MASTER EN SCIENCES INFORMATIQUES

Étude et réalisation d'un système auteur avec simulateur de systèmes d'équations différentielles intégré : profCOMP et simEDO

Léonard, Claude; Marchand, Olivier

Award date:
1988

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix - Namur
Institut d'informatique

Etude et réalisation d'un système auteur avec simulateur de systèmes d'équations différentielles intégré :

ProfCOMP et SimEDO

Mémoire présenté pour
l'obtention du titre de
licencié et maître en informatique

par Claude Léonard et
Olivier Marchand

Promoteur : Madame Monique Noirhomme-Fraiture
Directeur : Monsieur Jorge Muniz Barreto

Année académique 1987 - 1988

Résumé

Ce mémoire développe la conception et la réalisation de deux logiciels d'Enseignement Assisté par Ordinateur (EAO). Le premier, ProfCOMP, fait partie de la classe des systèmes auteurs et peut être utilisé dans l'enseignement de branches très diverses. Son originalité : il inclut un simulateur de systèmes d'équations différentielles ordinaires du premier ordre. Sur ce plan, les modèles créés sont totalement compatibles avec ceux du second logiciel SimEDO, outil entièrement spécialisé dans la simulation de tels systèmes, à ce titre entièrement utilisable aussi bien par des étudiants (EAO) que par des scientifiques.

Abstract

This dissertation relates the conception and realization of two Computer Assisted Instruction (CAI) software. The first one, ProfCOMP, is an author system which may be used in the teaching of various courses. Its originality : it includes an ordinary differential equations simulator. On this level, the SimEDO models are wholly compatible with those of ProfCOMP. SimEDO, specialized in simulation of these systems, is helpful either for students (CAI) or for scientists.

REMERCIEMENTS

Nous tenons à remercier Madame M. Noirhomme-Fraiture d'avoir accepté la promotion de ce mémoire, de même que Messieurs J.M. Barreto et S.D. Antunes, tout d'abord pour le stage très enrichissant que nous avons pu effectuer à l'Instituto Superior Técnico de Lisboa, ensuite pour les nombreux conseils qu'ils nous ont prodigués lors de la réalisation de ce travail.

N'oublions pas nos parents respectifs qui nous ont soutenus durant toutes ces études.

Nous remercions également Mademoiselle R.L. Pagano qui a testé les logiciels pour nous faire part de ses critiques.

Enfin, nous exprimons notre gratitude à Maria da Luz Faria, qui nous a si chaleureusement accueilli dans sa maison d'Encarnaçao, de même qu'aux étudiants qui ont facilité notre intégration à l'IST et dans la société portugaise.

Table des matières

Annexes

CHAPITRE 1
INTRODUCTION

Section 1
Cadre général du mémoire

Le but premier était de **rendre opérationnel, utilisable réellement par un enseignant, un logiciel d'enseignement assisté par ordinateur (EAO)**. Spécifier le résultat final s'avérait fort difficile. Avant l'étape de programmation, nous pouvions juste en tracer les grandes lignes et les grands principes de réalisation. Le logiciel devait être un moyen d'aborder l'EAO. Nous l'avons défini par rapport à ce qui existait déjà dans ce domaine : il devait être plus convivial, abordable par tout enseignant, même non spécialiste en informatique.

Par contre, l'aspect "matériel utilisé" comptait beaucoup moins. Nous ne devions pas dépendre des contingences matérielles actuelles. Nous voulons dire par là que nous ne pouvions réduire nos ambitions, par exemple à cause de la capacité des disquettes : la limite des 360 K du matériel actuel n'est que temporaire, de nouveaux drives aux capacités plus importantes s'imposeront comme standard du marché.

Toutefois, une espérance à la base du projet : diminuer de façon considérable le temps nécessaire à la création d'une leçon d'EAO. Comme une telle leçon comporte au moins du texte, des questionnaires et éventuellement des graphiques, la conception des outils permettant de réaliser ces différentes parties s'imposait. Mais nous voulions également rendre le projet original, en offrant, en plus, la possibilité de concevoir comme parties de leçons des simulations de systèmes d'équations différentielles ordinaires du premier ordre. Le logiciel, valable pour tous les types de cours, s'adapte particulièrement à l'électricité, la régulation, la résistance des matériaux,... , domaines où la simulation a énormément d'importance.

Objectif majeur : la transparence. Un des problèmes principaux des logiciels d'EAO est que l'enseignant doit souvent exécuter lui-même la leçon pour se rendre compte de

ce que l'étudiant voit à l'écran. Nous espérions supprimer cet inconvénient, en permettant au professeur de visualiser directement ce que verrait l'étudiant en suivant son cours.

Autre objectif : permettre l'erreur. Nous ne pouvons considérer l'utilisateur potentiel, en l'occurrence un professeur, comme professionnel de l'informatique. C'est pourquoi il faut le guider, vérifier tout ce qu'il frappe au clavier, l'empêcher de se "planter". Lui donner la possibilité de rectifier le travail terminé, de revenir en arrière dans le dialogue permettant la création d'une leçon. Une touche unique à travers tous les programmes lui offre cette chance, plutôt que d'incessantes demandes de confirmation qui alourdissent le dialogue.

Ce dialogue lui-même se doit d'être le plus simple possible, et pour cela il faut faciliter les choix faits par l'enseignant. Prévoir un système de valeurs par défaut est donc essentiel. Cela permet au professeur, dans bien des cas, de n'avoir qu'à appuyer sur **RETURN** pour confirmer les valeurs déjà affichées.

Nous voulons également que lors de l'exécution de la leçon par l'étudiant, plusieurs parties de cette leçon puissent cohabiter à l'écran. Un mécanisme de fenêtrage s'impose donc, mais au départ nous envisagions de découper statiquement l'écran, et d'affecter une fois pour toutes à chacune des parties ainsi obtenues le type d'information qu'elle contiendrait : du graphisme, du texte, ... Nous avons très vite abandonné cette idée pour laisser à l'enseignant le choix de la position et de la taille de chaque fenêtre.

C'était beaucoup et peu à la fois : il était très difficile d'envisager quel serait le résultat. Il s'agissait réellement d'une expérience, d'un ballon d'essai.

Le maître-mot de ce projet allait être la **transparence**.

Section 2 : Le laboratoire de deuxième licence

Une première version du logiciel a donc été réalisée dans le cadre d'un laboratoire de deuxième licence, durant l'année académique 86-87. Cinq étudiants (Sophie Jacob, Jean-Marc Verdure, Marc Paring, Claude Léonard et Olivier Marchand) se partagèrent le travail, sous la direction de Mme Noirhomme et de M. Barreto.

Il n'y avait alors aucune base sur laquelle s'appuyer. Ayant dégagé les grands principes déjà exposés, l'équipe s'est lancée dans la réalisation du projet. Les difficultés pour mener à bien un tel travail dans le cadre d'un laboratoire sont grandes : le temps que l'on peut y consacrer est relativement limité, car le projet doit se développer en parallèle avec les cours. De plus, c'est le premier programme original à réaliser en groupe important. Il n'est pas simple de s'organiser pour que la coopération soit efficace. Le travail a été réparti entre les différents membres du groupe, et ce n'est que lors de la mise en commun réelle, du regroupement du code en un seul programme, que nous nous sommes aperçus du relatif manque d'homogénéité du résultat.

Nous avons également dû faire une autre constatation : il est très difficile de garder une vue d'ensemble du projet lorsqu'on procède de cette façon. Certaines choses étaient ainsi très bien réalisées, mais sans s'intégrer correctement dans le reste du logiciel. Un autre problème : le manque de communication entre les membres de l'équipe lors de l'étape de programmation. Des difficultés ont ainsi été contournées, ou mal solutionnées, alors que la mise en commun des problèmes permet bien souvent de les résoudre plus rapidement et plus efficacement.

La simulation posait également problème : l'équipe a rencontré énormément de difficultés pour maîtriser le sujet. Les outils utilisés ne permettant pas de saisir les équations fournies par le professeur et de les évaluer directement sans transformation préalable, nous avions dû faire l'impasse sur un des nos objectifs de l'enseignant non

informaticien. En effet, nous lui demandions de compléter un programme standard avec son sytème d'équations, puis de le compiler. Chaque modèle prenait la forme d'un programme indépendant, sa création dans le logiciel d'EAO proprement dit se résumant à fournir quelques renseignements secondaires.

Cette solution posait problème : le passage entre exécution de la leçon et exécution de la simulation, et retour. Si le passage "aller" a pu être maîtrisé relativement rapidement, nous n'avons jamais pu obtenir un retour correct vers la leçon.

Le prototype réalisé était donc "bancal" : pas tout à fait sans erreur, ne répondant pas entièrement aux objectifs, pas très achevé, peu homogène. Mais il avait le mérite d'exister, et de représenter un essai dans une direction encore peu explorée jusqu'alors. Il était suffisamment démonstratif pour prouver que l'on était sur la bonne voie, et pouvait à présent servir de base pour un travail complémentaire. Avoir un tel point de départ est un avantage non négligeable, car cela permet d'utiliser le programme pour voir ce qu'il faudrait modifier ; de le montrer à un utilisateur potentiel, pour l'aider à exprimer ses désirs.

Dès cet instant, nous possédions donc une base importante, mais qu'il était nécessaire d'affiner, de compléter, de corriger pour le rendre réellement utilisable par des professionnels de l'enseignement. C'est alors que fut prise la décision d'en faire un mémoire. Le travail étant plus simple pour des gens ayant fait partie de l'équipe du laboratoire, puisque cela réduisait le temps de prise en main du programme, de prise de contact avec le domaine, nous nous sommes proposés pour prolonger notre travail de l'année précédente.

Section 3
Stage à Lisbonne

Pour qu'un stage soit réellement enrichissant, il fallait qu'il nous permette de rencontrer un utilisateur potentiel, réellement intéressé par le développement d'un tel projet. Cette rencontre devait nous permettre d'entendre de nouvelles idées, un nouveau point de vue sur le problème.

C'est ainsi que M. Barreto nous a proposé de partir à Lisbonne, à l'INSTITUTO SUPERIOR TECNICO, dans le département de génie mécanique, laboratoire de systèmes et machines, sous la direction de M. Antunes. Situation vraiment idéale, puisque M. Antunes a déjà longuement travaillé sur le problème de la simulation. Nous bénéficiions donc aussi de son expérience incomparable dans le domaine.

Nous nous sommes vite rendus compte que certaines parties de nos programmes étaient à refaire totalement. Cela a pris beaucoup de temps, trop sans doute. Après quoi, sur les conseils de M. Antunes, nous avons complètement modifié la partie simulation, pour qu'elle respecte enfin notre objectif d'être utilisable par un enseignant non informaticien. Pour cela, nous avons écrit un "compilateur" d'expressions, afin de les traduire en une structure qui facilite leur évaluation : nous avons donc refait dans notre programme une partie du travail que réalise normalement le compilateur.

Ayant obtenu une nouvelle version de la simulation, par cette approche totalement différente, M. Antunes nous a demandé d'en faire également un nouveau logiciel. C'est ainsi qu'est né SimEDO, venu doubler le logiciel d'EAO ProfCOMP. Ce logiciel de simulation a été étoffé petit à petit, pour en augmenter les possibilités. Tout cela, grâce à l'assistance de M. Antunes, pour qui ce programme représentait réellement un outil dont il envisageait l'utilisation dans le cadre de ses cours.

ProfCOMP et SimEDO ont alors évolué en parallèle, pour en arriver aux versions actuelles. Nous ne serions jamais arrivés à ce résultat sans ce stage de 3 mois à Lisbonne.

Section 4
Brève description des chapitres

Nous allons donc développer, dans ce mémoire, ce que sont réellement nos deux logiciels SimEDO et ProfCOMP. Mais avant cela, nous allons faire un bref résumé du contexte de l'EAO. Il est en effet important, lorsqu'on réalise un logiciel, d'arriver à le situer par rapport à d'autres produits existants. Nous n'avons pas voulu faire une étude exhaustive de l'histoire et de l'état de l'art en EAO; mais cela nous permettra de préciser par la suite les avantages de nos logiciels, et aussi leurs inconvénients.

ProfCOMP - PC_CREAT

Ensuite, nous détaillerons nos deux logiciels. Tout d'abord, ProfCOMP, dans sa partie "création d'une leçon" (PC_CREAT) : quelles en sont les idées générales, les spécifications ? Puis nous expliquerons comment un enseignant doit procéder pour créer une leçon. Cela passe par la découpe en noeuds de la matière à enseigner. Mais qu'est-ce qu'un noeud ? Nous préciserons aussi quels sont les types de noeuds existants, et comment créer chacun d'eux.

Pour le noeud "texte", nous dirons pourquoi nous avons dû avoir recours à un éditeur et un formateur distincts, quelles sont les fonctionnalités de l'éditeur, comment nous en sommes arrivés à la version actuelle. Pour le formateur, sans aller dans les détails, nous donnerons la liste des commandes qu'il autorise, ainsi que son algorithme général.

Ensuite, les noeuds de type "graphique" : comment les créer ? quelles sont les fonctions disponibles ? comment réaliser une animation ? Nous en développerons également l'algorithme le plus intéressant, celui qui assure la fonction "zoom".

Pour les noeuds "questions-réponses", nous établirons un "algorithme" du dialogue nécessaire à la conception d'un questionnaire. Nous justifierons nos choix en même temps que nous établirons les principaux concepts.

La simulation ne sera abordée ici que très superficiellement, et principalement sous le point de vue de

l'utilisateur : le dialogue qui en permet la création sera explicité, tout en introduisant les concepts les plus importants.

Enfin, nous expliquerons comment "relier" tout cela pour établir la leçon proprement dite : création et compilation de la structure de la leçon.

ProfCOMP - PC_EXEC

Nous aborderons aussi l'exécution de leçons, en expliquant comment elle se déroule, et en détaillant le contenu du fichier "rapport" créé automatiquement.

SimEDO

Après, nous développerons en détail la conception du module simulation. Mais nous commencerons d'abord par en présenter l'historique, c'est-à-dire la forme sous laquelle elle existait l'an dernier, les difficultés rencontrées, et les inconvénients de cette solution. Après quoi, nous aborderons les aspects plus techniques. Tout d'abord, nous expliciterons quels sont les types d'expressions admises pour les simulations : leur syntaxe précise, quels en sont les opérandes, quels sont les opérateurs permis.

Mais ces expressions nous sont données par l'utilisateur sous forme de chaînes de caractères : nous expliquerons comment nous les analysons, et sous quelle forme nous les représentons pour permettre leur utilisation rapide. Ensuite, comment nous les évaluons, et quelles sont les méthodes de résolution d'un système ? Il reste toutefois des limites à ce que l'on peut simuler ; nous essayerons de les expliquer. Nous donnerons aussi un exemple pratique de simulation.

Problèmes rencontrés

Le développement de tels logiciels ne se fait pas sans accrocs, nous avons rencontré des difficultés, certaines attendues, d'autres non, aussi bien avec le matériel utilisé (tant hardware que software) que dans la version de départ, résultat du laboratoire. Nous survolerons ces problèmes rapidement.

Domaines d'utilisation

Au fur et à mesure que nous avancions dans le développement, nous nous sommes aperçus, avec l'aide de M. Antunes, que ce que nous prenions uniquement pour des logiciels d'EAO pouvaient servir aussi à autre chose, dans d'autres domaines. Nous tenterons ensuite un jugement critique de notre travail, et donnerons une liste non exhaustive des améliorations possibles, qui rendraient encore plus performants SimEDO et ProfCOMP.

Avertissement

Nous avons joint en annexe les manuels d'utilisation de nos deux logiciels. Ils ont été conçus pour faciliter le contact avec les programmes, mais sont partiellement redondants avec ce mémoire, qui lui se veut plus "technique".

CHAPITRE 2

LE CONTEXTE DE L'EAO

Section 1

Introduction à l'EAO

L'Enseignement Assisté par Ordinateur est une des nouvelles technologies qui rencontre le plus grand intérêt auprès des professeurs. L'EAO se caractérise par l'utilisation de l'ordinateur comme support didactique, pouvant convenir à diverses formes et matières d'enseignement.

L'enseignant ne doit pas se sentir menacé par cet envahisseur qui viendrait lui prendre sa place et résoudrait tous les problèmes de l'enseignement. L'expression d'assistance est suffisamment explicite de ces limites et de l'importance de l'enseignant dans toute approche de l'enseignement. Nulle part encore des "machines à enseigner" n'ont remplacé un professeur. Au contraire même, l'ordinateur doit être considéré comme un outil qui complète la panoplie de l'enseignant et qui rend possible une amélioration et une rénovation de sa pédagogie. Le professeur ne doit pas non plus craindre un manque de compétences informatiques car les spécialistes en la matière devront proposer des logiciels adaptés à l'enseignement, et qui tiennent compte des utilisateurs. Pour que l'enseignement dispensé soit efficace, l'enseignant doit être volontaire.

L'introduction des nouvelles technologies dans les écoles ouvre des horizons nouveaux pour améliorer la qualité de l'enseignement dispensé : l'enseigné doit en être le principal bénéficiaire [Simon] :

- son rôle de spectateur dans l'école traditionnelle est tout à fait dépassé. L'étudiant apprend et devient actif en montrant qu'il a compris en répondant aux questions de l'ordinateur.
- Le didacticiel proposé doit s'adapter aux capacités de l'élève et plus particulièrement à son rythme de travail et d'assimilation. Cette individualisation du rythme de progression dans la matière résoud le problème de perte de

temps mais accentue les différences entre les élèves performants et ceux qui ont des difficultés d'apprentissage. Le professeur doit se rendre compte, en temps réel, de l'évolution de chaque élève et combler ces différences en jouant son nouveau rôle de précepteur personnel. Les liens professeurs-élèves se renforcent.

- On ne communique pas avec un ordinateur comme on communique avec un être humain. Les élèves prennent l'habitude de formuler leurs demandes de façon précise et rigoureuse pour se faire comprendre par leur outil de travail.
- La plupart des apprenants sont intéressés et motivés à utiliser l'ordinateur, à condition que les didacticiels leur laissent une part d'initiative. Il faut éviter les leçons dont le schéma d'exécution est prévu d'avance et où l'élève n'a aucune liberté (EAO directif).
- La créativité, l'adaptabilité peuvent s'exprimer dans les problèmes proposés.
- L'élève détermine lui-même le rythme de son travail. Il progresse dans l'exercice à la vitesse qui lui convient, peut prendre le temps de réfléchir ou de consulter des documents ou au contraire, avance très vite s'il connaît ou croit connaître les bonnes réponses.

L'EAO n'apporte pas que des avantages dans la formation et il faut veiller à éviter certains écueils :

- Si certains sont motivés par l'interaction avec la machine, d'autres n'apprécieront pas, notamment à cause des difficultés matérielles d'utiliser un clavier, une souris,...
- L'autre extrême est de voir certains élèves passionnés par la machine et constamment occupés à essayer de la maîtriser, négligeant tout contact avec leurs camarades de classe.
- L'EAO ne doit pas être uniquement le moyen de décharger le professeur des tâches répétitives et sans intérêt tel le contrôle des connaissances.

L'enseignant qui utilise ces moyens acquiert un double rôle : celui de concevoir les didacticiels qu'il utilisera et celui de suivre les apprenants, en accordant la priorité

aux élèves qui ont des difficultés d'apprentissage.

Section 2

La réalisation d'un didacticiel

On distingue 3 moyens classiques pour créer un programme d'EAO [Boyd]. Nous n'aborderons pas ici les systèmes "intelligents" d'EAO [Freed], [LFBN1], [LFBN2] ou [Athéna].

1. L'utilisation d'un langage de programmation

Parce qu'il permet d'acquérir une méthode de pensée exigeant rigueur et précision dans le raisonnement, l'apprentissage d'un langage de programmation peut être fort utile aux étudiants.

Oublions quelque peu cette perspective pour nous consacrer au professeur, créateur de didacticiels d'enseignement assisté par ordinateur à partir d'un langage de programmation traditionnel tel BASIC, PASCAL,... Cette approche astreint l'enseignant à maîtriser les fondements de l'informatique et le langage de développement. En outre, elle lui impose d'élaborer son cours de toutes pièces, en prévoyant toutes les possibilités d'exécution, de réponses possibles, ...

Un tel mode de conception de didacticiels comporte beaucoup d'inconvénients. Epinglons-en quelques-uns :

- Des études menées en France, suite à l'opération d'introduction de micro-ordinateurs dans les lycées, indiquent qu'une heure de cours demande au moins une centaine d'heures de mise au point (analyse, édition, compilation et débogage). Ce chiffre énorme exige une collaboration plus large entre écoles pour permettre une diffusion à grande échelle des didacticiels existants et ainsi éviter aux professeurs de concevoir des programmes que d'autres ont déjà réalisés. Malheureusement, la valeur pédagogique d'un grand nombre de programmes laisse à désirer.

- Beaucoup de professeurs, analphabètes informatiques, adopteront une attitude de refus face à la nécessité de devoir apprendre les rudiments de l'informatique et un

langage de programmation et de ce fait, utiliseront des programmes tout faits mais qui concordent rarement avec leurs besoins.

- Une leçon se résume souvent à un enchaînement séquentiel de textes, de questions et parfois de dessins. Le schéma d'exécution de la leçon, conçu par le professeur, laisse rarement l'initiative à l'élève d'orienter la leçon comme bon lui semble.

- Dans la réalisation d'un tel programme, l'enseignant doit prévoir toutes les réponses potentielles, leur traitement et le cheminement consécutif à ces réponses, ce qui peut devenir un véritable casse-tête.

- Les techniques d'analyse de réponse sont très pauvres, ce qui veut dire que, si l'étudiant commet une faute d'orthographe en tapant la réponse à la question, elle sera interprétée comme fausse. Les tolérances orthographiques sont en général très limitées sinon inexistantes. Ce problème peut amener le professeur à rédiger ses questions sous forme de choix multiples, où l'élève n'a plus qu'à taper la lettre qu'il estime correspondre à la bonne réponse. Ce type de question constitue indiscutablement un appauvrissement du dialogue entre l'ordinateur et l'apprenant, et peut être la cause d'un manque de motivation de la part de ce dernier.

- Le code et les données sont mélangées et une modification de la leçon est souvent malaisée et débouche sur une révision complète du programme.

Toutes ces faiblesses ne sont pas faites pour encourager les novices de l'informatique à se lancer dans le développement de didacticiels. Les langages de programmation ne sont pas assez spécifiques à leur utilisation par des non spécialistes désireux de créer des logiciels d'EAO. Un exemple de réalisation : [Péd].

2. Utilisation d'un langage auteur

Les langages de programmation traditionnels (BASIC, PASCAL, ...) sont de trop bas niveau pour réaliser les didacticiels nécessaires à l'enseignement, c'est-à-dire que

les instructions de base ne sont pas adéquates pour rédiger des programmes d'EAO. La solution passait par la mise au point de langages spécifiques à l'enseignement : les langages auteurs. Ces langages proposent des macros qui génèrent automatiquement les instructions les plus couramment utilisées pour l'établissement d'un EAO. Les progrès, certes évidents, ne dispensent toujours pas le professeur d'une connaissance des principes de programmation et de l'étude d'un langage, même si ce dernier est mieux adapté à l'EAO.

Pour acquérir une bonne maîtrise des outils mis à disposition, le temps d'apprentissage demandé est comparable à celui demandé par un langage classique, sinon plus important encore.

Les programmes qui constituent le mode auteur permettent à celui-ci de créer, modifier, gérer et encoder ces didacticiels. L'auteur dispose de verbes pour désigner les actions du didacticiel, considéré comme un programme. Les caractéristiques varient très fort d'un langage à l'autre mais voici, dans les grandes lignes, de quoi dispose l'auteur pour créer son didacticiel :

* Les instructions, dont le nombre dépend de la puissance qu'offre le système (exemple : EGO propose une vingtaine d'instructions alors qu'ARLEQUIN en offre environ cent-vingt). Quelques instructions classiques :
  + Afficher du texte à l'écran,
  + Effacer l'écran,
  + Indiquer le début d'un nouveau problème,
  + Attendre une réponse,
  + Comparer une réponse de l'étudiant à la bonne réponse. Ici, l'auteur dispose de plusieurs modèles pour analyser sa réponse :
    - Comparaison d'un radical à la réponse.
    - Utilisation de joker(s) dans la réponse type.
    - Définition de la réponse par plusieurs mots à retrouver dans la réponse de l'étudiant,...
  + Poursuivre à un autre endroit de la leçon,
  + définir une variable entière, réelle, booléenne, chaînes,...

+ Effectuer un calcul ou une fonction mathématique,...
+ Lire un graphique.

* Les variables de type entier, réel, caractère, booléen ou tableau d'éléments de types précédents peuvent être définis.

* Les opérations :
  + de tirage aléatoire,
  + arithmétiques,
  + sur les chaînes,
  + logiques,
  + correspondant aux fonctions mathématiques.

* Les structures algorithmiques :
  + alternative,
  + itération,
  + procédure,
  + récursivité.

En ce qui concerne le réalisation de scènes textuelles, certains langages offrent un éditeur d'écran, d'autres des instructions.

Exemple : le langage PILOT (Programmed Inquiry Learning Or Teaching) offre des instructions d'affichage de texte [Pilot].

```
tx:
t:        Le langage PILOT permet d'afficher
t:        du texte en utilisant la commande "t:".
t:        Dans cet exemple, la première commande
t:        "tx:" efface le contenu de l'écran avant
t:        d'afficher ce message.
```

Figure 2.1 - Affichage de texte dans PILOT

Un éditeur graphique propose des formes définies (traits, courbes, rectangles, cercles, caractères, ...) et des fonctions de dessin pour la réalisation des scènes graphiques. Une instruction dans le programme permettra d'afficher ces dessins créés par l'éditeur. Une animation graphique peut être réalisée par une succession d'images avec temporisation.

Les problèmes de temps de mise au point et de compétences informatiques ne sont toujours pas résolus. Toutes les critiques exprimées sur ces problèmes dans l'utilisation d'un langage de programmation sont toujours

valables ici.

3. Utilisation d'un système auteur

Un système auteur est un programme ou un ensemble de programmes qui permet(tent) au professeur de développer un cours assisté par ordinateur, sans devoir connaître quoi que ce soit en programmation, grande nouveauté par rapport aux langages précédents. L'ensemble des utilisateurs potentiels s'élargit considérablement puisque tous les enseignants qui ne possèdent pas les aptitudes de programmeurs sont succeptibles de faire connaissance avec les systèmes auteurs et l'EAO.

La tâche de création d'un didacticiel est rudement simplifiée par rapport à l'utilisation des langages précédents dans la mesure où le système limite le travail à la définition de ce que l'enseignant VEUT OBTENIR. Le créateur ne doit plus écrire des programmes instructionnels car l'étape COMMENT FAIRE n'existe plus. Avec la disparition des contraintes informatiques, l'enseignant peut concentrer ses efforts sur les objectifs pédagogiques de ses cours. La réalisation d'un cours se fait de façon totalement interactive et conviviale et le temps de mise au point diminue considérablement.

Dans de tels systèmes, le code du programme et les données sont clairement séparées, ce qui nous laisse l'espoir que les systèmes auteurs conduiront à des cours plus transportables.

La technique d'analyse des réponses y est sophistiquée et puissante : elle permet de définir un nombre quelconque de bonnes et de mauvaises réponses, avec pour chacune, le branchement et le commentaire adéquats.

On distingue plusieurs types de systèmes auteurs sur le marché. En voici 2 exemples [Kears]:

- "Form-driven authoring systems" : l'auteur utilise des formes prédéfinies qui contiennent l'information nécessaire pour créer les écrans de la leçon. Les formes sont des modèles pour présenter un texte, poser une question,... L'auteur n'a qu'à remplir les trous, c'est-à-dire introduire

l'énoncé d'une question, ses réponses bonnes ou mauvaises, le texte à afficher,...

Exemple : Définition d'une question dans le système Private Tutor :

```
================== DEFINITION DE L'ECRAN ==================
Nom de l'écran : .......... (maximum 10 caractères)
Ecran n° ..... de la leçon ..... (maximum 5 caractères)
Faut-il coter cette question ? ....
Indiquer N si la question ne doit pas être reprise dans la cote globale de la leçon

============== TEXTE, GRAPHIQUES et QUESTION ==============
_____________________________________________________________1
                              :
                              :
                              :
_____________________________________________________________15

==================== REPONSES CORRECTES ====================
Liste des réponses correctes. Les réponses équivalentes sont séparées par des blancs. Dans les
réponses de plusieurs mots, les mots sont groupés en une seule chaine de caractères.

_____________________________________________________________A
```

Figure 2.2 - Définition d'un écran dans Private Tutor

Quand l'utilisateur a conçu tous ces écrans, il lui reste à définir la structure de sa leçon : l'enchaînement des écrans.

- "Prompting systems" : le développement d'un cours se déroule de manière interactive. Le système pose des questions, à la fois sur le contenu et sur la logique du cours, et le travail du professeur consiste à répondre à ces questions.

Exemple : Le système COURSEMAKER

```
WHAT IS THE NAME OF YOUR COURSE ?
nom_leçon

TYPE YOUR INTRODUCTION, WHEN FINISHED PRESS ENTER TWICE
Ce texte apparaîtra comme texte introductif de la leçon

TYPE YOUR QUESTION ?
Enoncé de la question

IS YOUR QUESTION MULTIPLE CHOICE (1), TRUE FALSE (2),
COMPLETION (3) OR MATCHING (4)? TYPE 1, 2, 3 OR 4.
3

ENTER A LIST OF POSSIBLE CORRECT ANSWERS, PRESS ENTER TWICE
WHEN DONE
bonne_réponse1 bonne_réponse2 bonne_réponse3

ENTER A LIST OF POSSIBLE INCORRECT ANSWERS, PRESS ENTER
TWICE WHEN DONE
mauvaise_réponse1 mauvaise_réponse2
 ...
```

Figure 2.3 - Définition d'une question dans Coursemaker

D'autres exemples sont décrits dans [Leibl]

Section 3

Position du logiciel dans ce contexte

Soucieux d'atteindre une gamme d'utilisateurs la plus large possible et de tenir compte de la spécificité de chacun, nous ne pouvions proposer un langage, difficile à maîtriser et qui demande un apprentissage long et spécifique, surtout aux personnes non-informaticiennes.

Dans certains domaines, à côté des leçons, exercices et interrogations, l'expérimentation s'avère être un procédé essentiel du processus d'enseignement. Malheureusement, la plupart du temps, les difficultés pratiques (ou autres) empêchent la réalisation d'expériences. Dans cette perspective, l'ordinateur offre d'énormes possibilités tant au niveau individuel que pour illustrer la matière à enseigner, et nous avons voulu en profiter : le logiciel permet de simuler des systèmes d'équations différentielles ordinaires du premier ordre.

La solution adoptée offre un ensemble d'outils, d'utilisation aisée, pour créer toutes les parties de la leçon. Chaque outil est spécialisé dans un domaine précis : texte, graphisme, questionnaire ou simulation.

Un autre aspect important concerne la pédagogie. Pour être apprécié, notre logiciel doit être aussi souple que possible pour que l'enseignant puisse faire concorder les leçons qu'il crée à sa méthode pédagogique. Après la conception de toutes les parties de la leçon, l'enseignant peut la structurer selon une logique linéaire (une leçon = une suite séquentielle d'exécutions de parties de la leçon) ou conditionnelle (approche pédagogique de plus haut niveau). Les branchements conditionnels dépendent soit directement d'un choix de l'étudiant, soit de ses résultats obtenus pour les parties antérieures de la leçon.

Les programmes de création et d'exécution d'une leçon sont clairement séparés des données de la leçon (le contenu des différentes parties). Aucune codification n'est nécessaire après la création, le programme d'exécution travaille directement sur les données de la leçon.

Chaque exécution fournit un rapport complet qui permet au professeur d'apprécier le niveau de connaissances de l'apprenant.

Comme le logiciel dispense tout utilisateur de connaissances spécifiquement informatiques, à partir des considérations ci-dessus, nous proposons de classifier le logiciel dans les systèmes auteurs.

CHAPITRE 3
ProfCOMP

Section 1
Création d'une leçon (PC CREAT)

1. Introduction

Tout le logiciel ProfCOMP (**Profes**sor **COMP**atible) repose sur quelques principes très simples, que nous avons essayé de respecter afin d'en faciliter l'utilisation par n'importe qui.

Tout d'abord, une certaine transparence : il s'agit de montrer, lors de l'élaboration de la leçon, quelle sera la présentation à l'écran lors de son exécution par l'étudiant. Le but est d'éviter toute mauvaise surprise une fois le travail de création terminé, et ainsi de réduire les allers-retours entre la création et l'exécution de leçons avant d'aboutir à une version correcte.

Autre objectif : limiter le nombre de valeurs à introduire, grâce à un système de valeurs par défaut. Bien sûr, cela n'est pas toujours évident. De plus, dans la mesure du possible, nous avons privilégié les choix entre différentes options - sélectionnables grâce aux flèches - pour éviter l'utilisation de la partie classique du clavier.

Enfin, dans la plupart des cas, nous tolérons l'erreur et le retour en arrière (grâce aux touches de fonction). Cette facilité évite de rééditer 36 fois la même chose avant d'aboutir au résultat désiré.

Le logiciel est bien sûr installable sur une large gamme de matériels (grâce à l'utilitaire INST_EAO), et la plupart des messages figurent dans des fichiers indépendants, ce qui permet de les traduire éventuellement dans une autre langue (grâce à TRANS_ME) avant d'utiliser les programmes. L'explication concernant ces différents utilitaires se trouve au point 8 de l'annexe A (Manuel d'utilisation de ProfCOMP).

Toute leçon créée grâce à ProfCOMP se compose de plusieurs parties distinctes : du texte, des graphiques, des simulations, et des questions posées à l'étudiant. Le logiciel de création de leçons, PC_CREAT, comprend donc évidemment tous les outils nécessaires à la conception de ces différentes parties de leçons. Figurent ainsi dans le logiciel un éditeur et un formateur de textes pour créer les textes à présenter à l'étudiant, un éditeur graphique pour créer les graphiques, un éditeur de questionnaires, et enfin un "éditeur" de simulation qui saisit toutes les données nécessaires pour définir un modèle de simulation. Chacun de ces outils sera développé séparément par la suite. Retenons néanmoins que l'éditeur graphique permet également de créer des animations graphiques.

Pour "relier" les différentes parties d'une leçon, nous avons développé un éditeur d'enchaînements. Un compilateur y est associé, pour créer une leçon exécutable.

Avant d'aller plus loin dans les explications, quelques commentaires sur ces outils :

- L'éditeur de textes permet de taper un texte quelconque, en l'utilisant comme une machine à écrire ou en faisant appel à des fonctions beaucoup plus puissantes de traitement de texte. Cet éditeur offre différents "niveaux" d'utilisation, qui correspondent à différents degrés de familiarité.
- Un formateur de texte y est associé pour assurer la présentation de ce texte à l'écran lors de l'exécution de la leçon. Pour améliorer encore cette présentation, le professeur peut introduire différentes commandes dans le texte lors de son édition ; ces commandes sont traitées lors du formatage.
- L'éditeur graphique est, lui aussi, utilisable à plusieurs niveaux, par un débutant comme par un utilisateur chevronné.
- L'éditeur de questionnaires permet, lui, de créer des questions de types très différents : soit des questions à choix multiples (type QCM), soit des questions ouvertes. Ces questionnaires permettront de vérifier les connaissances de l'étudiant et, éventuellement, d'adapter la suite de l'enseignement à son niveau de connaissances.

- Dans des domaines d'utilisation plus spécifiques (cours d'électricité, de circuits, ...), l'éditeur de simulation est un outil très important, pour définir un système d'équations différentielles et la représentation de sa résolution.
- L'édition d'enchaînements permet non seulement de relier les différents noeuds entre eux, mais aussi de jouer sur l'affichage, le contenu de l'écran, lors de l'exécution de la leçon.
- Pour adapter le logiciel à l'écran dont vous disposez, ou encore à vos goûts en matière de couleurs, une option du menu principal permet de modifier celles-ci de manière définitive.
- Une dernière option offre la lecture d'une partie du manuel d'utilisation du logiciel.

Toutes ces options sont accessibles à partir d'un premier menu. Les options sont sélectionnables dans ce menu soit par les flèches, soit par les chiffres (**1** pour la première option,...), soit par les touches de fonction (**F3** pour la troisième option), soit par leur initiale. Enfin, la touche **Home** positionne sur la première option et **End** sur la dernière. Chacun dispose ainsi du mode de sélection auquel il est habitué.

L'interface est homogénéisée sur l'ensemble du programme. En général, une touche de fonction aura toujours le même effet. Le professeur est réellement guidé durant toute l'élaboration de la leçon.

Voici les touches qui conservent une signification identique à travers tous nos programmes :

- **F1** pour revenir un pas, une étape en arrière. C'est cette touche qui facilite la correction des erreurs précédemment introduites dans la leçon.
- **F2** entraîne l'abandon. Suivant la position dans le programme, il s'agira de l'abandon de la fonction en cours ou de l'outil utilisé (retour au menu précédent).
- **INS**, lors de la saisie de chaînes de caractères, d'entiers ou de réels, fait passer du mode surfrappe ("overwrite") au mode insertion et vice-versa.

- <-, -> déplace le curseur sur la ligne en cours d'édition.
- **space** change l'option sélectionnée en cas de choix entre deux possibilités.
- **Backspace** ou <—| efface le caractère à gauche du curseur, tandis que **del** efface le caractère courant.
- **F10** permet d'accéder au menu lorsqu'il y en a un de disponible dans l'outil utilisé.

2. La découpe en noeuds

Nous avons déjà parlé de noeuds à plusieurs reprises. Mais qu'est-ce qu'un noeud ? Sa qualité principale est l'homogénéité. Un noeud est une partie de leçon homogène au point de vue de son contenu - c'est-à-dire qu'il n'est pas sujet à choix ou à interruption lors de l'exécution de la leçon - et, également, homogène du point de vue de son type : il doit être uniquement du genre graphique, texte, simulation, ...

L'enseignant doit commencer par mettre sur papier le plan de sa leçon. Plan dans lequel les différents noeuds sont clairement distingués, et dans lequel il peut y avoir des branchements (le plan peut - doit ? - ne pas être linéaire). Ce plan se présente sous forme de graphe. Comme chacun des noeuds doit pouvoir être réalisé à l'aide des outils évoqués au point 1 (introduction), on les qualifiera du nom de cet outil : on parlera donc de noeuds graphique, de noeuds texte, de noeuds simulation, de noeuds questionnaire ou questions-réponses, et encore de noeuds animation (crées à l'aide de l'éditeur graphique).

Le professeur définit ce graphe à l'aide de l'éditeur d'enchaînements. Certains passages entre noeuds peuvent se faire de manière conditionnelle, et d'autres sans aucune condition.

Exemple :

Pour débuter sa leçon, l'enseignant va rappeler la matière vue au cours précédent. Ce rappel sera obligatoire, quel que soit le niveau de l'élève. Il aura par exemple la forme d'un noeud texte (on peut tout aussi bien imaginer que le rappel existe sous la forme d'un questionnaire). Suivra

alors un petit questionnaire, pour vérifier les connaissances sur la matière censée connue.

Surviendra ensuite la première condition de l'enchaînement : si l'étudiant a obtenu plus de 80 % des points, il peut passer à la suite de la matière, composée de noeuds de toutes espèces. S'il a moins de 80 %, il devra d'abord "subir" un rappel plus détaillé de la matière précédente, également composé de toutes sortes de noeuds.

A la fin de la matière normale, un petit questionnaire testera à nouveau la compréhension de cette nouvelle matière. En cas de mauvais résultats à ce test, l'étudiant subira une seconde fois la leçon dans son ensemble. Cette leçon, mise sous forme de graphe, se présente ainsi :

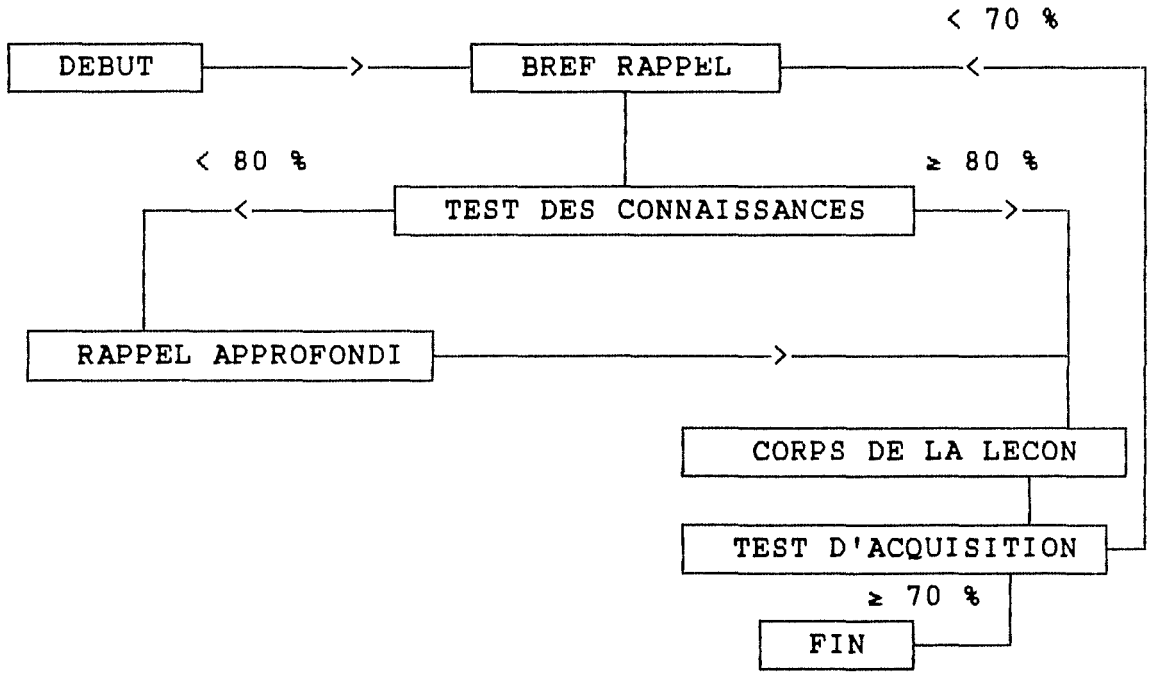


figure 3.1

Ceci n'est qu'un tout petit exemple de ce qu'on peut réaliser grâce à l'éditeur d'enchaînements. Nous détaillerons avec plus de précision le jeu d'instructions disponibles, et leur syntaxe exacte au point 4 de cette section (page 69).

Il reste toutefois un concept à développer avant d'aller plus loin dans les explications sur ProfCOMP. Il s'agit des fenêtres. A chaque noeud, va être associée une fenêtre. Une fenêtre est une partie de l'écran physique, à

la nuance près que plusieurs fenêtres peuvent se partager la même partie de l'écran. Dans ce cas, une seule des fenêtres sera visible (tout au moins dans leur partie commune). On dira alors qu'elle est au-dessus des autres. L'ensemble des fenêtres constitue une pile. La dernière fenêtre définie est au-dessus de la pile : elle est entièrement visible.

Chaque noeud est associé à une fenêtre. L'enseignant pourra en préciser la dimension, ainsi que la position. Lors de l'exécution d'une leçon, l'enseignant pourra s'arranger pour faire cohabiter plusieurs fenêtres à l'écran. Plusieurs primitives de manipulation de ces fenêtres lui seront offertes dans l'éditeur d'enchaînements.

3. Création des noeuds

3.1 Noeuds textes

3.1.1 L'éditeur (PC_EDIT)

Cette partie du programme est un peu particulière dans sa conception. En effet, nous nous sommes tout de suite rendus compte qu'il était impossible de réaliser nous-mêmes un éditeur efficace et agréable à utiliser, sans lui donner trop d'importance par rapport aux autres outils envisagés. Mais pour des raisons d'intégration de ces différents outils, il était tout à fait indispensable d'en implémenter un qui soit de plus "appelable" à partir du menu principal. Toutes ces considérations nous imposaient de trouver un éditeur existant, ainsi que ses sources, afin de pouvoir le modifier à notre guise. C'est pourquoi nous avons opté pour le Turbo Editor Toolbox de Borland, dont l'éditeur MicroStar forme la partie essentielle.

Nous avions donc tout à notre disposition pour réaliser notre éditeur à peu de frais, du moins nous l'espérions. Nous avons choisi de travailler uniquement à partir de l'éditeur MicroStar, de loin le plus complet du toolbox. La première étape, qui ne fut pas la plus aisée, a été de comprendre ce code, après quoi nous l'avons élagué pour supprimer toutes les fonctions qui nous semblaient inutiles dans le cadre d'un logiciel d'EAO.

Tout ce travail avait été réalisé une première fois dans le cadre du laboratoire de seconde licence, mais des fonctions intéressantes ayant été supprimées, et des erreurs graves introduites, nous avons préféré cette année le recommencer entièrement, plutôt que de perdre énormément de temps en "racommodages". Nous nous sommes en effet très vite aperçus qu'une erreur majeure du formateur avait été "corrigée" par une autre erreur, plus grave encore, dans l'éditeur (cfr pages 144-145).

Un avantage énorme que possède cet éditeur est le double système de commandes. Un utilisateur habitué peut en effet se servir de touches de contrôle, mais pour les autres un passage par des "pulldown menu" est possible. Cela facilite énormément l'apprentissage de l'éditeur, dont la plupart des commandes sont d'ailleurs celles que l'on retrouve dans WordStar.

Ce que l'on a conservé :

Il ne sert à rien de s'étendre très longuement sur ces fonctions. Ce sont toutes des fonctions très classiques, que l'on retrouve dans tous les éditeurs du commerce. Le système de menu les classe en différentes catégories.

Tout d'abord, les commandes de bloc : définir début et fin de bloc, copier ou déplacer le bloc défini, lire ou écrire un bloc dans un autre fichier, effacer un bloc ou encore "cacher" ou "montrer" le bloc courant (en utilisation normale, le bloc courant est affiché dans une couleur le différenciant par rapport au reste du texte).

Ensuite, il y a bien sûr les commandes de "find", "find and replace" ou "repeat last find" avec les options habituelles (ignorer majuscules/minuscules, remplacement dans tout le fichier, ...).

MicroStar posséde aussi des commandes de déplacement rapide : en début ou fin de fichier, en début ou en fin de bloc, vers la ligne n° x ou vers la colonne n° y, ou encore établir un "marker" ou se déplacer vers un "marker".

Quelques commandes de format de texte : mode insertion ou surfrappe, mode auto-indentation ou non, établir la largeur d'une tabulation, ou encore le nombre de "UNDO"

(**ESC**) que l'on accepte.

Enfin, toujours dans les commandes accessibles à partir des menus, les commandes de fichier : ouvrir un fichier de travail, le fermer, sauver le texte édité, le sauver sous un autre nom, afficher le répertoire courant, imprimer un fichier, le copier, l'effacer, le renommer ou encore sortir de l'éditeur pour retourner dans PC_CREAT (cfr infra, "Le lien entre PC_EDIT et PC_CREAT", page 30).

Quelques fonctions ne sont pas accessibles par le système de menu : se déplacer en début ou en fin de ligne courante (plus toutes les autres commandes habituelles de déplacement), effacer la fin de la ligne courante, effacer toute la ligne courante, insérer un caractère de contrôle, dérouler le texte vers le haut ou vers le bas, et aussi inverser des majuscules vers les minuscules et vice-versa.

Il y a encore d'autres fonctions disponibles, mais rien de vraiment original. La liste complète se trouve dans le manuel d'utilisation de ProfCOMP (cfr annexe A, page 11).

Ce que l'on a retiré :

MicroStar prévoyait la possibilité de travailler avec une ou deux fenêtres. Dans ce cas, l'écran était divisé en deux par une ligne horizontale. Ce système n'apportait rien dans le cadre d'un logiciel d'EAO. Il ne pouvait en aucun cas remplacer le formateur. Nous avons supprimé les options prévues (ouverture de la seconde fenêtre, sa fermeture, et la sélection d'une des deux fenêtres comme zone de travail) afin de simplifier au maximum ce qui doit rester un outil parmi d'autres.

Une autre option prévue était la correction orthographique. MicroStar prévoyait en effet l'interface avec un autre produit Borland, le Turbo Lightning. Comme nous ne disposions pas de ce logiciel, la suppression s'imposait, toujours dans le même but de simplification.

L'éditeur MicroStar de base offrait deux modes de travail différents : l'on pouvait en effet choisir de travailler en frappe kilométrique ("wordwrap") ou pas. Pour simplifier, et pour rapprocher notre éditeur de la majorité

des traitements de texte, nous avons choisi d'inhiber cette possibilité en fixant l'éditeur au mode "wordwrap". Cela nous a posé quelques problèmes, car certaines options ne fonctionnaient pas normalement dans ce mode (par exemple l'option print affichait le caractère ↓ en fin de ligne, s'il ne s'agissait pas en même temps d'une fin de paragraphe).

Il existait d'autres options permettant de fixer le format du texte introduit. Toutes n'ont pas été enlevées, mais les valeurs par défaut ont souvent été modifiées. L'option auto-indentation, par exemple, était à ON. Elle est maintenant à OFF, car ce mode est surtout utile pour la frappe de programmes, ce qui n'est pas souvent le cas dans le cadre de l'EAO.

La fixation des marges gauche et droite a également été otée. En effet, ces marges n'ont qu'une utilité de présentation du texte lors de l'édition et ne nous permettaient en aucun cas de supprimer le formateur indépendant, destiné, lui, à adapter le texte à une fenêtre lors de l'exécution de leçons.

L'éditeur MicroStar se rapprochant très fort d'un traitement de texte, il possédait bien sûr la possibilité de reformater un paragraphe. Cette option ne fonctionnant pas très bien et, encore une fois, ne nous permettant de toutes façons pas de nous passer d'un formateur indépendant, nous l'avons également inhibée.

Autre option fort liée aux précédentes et qui a été supprimée, le centrage de la ligne courante. En effet, sa place n'était plus dans un éditeur mais bien dans le formateur y-associé. Tout comme pour la fixation des marges, cette option a donc glissé de l'éditeur vers le formateur.

La plupart des options rassemblées sous le nom de "text format" ayant disparu, ou en tout cas les valeurs par défaut modifiées, la dernière option de cette rubrique, "save settings", nécessaire pour garder "définitivement" les options par défaut modifiées, ne justifiait plus sa raison d'être.

Dernière option inhibée : le changement du répertoire courant. ProfCOMP nécessite la fixation une fois pour toutes (INST_EAO - cfr annexe A, point 8.1) d'un répertoire pour les données, les textes de l'éditeur doivent bien sûr se retrouver à la même place. PC_EDIT, partie intégrante de ProfCOMP au même titre que PC_CREAT ou PC_EXEC, doit donc lui aussi être réinstallé grâce à INST_EAO lorsqu'on désire modifier le répertoire réservé aux données.

Toutes ces modifications ont simplifié l'éditeur, l'ont réduit à son rôle réel d'éditeur existant dans le cadre global d'un logiciel d'EAO (ProfCOMP). Mais nous nous sommes très vite rendus compte que nous ne pouvions pas nous contenter de supprimer, de détruire certaines parties du "package" standard, et que nous devrions également construire en rajoutant certaines options dont l'absence se faisait cruellement sentir pour des personnes non-informaticiennes, et utilisatrices occasionnelles d'un programme. Or il s'agissait évidemment de la "cible" privilégiée du logiciel ProfCOMP, puisque le but premier de ce projet était, bien entendu, de donner un accès vers l'EAO à des professeurs n'ayant que des connaissances restreintes en informatique.

Ce que l'on a rajouté :

Dans cette perspective, il était bien sûr essentiel, dans un outil très complexe pour un utilisateur profane, d'incorporer de l'aide "on-line". Nous avons finalement décidé d'intégrer deux "help" différents, avec rappel permanent à l'écran de la façon d'accéder au premier d'entre eux. Celui-ci explique en détail le fonctionnement de l'éditeur : comment profiter de toute sa puissance, quelles sont les fonctions qu'il offre, ... Le second est encore plus nécessaire : il s'agit, en fait, d'un aide-mémoire rappelant les différentes commandes offertes par le formateur. En effet, même s'il y a souvent des moyens mnémotechniques de les retenir (par exemple **^mg** correspond à **M**arge **G**auche, **^a** à **A**linéa), l'absence d'un help pour ces commandes limiterait sans aucun doute l'utilisation réelle de celles-ci à deux ou trois d'entre elles.

Autre aide, indirecte celle-là, nécessaire dans un tel

logiciel : alors que MicroStar prévoyait l'affichage du répertoire courant comme un option à part entière (avec saisie d'un masque de recherche), nous avons offert en plus cette possibilité (avec masque fixé à *.TXT, soit les fichiers textes déjà introduits grâce à PC_EDIT) à chaque saisie d'un nom de fichier. Nous avons d'ailleurs étendu cette possibilité aux autres parties de ProfCOMP, avec à chaque fois le masque de recherche fixé à la sorte de noeud que l'utilisateur est en train de définir (par exemple *.GRF dans l'éditeur graphique). L'accès à cette possibilité se fait en appuyant sur la touche **F5** lorsqu'on demande un nom de fichier.

Autre option ajoutée pour les fichiers : la possibilité de "restaurer" un fichier. L'utilisateur peut ainsi rétablir l'avant-dernière version d'un texte, créée automatiquement par l'éditeur à chaque sauvetage (".BAK"), comme version courante de ce texte (".TXT"). L'ajout de cette option a été rendu nécessaire par une de nos modifications. En effet, en fixant nous-mêmes l'extension de tous les fichiers créés par notre éditeur à ".TXT", nous supprimions la possibilité d'obtenir le même résultat par un "rename" du fichier ".BAK". Bien entendu, dans cette option "restore", le masque de recherche pour l'affichage du répertoire n'est plus "*.TXT" mais "*.BAK". Cette option n'ouvre pas le texte "restauré"; elle se contente d'opérer sur les fichiers. Tout d'abord, elle efface le fichier ".TXT" correspondant avant de rebaptiser le fichier ".BAK" avec l'extension ".TXT".

Puisqu'on peut changer les couleurs de travail dans PC_CREAT, par souci d'homogénéisation, nous avons préféré offrir cette possibilité également dans PC_EDIT. Cela peut provoquer quelques "troubles" temporaires dans les couleurs à l'écran lorsqu'au moins deux des six couleurs de départ sont identiques. En effet, ce changement nous oblige à modifier les couleurs de ce qui est déjà affiché à l'écran. S'il existait des couleurs identiques, il nous est impossible de savoir par quel nouveau "ton" les remplacer.

Toujours dans le même but, nous avons "sorti" la plupart des messages de l'éditeur pour les mettre dans un

fichier indépendant, comme pour les autres parties de ProfCOMP. Pour des raisons de facilité, nous ne l'avons pas fait pour les différentes options des menus. Ces messages, contenus dans les fichiers EDITERR.* (l'extension désignant précisément la langue de ces messages) peuvent, bien entendu, être traduits aisément grâce à l'utilitaire TRANS_ME (cfr Annexe A, point 8.2).

L'homogénéisation avec le reste de ProfCOMP

Pour simplifier la vie de l'utilisateur, nous avons essayé d'homogénéiser au maximum l'interface de l'éditeur avec ceux de nos autres programmes. C'est pourquoi nous avons remplacé dans la plupart des cas les procédures de saisie de données utilisées dans MicroStar par celles que nous avions déjà définies. C'est ainsi que, par exemple, la saisie de noms de fichiers se fait exactement de la même manière que dans le reste de ProfCOMP. Une exception : la saisie de nombres entiers étant très rare dans un éditeur (exemple : saisie du nombre d'UNDO maximum, ou de la largeur des tabulations), nous n'avons pas jugé utile de remplacer la procédure qu'utilise MicroStar par la nôtre, beaucoup plus lourde.

Le lien entre PC_EDIT et PC_CREAT

L'éditeur obtenu, encore trop important que pour pouvoir être "fondu" dans PC_CREAT, nous avons été contraints de le laisser comme programme indépendant. Pour faciliter l'utilisation de ProfCOMP, cela doit toutefois être transparent à l'utilisateur. Celui-ci doit avoir accès à l'éditeur exactement de la même façon que pour les autres outils, par exemple le formateur et l'éditeur graphique. L'édition d'un texte figure dans le menu principal de PC_CREAT. Une fois cette option sélectionnée, on vérifie si PC_EDIT se trouve bien là où il doit être. S'il n'y est pas, un message apparaîtra à l'écran pour en informer l'utilisateur qui peut alors l'y placer (cas d'un système sans disque dur). Si au bout de trois tentatives, il n'y est toujours pas, on revient alors directement au menu principal. Le même système est, bien entendu, établi dans l'autre sens, pour le retour de PC_EDIT vers PC_CREAT. Toutefois, dans ce cas-là, l'échec au bout des trois essais causera le retour au niveau du DOS. Cette procédure permet de ne pas distinguer les

systèmes avec ou sans disque dur : en effet, si le logiciel est installé sur disque dur, il n'est pas possible de corriger directement cette erreur d'installation du logiciel; les trois tentatives ne permettront aucune modification de la situation, mais après retour au DOS il suffira d'utiliser INST_EAO pour apporter la correction nécessaire (cfr annexe A, point 8.1).

Divers

Nous n'avons pratiquement pas modifié l'option Print, très intéressante car elle permet à l'utilisateur de changer de mode d'impression. Cela se fait par le biais de l'introduction, dans le texte, de caractères de contrôle (pour introduire ^a dans le texte, frapper ^p suivi de a, ce qui correspond au passage par le menu Impression en WordStar). Il est important de noter que le formateur ignore totalement ces caractères de contrôle, éliminés avant d'accomplir le formatage.

Voici la liste exhaustive des modes permis (avec, entre parenthèses, les codes envoyés à l'imprimante) :

- ^a permet de passer en mode condensé (envoi de **SI** ou ^o à l'imprimante),
- ^n de relâcher ce mode pour revenir aux caractères normaux (DC2),
- ^h de reculer la tête d'impression d'un caractère (**BS** ou **Backspace**).

D'autres commandes permettent de "switcher" entre deux modes d'impression :

- ^b par exemple sélectionne ou désélectionne le mode "bold" (**ESC E, ESC F**),
- ^d choisit la double impression ou non (**ESC G, ESC H**),
- ^s souligne le texte (**ESC - 1, ESC - 0**),
- ^t les indices supérieurs (**ESC S 0, ESC T 18**),
- ^v les indices inférieures (**ESC S 1, ESC T 18**).

Bien sûr, tous les caractères de contrôle immédiats, tels que ^l (ou **FF** : Form Feed) fonctionnent également. Nous avons rajouté trois nouvelles possibilités :

- ^w pour les caractères larges (**ESC W 1, ESC W 0**),
- ^c pour la qualité courrier (si l'imprimante le permet : **ESC x 1, ESC x 0**),

- et ^i pour le mode italique (si l'imprimante le permet : **ESC 4, ESC 5**).

Pour mieux faire comprendre ce qui précède, voici un petit exemple, illustré par un extrait de "Joachim a des ennuis - Une aventure du petit Nicolas" de Sempé et Goscinny. Pour une question de facilité de compréhension, alors que, dans le texte édité, les caractères de contrôle sont symbolisés par les caractères ASCII correspondants, nous les remplacerons ici par ^ (pour CTRL) suivi du caractère concerné.

Exemple:

```
^s^w^cJONAS^w^s

^aEudes, qui est un copain qui est très fort et qui aime
bien donner des coups de poing sur le nez des copains, a un
grand frère qui s'appelle Jonas et qui est parti faire le
soldat. Eudes est très fier de son frère et il nous en parle
tout le temps.^n
  -^i Nous avons reçu une photo de Jonas en uniforme^i, il
nous a dit un jour.^i Il est terrible ! Demain, je vous
apporte la photo.^i
  ^aEt Eudes nous a apporté la photo, et Jonas était très
bien, avec son béret et un grand sourire tout content.^n
  -^i Il a pas de galons^i,^t a dit Maixent.^t
  -^i Ben, c'est parce qu'il est nouveau^i, ^b a expliqué
Eudes^b, ^imais il va sûrement devenir officier et commander
des tas de soldats. En tout cas, il a un fusil.
  - Il a pas de revolver ?^i^v a demandé Joachim^v.
  -^i Bien sûr que non^i, a dit Rufus.^i Les revolvers,
c'est pour les officiers. Les soldats, ils n'ont que des
fusils.^i
  ^aÇa, ça ne lui a pas plu, à Eudes.^n
  -^i Qu'est-ce que tu en sais ?^i il a dit.^i Jonas a un
revolver, puisqu'il va devenir officier.
  - Ne me fait pas rigoler^i, a dit Rufus.^i Mon père, lui,
il a un revolver.
  - Ton père^i,^d a crié Eudes^d,^i il n'est pas officier !
Il est agent de police. C'est pas malin d'avoir un revolver
quand on est agent de police !
  - Un agent de police, c'est comme un officier^i, a crié
Rufus.^i Et puis d'abord, mon père, il a un képi ! Il a un
képi, ton frère ?^i
  ^a^wEt Eudes et Rufus se sont battus.^w^n
```

figure 3.2

Imprimé avec PC_EDIT, cela donne :

JONAS

Eudes, qui est un copain qui est très fort et qui aime bien donner des coups de poing sur le nez des copains, a un grand frère qui s'appelle Jonas et qui est parti faire le soldat. Eudes est très fier de son frère et il nous en parle tout le temps.

- *Nous avons reçu une photo de Jonas en uniforme, il nous a dit un jour. Il est terrible ! Demain, je vous apporte la photo.*

Et Eudes nous a apporté la photo, et Jonas était très bien, avec son béret et un grand sourire tout content.

- *Il a pas de galons ,* **a dit Maixent.**

- *Ben, c'est parce qu'il est nouveau , a expliqué Eudes, mais il va sûrement devenir officier et commander des tas de soldats. En tout cas, il a un fusil.*
- *Il a pas de revolver ? a demandé Joachim .*
- *Bien sûr que non , a dit Rufus. Les revolvers, c'est pour les officiers. Les soldats, ils n'ont que des fusils.*

Ça, ça ne lui a pas plu, à Eudes.

- *Qu'est-ce que tu en sais ? il a dit. Jonas a un revolver, puisqu'il va devenir officier.*
- *Ne me fait pas rigoler , a dit Rufus. Mon père, lui, il a un revolver.*
- *Ton père , a crié Eudes, il n'est pas officier ! Il est agent de police. C'est pas malin d'avoir un revolver quand on est agent de police !*
- *Un agent de police, c'est comme un officier , a crié Rufus. Et puis d'abord, mon père, il a un képi ! Il a un képi, ton frère ?*

Et Eudes et Rufus se sont battus.

figure 3.3

Pour en terminer avec PC_EDIT, il reste à signaler que, de l'ensemble des programmes, c'est sans aucun doute celui qui est le mieux protégé contre les erreurs à l'exécution (run-time error). Il teste en effet entièrement toutes les opérations d'entrée-sortie qu'il effectue, mais également la mémoire disponible pour ses besoins dynamiques (listes chaînées, ...). Il est vrai que ceux-ci sont énormes étant donnée la nature même du programme, le texte édité étant mémorisé dans des listes.

Conclusion

Arrivés à ce stade de développement, nous avons estimé que l'éditeur ainsi obtenu possédait les principales caractéristiques nécessaires dans un logiciel d'EAO.

3.1.2 Le formateur

Deux possibilités s'offraient à nous : soit intégrer le formateur à l'éditeur, soit faire du formateur un module à part. Comme l'intégration impliquait une modification en profondeur de notre éditeur, ce qui n'allait pas sans risques, et que l'aspect essentiel de ce que nous attendions de ce formateur était l'adaptation d'un texte à une fenêtre, il nous a semblé préférable de dissocier ces deux aspects. Le choix de la taille de la fenêtre se fait en début de formatage.

Le formateur, lorsqu'il adapte le texte aux dimensions de la fenêtre, respecte les paragraphes (c'est le **RETURN** qui

sépare les différents paragraphes). Mais à l'intérieur de chacun des paragraphes, il "remodèle" complètement le texte de façon à répartir les blancs le plus uniformément possible. Ce formateur ne fait pas de césure : aucun mot n'est donc coupé en fin de ligne, sauf, bien sûr, s'il est plus long que cette ligne.

Une fois établie la nécessité de ce formateur, nous l'avons voulu le plus complet possible : c'est pourquoi il a évolué au fil du temps, par l'ajout de nouvelles commandes. Ainsi, bien qu'il serve toujours à adapter le texte édité (cfr point 3.1.1, L'éditeur) à une fenêtre, il offre un grand nombre d'autres possibilités. Certaines sont classiques dans un formateur, d'autres tout à fait spécifiques de par son rôle dans un logiciel d'EAO. Ces fonctions doivent, bien entendu, être inclues dans le texte lors de son édition.

Ces commandes obéissent à une certaine syntaxe. Toute commande, pour être interprétée comme telle, se trouve en début de paragraphe dans le texte. Naturellement, plusieurs commandes peuvent y être regroupées. Toute commande commence par un ^ suivi du nom de la fonction. Si la fonction nécessite un paramètre, celui-ci sera séparé du nom de la commande par un point. Toute commande doit se terminer par un espace. L'effet d'une commande n'est pas limité au paragraphe courant ; pour stopper son effet, il faut donc donner explicitement une commande inverse.

Voici tout d'abord les fonctions courantes :
- le changement des positions des marges : **^mg** modifie la **m**arge **g**auche, et **^md** la **m**arge **d**roite. Pour cette fonction, l'argument n'a pas le sens courant de la marge de droite. Habituellement, on désigne la position du dernier caractère de chaque ligne. Ici, cela n'a pas de sens, puisque la longueur maximum de chaque ligne dépend du choix de la taille de la fenêtre pour l'exécution de la leçon. Or, si la marge de droite est définie lors de l'édition du texte, la dimension de la fenêtre n'est choisie que lors du formatage. Le paramètre (un nombre entier) indique donc la position du dernier caractère par rapport au bord droit de la fenêtre. Tout comme **^mg.3** place le premier caractère en troisième

colonne dans la fenêtre, **^md.3** fixe le dernier caractère à l'antépénultième position disponible dans cette fenêtre. Par défaut, les marges sont établies à la première et à la dernière colonne de la fenêtre (pas d'espace ni en début ni en fin de ligne).

- Il peut être intéressant d'avoir un alinéa en début de paragraphe. Cela ne peut se faire dans l'éditeur, car pour avoir une bonne répartition des espaces, le formateur commence par supprimer tous les blancs superflus : ceux en début de paragraphe, et ceux qui sont redondants entre deux mots. Une commande du formateur permet de créer un alinéa : **^a**. Le paramètre qui suit donne la largeur de cet alinéa. La commande correspondante, qui supprime l'alinéa, est **^fa**. Elle a le même effet que **^a.0**.

- D'autres commandes influent sur la présentation du texte à l'écran. Par défaut, le formateur justifie le texte à droite et à gauche, c'est-à-dire que les premiers et derniers caractères de chaque ligne s'alignent verticalement. Bien sûr, le texte peut aussi être justifié uniquement à droite (seuls les derniers caractères de chaque ligne s' alignent verticalement : **^jd**), uniquement à gauche (on supprime en fait la justification : **^fj**). Il peut aussi être centré : **^m**. **^j** pour établir la justification gauche-droite, **^fjd** pour la fin de justification à droite, et **^fm** pour la fin du centrage ont le même effet. Ces trois commandes ramènent l'option par défaut de justification gauche-droite.

- Il est aussi possible, par **^l**, de définir des littéraux. Un littéral est une partie de texte que le formateur ne peut en aucun cas modifier. Les espaces ne sont donc pas répartis uniformément. **^fl** indique la fin d'un littéral.

D'autres commandes doivent uniquement leur présence au but premier de ce formateur :
- Lors de l'exécution de la leçon, deux modes d'affichage du texte sont possibles : soit l'élève lit le texte à son rythme, peut revenir en arrière, etc.; soit l'enseignant fixe un délai d'affichage valable pour chaque page du texte. Ce délai est donné en secondes, par la fonction **^d.x**. x représente ce délai. Normalement, cette commande ne doit figurer au plus qu'une seule fois dans un texte. Dans tous

les cas, ce sera la dernière valeur qui sera retenue. En l'absence d'une telle commande, l'élève pourra progresser à son rythme dans le texte.

- Par défaut aussi, on numérote à partir de 1 les pages de texte affichées dans la fenêtre à l'écran. La commande **^fn** supprime cette numérotation, et **^n.x** commence la numérotation à partir du chiffre x. Ces commandes peuvent se trouver n'importe où dans le texte, et leur effet prend cours immédiatement.

- Pour bien dissocier des points importants, on peut également forcer un changement de page à l'écran (**^p**). Cela correspond au .pa de WordStar. L'insertion de lignes blanches lors de l'édition ne garantit pas cet effet puisque le nombre de lignes de la fenêtre n'est pas connu à ce moment.

- Enfin, il est aussi possible de modifier la couleur d'affichage du texte à l'écran lors de l'exécution de la leçon, pour attirer l'attention sur des points importants. Signalons néanmoins que le Turbo Graphix Toolbox avec lequel nous travaillons n'autorise pas plus d'une couleur à la fois, c'est tout l'écran qui change alors de couleur. Cela se fait par **^c.x**. x est le numéro de la nouvelle couleur. **^fc** rétablit la couleur initiale. Pour vérifier la signification de x, il faut utiliser INST_EAO pour PC_EXEC (cfr annexe A, 8.1).

Pour illustrer ces commandes, nous allons reprendre un autre extrait de "Joachim a des ennuis" par Sempé et Goscinny.

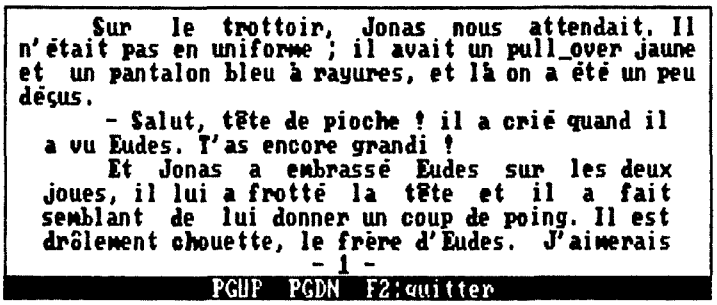
Exemple :

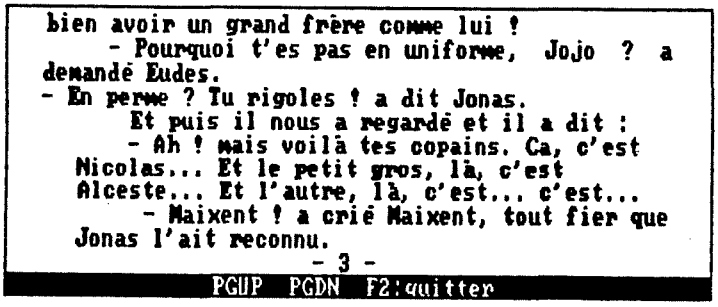
```
^a.5 Sur le trottoir, Jonas nous attendait. Il n'était pas
en uniforme ; il avait un pull-over jaune et un pantalon
bleu à rayures, et là on a été un peu déçus.
^mg.3 ^md.3 - Salut, tête de pioche ! il a crié quand il a
vu Eudes. T'as encore grandi !
Et Jonas a embrassé Eudes sur les deux joues, il lui a
frotté la tête et il a fait semblant de lui donner un coup
de poing. Il est drôlement chouette, le frère d'Eudes.
J'aimerais bien avoir un grand frère comme lui !
^n.3 - Pourquoi t'es pas en uniforme, Jojo ? a demandé
Eudes.
^l - En perme ? Tu rigoles ! a dit Jonas.
^fl ^mg.5 ^md.2 Et puis il nous a regardé et il a dit :
^fj - Ah ! mais voilà tes copains. Ça, c'est Nicolas... Et
le petit gros, là, c'est Alceste... Et l'autre, là, c'est...
c'est...
```

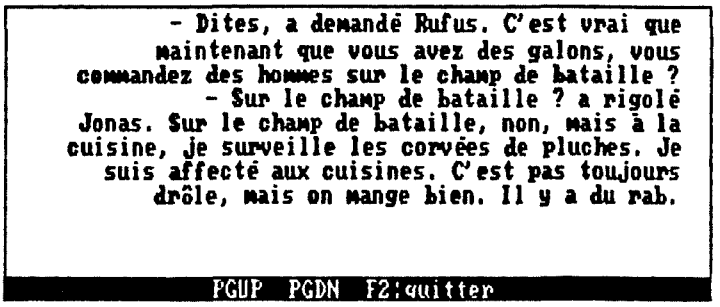
- Maixent ! a crié Maixent, tout fier que Jonas l'ait reconnu.
^jd - Dites, a demandé Rufus. C'est vrai que maintenant que vous avez des galons, vous commandez des hommes sur le champ de bataille ?
^fn - Sur le champ de bataille ? a rigolé Jonas. Sur le champ de bataille, non, mais à la cuisine, je surveille les corvées de pluches. Je suis affecté aux cuisines. C'est pas toujours drôle, mais on mange bien. Il y a du rab.
^p Alors, Eudes a regardé Jonas, il est devenu tout blanc et il est parti en courant.
^m - Eudes ! Eudes ! a crié Jonas. Mais qu'est-ce qu'il a, celui-là ? Attends-moi, tête de pioche ! attends-moi !
Et Jonas est parti en courant, après Eudes.
^j Nous, nous sommes partis aussi, et Alceste a dit qu'Eudes devait être fier d'avoir un frère qui avait si bien réussi dans l'armée.

figure 3.4

Cela donne comme résultat, après passage dans le formateur :







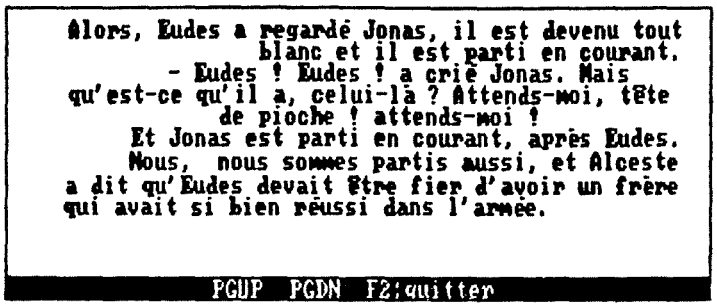


figure 3.5

Algorithme du formateur

Le formateur accepte comme données en entrée un fichier de type texte **fichd** contenant le texte à formater. En sortie, le texte formaté **fichres** : c'est bien sûr également un fichier, mais de "records" cette fois. Chaque "record" est composé d'une chaîne de 80 caractères (**tabl**) et d'un entier indiquant la couleur à employer pour cette ligne (**coul**). Chacun de ces "records" contient les données nécessaires à l'affichage d'une ligne dans la fenêtre. Les informations concernant la taille de la fenêtre sont conservées, comme pour les noeuds des autres types, dans le fichier WINDOWS.DIM. S'il se produit des erreurs durant le formatage, un fichier temporaire **ficherr** (de même structure que le fichier résultat, mais adapté à une fenêtre de taille fixe) est créé. Une fois le formatage terminé, le professeur peut consulter ce fichier de messages d'erreurs exactement de la même façon que l'étudiant lisant un noeud texte.

Précisons également les variables essentielles. Tout d'abord celles qui indiquent l'état courant, d'après les commandes introduites dans le texte à formater :

- **state** indique quel est le type de formatage demandé : justification gauche-droite (**jgd**), justification à gauche (**rien**), justification à droite (**jd**), ou encore centrage (**centrage**).
- Pour les commandes concernant la couleur d'affichage, **coul** est un entier qui indique quelle est cette couleur. Lorsque **coul** vaut zéro, cela signifie qu'aucune demande de changement de couleur n'a été effectuée et qu'il faudra dès

lors conserver la couleur normale lors de l'affichage de cette ligne.

- Pour la numérotation des pages, le booléen **pgn** indique si elle est demandée actuellement et **nbpage** est un entier désignant le numéro de page courant.
- Pour l'affichage du noeud texte à durée fixée par l'enseignant, **dem_del** (entier) indique le délai d'affichage en secondes. Si **dem_del** vaut zéro, l'étudiant est libre de circuler à sa guise dans le texte. Cette valeur est également mémorisée dans le fichier **WINDOWS.DIM**.
- Le booléen **litteral** signale si la fonction littéral est activée ou non.
- Les deux entiers **marge_g** et **marge_d** donnent les valeurs courantes des marges. Pour plus de facilité, **marge_d** contient ici le nombre de blancs nécessaires en fin de ligne (après la commande **^md.3**, **marge_d** vaudra 2)
- L'entier **alinea** indique la valeur courante de l'alinéa. La variable de travail **alineer** indique si la ligne courante en sortie doit être "alinéée", c'est-à-dire si elle est la première ligne du paragraphe.

Il existe encore bien d'autres variables de travail. Nous n'expliquerons ici que les plus importantes, les autres seront définies lorsque nous en aurons besoin.

- Tout d'abord, les lignes courantes, de même type que les enregistrements des fichiers en sortie. **Taban** (**tab**leau d'**an**alyse) est la suite de caractères en entrée, avant traitement. Il contient au plus 80 caractères ne provenant jamais de deux paragraphes différents. **Tabanbis**, tableau de secours, mémorise les caractères déjà lus dans **fichd** et ne pouvant être contenus dans **taban**. Par exemple, un mot ne sera jamais mis partiellement dans **taban**. Si sa lecture est entamée et non terminée, alors que **taban** est rempli, le début du mot sera mémorisé dans **tabanbis** et retiré de **taban**. Enfin, un troisième tableau conserve la suite de caractères en sortie, après traitement : il s'agit de **tabres** (**tab**leau **rés**ultat).
- Deux variables de type booléen concernent les paragraphes. La première **fin_paragraphe** joue sur la suite de caractères en entrée ; elle est à vrai si l'on est arrivé à une fin de paragraphe dans la lecture du fichier **fichd**. La

seconde **partie_paragraphe_non_traitée** concerne le traitement lui-même : elle est à vrai tant qu'il reste des caractères du paragraphe courant en entrée à traiter et à écrire dans **fichres**.

- **Lcour** et **hcour** donnent respectivement la longueur et la hauteur de la fenêtre pour laquelle on formate le texte. **Hact** est un compteur qui indique la hauteur actuelle, c'est-à-dire le numéro de ligne à l'intérieur de la fenêtre de la prochaine ligne résultat. Ce compteur assure les fonctions de numérotation des pages (**^n.x**) et de saut de page (**^p**).

Il est temps de passer à l'algorithme lui-même. Pour plus de facilité, nous allons le développer par raffinements successifs. Les "identificateurs" soulignés sont développés par la suite.

P0 :

```
Initialisation des variables
Ouverture des fichiers
debut_trt :
Tant que non fin de fichd faire
      taban.tabl <-- ''
      tabanbis.tabl <-- ''
      fin_paragraphe <-- false
      partie_paragraphe_non_traitée <-- true
      alineer <-- true
      Lire
      Traiter appels fonctions

      (* à cet endroit, aucune partie texte du paragraphe
      courant n'a été traitée. Si l'on est à la fin du
      paragraphe et que le tableau d'analyse taban est vide,
      il suffit d'écrire une ligne vide dans le fichier
      résultat fichres et de retourner en début de boucle *)

      Si fin_paragraphe et taban.tabl='' alors
            tabres.tabl <-- ''
            ecrire (tabres)
            goto debut_trt
      Fin Si

      (* Les fonctions étant traitées, reste le texte
      proprement dit du paragraphe courant *)

      Si litteral
      Alors traitement litteral
      Sinon traitement normal

Fin tant que

(* le texte en entrée a été entièrement traité, reste à
   numéroter la dernière page et à fermer les fichiers *)

tabres.tabl <-- ''
Tant que pgn et (hact<>1) faire ecrire(tabres)
Fermer les fichiers
```

P1 = **P0** où :

Initialisation des variables =

```
coul <-- 0
dem_del <-- 0
marge_g <-- 1
marge_d <-- 0
alinea <-- 0
pgn <-- true
nbpage <-- 1
state <-- jgd
litteral <-- false
hact <-- 1
```

---

traiter appels fonctions =

```
cont <-- true
tant que cont faire
     repérer prochain caractère non "blanc" de taban.tabl,
     et effacer tous les caractères "blancs" le précédant
     Si pas de caractère non "blanc"
     Alors Si fin_paragraphe alors cont <-- false
                              sinon lire
     Sinon Si ce premier caractère est un ^
             (* signifie une fonction *)
           Alors l'effacer
                 lirefonc (argument,nfonc,arg)

                 (* Tous ces paramètres sont des résultats
                   nfonc nom de la fonction,
                   arg l'argument sous forme de string et
                   argument le même sous forme d'entier *)

                 verifappel (nfonc,argument,arg,taban)
           Sinon cont <-- false
```

---

traitement litteral =

```
Si lcour-marge_d-(marge_g-1) < longueur(taban.tabl)
                               +longueur(tabanbis.tabl)
Alors (* le littéral est trop grand pour la ligne *)
     errfonc <-- true (* indique que l'erreur provient
                         d'une fonction *)

     nfonc <-- '^1'
     traiterr (15,taban.tabl,nfonc)
     Tant que partie_paragraphe_non_traitée faire
          mettre plus possible in tabres
          case state of
            centrage : centrer le texte entre les marges
               par insertion de blancs en début de tabres.
            jd : plaquer le texte contre la marge de
               droite par insertion de blancs en début de
               tabres.
          fin case
          ecrire (tabres)
     Fin tant que
Fin Alors
Sinon (* pas de problème de longueur du littéral *)
     tabres.tabl <-- marge_g - 1 blancs
     tabres.tabl <-- tabres.tabl+taban.tabl+tabanbis.tabl
     case state of
       centrage : centrer le texte entre les marges
          par insertion de blancs en début de tabres.
       jd : plaquer le texte contre la marge de
          droite par insertion de blancs en début de
          tabres.
     fin case
     ecrire (tabres)
Fin Sinon
```

```
traitement normal =

 Tant que partie_paragraphe_non_traitée faire
      mettre plus possible in tabres
      case state of
        centrage : centrer le texte entre les marges
           par insertion de blancs en début de tabres.
        jgd : si partie_paragraphe_non_traitée alors traiter

           (* la dernière ligne en sortie d'un paragraphe ne
           doit pas être justifiée gauche-droite *)

        jd : plaquer le texte contre la marge de
           droite par insertion de blancs en début de
           tabres.
      fin case
      ecrire (tabres)
 Fin Tant que
```

P2 = P1 où :

```
mettre plus possible in tabres =

 tabres.tabl <-- marge_g - 1 blancs
 Si alineer
 Alors Si marge_g + marge_d + alinea - 1 < lcour

       (* assez de place dans la ligne pour l'alinea *)

       Alors ajouter alinea blancs à tabres.tabl
       Sinon alineer <-- false
 Fin du Si alineer
 qqchose_in_tabres <-- false

      (* vrai dès qu'on a pu placer des caractères non
         blancs dans tabres.tabl *)

 deb <-- tabres.tabl

      (* contient le "préfixe" de la ligne résultat courante
         du paragraphe courant ; utile lorsque l'on devra
         répartir le prochain mot sur plusieurs lignes  *)

finmot :
 place_restante <-- true

      (* faux dès que le mot suivant du paragraphe est trop
      long pour être ajouté à tabres.tabl *)

 Tant que place_restante et partie_paragraphe_non_traitee
      prendre mot svt (word) (* word, résultat, est le mot
           suivant du paragraphe s'il existe. word sera
           toujours préfixé par un blanc *)

      Si partie_paragraphe_non_traitee
      Alors Si on peut rajouter ce mot à tabres.tabl tout
             en respectant les marges définies
            Alors Tabres.tabl <-- tabres.tabl + word
                  qqchose_in_tabres <-- true
                  Si c'est le premier mot placé dans tabres
                  Alors en effacer dans Tabres le 1er
                       caractère (* qui est un blanc *)
            Sinon replacer word à sa place dans taban.tabl
                  place_restante <-- false
 Fin du tant que
 Si non qqchose_in_tabres et non place_restante
 Alors (* word est trop long pour la ligne lorsqu'il est
           préfixé par un blanc *)

      effacer le premier caractère (un blanc) de word
      S'il ne rentre toujours pas dans les marges
      Alors errfonc <-- false
            traiterr(14,taban.tabl,nfonc)
      Fin du Si
      Retirer Word de taban.tabl
      Tant que word n'est pas vide faire
           ajouter le plus de caractères possibles, tout en
```

```
            respectant les marges, de word à tabres.tabl
        effacer de word les caractères placés dans
            tabres.tabl
        Si alineer (* modifier deb car les lignes
                suivantes ne doivent pas être alinéées*)

        Alors retirer alinea blancs de deb
        Si word est vide
        Alors (* peut-être reste-il assez de place dans
             tabres.tabl pour rajouter le mot suivant
             de taban.tabl ? *)

             qqchose_in_tabres <-- true
             aller en finmot
        Sinon ecrire(tabres)
    Fin du tant que
Fin du Si
```

---

traiter =

(* utilisée en cas de justification gauche-droite pour répartir les blancs à l'intérieur de la ligne. Si le numéro de la ligne est pair, on mettra plus de blans sur la droite de la ligne ; s'il est impair, on en mettra plus à gauche. Quelques explications sur les variables locales : **nb_a_ajouter** : nombre total d'espaces à insérer entre les mots ; **nb_espaces** : nombre d'inter-mots existant dans la ligne (= nombre de blancs entre le premier et le dernier mot de **tabres.tabl**) ; **nb_a_gauche** : nombre de blancs à insérer dans les inter-mots à gauche dans **tabres.tabl** ; **nb_a_droite** : nombre de blancs à insérer dans les inter-mots à droite dans **tabres.tabl** ; **nb_esp_a_gauche** et **nb_esp_a_droite** : nombre d'inter-mots respectivement à gauche et à droite dans **tabres.tabl**, dans chacun desquels il faut insérer respectivement **nb_a_gauche** et **nb_a_droite** blancs. *)

```
calcul de nb_a_ajouter
calcul de nb_espaces
Si nb_espaces <> 0
Alors calcul de nb_a_droite,nb_a_gauche,nb_esp_a_droite,
                nb_esp_a_gauche en tenant compte de la
                parité ou non du numéro de la ligne (hact)
     insérer nb_a_droite blancs dans chacun des
         nb_esp_a_droite inter-mots à droite
         dans tabres.tabl
     insérer nb_a_gauche blancs dans chacun des
         nb_esp_a_gauche inter-mots à gauche
         dans tabres.tabl
```

---

lirefonc =

Ne nécessite pas le développement de son algorithme. Son but est de retirer les valeurs de **nfonc**, **arg**, et **argument** (cfr signification supra) de **taban.tabl**, de vérifier la légalité syntaxique de la fonction et de l'argument (si non, appel à traiterr), d'effacer ce qu'il a lu de **taban.tabl**, et d'effacer éventuellement tous les blancs qui précéderaient une autre fonction dans **taban.tabl**.

---

verifappel =

```
errfonc <-- true
case nfonc of
     J : state <-- jgd
     FJ : state <-- rien
     FL : litteral <-- false
     FC : coul <-- 0
     FN : pgn <-- false
     D : Si argument valide
         Alors dem_del <-- argument
         Sinon traiterr (16,taban.tabl,nfonc)
     L : litteral <-- true
     P : tabres.tabl <-- ''
         tant que hact<>1 faire ecrire(tabres)
     N : pgn <-- true
```

```
        si argument > 0 alors nbpage <-- argument
    C : si argument valide
        alors coul <-- argument
        sinon traiterr(16,taban.tabl,nfonc)
   MG : si argument valide par rapport aux valeurs
        courantes de marge_d et lcour
        alors marge_g <-- argument
        sinon traiterr(16,taban.tabl,nfonc)
   MD : si argument valide par rapport aux valeurs
        courantes de marge_g et lcour
        alors marge_d <-- argument - 1
        sinon traiterr(16,taban.tabl,nfonc)
FM,FJD : state <-- jgd
    M : state <-- centrage
   JD : state <-- jd
    A : si argument valide (≥0)
        alors alinea <-- argument
   FA : alinea <-- 0
  else traiterr(13,taban.tabl,nfonc)
```

P3 = P2 où :

prendre mot svt =

```
début :
 effacer tous les blancs en début de taban.tabl
 Si taban.tabl = ''
 Alors si fin_paragraphe
       alors partie_paragraphe_non_traitee <-- false
       sinon lire
             aller en début
 Si partie_paragraphe_non_traitee
 Alors repérer le premier caractère blanc de taban.tabl
          s'il existe (soit le ième ; s'il n'existe pas,
          i vaudra la longueur de taban.tabl - 1)
       word <-- ' ' + les i-1 premiers caractères de
                taban.tabl
       effacer les i-1 premiers caractères de taban.tabl
```

---

ecrire =

```
 tabres.coul <-- coul
 copier tabres dans fichres
 alineer <-- false (* si ceci était la 1ère ligne en sortie
                      d'un paragraphe, la suivante ne le
                      sera plus *)

 hact <-- hact + 1
 Si (hact = hcour) and pgn
 alors écrire le numéro de page, centré, dans fichres
       hact <-- hact + 1
 Si hact > hcour alors hact <-- 1
 tabres.tabl <-- ''
```

---

traiterr =

```
   (* ne pas confondre avec traiter ; ici, il s'agit de
traiter les erreurs à écrire dans ficherr *)

  (* il n'est pas nécessaire ici de développer l'algorithme.
     Retenons uniquement que connaissant les dimensions de
     la fenêtre, on commente l'erreur sur une demi-hauteur
     de cette fenêtre, grâce au numéro communiqué comme
     premier paramètre *)
```

P4 = **P3** où :

lire =

```
(* i est un simple compteur du nombre de caractères dans
   taban.tabl
   Cette procédure est basée sur le fait que PC_EDIT
   indique un passage à la ligne sans RETURN (frappe
   kilométrique) par insertion des deux caractères 141 et
   10 (à noter que 141 = CR (13) +128 et 10 = LF *)

taban.tabl <-- tabanbis.tabl
tabanbis.tabl <-- ''
i <-- longueur (taban.tabl)
Tant que non fin de fichd et non fin de ligne
     (* paragraphe *) de fichd et i < 80 faire
     ch <-- caractère suivant de fichd
     Si ch n'est pas un des caractères de contrôle
          destinés à enjoliver l'impression à partir
          de PC_EDIT (* cfr point 3.1.1. *)
     Alors i <-- i + 1
           Si ch <> #141
           alors ajouter ch à taban.tabl
           sinon Si pas fin de fichd
                 alors ch <-- caractère suivant de fichd
                 sinon ch <-- #10
                 Si ch = #10  (* frappe kilométrique *)
                 alors ajouter ' ' à taban.tabl
                 sinon ajouter #141 à taban.tabl
                       i <-- i + 1
                       si i ≥ 80
                       alors tabanbis.tabl <-- ch
                       sinon ajouter ch à taban.tabl
Fin du tant que
fin_paragraphe <-- fin de fichd ou fin de ligne de fichd
Si fin de ligne de fichd, la lire
Si pas fin_paragraphe
Alors repérer le dernier blanc de Taban.tabl
      s'il existe, retirer les caractères qui le suivent
          pour les mettre dans tabanbis.tabl, et effacer
          de  taban.tabl tous les caractères à partir de
          ce blanc (blanc compris). Cela va permettre
          d'éviter de ne prendre qu'un début de mot dans
          taban.tabl et donc de répartir un mot sur deux
          lignes différentes dans fichres.
```

figure 3.6

Voilà qui clotûre la description du formateur. Nous avons expliqué uniquement la partie formatage pur; les autres parties, notamment la saisie de la taille de la fenêtre pour laquelle le texte doit être formaté et l'affichage du texte résultat (**fichres**) ou du fichier des erreurs (**ficherr**) offrent beaucoup moins d'intérêt.

3.2 Noeuds graphique et animation

Nous avons montré comment créer un noeud texte, mais il ne s'agit là que d'un type de noeud sur les cinq disponibles dans ProfCOMP. Nous allons voir maintenant comment élaborer les deux types de noeuds graphique. En effet, un noeud animation n'a pas réellement d'existence propre. Il est composé d'une collection un peu spéciale de noeuds graphique, tous crées avec l'éditeur graphique.

Comme dans tout le programme, nous avons ici aussi voulu privilégier la facilité d'utilisation. C'est pourquoi nous avons prévu un double système de commandes, et une aide à tous les niveaux.

Tout d'abord, les systèmes de commandes. Le premier, très simple pour les utilisateurs occasionnels de l'éditeur graphique, est un choix par menu. Pour "appeler" le menu, il suffit d'appuyer sur **F10** (comme dans l'éditeur PC_EDIT); ce "moyen d'accès" est d'ailleurs rappelé en permanence à l'écran. Le choix dans ce menu se fait par les flèches (déplacement) et **RETURN** (sélection). Comme les différents choix de ce menu figurent dans les fichiers de messages, ils sont également traduisibles, ce qui augmente la facilité d'utilisation.

Pour les utilisateurs plus chevronnés, un choix par commande existe également. A chacune des fonctions disponibles, est associée une lettre. Pour exécuter cette fonction, il suffit donc de frapper cette lettre. Par exemple, la fonction Aide est identifiée par la lettre H (Help). En frappant H au niveau général, on obtient donc la liste complète des fonctions disponibles. Si, par contre, on se trouve à l'intérieur d'une fonction, l'aide permettra de préciser ce que le système attend de l'utilisateur. Il est important de noter que la fonction aide est une des deux fonctions qui peuvent être exécutées à l'intérieur d'une commande.

Ce système de choix est valable également à l'intérieur du menu, comme nous l'avons annoncé ci-dessus. Donc, contrairement à ce qui existe dans la plupart des logiciels (mais aussi dans PC_EDIT, et les autres menus de

ProfCOMP), le choix ne se fait pas par l'initiale de l'option que l'on veut sélectionner, mais par la lettre associée à cette commande. Exemple : pour sélectionner l'aide dans le menu autrement que par les flèches, il faut frapper la lettre H et non la lettre A. En traduisant les messages (cfr troisième section de ce chapitre, Les messages), il peut donc être intéressant de faire précéder les options de ce menu par la lettre y associée.

L'aide est ici très complète : elle reprend donc, au niveau général, une liste exhaustive des fonctions disponibles, et à l'intérieur de ces fonctions un rappel de la commande choisie et de ce que l'utilisateur doit faire. Mais ce n'est, bien sûr, pas la seule aide : comme partout ailleurs dans ProfCOMP, **F5** visualise le répertoire (ici des noeuds graphique existants) lorsque le système demande l'introduction d'un nom de fichier, et les touches **F1** et **F2** permettent respectivement de revenir un pas en arrière (lorsqu'on se trouve à l'intérieur d'une fonction) ou d'abandonner la fonction en cours. Lorsqu'on ne se trouve pas dans une fonction, **F2** permet d'abandonner l'édition.

Comme nous utilisons le Turbo Graphix Toolbox de Borland, nous sommes limités à une seule couleur. Cela peut paraître désespérant d'être confiné à une présentation monochrome en mode graphique, même si l'utilisateur dispose d'un bel écran couleur. Mais lorsqu'on se rend compte de la place prise pour mémoriser une fenêtre graphique même monochrome, l'on se dit que pour faire le même logiciel en exploitant au mieux les possibilités de couleurs d'une carte EGA par exemple, il faudrait tout d'abord faire sauter le goulot des 640 K de mémoire centrale, limitation due au choix de MS-DOS comme système d'exploitation.

L'éditeur graphique existait bien sûr déjà l'an dernier, mais dans une version plus limitée : pas de sélection des fonctions par menu, moins de fonctions disponibles, pas de "curseur" réellement utilisable (il était représenté par un point en inverse),... Ces défauts étaient acceptables dans un prototype, ils ne l'étaient plus dans un logiciel voulu plus "professionnel".

Avant de passer à la liste des fonctions disponibles, notons une dernière chose importante, valable pour toutes les parties de ProfCOMP où l'on travaille en mode graphique. L'an dernier, nous utilisions la partie concernant les cartes graphiques IBM (CGA,...). Mais lors de notre stage à Lisbonne, nous avons travaillé avec une carte Hercules. Le Turbo Graphix Toolbox prévoyait bien ce type de matériel, mais il fallait alors utiliser à la compilation du programme d'autres fichiers du toolbox. Notre logiciel aurait donc été limité à un type de matériel, ou encore nous aurions dû maintenir deux versions en parallèle, une pour chaque type de carte. Nous avons alors décidé de modifier les fichiers du Toolbox, pour pouvoir travailler avec une seule version quel que soit le matériel. Un même utilisateur, qui aurait deux ordinateurs avec cartes graphiques différentes, n'aura aucun problème : il pourra travailler avec la même version de façon totalement transparente : c'est à présent le logiciel qui détecte quel type de carte il doit piloter.

Les commandes

Il y a tout d'abord les commandes de déplacement. En plus des flèches, qui déplacent le curseur dans le graphique édité, il existe quatre autres commandes (non accessibles via le menu) pour accélérer le mouvement : **B** (pour **B**ackward) permet de se déplacer à l'extrême gauche de la fenêtre (tout en restant sur la même ligne), et **F** (pour **F**orward) à l'extrême droite ; **U** (pour **U**p) positionne le curseur en haut de la fenêtre (sur la même colonne) et **D** (pour **D**own) en bas.

En règle générale (nous verrons les exceptions plus tard), l'utilisateur peut modifier la vitesse de déplacement du curseur. Il utilisera la commande **V** pour fixer lui-même la vitesse, c'est-à-dire le nombre de points dont le curseur "bougera" à chaque déplacement. Les vitesses horizontales et verticales peuvent varier de 1 à 15. Par défaut, elles sont toutes deux fixées à 9. Ainsi, si l'utilisateur les établit respectivement à 13 et à 4, unr frappe sur <- déplacera (si possible) le curseur de 13 pixels vers la gauche, et l'appui sur la flèche vers le haut de 4 pixels vers le haut. Pour modifier ces vitesses, il ne faut pas introduire un chiffre : une simple pression sur le flèche vers le haut augmente la vitesse d'une unité, alors qu'elle décroit

lorsqu'on appuie sur la flèche vers le bas. Il est important de noter que comme pour la commande **H** (Help), on peut exécuter cette option d'ajustement de la vitesse à l'intérieur des autres commandes. C'est bien sûr également le cas des commandes de déplacement (à l'intérieur des commandes exigeant un déplacement du curseur).

Passons maintenant aux commandes de dessin. **L** (Line) permet de tracer une ligne. Une fois cette option sélectionnée, il suffit de se déplacer à l'autre extrémité de la ligne, et d'appuyer une seconde fois sur **L**. La ligne désirée est alors tracée. L'utilisateur veut dessiner en même temps qu'il se déplace ? Il sélectionne pour cela l'option **W** (While deplacing). Tout son parcours dans le graphe sera alors tracé. Pour revenir au mode normal, il lui suffit de frapper une seconde fois sur **W**.

L'éditeur graphique propose de tracer très facilement deux formes géométriques : le rectangle et le cercle. Pour le premier, il se positionne tout d'abord sur l'un des coins (par exemple le coin supérieur gauche) du futur rectangle et appuie sur **S** (Square). Il se déplace ensuite vers le coin opposé (dans notre exemple le coin inférieur droit) et frapper un seconde fois **S**. Il reste ensuite le choix de remplir ou non le rectangle avec la couleur (et le style de ligne - cfr page suivante) courante. C'est ici le premier exemple que nous rencontrons d'un choix entre deux options : les flèches verticales ou la barre d'espacement changent l'option courante, tandis que **RETURN** fixe la sélection définitive.

Pour tracer un cercle, l'utilisateur positionne le curseur en son centre avant de frapper **C**; après quoi il le déplace HORIZONTALEMENT (<-, ->, **B** ou **F**) pour déterminer la taille du rayon. Une seconde pression sur **C** et le cercle est tracé.

Mais il existe une multitude de formes géométriques. Comme nous ne pouvions les offrir toutes, nous avons ajouté la commande **A** pour tracer une figure fermée. Elle est très importante car elle évite à l'enseignant de jongler pour retomber exactement à son point de départ, comme il devrait

le faire pour réaliser la même figure uniquement avec la commande **L**. Du point de vue utilisation, l'enseignant se positionne au point de départ (appelons-le point a) de sa figure avant d'appuyer sur **A**. Il se déplace alors de sommet en sommet, jusqu'à l'avant-dernier (nommons le y), en appuyant sur **A** à chaque fois. Chacun des sommets est alors relié au précédent. Reste alors à se positionner sur le dernier sommet (z). Là, pour fermer la figure, il suffit d'appuyer sur **RETURN**, et une ligne sera tracée entre y et z, et une autre entre a et z.

Nous avons annoncé que le style des lignes pouvait être changé. Par défaut, le trait est continu, mais quatre pointillés différents (* * * * , ***** ***** , *** * *** * , *** *** *** ***) sont disponibles. Toute modification du dessin s'affiche dans le style de ligne courant. Le choix du style se fait dans la commande **G**. Un menu affiche les cinq styles existants, chacun préfixé par un chiffre de 0 à 4 (facilité supplémentaire pour sélectionner une option).

Il est toujours intéressant de pouvoir mettre du texte dans un graphique, grâce à l'option **T**. L'utilisateur doit d'abord introduire son texte, puis choisir le type de caractères : graphiques (matrice de 4 points sur 6) ou normaux (les caractères habituels de l'écran - matrice de 8 sur 8 pour une carte graphique IBM, de 9 sur 14 pour une carte HERCULES). Après ce choix, la fenêtre de saisie disparaît, mais un rectangle d'une surface égale à celle occupée par le texte apparaît à l'endroit où se trouve le curseur. L'utilisateur peut déplacer ce rectangle, avant d'appuyer sur **RETURN** pour confirmer. Ensuite, il détermine la taille du texte en utilisant les flèches (Droite ou Bas pour agrandir, Gauche ou Haut pour rétrécir). Une dernière pression sur **RETURN** et le texte s'affiche, en surimpression du graphique existant.

Il faut évidemment pouvoir sauver un graphique. Cela se fait soit via l'option **I** (fIle), qui sauve le graphique entier dans un fichier dont vous préciserez le nom, soit via l'option **J**. Celle-ci permet de ne sauver qu'une partie du graphique en cours d'édition. Il vous faudra donc, avant de

donner un nom de fichier, préciser l'endroit et la taille de ce qui doit être sauvé. Pour afficher à l'endroit du curseur un graphique précédemment sauvé, il faut utiliser l'option **R** (Restore).

Pour imprimer un graphique, il y a l'option **P** (Print). L'enseignant peut préciser un titre, qui s'imprimera en entête. **O** permet de remplir dans la couleur courante la figure fermée dans laquelle se trouve le curseur. On interrompt cette commande en appuyant sur n'importe quelle touche du clavier avant sa fin normale.

Il est également possible, tout comme on peut manipuler des blocs avec un traitement de texte, de traiter des zones de graphique. On peut copier (commande **K**) ou encore déplacer (commande **M** - **M**ove), tout en agrandissant. Le déplacement diffère de la copie parce qu'il efface la zone source. L'un comme l'autre se font en surimpression de ce qui existe déjà dans la zone cible. La commande **N** permet, elle, d'inverser une zone, c'est-à-dire d'en obtenir le négatif. Enfin, on peut faire un "**Z**oom" (commande **Z**) sur une zone de taille fixe. Cela permet de travailler point par point (la barre d'espacement permettant d'inverser le point courant).

Tout cela nous a permis de dessiner, mais encore faut-il pouvoir effacer ! **E** (**E**rase) demande de choisir la taille de la gomme, puis de la déplacer pour effacer ce qui se trouve sur son passage. On quitte cette fonction en appuyant sur **RETURN**.

La commande **X**, elle, permet d'effacer tout le graphique d'un seul coup. **A** utiliser avec prudence ! Heureusement, il y a la dernière commande, **Y**, qui vous rétablit le graphique tel qu'il était avant la dernière commande (dont **E** et **X**).

Nous avons laissé entendre qu'il n'était pas toujours possible de varier la vitesse de déplacement. En effet, certaines commandes (**J**, **K**, **M**, **N**, **Z**) travaillent directement sur l'écran. Or, comme chaque pixel est représenté par un bit, par souci de rapidité de traitement, on préfère travailler sur des bytes entiers, c'est-à-dire huit points

consécutifs. Pour ces commandes, la vitesse de déplacement horizontal restera donc fixée à 8.

L'animation

Ce type de noeud exige plusieurs graphiques de mêmes dimensions. Chacun des graphiques doit porter le même nom, postfixé par son numéro d'ordre dans l'animation. Cela donnera donc par exemple STARS1, STARS2 et STARS3 pour une animation (STARS) comportant 3 graphiques. Il y a au plus cinq graphiques par animation.

La procédure la plus simple pour créer une telle animation est de dessiner le premier graphique, de le sauver par l'option **I**, et de continuer à dessiner à partir de la même base pour créer les graphiques suivants. L'éditeur graphique permet uniquement la création des différentes parties d'une animation. Le reste (délai d'affichage, ...) est spécifié lors de la création de l'enchaînement (cfr point 4).

Modification au toolbox

Comme nous l'avons déjà dit, nous avons adapté toutes ces procédures pour qu'elles fonctionnent quelle que soit la carte graphique employée. Ce n'est pas la seule modification que nous avons apportée au "package" standard.

Il nous a semblé intéressant, lorsqu'on utilise la commande **R**, de ne pas effacer le graphique existant. Cela permet notamment de créer une bibliothèque de petits symboles, employés couramment (par exemple transistors, résistances, ... en électricité), et de les rajouter au schéma en cours d'édition en une seule opération. Dans la version prototype de l'an dernier, l'éditeur graphique était paramétré sur le type de graphique : noeud graphique ou symbole prédéfini. Le chargement d'un symbole prédéfini n'était possible que si l'on travaillait sur un noeud graphique. Le travail avec des symboles prédéfinis est donc beaucoup plus simple maintenant. De plus, un graphique normal peut être utilisé comme un symbole prédéfini.

Nous avons donc modifié la procédure du Toolbox affichant un graphique sauvé sur fichier, pour qu'elle ne détruise plus ce qui existait déjà. Ce n'est pas la seule

modification apportée à cette procédure : à présent, s'il n'y a pas assez de place dans la fenêtre courante pour afficher un graphique, elle ne refuse plus "d'accomplir sa mission". Au contraire, elle affiche maintenant la plus grande partie possible du graphique tout en respectant les limites de la fenêtre.

Algorithme de la fonction "zoom"

Pour des raisons de rapidité, les points corrigés dans cette option sont automatiquement modifiés dans la copie de l'écran conservée précédemment. Pour permettre l'option **Y** après être sortis du zoom, il faut conserver une autre copie intacte du graphique édité : on mémorise donc aussi une copie de la fenêtre de travail.

La taille de la zone grossie est fixe : elle est de 40 points sur 20. La vitesse de déplacement horizontal, lors du choix de la zone à grossir, est fixée à 8. Comme chaque pixel est représenté par un bit seulement (grâce au choix du travail en monochrome), cela représente 5 * 20 bytes susceptible de modification pendant le zoom. Chaque point sera représenté, durant le zoom, par un carré de 8 points sur 8, donc par 8 bytes.

p0 :

```
sauvetage de l'écran courant.
dessin d'un rectangle de 40 points sur 20 à la place du
   curseur
choix :
choix de la zone à agrandir
effacer l'écran
remplissage ecran zoom
dessin croix(0,0)
px <-- 0 (* position en x du coin supérieur gauche du point
          courant, par rapport au coin supérieur gauche
          du 1er point agrandi *)

py <-- 0 (* position en y du coin sup. gauche du point
          courant ; le point courant est indiqué par une
          croix au centre du carré le représentant *)
Tant que non RETURN (* par lequel l'utilisateur indique
          la fin du zoom *)

   Si l'utilisateur frappe :
     H : afficher l'aide concernant le zoom
     flèche vers le haut :
          Si py - 8 ≥ 0
          Alors dessin croix (px,py) (* effacer la croix
                              représentant la position actuelle
                              du curseur *)

                py <-- py - 8
                dessin croix (px,py) (* pour indiquer la
                              nouvelle position du curseur *)

          Sinon beep !
     flèche vers le bas :
          Si py + 15 < 160 (* 160=20 points * 8, où 8 est
                              la taille d'un point ;
```

```
                                    + 15 car il faut en plus
                                    tenir compte de la taille
                                    du point courant *)
            Alors dessin croix (px,py)
                  py <-- py + 8
                  dessin croix (px,py)
            Sinon beep !
      flèche vers la gauche :
            Si px - 8 ≥ 0
            Alors dessin croix (px,py)
                  px <-- px - 8
                  dessin croix (px,py)
            Sinon beep !
      flèche vers la droite :
            Si px + 15 < 320
            Alors dessin croix (px,py)
                  px <-- px + 8
                  dessin croix (px,py)
            Sinon beep !
      espace : (* inversion de la valeur du point courant *)
            changer valeur point (px div 8,py div 8)
               (* modifier la valeur de ce point dans la copie
                  de l'écran *)

            faire un not, directement sur l'écran, sur chacun
             des 8 bytes représentant le point courant
               (* inverser les couleurs du carré représentant
                  le point courant *)

Fin du tant que
Réaffichage de l'écran mémorisé (* et sur lequel on a opéré
     toutes les modifications *)

Réaffichage du curseur à son ancienne position (avant zoom)
```

P1 = P0 où :

remplissage ecran zoom =

.Définir une nouvelle fenêtre, de 336 points sur 176, ce qui permettra de représenter les 40 * 20 points grossis, et de laisser en plus de chaque côté un bord de 8 points (y compris le cadre)
.Sélectionner cette fenêtre
.En dessiner le cadre
.Pour chacun des points de la zone à "zoomer",
  s'il est en couleur (* valeur du bit le représentant = 1*)
     le représenter par un carré de 8 sur 8 à l'endroit
     adéquat de la fenêtre (* cette opération se fait en
          mettant chacun des 8 bytes de l'écran, décrivant
          à présent le point "originel", à 255 *)

---

changer valeur point =

Si le bit décrivant ce point dans la copie de l'écran vaut 1, le mettre à zéro ; sinon, le mettre à 1, toujours sans modifier le reste de la copie de l'écran.

---

dessin croix =

inverser la croix actuelle dans le point donné en paramètre. Chaque croix est composée de deux rectangles de deux points sur 8.

---

figure 3.7

Il est important de noter que lorsque l'utilisateur va définir la zone qu'il veut "zoomer", il peut modifier la vitesse de déplacement. Or la procédure qui exécute cela prend une copie de l'écran avant d'effectuer les I/O qui lui permettent de déterminer les nouvelles vitesses. Pour terminer, elle inverse l'écran et la copie qu'elle en avait prise (elle remplace l'un par l'autre). Cela a pour effet majeur de détruire la première copie d'écran prise en tout début de traitement. Pour que la procédure de Zoom fonctionne correctement, elle doit donc reprendre une nouvelle copie de l'écran, vierge de tout curseur, après passage dans la procédure vitesse.

Beaucoup d'autres fonctions jouent directement sur l'écran, comme par exemple la commande **M** pour déplacer des zones du graphique. Mais le zoom est la seule commande qui travaille directement sur la copie de l'écran.

la commande N (inverse)

Le traitement est ici très simple : comme la vitesse de déplacement horizontal est fixée à 8, et que la largeur de la zone choisie, si elle n'est pas fixe, doit tout de même être un multiple de 8, il suffit de travailler sur des bytes entiers. Par exemple, si la zone à inverser est de 40 points sur 50, il suffira de faire un NOT logique sur chacun des 250 bytes qui "définissent" la zone.

Ainsi donc, l'enseignant peut créer des noeuds de type texte, des graphiques et des animations. Mais comme dans tout bon logiciel d'EAO, il doit également pouvoir interroger ses élèves, soit pour vérifier leurs connaissances ou en forcer l'assimilation, soit pour y adapter la suite de son enseignement.

3.3. Noeud questions-réponses

L'éditeur de ce type de noeud guide totalement l'enseignant : il lui suffit de répondre à une série de questions. Pour lui simplifier encore la vie, la majorité des questions se résument à un choix entre deux options. L'enseignant n'utilise donc que les flèches de déplacement, ou la barre d'espacement, pour faire ses choix. Il peut aussi à tout moment revenir en arrière (touche **F1**) ou abandonner la question en cours d'édition (**F2**). La possibilité pour l'utilisateur de revenir en arrière grâce à **F1** permet également de ne pas alourdir le dialogue, par des demandes répétées de confirmation (Etes-vous sûr ?, Confirmez votre choix, ...). Evidemment, pour l'écriture de la question et de la (des) réponse(s), nous n'avons pas pu lui éviter l'emploi du clavier.

Un autre souci dans la conception de cette partie a été que l'enseignant puisse se rendre compte immédiatement de ce que l'étudiant verra à l'écran lorsqu'il devra répondre au questionnaire. Pour ce faire, il y a toujours deux fenêtres définies : la première (fenêtre enseignant), pour poser les questions à l'enseignant, et une seconde (fenêtre étudiant), où on affiche la question au fur et à mesure de son édition. Bien entendu, la première peut recouvrir partiellement la seconde. C'est pourquoi à tout moment l'utilisateur peut effacer temporairement l'une pour visualiser complètement l'autre, grâce à la touche **F3**. Pour revenir à l'édition de la question courante, il lui suffit alors de frapper n'importe quelle touche.

L'édition d'un noeud questions-réponses est divisée en deux options : l'édition proprement dite, ou la consultation d'un noeud existant. Celle-ci permet de vérifier une à une, chaque fois d'un seul coup d'oeil, toutes les questions d'un noeud.

Dans l'édition elle-même, nous avons essayé de réduire au strict minimum le nombre de questions à poser à l'enseignant. Pour cela, nous tirons le maximum de renseignements des réponses qu'il a déjà fournies. Nous allons maintenant présenter, sous forme d'algorithme, cette séquence nécessaire à l'édition d'un noeud.

saisie_nom :

.Saisie du nom du questionnaire

.Si ce questionnaire existe déjà,

- Demander à l'enseignant s'il veut choisir un autre nom, ou effacer/modifier ce noeud.
- S'il veut choisir un autre nom, aller en **saisie_nom**
- Demander à l'enseignant s'il veut modifier, ou effacer le questionnaire existant.

.S'il n'y avait pas de questionnaire de ce nom, ou que l'enseignant a choisi de l'effacer, ouvrir le fichier en écriture. Sinon, l'ouvrir en lecture, et mettre **modifie** à vrai.

.Si non **modifie**, déterminer la taille de la fenêtre du noeud

.Pour chaque question (maximum 50)

Saisie_numero_question :

- S'il y a déjà une question dans le fichier, demander à l'utilisateur s'il veut définir une nouvelle question (à ajouter en fin de fichier) ou en corriger une existante. S'il choisit la correction, lui demander le numéro de la question à rééditer.
- S'il s'agit d'une réédition d'une question existante, la lire dans le fichier; sinon l'initialiser.
- Saisie du texte de la question (l'énoncé)
- Vérifier si la question comporte bien du texte ; sinon retour à la saisie.
- Demander s'il s'agit d'une question normale, ou d'une question-menu, demandant à l'étudiant comment il veut poursuivre la leçon (auquel cas, la question sera la dernière du questionnaire).
- S'il s'agit d'une question normale :
  + Demander les différentes réponses possibles. Vous pouvez donner de 0 à 5 réponses différentes.
  + La vérification va porter ici sur le nombre de réponses non vide. En même temps, on va procéder au regroupement de ces réponses dans les premières "positions".
  + Si plus d'une réponse introduite, demander quelle est la réponse correcte.
  + Si une seule réponse, il ne s'agit pas d'un QCM, la réponse correcte est celle introduite. Avertir l'enseignant que cette réponse n'apparaîtra pas

dans la fenêtre étudiant.

+ Si pas de réponse introduite, c'est une question sans "réponse correcte". L'étudiant devra répondre par un texte libre, sans que cela puisse avoir une influence sur le déroulement de la leçon, mais sa réponse sera enregistrée dans le fichier compte-rendu de la leçon.
+ Demander si un modèle de simulation est associé à la question. Si oui, en demander le nom. S'il n'existe pas encore, prévenir l'enseignant qu'il devra le créer.
+ Si le professeur a introduit au moins une réponse :
  * Lui demander s'il désire coter la question.
  * Si oui, lui demander le nombre de points par réponse juste et le nombre de points par réponse fausse.
  * Demander le nombre d'essais auquels l'étudiant aura droit.
  * Si l'étudiant n'a droit qu'à un seul essai, demander s'il faudra lui annoncer si sa réponse est bonne ou mauvaise. Appelons cela la confirmation de la réponse.
  * Si oui ou qu'il a droit à plus d'un essai, demander s'il faudra lui afficher la réponse correcte.

- S'il s'agit d'une question "menu", demander quelles seront les différentes options. Vérifier que deux options au moins ont été introduites. Procéder au même regroupement que pour une question normale.
- Réafficher dans la fenêtre enseignant quelles sont les options qu'il a choisies.
- Lui demander une confirmation (elle est enregistrée définitivement dans le fichier) ou un abandon de la question.

figure 3.8

Précisons à présent les notions introduites dans cet algorithme. Tout d'abord la notion de question-menu. Ce type de question n'est pas "autonome", c'est-à-dire qu'il ne suffit pas de l'avoir créée pour que ce qu'elle annonce se réalise à l'exécution des leçons contenant ce questionnaire.

Il faudra en effet qu'en créant la leçon elle-même (cfr point 4, l'enchaînement des noeuds d'une leçon), l'enseignant tienne compte de ce qu'il a proposé comme menu, et qu'il réalise son enchaînement de telle manière que l'étudiant ait réellement ce que le menu offre. Il paraît évident qu'une question de ce type doit être la dernière du questionnaire.

Pour les questions normales, suivant le nombre de réponses introduites, on peut déduire beaucoup de renseignements. Si aucune réponse n'est donnée, le logiciel n'a aucun moyen de vérifier l'exactitude de la réponse fournie par l'étudiant. Du même coup, il ne peut y avoir ni confirmation, ni cotation, ni affichage de la réponse correcte, ni plus d'un essai. ProfCOMP ne pose donc pas ces différentes questions à l'enseignant. Ce type de question convient parfaitement pour permettre des réponses en langue naturelle, sur lesquelles des vérifications sont très difficiles à effectuer.

Parler de simulation dans un questionnaire peut sembler bizarre à première vue. Mais dans des domaines très particuliers (électricité, circuits, ...) il peut être très intéressant de poser des questions par rapport aux résultats d'une simulation, ou encore mieux de demander à l'étudiant de jouer avec cette simulation pour trouver la réponse correcte.

La cotation n'est jamais assez fine dans un tel système. Néanmoins, elle peut être très utile pour adapter le reste de l'enseignement à la valeur de l'étudiant. L'enseignant se base sur les résultats obtenus (questions normales ou questions-menus) pour créer la structure de la leçon.

L'enseignant peut lui-même fixer le nombre de tentatives permises pour répondre à la question. Il veillera à la cohérence par rapport au nombre de réponses qu'il a introduites. Il serait en effet ridicule de proposer 5 essais pour une question où l'étudiant aurait le choix entre 3 réponses. C'est pourquoi l'intervalle des valeurs permises sera de 1 à 9 s'il n'y a qu'une réponse introduite, et de 1 au nombre de réponses s'il s'agit d'un QCM.

Le nombre d'essais permet aussi d'adapter la suite du dialogue avec l'enseignant. En effet, si l'étudiant a droit a plus d'un essai, il serait ridicule de ne pas lui donner confirmation de sa réponse. Car le simple fait de lui redonner ou non une autre chance équivaut à confirmation. Si l'on posait donc cette question à l'enseignant, et que par hasard il répondait "non", il n'y aurait que pour le dernier essai auquel il a droit que l'étudiant n'aurait pas de confirmation ! La confirmation est donc automatique si le nombre d'essais autorisés dépasse 1.

Enfin, faut-il ou non afficher la réponse correcte ? Cette question n'a bien entendu à être posée que si l'étudiant a confirmation de sa réponse.

Voilà en ce qui concerne la question proprement dite. L'enseignant a maintenant le choix entre l'abandon de la question ou son enregistrement. Que faire s'il n'a qu'une petite modification à y apporter avant de la sauver ? Il peut toujours utiliser la touche **F1** qui le ramènera au niveau de son dernier choix.

Et pour l'utilisation de la touche **F2** ? Cela dépend du niveau où l'utilisateur se trouve. Tant qu'il n'est pas dans la définition d'une question (sauf si le fichier-questionnaire édité est encore vide), ou au niveau de la question lui demandant s'il confirme (sauve) ou s'il abandonne la question courante, son effet est d'abandonner l'édition du questionnaire, et donc de revenir au menu précédent. Par contre, si l'enseignant édite une question (et que le fichier-questionnaire n'est pas vide), il n'abandonne que l'édition de la question courante, et on lui repropose le choix entre définition d'une nouvelle question, ou redéfinition d'une question déjà introduite.

Voyons maintenant les autres fonctions disponibles : les flèches verticales permettent à l'enseignant d'aller et venir entre les différentes lignes de la question, les différentes réponses ou options. La touche **End** ou **Fin** lui permet d'indiquer qu'il est satisfait de ce qu'il a déjà introduit; comme les flèches verticales, elle n'est utilisable que dans l'introduction de l'énoncé de la question, des réponses ou des options. L'emploi de cette touche évite à

l'enseignant de parcourir les 15 lignes de texte disponibles pour l'énoncé alors que les deux premières lignes lui suffisent. En correction, ayant l'énoncé complet sous les yeux, il passe immédiatement à la suite du dialogue s'il n'a pas de modification à y apporter. Enfin, comme d'habitude, **F5** affiche le répertoire lorsqu'on lui demande un nom de fichier.

Consultation

Il est également possible de consulter rapidement un questionnaire déjà créé. La consultation se fait également sur deux fenêtres, le passage de l'une à l'autre se faisant par **F3**. Une fois introduit le nom du questionnaire à consulter, les questions seront affichées une à une. Pour passer à la question suivante, il faut utiliser la flèche vers le bas. Pour revenir à la précédente, deux méthodes sont acceptées : **F1** ou la flèche vers le haut. On peut également abandonner prématurément la consultation en utilisant la touche **F2**.

3.4 Noeud simulation

Le dernier type de noeud que l'on peut définir est donc la simulation. Après l'avoir réalisée dans le cadre de ProfCOMP, nous avons été amenés à en faire un programme indépendant, SimEDO, que nous développerons plus tard (cfr chapitre 4). C'est pourquoi nous n'expliquerons maintenant que ce qu'il est nécessaire de savoir pour créer soi-même une simulation. Nous n'aborderons donc pas ici les explications plus techniques (traitement d'une expression, méthodes de résolution,...), les laissant de côté pour ce chapitre sur SimEDO.

Ces deux logiciels, ProfCOMP et SimEDO, sont donc compatibles. C'est très important, dans la mesure où il est plus agréable pour le professeur de créer la simulation avec SimEDO, où il peut la tester immédiatement, et la réutiliser dans une leçon de ProfCOMP. La seule limite à la compatibilité est la taille des systèmes simulés. Pour des raisons de place mémoire, le nombre maximum de variables d'états,... est plus restreint dans ProfCOMP.

Mais que peut-on simuler ? Des systèmes d'équations différentielles ordinaires du premier ordre. Voici le modèle mathématique de tels systèmes pour ProfCOMP :

```
d/dt [X(i)] = f1(t,X(m),P(n),D(q),C(r))      (a)
P(j) = f2(t,X(m),P(n),D(q),C(r))             (b)
D(k) = f3(t,X(m),P(n),C(r))                  (c)
C(l) = réel                                  (d)
avec
    i,m = 1..nombre de variables d'état
    j,n = 1..nombre de paramètres
    k,q = 1..nombre de variables de retard
    l   = 1..nombre de constantes du modèle
et
    1 ≤ nombre de variables d'état ≤ 8
    0 ≤ nombre de paramètres ≤ 8
    0 ≤ nombre de variables de retard ≤ 4
    0 ≤ nombre de constantes ≤ 10
```

figure 3.9

Les seules équations différentielles sont celles des variables d'état (a) mais ProfCOMP autorise également la définition de paramètre(s) (b) et/ou de variable(s) de retard (c) par une équation (non différentielle). Enfin, des constantes (d) peuvent faire partie du modèle. Chacun de ces élément sera expliqué plus en détail par la suite, dans le chapitre 4.

Un système d'ordre supérieur à un peut également être simulé si on sait le ramener à un système d'ordre un.

Chaque fonction f, qui définit le membre de droite de l'équation différentielle de chaque variable d'état ou encore l'équation d'un paramètre ou d'une variable de retard, doit vérifier une certaine syntaxe; celle-ci sera développée de manière précise (Backus-Naur Form) dans le chapitre consacré à SimEDO.

Dialogue

Nous allons maintenant expliciter le dialogue nécessaire à la création d'un modèle. Comme pour tout noeud, il s'agit tout d'abord d'en donner le nom. S'il existe déjà,

une nouvelle question demande au professeur s'il désire la modifier, ou alors l'effacer. On ne laisse donc apparemment pas la possibilité de changer de nom pour ce noeud. Mais, grâce à la touche **F1**, qui a la signification habituelle, on peut revenir au choix de ce nom. D'autres touches de fonction ont une signification importante à travers tout le dialogue. **F2** permet d'abandonner la définition d'une simulation sans la sauver. **End** a le même effet, mais en sauvant les modifications effectuées ; toutefois, si le modèle n'est pas entièrement défini (ce ne peut être le cas en modification), on ne le sauve pas, et **End** a alors exactement le même effet que **F2**. Enfin, comme nous l'avons déjà dit, **F1** permet de revenir une étape en arrière dans le dialogue.

Maintenant, le logiciel connait le nom de la simulation à définir. Si elle existe déjà, et que l'enseignant a choisi de la modifier, ProfCOMP proposera par défaut les valeurs de la simulation existante. Sinon, on initialisera toutes les variables à des valeurs "acceptables". Après cette étape débute la définition de la simulation proprement dite.

La première chose à déterminer sera la taille de la fenêtre à l'exécution. S'il s'agit d'une simulation déjà existante, on va demander à l'enseignant s'il veut modifier les dimensions de cette fenêtre. S'il répond oui, ou qu'il est en train de définir une nouvelle simulation, il fixe la taille en la faisant varier à l'aide des flèches.

Poursuivons le dialogue. Le choix suivant est très important dans le contexte de l'EAO. L'étudiant pourra-t-il ou non créer lui-même sa propre simulation ? Si l'enseignant répond oui à cette question, après avoir "vu" le résultat de la simulation professeur, l'étudiant pourra essayer de l'imiter, ou d'affiner les résultats, en définissant lui-même sa propre "expérience". Il est en effet très important, dans les domaines où la simulation est applicable, que l'étudiant en maîtrise lui aussi les principes et les techniques, et qu'il ne se contente pas d'observer les résultats obtenus par son professeur. Le cours peut donc, si l'enseignant le désire, porter en même temps sur un domaine particulier (électricité, ...) et sur le moyen d'étudier ce

domaine (la simulation).

Ensuite, il s'agit de préciser quelle méthode de résolution sera appliquée. Nous en proposons deux : Euler amélioré (méthode trapézoïdale - deuxième ordre) ou Runge-Kutta de quatrième ordre. Ces méthodes seront développées dans le chapitre consacré à SimEDO, mais retenons néanmoins que la méthode de Runge-Kutta perd en rapidité ce qu'elle gagne en précision par rapport à Euler.

Quel va être le nombre d'équations (ou nombre de variables d'état) du système ? C'est l'objet de la question suivante. Ce nombre peut varier de 1 à 8 dans ProfCOMP ; c'est une des différences avec SimEDO qui permet jusqu'à 20 variables d'état. Cette différence peut poser problème. C'est pourquoi, lorsqu'il s'agit d'une modification d'un modèle, on va vérifier lors de la lecture des valeurs existantes si celles-ci sont supérieures à la limite permise par ProfCOMP. Si oui, l'utilisateur sera informé du problème et renvoyé au menu principal. Cette différence entre les deux logiciels, si elle peut sembler être un accroc grave à la compatibilité, doit cependant être vue comme un souci de rendre SimEDO plus performant. Cette amélioration peut être très importante lorsqu'on utilise ce logiciel de manière autonome. Malheureusement, ce "turbo" placé sur SimEDO ne pouvait l'être sur ProfCOMP, qui puise déjà largement dans les ressources mémoires disponibles.

Revenons au dialogue : après le nombre d'équations, le nombre de paramètres. Les paramètres seront définis par des expressions identiques à celles utilisées pour les variables d'état, sans être évaluées de manière différentielle. ProfCOMP autorise jusqu'à 8 paramètres, alors que SimEDO en offre 20.

Ensuite, le nombre de constantes. On peut en définir 10 au maximum. Il est important d'utiliser les constantes, au lieu de nombres, à l'intérieur des expressions. En effet, si l'étudiant décide de modifier les conditions initiales de la simulation, il pourra également jouer sur les valeurs des constantes. Par contre, les expressions elles-mêmes ne pourront jamais être modifiées lors de l'exécution d'un

modèle-professeur.

Dernière amélioration en date à la simulation, les variables de retard, pour simuler des modèles dans lesquels certaines variables jouent avec un retard dans le temps. Par exemple, si l'on veut simuler l'effet d'un médicament sur le corps humain, il faudra tenir compte du fait que son résultat n'est pas immédiat. Pour y palier, on utilisera les variables de retard. La question suivante porte donc sur leur nombre. Il peut y en avoir quatre au maximum. A ce moment-ci du dialogue, on connait enfin la taille exacte du système à simuler.

A présent, l'enseignant précise les valeurs de temps, primordiales dans la résolution d'un système d'équations différentielles. Tout d'abord, il introduit la valeur du temps en début de simulation (valeur comprise entre -10^5 et 10^5), après quoi il détermine l'incrément de temps, ou pas d'intégration. De cette valeur va dépendre la précision des résultats ; elle doit être comprise entre 10^{-20} et 10^3. Voilà en ce qui concerne le temps pour la résolution du système.

Pour chacune des variables d'état, l'enseignant doit alors donner l'équation différentielle correspondante. En fait, il ne doit en introduire que le membre de gauche, la partie droite étant déjà affichée. Il n'a donc qu'à la compléter. Chaque expression introduite (et cela sera vrai tout au long du programme) est aussitôt vérifiée du point de vue syntaxique, et refusée en cas d'erreur. Nous reviendrons sur la syntaxe exacte des expressions au chapitre 4, consacré à SimEDO.

S'il y a des paramètres dans le système, l'enseignant introduit pour chacun d'eux l'expression qui la définit. Ici, la vérification des expressions est encore plus poussée puisque, si un paramètre d'indice supérieur ou égal à celui que l'on est en train de définir intervient dans l'expression, il faudra, par la suite, demander à l'enseignant la valeur initiale de ce paramètre.

Pour chacune des constantes utilisées (s'il y en a), l'enseignant donne sa valeur, comprise entre $-(10^{20})$ et 10^{20}.

La question suivante est aussi importante que celle demandant si l'étudiant peut introduire son propre modèle. Ici, on demande à l'enseignant si l'étudiant pourra corriger les conditions initiales de la simulation qu'il est en train de définir. C'est également un exercice important pour observer quelle différence de résultat apporte un changement mineur à un modèle. L'étudiant peut ainsi mieux saisir le rôle de chacune des valeurs du système, en modifiant les valeurs initiales des variables d'état et celles des constantes.

Le dialogue se poursuit par la définition des variables de retard : pour chacune d'elles, on va tout d'abord demander l'expression correspondante, et ensuite la valeur même du retard (dans notre exemple ci-dessus, le décalage entre l'absorption du médicament et ses premiers effets). L'expression définissant une variable de retard ne peut contenir d'autres variables de retard. Le retard - ou délai - dépend de l'incrément de temps qui a été choisi pour la résolution du système. Sa valeur doit être comprise entre 1 et 50 fois cet incrément de temps. Donnée en temps, elle sera traduite en nombre de pas d'intégration lors de la résolution du système. Si par malheur la valeur introduite n'appartenait pas à l'intervalle permis (ce qui est possible si après avoir modifié l'incrément de temps d'une simulation existante, l'enseignant utilisait la fonction **End** avant d'avoir corrigé les retards), lors de la résolution, elle sera ramenée à la borne la plus proche de cet intervalle.

Pour terminer la définition du système d'équations différentielles, l'enseignant spécifie les valeurs initiales des variables d'état, ainsi que celles des paramètres (pour ceux dont l'expression contient d'autres paramètres d'indices supérieurs), valeurs comprises entre $-(10^{20})$ et 10^{20}.

Reste à préciser la sortie désirée pour la simulation. Trois possibilités existent : sortie graphique uniquement, numérique uniquement, ou les deux combinées. La sortie numé-

rique seule n'a que peu d'intérêt dans le cadre de l'EAO, cette sortie se faisant vers un fichier de type texte que l'étudiant peut consulter après avoir terminé la leçon. Néanmoins, elle peut être nécessaire pour analyser les résultats obtenus de manière plus affinée.

Si l'enseignant désire un sortie graphique, il peut ne représenter que les points calculés réellement, ou relier ces points entre eux. S'il choisit cette seconde possibilité, deux points successifs seront reliés par une droite. Il n'est nullement question ici de faire une interpolation pour affiner la courbe. D'ailleurs les différents points à représenter, généralement assez rapprochés, éliminent l'intérêt d'une interpolation, qui empêcherait l'affichage progressif des résultats, puisque cette technique ne s'applique que lorsqu'on connait l'ensemble des points d'une courbe.

Ensuite, l'enseignant précise le nombre de graphiques qu'il désire. Il peut afficher jusqu'à cinq graphiques en même temps dans la même fenêtre. Si les différents points sont reliés entre eux, on représente chacun des graphiques dans un style de ligne différent, pour les distinguer clairement les uns des autres. ProfCOMP affiche le style de ligne lors de la définition de chacun des graphes.

Dernière question qui se rapporte à la sortie graphique en général : la fréquence de représentation des points. L'enseignant décide ici s'il affichera tous les points calculés (fréquence=1) ou s'il n'en désire qu'un sur x. La fréquence x sera un nombre entier compris entre 1 et 100.

Il faut maintenant définir chaque graphe séparément. Tout d'abord, en donnant l'expression de chaque axe : par exemple, on représentera t (le temps) en abscisse et x(1) (première variable d'état) en ordonnée. Ces expressions peuvent être évidemment beaucoup plus complexes. Leur affichage sur les axes lors de l'exécution est malheureusement impossible, puisqu'on peut représenter plusieurs graphes simultanément. De même, il est impossible de représenter une échelle de valeur sur les axes.

Après, l'enseignant donne, pour chacune de ces expressions, l'intervalle de représentation, c'est-à-dire les valeurs de ces expressions pour lesquelles il y aura effectivement affichage du résultat. Pour chacun des axes, l'intervalle est compris entre $-(10^{35})$ et 10^{34} pour la borne inférieure, et entre la borne inférieure et 10^{35} pour la borne supérieure. Ces valeurs définissent l'échelle de représentation lors de l'exécution. Devoir introduire ces valeurs peut sembler contraignant pour l'enseignant : mais d'une part, cela permet d'afficher chaque résultat dès qu'il est calculé, sans attendre d'avoir terminé la résolution du modèle pour calculer l'échelle, et d'autre part, l'enseignant peut toujours, grâce à SimEDO, tester son modèle avec une sortie numérique pour observer les variations des valeurs.

Pour les sorties numériques, s'il y en a, l'enseignant va tout d'abord en préciser le nombre (trois maximum). Ensuite, il définit l'expression de chacune des sorties, et enfin la fréquence d'écriture des résultats dans le fichier. Cette fréquence est un entier compris entre 1 et 100, comme pour la sortie graphique. Un exemple de sortie numérique est montré en annexe B, dans le manuel de SimEDO, page 29.

Pour terminer, quel que soit le type de sortie demandé, il va falloir préciser l'intervalle de temps pendant lequel on effectue ces sorties. En effet, on peut commencer une simulation au temps 0, mais de n'en afficher les résultats qu'à partir du temps 10 par exemple. La valeur inférieure est un réel compris entre -10^5 et 10^5, la valeur supérieure étant elle comprise entre la borne inférieure et 10^6.

La simulation est maintenant entièrement définie, et peut être exécutée. Si on n'a pas utilisé la touche **F2** d'abandon d'édition, elle est alors sauvée dans le fichier.

4. L'enchaînement des noeuds d'une leçon

4.1. Introduction

Avec les outils présentés ci-dessus, l'enseignant crée les noeuds qui composent une leçon. Reste à définir la structure de la leçon (synonyme d'enchaînement d'une leçon), c'est-à-dire les différents schémas possibles d'exécution de noeuds, le positionnement des fenêtres à l'écran et les opérations de manipulation de celles-ci.

Pour mener à bien cette mission, un éditeur d'enchaînement propose un petit langage indispensable pour spécifier les actions à effectuer sur les noeuds ou sur les fenêtres lors de l'exécution.

Enfin, un compilateur se chargera de vérifier la correction sémantique de la suite d'instructions et de la rendre exécutable.

4.2. Le langage d'enchaînement

La structure de la leçon s'exprime à l'aide d'un petit langage, composé de verbes anglais qui correspondent aux actions permises sur un noeud ou sur une fenêtre à l'écran. Dans un premier temps, nous décrirons la syntaxe de ce langage avant d'expliciter la sémantique de chaque instruction.

4.2.1. La syntaxe du langage d'enchaînement

Nous exprimons la syntaxe dans le formalisme BNF. Voici la définition des quelques métasymboles que nous utilisons mais qui ne font pas partie du langage d'enchaînement :

::= signifie "est défini comme",

| indique une alternative,

[] précise l'aspect facultatif,

{} délimite un ensemble,

.. désigne une des valeurs entières comprises dans l'intervalle défini par le nombre qui précède les 2 points et celui qui suit ces 2 points, et

... désigne une des lettres comprises entre les bornes qui entourent ces 3 points.

Nous soulignerons les constructions syntaxiques et les mots du langage sont en caractères gras.

```
LECON ::= enchaînement
amener-fenêtre-au-dessus ::= GET WINDOW
                             identificateur-de-fenêtre UP ;
coin-sup-gauche-X ::= 0..79 (carte IBM)
                    | 0..89 (carte HERCULES)
coin-sup-gauche-Y-GRF ::= 0..191 (carte IBM)
                        | 0..335 (carte HERCULES)
coin-sup-gauche-Y-TXT ::= 0..23
déclaration-étiquette ::= identificateur :
délai-affichage ::= 1..30
effacement-écran ::= CLEAR ALL THE SCREEN ;
effacement-fenêtre ::=
                    ERASE WINDOW identificateur-de-fenêtre ;
enchaînement ::= {ligne}
étiquette ::= identificateur
exécution-noeud ::= exécution-noeud-animation |
                    exécution-noeud-graphe |
                    exécution-noeud-qcm |
                    exécution-noeud-simulation |
                    exécution-noeud-texte
exécution-noeud-animation ::= EXECUTE nom-noeud.ANI
                              fenêtre-animation ;
exécution-noeud-graphe ::= EXECUTE nom-noeud.GRF
                           fenêtre-graphe ;
exécution-noeud-qcm ::= EXECUTE nom-noeud.QCM fenêtre-texte
                        [WITH SIMULATION fenêtre-graphe] ;
exécution-noeud-simulation ::= EXECUTE nom-noeud.SIM
                               fenêtre-graphe ;
exécution-noeud-texte ::= EXECUTE nom-noeud.TXT
                          fenêtre-texte ;
fenêtre-animation ::= (identificateur-de-fenêtre,
                       coin-sup-gauche-X,
                       coin-sup-gauche-Y-GRF,
                       nombre-graphe-animation,
                       délai-affichage)
fenêtre-graphe ::= (identificateur-de-fenêtre,
                    coin-sup-gauche-X,
                    coin-sup-gauche-Y-GRF)
```

```
fenêtre-texte ::= (identificateur-de-fenêtre,
                   coin-sup-gauche-X,
                   coin-sup-gauche-Y-TXT)
identificateur  ::= une suite de 1 à 15 caractères dont le
code ASCII est compris entre 32 et 126 ou entre 128 et 254
identificateur-de-fenêtre ::= 1 | 2 | 3 | 4 | 5 | 6 | 7
identificateur-de-fichier-sans-extension ::= une suite de 1
à 8 caractères choisis parmi a...b, A...Z, 0..9 et _
instruction ::= amener-fenêtre-au-dessus |
                effacement-écran |
                effacement-fenêtre |
                exécution-noeud |
                instruction-conditionnelle |
                instruction-de-branchement |
                inversion-deux-dernières-fenêtres
instruction-conditionnelle ::=
                                 instruction-conditionnelle-menu
                               | instruction-conditionnelle-simple
instruction-conditionnelle-menu ::= IF MENU CHOICE option
instruction-conditionnelle-simple ::=
                  IF   premier-membre opérateur second-membre
instruction-de-branchement ::= GOTO étiquette ;
inversion-deux-dernières-fenêtres ::=
                                    INVERT THE 2 LAST WINDOWS ;
ligne ::= instruction | déclaration-étiquette
nombre-entier ::= -32768..32767
nombre-graphe-animation ::= 2..5
nom-noeud ::= identificateur-de-fichier-sans-extension
opérateur ::= < | <= | > | >= | = | <>
option ::= A | B | C | D | E
premier-membre ::= MCQ POINTS | MCQ TRIALS |
                   TOTAL POINTS | TOTAL TRIALS |
                   (MCQ POINTS/POSSIBLE MCQ POINTS) |
                   (TOTAL POINTS/POSSIBLE TOTAL POINTS)
second-membre ::= nombre-entier
```

Figure 3.10 - la syntaxe du langage d'enchaînement

4.2.2. Sémantique des instructions

a) L'exécution d'un noeud

TOUTE LECON DOIT COMPORTER AU MOINS UNE EXECUTION DE NOEUD

Syntaxe : EXECUTE name.ext (n1,x1,y1[,z,d])
[WITH SIMULATION (n2,x2,y2)];
exécute le noeud nommé "name" de type symbolisé par "ext" dans la fenêtre identifiée par l'entier n1 dont le coin supérieur gauche se trouve en (x1,y1) par rapport au coin supérieur gauche de l'écran. Les éléments entre crochets seront détaillés par la suite car ils ne sont définis que pour l'exécution de type de noeuds bien précis.

Comme nous l'avons vu précédemment, les outils offerts pour la mise au point d'une leçon permettent de créer des noeuds questions-réponses, texte, simulation, graphique ainsi que les composants d'une animation. Dans une instruction d'exécution, c'est l'extension "ext" qui identifie le type de noeud exécuté :
- TXT pour un noeud texte,
- GRF pour un noeud graphique,
- SIM pour une simulation,
- QCM pour un noeud questions-réponses et
- ANI pour une animation.

Le nom du noeud, "name", correspond à celui qui a été défini au moment de sa création, sauf pour une animation. En effet, une animation se compose de plusieurs graphiques de même dimension affichés circulairement dans une même fenêtre.
Exemple : soient GRAPH1.GRF, GRAPH2.GRF et GRAPH3.GRF les trois graphiques qui composent une animation. L'ordre d'affichage d'un composant dans la séquence est obligatoirement déterminé par le dernier caractère de son nom. Ces trois composants seront affichés circulairement, c'est-à-dire qu'après l'affichage de GRAPH1.GRF, GRAPH2.GRF et GRAPH3.GRF, on réaffichera GRAPH1.GRF, et ainsi de suite jusqu'au moment où l'étudiant décide la fin de l'animation.

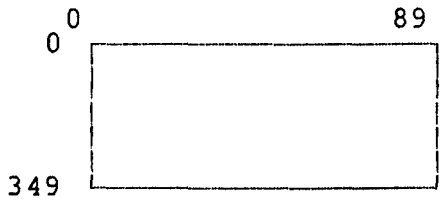
Quel nom donner pour une animation ? Le **radical commun des noms de tous les composants,** obtenu en en supprimant le dernier caractère (qui correspond en fait à l'ordre d'affichage dans la séquence). Dans l'exemple ci-dessus, le radical commun est GRAPH. On aura donc une instruction du type EXECUTE GRAPH.ANI (...);.

Tout noeud s'exécute dans une fenêtre. Le logiciel autorise la représentation d'au plus sept fenêtres à l'écran, chacune identifiée par un entier compris entre 1 et 7. Le paramètre "nl", dans la syntaxe ci-dessus, identifie précisément cet entier. La redéfinition d'une fenêtre, c'est-à-dire la réutilisation de son numéro dans une instruction, efface son ancien contenu à l'écran.

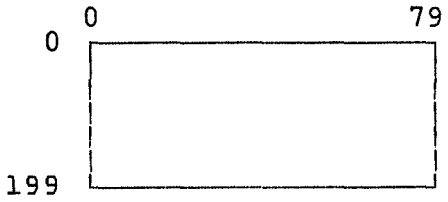
Pour déterminer la position d'une fenêtre à l'écran, l'utilisateur doit choisir l'endroit de l'écran où il désire que le coin supérieur gauche de sa fenêtre vienne se placer. Pour situer précisément cet endroit, nous avons défini un système de coordonnées cartésiennes à deux dimensions, d'origine le coin supérieur gauche de l'écran :

On distingue deux types de coordonnées cartésiennes :

- Les coordonnées GRAPHIQUES utilisées pour situer la fenêtre d'un noeud de type graphique, animation ou simulation. Etant donné que ProfCOMP travaille indifféremment sur un ordinateur qui dispose d'une carte Hercules ou IBM (Ega ou Cga), le nombre de subdivisions horizontales et verticales de l'écran change selon le type de carte utilisée. L'utilisateur doit savoir de quelle carte dispose son ordinateur pour situer efficacement les fenêtres. En mode Hercules, un écran graphique se définit de la manière suivante :

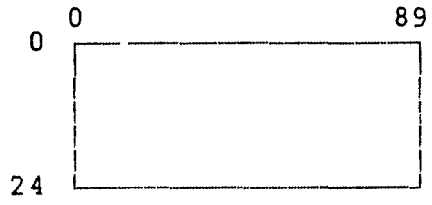


tandis que le mode IBM offre :

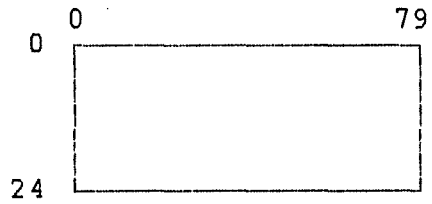


- Les coordonnées TEXTES pour situer la fenêtre d'un noeud texte ou questions-réponses. Même distinction pour les deux types de cartes :

* Hercules :



* IBM :



Dans la syntaxe, "x1" correspond à la position horizontale du coin supérieur gauche de la fenêtre tandis que "y1" correspond à la position verticale de ce même point. A l'utilisateur d'introduire les bons nombres pour situer correctement ses fenêtres en tenant compte de la carte graphique de l'ordinateur et du système de coordonnées qui correspond au type du noeud à exécuter. En observant la syntaxe, on remarque que la position verticale d'une fenêtre de type texte se situe dans l'intervalle 0..23. Vous comprendrez très facilement cette diminution d'une unité; en effet, toute fenêtre TEXTE contient au moins une ligne. Dès lors, pour qu'elle soit affichable sur l'écran, la position de son coin supérieur gauche peut se trouver au plus bas sur la ligne 23. C'est l'explication de cette diminution d'une unité. On peut faire la même remarque pour la coordonnée verticale d'un noeud graphique, animation et simulation. Si l'on donne une position trop basse ou trop à droite pour une fenêtre dont on ne verrait qu'une portion sur l'écran,

Remarque : toutes les simulations qui se rappportent à un même noeud questions-réponses s'exécutent dans une fenêtre identifiée par le même numéro et dont le coin supérieur gauche se trouve toujours en (x2,y2). Les dimensions varient selon le modèle exécuté.

Quelques exemples d'instructions d'exécution de noeuds :

* EXECUTE TOTO.QCM (1,5,12) ;
  ≡ exécuter le noeud questions-réponses TOTO dans la fenêtre 1 positionnée en (5,12)

* EXECUTE GRAPH.ANI (3,10,180,3,1) ;
  ≡ exécuter le noeud animation GRAPH dans la fenêtre 3 positionnée en (10,180). Trois graphiques composent cette animation : GRAPH1.GRF, GRAPH2.GRF et GRAPH3.GRF. Le délai d'affichage de chaque composant est d'une seconde.

b) Effacer l'écran

Syntaxe : ERASE ALL THE SCREEN ;

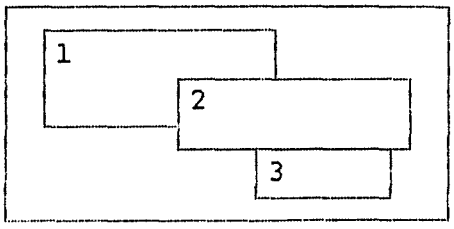
Le contenu de toutes les fenêtres affichées à l'écran est irrémédiablement perdu. On retrouve l'écran dans un état semblable à celui du début de la leçon.

c) Effacement d'une fenêtre

Syntaxe : ERASE WINDOW n ;

L'écran peut représenter jusqu'à 7 fenêtres différentes. Cette instruction permet d'en effacer une bien précise : celle identifiée par l'entier "n". Si la fenêtre "n" n'a pas encore été utilisée pour une exécution de noeud ou qu'elle a déjà été effacée, cette instruction restera sans effet.

Exemple : trois fenêtres, identifiées par les numéros 1,2 et 3 sont utilisées à l'écran :



Après exécution de l'instruction ERASE WINDOW 2, l'écran n'affichera plus que les fenêtres 1 et 3 :

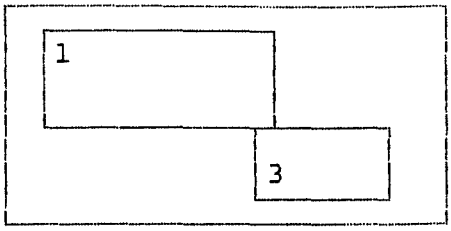


Figure 3.11 - Effacement d'une fenêtre

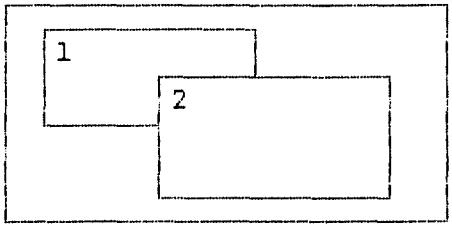
d) Inverser les deux dernières fenêtres

Syntaxe : INVERT THE TWO LAST WINDOWS ;

Cette instruction n'a de sens que si au moins deux fenêtres sont présentes à l'écran. Supposons que les deux dernières instructions exécutées soient

```
EXECUTE TOTO.TXT (1,5,4) ;
EXECUTE TITI.GRF (2,25,70) ;
```

et que l'écran affiche les fenêtres 1 et 2 comme suit :



La fenêtre 2 recouvre partiellement la 1 car sa définition est postérieure. Il peut être intéressant de ramener la fenêtre de texte numéro 1 à l'avant-plan avant de poursuivre l'exécution des noeuds suivants. C'est le but de cette instruction. Comme les fenêtres 1 et 2 ne sont pas disjointes, la fenêtre 1, à son tour, recouvrira partiellement la 2 après cette demande d'inversion :

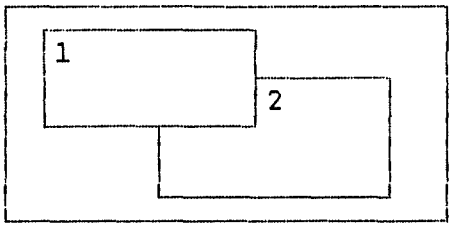


Figure 3.12 - Inversion des 2 dernières fenêtres

L'instruction reste sans effet quand moins de 2 fenêtres sont présentes à l'écran.

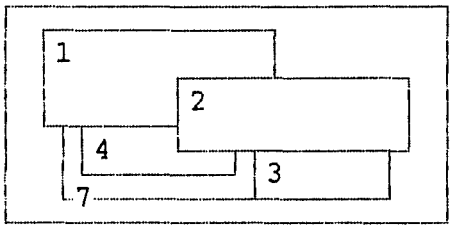
e) Ramener une fenêtre à l'avant-plan

Syntaxe : GET WINDOW n UP ;

Les dimensions de l'écran ne permettent pas toujours une représentation disjointe des fenêtres. Il est parfois utile, à un moment déterminé de la leçon, de ramener à l'avant-plan une fenêtre cachée partiellement ou totalement par d'autres. C'est précisément l'action qu'effectue cette instruction. Si la fenêtre de numéro spécifié n n'existe pas à l'écran ou si elle n'est obstruée par aucune autre fenêtre, l'instruction restera sans effet.

Remarque : on peut réaliser l'instruction INVERSE THE 2 LAST WINDOWS à l'aide de cette instruction mais nous avons voulu la conserver pour une question de facilité lorsqu'il y a couplage entre 2 noeuds.

Exemple : La fenêtre 7 est partiellement cachée par les fenêtres 1, 2 et 4 :



Après exécution de GET WINDOW 7 UP, l'écran affichera la fenêtre 7 en premier plan; la fenêtre 4 est maintenant totalement cachée tandis que les fenêtres 1 et 2 sont partiellement recouvertes par la 7 :

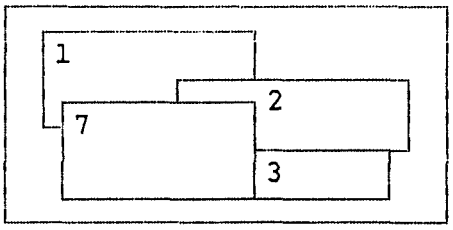


Figure 3.13 - Ramener une fenêtre à l'avant-plan

f) Etiquette et GOTO

Syntaxe : - d'une étiquette : nom :
- d'un GOTO : GOTO nom ;

En principe, les instructions de la structure sont interprétées séquentiellement, les unes à la suite des

autres. On peut déroger à ce principe en forçant le programme d'exécution à ALLER VERS (goto en anglais) une autre partie de la leçon. Comme cette autre partie doit être spécifiée, une étiquette sera placée avant sa première instruction.

Exemple :

```
EXECUTE TOTO.GRF (1,1,1) ;
EXECUTE TITI.QCM (2,10,15) ;
GOTO TOTO ;
EXECUTE TUTU.SIM (3,50,52) ;
TOTO :
EXECUTE TETE.ANI(4,1,1,4,1) ;
...
```

Après l'affichage de TOTO.GRF et l'exécution du noeud questions-réponses TITI.QCM, l'exécution se poursuivra par l'animation TETE.ANI. Cet exemple est stupide car on n'exécutera jamais TUTU.SIM. Pour le moment, l'utilisation d'étiquettes et de GOTO n'a pas beaucoup de sens puisqu'on peut toujours supprimer toutes les étiquettes et les GOTO qui apparaissent dans la suite d'instructions. On comprendra mieux l'utilité des étiquettes et branchements inconditionnels à la présentation de l'instruction suivante.

g) L'instruction conditionnelle

Syntaxe : IF cond

Si la condition "cond" est évaluée à VRAI, l'exécution poursuit son cours à la ligne d'instruction suivante. Si par contre la condition n'est pas vérifiée, l'exécution déroge aussi au principe de séquentialité et se poursuit sur la deuxième ligne d'instruction qui suit ce IF. Nous parlons ici d'instructions : les lignes contenant une étiquette ne sont pas prises en compte !!!

Que peut-on trouver comme condition ?

- un choix-menu de l'étudiant :
  Syntaxe : MENU CHOICE lettre
  avec lettre = A, B, C, D ou E

Dans un noeud questions-réponses, la dernière question peut être un choix-menu destiné à orienter le suite de la leçon. Par exemple, on pourrait poser la question suivante à l'étudiant, après qu'il ait exécuté le premier chapitre de la leçon :

Voulez-vous :
A passer au chapitre 2
B relire la chapitre 1
C faire les exercices du chapitre 1
Votre choix ?

Dans la structure de la leçon, après l'instruction d'exécution de ce noeud, 2 instructions conditionnelles IF examineront la réponse de l'étudiant pour diriger la leçon à l'endroit désiré :

```
IF MENU CHOICE A
GOTO CHAPITRE2 ;
IF MENU CHOICE B
GOTO CHAPITRE1 ;
GOTO EXERCICE1 ;
...
CHAPITRE1 : ...
...
EXERCICE1 : ...
...
CHAPITRE2 : ...
...
```

Nous supposerons les étiquettes CHAPITRE1, EXERCICE1 et CHAPITRE2 disposées aux bons endroits.

- Une condition portant sur une variable prédéfinie du système

Syntaxe : variable opérateur nombre

L'orientation d'une leçon dans une direction plutôt qu'une autre peut également dépendre des performances de l'étudiant. Nous proposons un ensemble de variables prédéfinies, gérées par le module d'exécution d'une leçon et qui portent soit sur l'exécution du dernier noeud questions-réponses, soit sur l'ensemble des noeuds questions-réponses exécutés. Quand on interroge l'étudiant, chaque tentative de réponse est enregistrée : une première variable comptabilise le nombre total de tentatives de réponses pour toutes les questions d'un noeud : MCQ TRIALS. Le nombre de points obtenus par l'étudiant pour les questions qui lui sont posées constitue une autre variable : MCQ POINTS. La dernière divise le nombre de points obtenus par le nombre de points maximum pour le noeud : (MCQ POINTS / POSSIBLE MCQ POINTS). La valeur de cette variable est un nombre décimal compris entre 0 et 1, c'est pourquoi nous multiplions cette valeur par 100 pour que l'utilisateur introduise un nombre entier comme second membre (nombre dans

la syntaxe) et non un réel. Les questions libres pour lesquelles le professeur n'a pas défini de réponses ne sont pas prises en compte dans la comptabilisation des essais et points. On retrouve les mêmes variables mais qui portent sur l'ensemble des noeuds questions-réponses auxquels l'étudiant a déjà répondu dans la leçon :

- le nombre total des essais de réponses : TOTAL TRIALS,
- le nombre total des points obtenus : TOTAL POINTS et
- le rapport entre le nombre de points obtenus pour tous les noeuds et le nombre de points maximum (également multiplié par 100) : (TOTAL POINTS / POSSIBLE TOTAL POINTS).

Les opérateurs admis : <, ≤, >, ≥, = et <>.
Le second membre est toujours un nombre entier.

Toutes les variables sont initialisées à zéro au début de la leçon de sorte que si on rencontre une instruction conditionnelle dans la structure de la leçon alors qu'aucun noeud questions-réponses n'a été exécuté ou qu'aucune question côtée n'a été posée à l'étudiant, la condition puisse quand même être évaluée.

Exemple : Après la présentation textuelle du chapitre 1 (CHAP1.TXT) et l'interrogation de l'étudiant sur ce même chapitre (CHAP1.QCM), on veut lui rappeler les notions principales s'il n'a pas atteint 70 % des points avant de passer au chapitre 2 :

```
EXECUTE CHAP1.TXT (1,1,1) ;
EXECUTE CHAP1.QCM (2,20,7) ;
IF (MCQ POINTS/POSSIBLE MCQ POINTS) < 70
EXECUTE RAPPEL.TXT(4,12,56);
EXECUTE CHAP2.TXT (1,10,10) ;
...
```

Remarquons que le noeud CHAP2.TXT réutilise la fenêtre 1. On suppose donc que le contenu de cette fenêtre est devenu inutile à cet endroit de la leçon.
Après l'introduction de cette instruction conditionnelle, on comprend mieux l'utilité des GOTO et étiquettes.

4.3. L'éditeur d'enchaînement

Pour éditer la structure d'une leçon, nous avons essayé de simplifier cette tâche au maximum en offrant à l'utilisateur un éditeur d'enchaînement, orienté vers le langage explicité ci-dessus et qui permet d'éditer une structure efficacement et rapidement. En début d'utilisation, l'enseignant tapera le nom du fichier qui contient la structure de leçon qu'il désire modifier. S'il désire créer une nouvelle structure, il tapera **RETURN** ou introduira le nom qu'elle portera. En fin d'édition, l'éditeur proposera par défaut le nom défini en début d'utilisation pour sauver le fichier d'instructions sur l'unité des données. Si l'utilisateur désire changer ce nom par défaut, il pourra le modifier. L'édition se termine sans sauvetage si le nom donné est vide. L'extension choisie par l'éditeur pour de tels fichiers : ENC.

L'éditeur propose toujours deux fenêtres (voir figure 3.14 page suivante) :

- La fenêtre supérieure affiche les instructions déjà introduites ou du moins une partie si leur nombre est supérieur au nombre de lignes que peut afficher cette fenêtre. Le curseur, symbolisé par "**>>**" en début de ligne et "**<<**" en fin de ligne désigne l'instruction courante.

- La fenêtre inférieure guide l'utilisateur en lui proposant les options courantes ou en lui demandant d'introduire un élément bien précis (un paramètre d'instruction par exemple). C'est donc cette fenêtre qui permet d'éditer la structure de la leçon, le résultat étant visible dans la partie supérieure.

Chaque fois que l'utilisateur a le choix parmi plusieurs options, un menu de la fenêtre inférieure les présente : un rectangle en vidéo inversé y indique l'option courante. Un symbole représentant une touche particulière du clavier (**F1**, **F2**, **END**, **1**, **2**,...) précède chaque option. Pour sélectionner une option, l'utilisateur peut :

- frapper la touche qui correspond au symbole placé devant l'option, ou
- déplacer le rectangle en vidéo inversé à l'aide des flèches sur l'option désirée et confirmer par **RETURN**.

Le menu principal affiche toutes les possibilités : définir une instruction, une étiquette, insérer ou supprimer une instruction de la structure ou encore terminer l'édition.

Quand l'utilisateur se lance dans la définition d'une instruction, l'éditeur le guide pour définir chacun de ses paramètres (s'il y en a), par des menus ou par une demande d'introduction manuelle lorsqu'il n'est pas possible de proposer toutes les possibilités.
Exemples : le nom du noeud exécuté, une étiquette, un entier ou un réel.
L'éditeur vérifie syntaxiquement les entrées manuelles avant de les accepter.
Exemples :
- Un numéro de fenêtre ne sera accepté que si l'utilisateur introduit un chiffre entre 1 et 7.
- Une position de coin supérieur gauche en dehors des bornes acceptées par le système de coordonnées cartésiennes adéquat sera refusée.

L'éditeur met à jour la fenêtre supérieure après chaque sélection ou introduction manuelle par écriture de l'élément.
Exemple : déroulement de la définition de l'instruction IF MENU CHOICE B
L'utilisateur doit d'abord sélectionner l'instruction IF du menu principal :

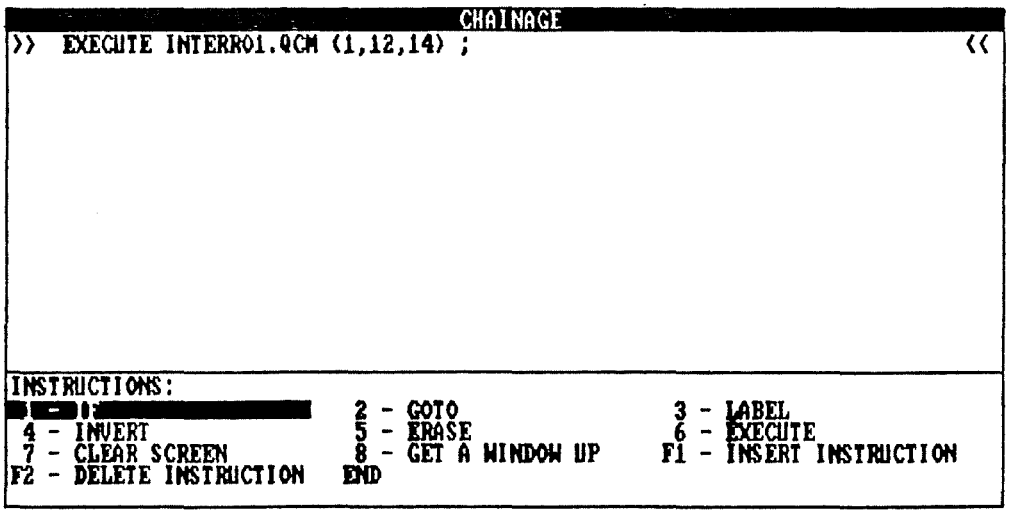


Figure 3.14 - Sélection de l'option IF dans le menu

Ensuite, un second menu propose toutes les variables que l'on peut tester après un IF mais aussi l'option du choix menu de l'étudiant :

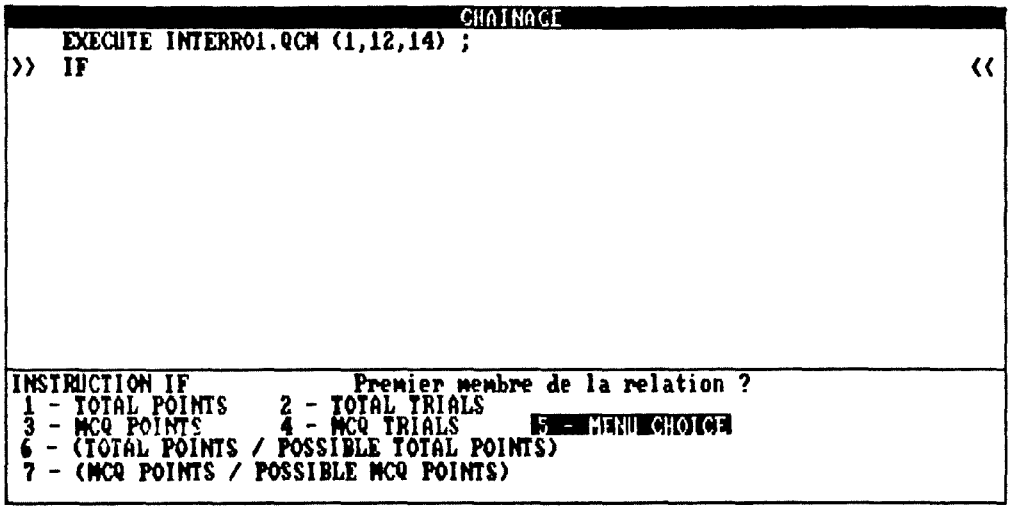


Figure 3.15 - Sélection du premier membre de la relation

Enfin, on clôture l'instruction en tapant la lettre qui correspond au choix menu désiré (introduction manuelle) :

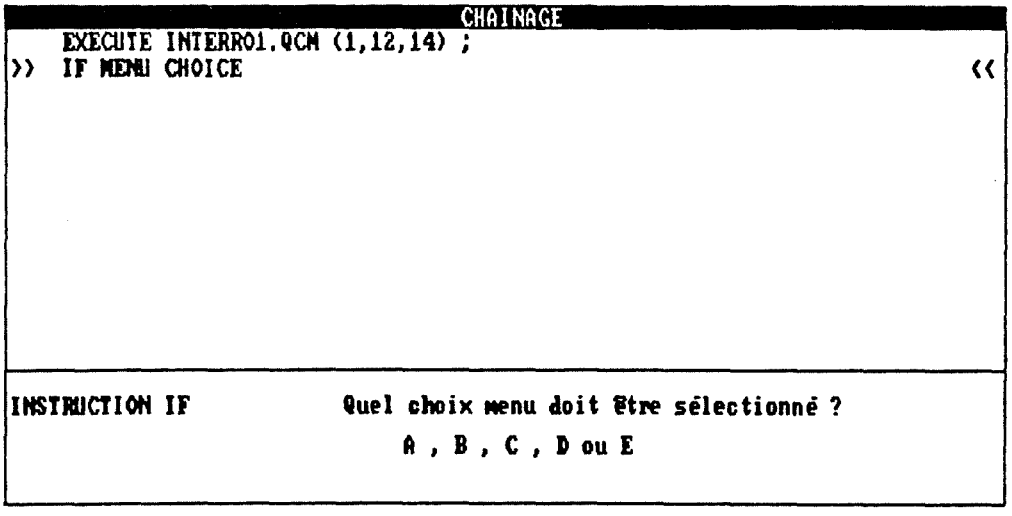


Figure 3.16 - Choix de l'option menu

Toute l'édition de la structure de la leçon se passe via la deuxième fenêtre. L'utilisateur n'a accès à la fenêtre supérieure que pour détruire ou insérer une instruction. Dans ce cas-là, il doit, bien entendu, positionner le curseur au bon endroit : il le fera à l'aide des flèches HAUT et BAS. Dans un cas de suppression, il s'agit de placer le curseur sur la ligne de l'instruction à détruire tandis que pour une insertion, on positionnera d'abord le curseur sur la ligne au-dessus de laquelle on désire insérer une instruction. Toute pression de la flèche HAUT (BAS) quand le curseur se trouve sur la première (dernière) ligne de la fenêtre provoquera un déroulement d'une ligne pour que l'instruction précédente (suivante) soit visible à l'écran. Utiliser la flèche HAUT (BAS) sur la première (dernière) ligne d'instruction restera sans effet. Après une de ces deux opérations, le curseur indiquera la dernière instruction de la structure.

Signalons également que l'utilisateur peut abandonner à tout moment l'instruction qu'il définit (touche **F3**).

4.4. La compilation d'un enchaînement

L'édition de la structure d'une leçon produit un fichier d'enregistrements dont chacun contient soit une instruction et ses paramètres (s'il y en a), soit une étiquette. Le module PC_EXEC est incapable d'exécuter un tel fichier, pour 2 raisons : d'une part l'éditeur d'enchaînement n'effectue aucun contrôle sémantique sur les instructions, d'autre part les étiquettes doivent être remplacées par un mécanisme plus rapide qui désigne directement l'endroit où poursuivre la leçon dès qu'on rencontre un GOTO.

Dès lors, un traitement complémentaire et obligatoire s'impose à l'issue de l'édition d'une structure de leçon : la compilation. Son but est d'obtenir un fichier d'instructions sémantiquement correct et exécutable qui portera le même nom que le fichier de la structure éditée mais l'extension LES. Le contrôle sémantique :

- détecte les erreurs fatales (fatal error) qui stoppent immédiatement la compilation et empêchent toute exécution de la leçon. Ces erreurs doivent être corrigées via l'éditeur d'enchaînement avant d'essayer de recompiler,
- donne des avertissements (warning) qui n'empêchent pas la compilation mais dont il vaudrait mieux tenir compte avant l'exécution. Dans la plupart des cas, il s'agit d'erreurs qui pourraient donner des effets non désirés par l'utilisateur.

La démarche d'utilisation du compilateur est la suivante : l'utilisateur introduit le nom de la structure qu'il désire compiler, ce qui déclenche la compilation, puis le diagnostic s'inscrit à l'écran : la compilation s'est bien déroulée (aucune erreur syntaxique ou sémantique) ou bien les messages d'avertissements et/ou d'erreur fatale avertissent l'utilisateur de l'incohérence. Pour rendre un fichier d'enregistrements exécutable (en l'absence d'erreurs, il suffit de supprimer tous les enregistrements "étiquette" et modifier les enregistrements GOTO en remplaçant l'étiquette associée par l'adresse de l'enregistrement qui suit la déclaration de cette étiquette.

La procédure de compilation s'effectue en parcourant deux fois SEQUENTIELLEMENT la suite d'instructions. Le premier passage consiste principalement à construire une table qui, pour chaque étiquette, reprend son nom et son adresse. Pour déterminer l'adresse d'une étiquette, on utilise un compteur d'instructions, initialisé à zéro au début de la compilation, et incrémenté d'une unité à chaque instruction rencontrée (voir exemple en 4.5). Seules les étiquettes ne sont pas considérées comme instruction (cfr syntaxe). Quand on rencontre une étiquette dont le nom ne figure pas encore dans la table, on l'y ajoute avec la valeur du compteur d'instructions comme adresse. Si, par contre, on tombe sur un identifiant d'étiquette déjà déclaré, on en déduit qu'il s'agit d'une erreur fatale : la double déclaration d'une étiquette.

A la fin du premier passage, d'autres vérifications peuvent être opérées :

- Toute leçon, par définition, consiste à exécuter une série de noeuds. Par conséquent, si le compilateur n'a pas rencontré l'instruction EXECUTE dans l'enchaînement, ça n'a aucun sens d'exécuter cette leçon. Une erreur fatale signale au concepteur qu'il n'est pas possible d'avoir une leçon sans exécution de noeud.

- Si une étiquette termine la leçon, un avertissement signale ce fait. Ce n'est pas à proprement parler une erreur s'il s'agit d'un moyen de sortir de la leçon mais nous avons jugé bon de le signaler.

- Nous avons vu qu'une instruction IF doit être suivie de deux instructions. La première s'exécute quand la condition est évaluée à VRAI tandis que le seconde l'est quand elle est évaluée à FAUX. Si un IF est suivi de moins de 2 instructions, un avertissement signale ce fait à l'utilisateur. Un IF comme dernière instruction d'une structure constitue manifestement une erreur puisque cette instruction ne servirait à rien. S'il n'y a qu'une instruction qui suit, ça peut être un moyen de terminer la leçon dans un cas et de faire autre chose dans l'autre cas.

Le second passage s'effectue si le premier parcours n'a détecté aucune erreur fatale et se poursuit tant qu'une telle erreur n'est pas rencontrée. Examinons les contrôles particuliers opérés suivant le type de l'instruction rencontrée :

- Pour chaque exécution de noeud, on vérifie d'abord que le fichier qui s'y rapporte existe bien sur le répertoire des données (point 8.1. de l'annexe A) , c'est-à-dire que le noeud existe. Sinon, un avertissement signale qu'un noeud inexistant doit être exécuté. Le professeur devra créer ce noeud avant exécution de la leçon par l'étudiant.
Exemple : EXECUTE TOTO.TXT (1,2,3) ;
On regarde si le fichier TOTO.FRM (le noeud TEXTE doit être formaté !!!) se trouve sur la directory des données. S'il ne s'y trouve pas, avertissement : le professeur doit créer ce noeud avant toute exécution de la

leçon.

Chaque composant d'une animation est testé de la même manière et s'il en manque un, l'animation ne peut s'exécuter.

De plus, tout noeud s'exécute dans le cadre d'une fenêtre. Les dimensions de la fenêtre de chaque noeud, fixées en début de création, sont conservées dans le fichier WINDOWS.DIM du répertoire des données. Pour chaque noeud, un enregistrement reprend son nom, son extension (qui détermine son type) et les dimensions horizontale et verticale de la fenêtre. Le compilateur vérifie qu'un enregistrement de WINDOWS.DIM porte le nom et le type du noeud concerné par l'exécution. Dans le cas contraire, un avertissement prévient l'utilisateur d'une absence des dimensions (voir la description de l'utilitaire COPYNODE au point 8.3. de l'annexe A pour découvrir la raison à cette absence d'enregistrements).

Grâce à WINDOWS.DIM, on peut en plus vérifier que tous les composants graphiques d'une animation ont la même dimension, condition indispensable à une bonne exécution. Le cas échéant, une erreur fatale stoppe la compilation.

Comme nous l'avons expliqué dans la sémantique de l'instruction EXECUTE, un ou plusieurs noeuds simulation peuvent être lié(s) à l'exécution d'un noeud questions-réponses. Il est exclu que les fenêtres simulation et questions-réponses portent le même numéro (erreur fatale).

Afin de vérifier la sémantique des opérations sur les fenêtres, une liste des numéros de fenêtres utilisées est mise à jour à chaque EXECUTE rencontré : on ajoute le(s) numéro(s) de fenêtre(s) de l'instruction s'il(s) ne se trouve(nt) pas encore dans cette liste.

- Si l'étiquette qui apparaît après un GOTO ne figure pas dans la table construite au premier passage, une erreur fatale est identifiée : étiquette inexistante. Sinon, on remplace l'étiquette par son adresse trouvée dans la table.

- Les étiquettes sont supprimées de la structure.

- Deux instructions exigent un numéro de fenêtre en paramètre : ERASE WINDOW x et GET WINDOW x UP. Pour effacer ou ramener une fenêtre à l'avant-plan sur l'écran, il faut qu'elle ait précédemment été utilisée, c'est-à-dire que son numéro doit se trouver dans la liste des numéros de fenêtres utilisées. La rencontre d'une instruction d'effacement d'une fenêtre élimine son numéro de cette liste. L'utilisateur est averti si la fenêtre concernée par une de ces 2 instructions n'est pas utilisée. Nous n'avons aucune raison de considérer ce fait comme erreur fatale puisque le compilateur constitue la liste des numéros de fenêtre en parcourant séquentiellement le fichier des instructions, alors que la structure de contrôle GOTO et l'instruction IF autorisent une dérogation au principe de séquentialité.

- Pour exécuter INVERSE THE TWO LAST WINDOWS, au moins deux numéros de fenêtres doivent se trouver dans le liste, sinon avertissement.

Quand le compilateur arrive au terme de ce second passage,on considère que la structure compilée (extension LES) constitue une leçon que PC_EXEC peut exécuter, même si certains avertissements sont donnés.

4.5. Exemple de structure à compiler

Soit la structure suivante à compiler :

```
EXECUTE TEST.QCM (1,1,1) ;
GET WINDOW 2 UP ;
IF MCQ POINTS < 60
GOTO FAIBLE ;
EXECUTE RAPPEL1.TXT (2,10,10) ;
GOTO SUITE ;
FAIBLE :
EXECUTE RAPPEL2.TXT(2,10,10) ;
EXECUTE DESSIN.GRF (3,50,100) ;
EXECUTE RAPPEL3.TXT (4,40,20) ;
ERASE WINDOW 7 ;
SUITE :
EXECUTE COMMUN.TXT (7,3,10) ;
```

Examinons l'évolution de la valeur du compteur d'instructions et le contenu de la table des étiquettes lors du premier passage :

```
COMPTEUR :                                              TABLE :
 0                                                      vide
          EXECUTE TEST.QCM (1,1,1) ;
 1
          GET WINDOW 2 UP ;
 2
          IF MCQ POINTS < 60
 3
          GOTO FAIBLE ;
 4
          EXECUTE RAPPEL1.TXT (2,10,10) ;
 5
          GOTO SUITE ;
 6
          FAIBLE :                                        {(FAIBLE,6)}
 6
          EXECUTE RAPPEL2.TXT(2,10,10) ;
 7
          EXECUTE DESSIN.GRF (3,50,100) ;
 8
          EXECUTE RAPPEL3.TXT (4,40,20) ;
 9
          ERASE WINDOW 7 ;
 10
          SUITE :
 10                                  {(FAIBLE,6),(SUITE,10)}
          EXECUTE COMMUN.TXT (7,3,10) ;
 11
```

Figure 3.17 - Première phase de compilation d'une structure

Le second passage constitue le fichier des instructions exécutables tant qu'aucune erreur fatale n'est rencontrée. On remplace les étiquettes d'un GOTO par l'adresse que le premier passage leur a associés :

```
EXECUTE TEST.QCM (1,1,1) ;
GET  WINDOW 2 UP ;                    (*)
IF MCQ POINTS < 60
GOTO FAIBLE  6;
EXECUTE RAPPEL1.TXT (2,10,10) ;
GOTO SUITE  10;
FAIBLE :
EXECUTE RAPPEL2.TXT(2,10,10) ;
EXECUTE DESSIN.GRF (3,50,100) ;
EXECUTE RAPPEL3.TXT (4,40,20) ;
ERASE  WINDOW 7 ;                     (**)
SUITE :
EXECUTE COMMUN.TXT (7,3,10) ;
```

Figure 3.18 - Seconde phase de compilation d'une structure

Le fichier exécutable contiendra autant d'enregistrements qu'il y a d'instructions. Chaque enregistrement est accessible à partir de son numéro d'ordre. Le premier enregistrement porte le numéro 0. Remarquez que l'adresse figurant comme paramètre d'une instruction GOTO correspond bien à l'instruction qui suivait l'étiquette dans la structure initiale.

Cette compilation s'est bien passée puisqu'elle n'a détecté aucune erreur fatale. Néanmoins, 2 avertissements s'afficheront à l'écran :

```
                *** Erreurs de compilation ***
WARNING: Numéro de fenêtre inconnu à ce niveau: 2
WARNING: Numéro de fenêtre inconnu à ce niveau: 7
```

En effet, le deuxième instruction de la structure (*) demande de ramener la fenêtre 2 à l'avant-plan sur l'écran alors qu'elle n'a pas encore été définie. Si le programme n'est pas modifié, cette instruction n'aura aucun effet. Le deuxième avertissement concerne l'instruction d'effacement de la fenêtre 7 (**), également non utilisée jusque là.

Section 2

Exécution d'une leçon (PC_EXEC)

1. Introduction

Dès que le professeur a créé tous les noeuds, édité puis compilé la structure de la leçon, le second module d'exécution d'une leçon peut prendre le relai. Le principal utilisateur de ce module est évidemment l'étudiant qui doit suivre la leçon assistée par ordinateur, mais aussi le professeur qui peut observer si l'exécution d'une leçon se déroule conformément à ses espérances. Dans le cas contraire, il peut toujours retourner dans le module création PC_CREAT pour modifier le(s) noeud(s) et/ou la structure de la leçon non satisfaisants. Nous examinerons d'abord à quoi correspond l'exécution de chaque type de noeuds, après quoi nous approfondirons le déroulement d'une leçon et le rapport d'exécution destiné au professeur.

2. L'exécution des différents types de noeuds

2.1. Un noeud graphique

L'exécution d'un noeud **graphique** consiste simplement à afficher le dessin. L'étudiant peut admirer ce dessin autant de temps qu'il le désire avant de presser une touche pour progresser dans le déroulement de la leçon.

2.2. Un noeud animation

Une **animation** consiste à afficher circulairement des graphiques de même dimension dans une même fenêtre. Ces composants portent le même nom à l'exception du dernier caractère qui, en fait, est un chiffre indiquant le numéro d'ordre du composant dans la séquence d'affichage. L'animation se déroule en 2 phases. La première consiste à lire chaque composant tout en affichant le dessin au fur et à mesure de la lecture. Quand un composant est affiché entièrement, la temporisation spécifiée dans l'instruction est respectée avant de lire et d'afficher le composant suivant (s'il existe). La lenteur de cette première phase s'explique par le temps de lecture des différents fichiers qui contiennent les composants. A l'issue de cette étape, les composants se retrouvent en mémoire centrale. La seconde

phase consiste à afficher successivement les composants selon l'ordre prescrit tout en respectant le délai d'affichage. Quand tous les composants ont été affichés à l'écran, on recommence le cycle d'affichage. L'étudiant décide la fin de l'animation en pressant une touche.

2.3. Un noeud texte

L'exécution d'un noeud **texte** présente le texte formaté page par page : une page correspond au contenu de la fenêtre. Si le professeur, en éditant le texte, a introduit la commande de formatage qui fixe le délai d'affichage d'une page, alors le rôle de l'étudiant consiste seulement à lire les pages qui défilent dans la fenêtre. Il n'a aucune possibilité de ralentir ou de stopper le défilement. Par contre, si cette commande de fixation du délai d'affichage est absente, l'étudiant choisit lui-même son rythme de lecture. Il dispose des touches **PGDN** pour passer à la page suivante, **PGUP** pour revenir à la page précédente et **F2** pour avorter l'exécution de ce noeud texte et passer au noeud suivant, même s'il n'a pas pris le temps de lire tout le texte.

2.4. Un noeud simulation

Une **simulation** commence toujours par la représentation graphique des courbes définies dans le modèle du professeur. Le temps correspondant à l'itération courante s'affiche dans le coin supérieur gauche de la fenêtre. L'étudiant peut arrêter cette représentation à tout moment en pressant une touche. Ensuite, plusieurs possibilités s'offrent à lui : il peut

- redémarrer la représentation des courbes à l'itération du moment d'interruption,
- si le professeur lui en a donné la possibilité, changer les conditions initiales des variables d'état et les valeurs des constantes (s'il y en a) du modèle-professeur et recommencer une nouvelle simulation. Les courbes des essais précédents ne sont pas effacées,
- effacer toutes les courbes de la fenêtre.

L'étudiant peut stopper la simulation et effectuer une de ces 3 opérations autant de fois qu'il le désire. Il termine la simulation-professeur quand il le souhaite.

Après la simulation-professeur, l'étudiant peut simuler son propre modèle, si le professeur lui a accordé cette option lors de la création du noeud. L'étape de définition du modèle est identique à ce qui se passe quand le professeur crée un modèle dans le module de création, à quelques exceptions près. Ainsi, on ne demandera pas à l'étudiant "L'étudiant peut-il modifier les conditions initiales de la simulation-professeur ?" ou encore "L'étudiant peut-il créer son propre modèle ?",... ces questions sont uniquement destinées au professeur. La définition s'effectue dans une fenêtre texte dont nous avons déterminé la taille. L'étudiant doit définir le nombre de variables d'état, de paramètres, de constantes et de variables de retard, le temps initial pour la première itération, l'accroissement de temps, les équations et valeurs initiales des variables d'état et paramètres, les équations des variables de retard et leur retard respectif, les valeurs des constantes, les abscisses et ordonnées des courbes qu'il désire observer, les expressions dont il souhaite enregistrer les valeurs dans un fichier qu'il pourra consulter après l'exécution de la leçon et l'intervalle de temps pendant lequel les courbes seront représentées sur l'écran. Il peut résoudre son modèle par la méthode d'Euler amélioré (deuxième ordre) ou de Runge-Kutta (quatrième ordre). Dès que l'étudiant introduit une erreur dans le modèle, un message d'erreur la lui signale (comme dans la définition d'un modèle-professeur dans le module PC_CREAT). Dès que la définition du modèle est complète et correcte, la fenêtre de définition disparaît et les courbes demandées par l'étudiant sont représentées dans la fenêtre de la simulation professeur. Les courbes antérieures ne sont pas effacées. L'étudiant peut arrêter l'exécution de son modèle à tout moment. Il peut alors passer à l'exécution du noeud suivant, modifier le modèle puis recommencer sa simulation ou encore effacer les courbes de la fenêtre. Quand il décide de modifier le modèle, la fenêtre texte utilisée à la création du modèle réapparaît à l'écran. Tous les renseignements collectés pour la création d'un modèle vont être redemandés à l'étudiant, mais les valeurs par défaut seront celles du modèle que l'étudiant modifie. Quand l'étudiant ne désire

pas changer l'élément du modèle sur lequel le curseur est positionné, il tape simplement RETURN pour passer à l'élément suivant. S'il désire le changer, il modifie la valeur par défaut. Dès qu'il a effectué les modifications qu'il désirait, il peut passer directement à la simulation du nouveau modèle dans la fenêtre de la simulation-professeur, sans passer en revue toutes les données (touche **END**).

2.5. Un noeud questions-réponses

L'**interrogation** de l'étudiant constitue une partie importante de la leçon puisqu'elle permet d'orienter la suite de la leçon suivant le comportement de l'étudiant dans ces questionnaires (voir les variables qu'on retrouve dans le premier membre de la condition d'un IF). Les questions d'un noeud questions-réponses sont posées à l'étudiant dans l'ordre de leur création. La fenêtre et la disposition des questions et réponses sont identiques a ce qui était présenté au professeur quand il créait son questionnaire. S'il n'a défini qu'une seule réponse, elle ne sera évidemment pas affichée à l'écran. Chaque fois qu'au moins une réponse est connue pour la question courante, une fenêtre de plus petite taille apparaîtra dans le coin inférieur droit de la fenêtre du noeud questions-réponses. C'est dans celle-ci que l'étudiant recevra les directives, introduira ses réponses et aura confirmation du résultat. Il se peut que l'énoncé et la (les) réponse(s) de la question soient cachés par cette fenêtre supplémentaire. L'étudiant peut la faire disparaître un moment afin de lire la question (touche **F3**). Avant de répondre à une question, l'étudiant peut exécuter un noeud simulation (cfr ci-dessus). Le professeur peut poser une question libre à l'étudiant, par exemple pour lui demander son avis sur un point bien précis. Une telle question n'est pas cotée. L'étudiant introduira sa réponse sous forme d'un ensemble de lignes de caractères (texte) dans l'espace de la fenêtre courante qui n'est pas occupé par l'énoncé de la question. Pour une question dont le programme ne connaît qu'une seule réponse, l'étudiant devra répondre sous forme d'une chaîne de caractères, comparée ensuite à la réponse du professeur, abstraction faite des différences de majuscules/minuscules

et du nombre de blancs entre les deux. Pour répondre au dernier type de questions (choix multiple), l'étudiant frappe la touche qui correspond au choix qu'il considère le meilleur.

3. Déroulement d'une leçon

Interprétation de la structure

PC_EXEC parcourt ce fichier séquentiellement, en exécutant chaque fois l'opération que caractérise l'instruction courante. Deux instructions dérogent à ce principe de séquentialité :

- Un branchement inconditionnel GOTO positionne le pointeur du fichier sur l'enregistrement qui porte le numéro d'ordre spécifié par son paramètre (le premier enregistrement porte le numéro 0).
- Quand la condition booléenne associée à un IF est évaluée à faux, le pointeur se positionne sur le deuxième record après ce IF ou la leçon se termine si moins de 2 instructions suivent cette instruction conditionnelle.

La dernière ligne de l'écran

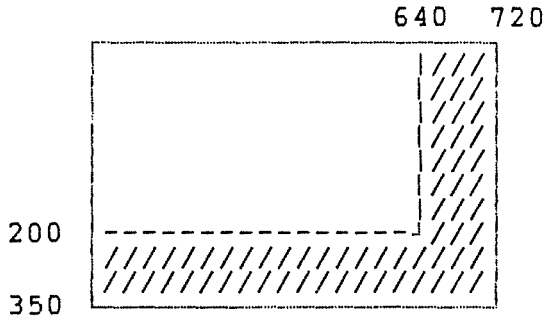
Pendant l'exécution d'une leçon, la dernière ligne de l'écran a une signification particulière : représentée en vidéo inversé, elle indique à l'étudiant toutes les possibilités qui lui sont offertes à l'instant courant, c'est-à-dire qu'on y trouve la liste des touches utiles et les fonctions spécifiques qui y sont affectées.

Problèmes d'exécution

Normalement, l'exécution se déroule exactement comme le professeur l'a prévu dans la structure de la leçon. Cependant, il peut y avoir quelques problèmes, notamment lorsque le professeur n'a pas tenu compte des avertissements signalés à la compilation. Ainsi, lorsqu'un noeud mentionné dans la leçon mais introuvable sur le répertoire des données doit être exécuté, PC_EXEC affichera un message d'erreur à l'écran. Quand un noeud, provenant d'un autre répertoire que celui des données, a été copié vers ce répertoire par la commande MS-DOS "copy" sans utiliser l'utilitaire de copie, les dimensions de sa fenêtre ne se trouvent pas dans le fichier WINDOWS.DIM. Par conséquent, le programme ne peut exécuter ce noeud et un message d'erreur signale que les

dimensions sont absentes du fichier spécialisé. L'étudiant n'est évidemment pas responsable de ce type d'erreur et la leçon poursuivra son cours normal après l'affichage du message d'erreur.

Il se peut très bien que les dimensions d'une fenêtre trouvées dans WINDOWS.DIM et la position du coin supérieur gauche de cette même fenêtre (spécifiée dans la structure de la leçon) empêchent de représenter la fenêtre dans son entièreté à l'écran. Dans ce cas, la position du coin supérieur gauche sera corrigée pour que la fenêtre soit vue entièrement à l'écran : elle sera décalée vers le haut si le bas de la fenêtre n'est pas visible et/ou vers la gauche si sa partie droite n'apparaît pas à l'écran. Il se peut que le logiciel ait déplacé la fenêtre dans le coin supérieur gauche de l'écran et qu'elle ne soit encore que partiellement visible. Ce problème se présente, par exemple, lorsqu'on essaie d'afficher, sur un ordinateur disposant d'une carte IBM, un graphique de taille maximum créé sur un autre ordinateur qui travaille en mode Hercules. En effet, la taille maximum d'un graphique en mode Hercules est de 720X350 alors qu'une carte IBM offre 640X200. Par conséquent, tout graphique de taille supérieure à 640X200 créé sur un ordinateur disposant d'une carte Hercules ne sera représenté que partiellement avec une carte IBM; la partie qui excède la dimension maximale permise par la carte IBM étant tronquée (partie hachurée) :



4. Le rapport d'exécution

Avant que ne débute réellement la leçon, le module d'exécution demande à l'étudiant d'introduire 2 renseignements : le nom de la leçon et un code qui identifie l'étudiant pour cette exécution particulière, par exemple son nom. On utilise ce second renseignement pour créer un fichier de même nom, avec l'extension REL, qui représentera la "boîte noire" de l'exécution de la leçon, à savoir un rapport complet où on trouve le cheminement suivi par l'étudiant à travers la leçon et les scores obtenus. Ce fichier intéresse tout particulièrement le professeur qui doit veiller à ce que l'étudiant ne falsifie pas les résultats à son avantage. On consulte ce fichier par la simple commande "TYPE" du MS-DOS !!

Si un étudiant exécute 2 leçons de suite, il devra donner un nom ou un code différent pour chaque leçon, sinon le professeur ne pourra consulter que le deuxième rapport, puisque le premier fichier sera effacé par le deuxième qui porte le même nom.

On trouve dans ce fichier une trace de tous les noeuds exécutés. Pour les noeuds graphique, animation et texte, on y trouve simplement une phrase qui rapporte leur exécution.

Exemples : EXECUTION DU NOEUD GRAPHIQUE : CIRCUIT
EXECUTION DU NOEUD ANIMATION : MOVE
EXECUTION DU NOEUD TEXTE : CHAP_1
_____/
Nom du noeud

Les noeuds simulation et questions-réponses sont plus détaillés:

- un noeud **simulation** :

La première ligne spécifie le nom du modèle simulé. Si le professeur a autorisé le changement des valeurs initiales des variables d'état et des valeurs des constantes de son modèle, on y trouve ces valeurs, modifiées ou non, chaque fois que l'étudiant utilise l'option. Lorsque l'étudiant crée ou modifie un modèle, on y retrouve, après le titre "SIMULATION ETUDIANT", les équations des variables d'état, des paramètres, des variables de retard et les valeurs des constantes. Si l'étudiant n'a pas jugé bon de créer et de

simuler son propre modèle, la mention "ABANDON" figure dans le fichier.

Exemple : Dans l'exécution du modèle CERCLE, l'étudiant a changé 2 fois les valeurs initiales des variables d'état X(1) et X(2) de la simulation professeur, avant de passer à la création de son propre modèle dont on retrouve les équations des variables d'état (pas de paramètres, de variables de retard ni de constantes) :

```
EXECUTION DU NOEUD SIMULATION : CERCLE
  Changement des conditions initiales du professeur:
  X(1)=    4.8000000000E+00
  X(2)=   -4.8000000000E+00
  Changement des conditions initiales du professeur:
  X(1)=    4.9500000000E+00
  X(2)=    3.2510000000E-01
  SIMULATION ETUDIANT
  d/dt[X(1)]= X(2)
  d/dt[X(2)]= -X(1)
```

Figure 3.19 - Contenu du rapport d'exécution pour une modification des conditions initiales

- un noeud **questions-réponses** :

La partie concernant l'exécution d'un noeud questions-réponses commence toujours par une ligne d'en-tête qui indique le nom du noeud. ensuite, on trouve les renseignements pour chaque question du noeud : le numéro d'ordre de la question dans le questionnaire et son énoncé tel qu'il apparaît à l'étudiant. La suite dépend du type de la question :

- Pour les questions dont on connaît au moins une réponse, on donne, pour chaque tentative de réponse, le numéro d'ordre de l'essai, la réponse choisie par l'étudiant et une conclusion : la réponse de l'essai est bonne ou mauvaise. La dernière ligne d'une question cotée mentionne le nombre de points obtenus pour la question. Attention, si par exemple le professeur a prévu 3 tentatives de réponses et que l'étudiant répond bien au troisième essai, le nombre de points n'est comptabilisé que lors de ce troisième essai. Aucun point ne sera décompté pour les mauvaises tentatives de réponses. Le nombre d'essais est incrémenté à chaque tentative.

- Pour une question-menu, le choix de l'étudiant est mentionné.

- Pour une réponse libre, le professeur découvre la réponse de l'étudiant sous forme d'un texte. Pour rappel, on ne peut coter une telle question.

Un noeud se termine par la note obtenue pour l'ensemble des questions.

Les 2 dernières lignes du fichier mentionnent la note globale de la leçon et le temps d'exécution.

Section 3 : Les messages

Un des objectifs prioritaires que nous nous étions fixé dès le départ était de permettre à l'utilisateur de travailler dans sa langue maternelle. Tous les messages du logiciel concernant options, titres, commentaires, messages d'erreur, ... qui apparaissent dans l'ensemble des programmes peuvent être modifiés ou traduits suivant le goût de celui qui l'utilise. Une seule restriction : chaque message a une longueur maximale imposée, qui dépend du contexte dans lequel il doit s'afficher. Nous avons réalisé cet objectif en regroupant tous les messages dans des fichiers externes aux programmes de création ou d'exécution de leçon. Tout message fait partie d'un enregistrement composé de 2 parties : le message proprement dit et sa longueur maximale, représentée par un entier, qui indique le nombre maximal de caractères qu'il peut contenir. Ces enregistrements forment les fichiers de messages cités ci-dessous. Un programme qui doit afficher un message connaît le nom du fichier dans lequel il se trouve et le numéro de l'enregistrement qui contient ce message. Chaque affichage de message requiert donc l'accès à un fichier auxiliaire. Nous avons choisi l'ordre des enregistrements dans le fichier de telle manière que si un programme doit afficher successivement 2 messages, il est fort probable qu'ils occuperont 2 enregistrements consécutifs. Les noms de fichiers de messages sont imposés mais l'extension, identique pour tous les fichiers d'un même programme, peut être installée (voir l'utilitaire INST_EAO au point 8.1. de l'annexe A). La modification ou la traduction de messages se fait via l'utilitaire TRANS_ME (point 8.2 de l'annexe A).

Les fichiers de messages

Quatre fichiers de messages sont nécessaires pour créer une leçon, à savoir :

- le fichier MESSAGES qui regroupe tous les messages de la création, à l'exception de l'aide générale de l'éditeur graphique et des messages qui apparaissent dans l'éditeur de texte.
- Le fichier GRAPHIX qui contient le texte d'aide général de l'éditeur graphique.
- Le fichier EDITERR qui rassemble tous les messages de l'éditeur de texte, le texte d'aide des commandes de cet éditeur et l'aide concernant les commandes de formatage que l'utilisateur peut insérer lors de l'édition d'un noeud texte.
- Le fichier READ qui contient la manuel d'utilisation du logiciel.

Vous comprendrez cette répartition en plusieurs fichiers lors de la présentation de l'utilitaire d'installation INST_EAO (point 8.1 de l'annexe A).

Tous les messages de l'exécution d'une leçon forment le fichier EXEC. L'utilisateur accède au contenu de ces fichiers par les utilitaires proposés au point 8 de l'annexe A.

CHAPITRE 4

SimEDO

Section 1 : Introduction

L'abréviation SimEDO signifie **Sim**ulation d'**E**quations **D**ifférentielles **O**rdinaires. Le logiciel offre la possibilité de créer puis de simuler des modèles représentant un système d'équations différentielles ordinaires du premier ordre totalement compatibles avec les modèles de ProfCOMP. C'est pourquoi tout ce chapitre s'adresse également à ProfCOMP.

Définition : Une équation différentielle ordinaire du premier ordre est une relation liant une fonction y d'une variable x, sa dérivée y' et la variable indépendante

$$F(x,y,y') = 0$$

Admettons qu'on puisse expliciter l'équation par rapport à y' : $\boxed{y' = f(x,y)}$

SimEDO résoud des modèles d'équations présentées sous cette forme.

Le mécanisme de création d'un modèle et les possibilités offertes à sa simulation ont fait l'objet des sections 1 (point 3.4.) et 2 (point 2.4.) du chapitre précédent. Il faut également savoir que SimEDO offre des possibilités d'impression des courbes ou d'un rapport du modèle simulé (pour plus de détails, se référer au manuel d'utilisation de SimEDO en annexe B).

Dans ce chapitre, nous n'avons nullement l'intention de développer la théorie des systèmes d'équations différentielles. Après quelques mots sur l'historique de SimEDO, nous évoquerons la version actuelle du logiciel. Nous commencerons par examiner la syntaxe des expressions et les composants qu'on y trouve. Nous verrons comment le logiciel assure la correction syntaxique d'un modèle et quelles sont les représentations internes adoptées. Nous nous intéresserons également à l'évaluation des expressions et aux méthodes de résolution avant de détailler un exemple de modèle simulé. Nous conclurons par les limites de SimEDO.

Section 2 : L'historique d'un noeud simulation

A l'issue du laboratoire de deuxième licence, le logiciel SimEDO n'existe pas. Quant à ProfCOMP, il offre déjà la possibilité de simuler des systèmes d'équations différentielles mais exige une bonne dose de courage et de persévérance avant d'obtenir un modèle "correct". La mise au point d'une simulation s'effectue en 2 phases : l'utilisateur doit répondre à une série de questions dans le module de création d'une leçon avant de compléter un squelette de programme Pascal qui simulera le modèle à l'exécution de la leçon. Une fois rempli, ce programme est compilé en version exécutable puis copié sur l'unité des données.

Première phase

Le module création interroge le concepteur du modèle. La première question concerne le nom du noeud simulation. Ce nom, auquel on ajoute l'extension ".DAT", est utilisé pour créer un fichier de type texte qui enregistre toutes les réponses aux questions suivantes. Chaque réponse occupe une ligne de ce fichier.

Comme nous l'avons déjà dit, tout noeud s'exécute dans une fenêtre dont l'utilisateur détermine les dimensions. Dans cette version, la détermination de la taille de fenêtre par un ajustement visuel des dimensions à l'écran à l'aide des flèches n'existe pas. L'enseignant doit connaître les dimensions de l'écran sur lequel il travaille, en l'occurrence 80 divisions horizontales et 200 verticales (ce logiciel n'existe que pour une carte IBM). L'échelle horizontale correspond en fait au nombre de caractères que peut contenir une ligne. Nous avons pris ces divisions parce qu'elles nous sont imposées par les procédures de définition de fenêtres du GRAPHIX TOOLBOX que nous utilisons. Evidemment, pour qu'une fenêtre serve à quelque chose, nous lui avons imposé une taille minimale d'un quart d'écran, c'est-à-dire de dimension 20X50. L'enseignant introduit 2 nombres qui correspondent aux dimensions horizontale et verticale de la fenêtre.

Toutes les autres questions concernent le modèle proprement dit : le nombre d'équations, de paramètres et les temps de début et fin de simulation.

Pour démarrer la résolution d'un modèle, le logiciel a besoin de connaître la valeur initiale de chaque variable d'état. Pour chaque valeur initiale nécessaire, le professeur peut soit en déterminer la valeur immédiatement, soit laisser la possibilité à l'étudiant d'introduire cette valeur à l'exécution. Dans ce dernier cas, l'enseignant introduit 5 lignes de texte qui informeront l'étudiant de la demande d'introduction de la valeur initiale concernée.

Toutes ces données seront utilisées par le programme, constitué à la seconde phase, qui simulera le modèle à l'exécution de la leçon.

Dès que l'enseignant a terminé de répondre à ce questionnaire, il peut passer à la seconde étape de création du modèle.

Seconde phase

La seconde étape d'élaboration d'un modèle se déroule dans l'environnement du Turbo Pascal version 3 (Borland) et consiste à remplir un squelette de programme Pascal afin que celui-ci simule le modèle désiré et en représente les courbes à l'exécution. Une bonne compréhension du travail demandé requiert la connaissance du langage de programmation Turbo Pascal mais également des spécifications des procédures du Graphix Toolbox qu'on utilise pour gérer l'application graphique. Ces contraintes restreignent fortement l'ensemble des utilisateurs potentiels du logiciel à ceux qui ont des aptitudes de programmation.

Détaillons maintenant ce squelette de programme Pascal afin de mieux comprendre ce que le concepteur d'un modèle doit faire. Signalons qu'un exemple de programme complété se trouve en annexe E.

Pour bénéficier du Graphix Toolbox, nous devons d'abord incorporer, en début du programme, une série de fichiers au moyen de la directive "include" du Turbo Pascal (cette directive informe le compilateur de lire le programme qui se trouve dans le fichier spécifié) :

- Les fichiers TYPEDEF.SYS, GRAPHIX.SYS et KERNEL.SYS constituent le noyau de l'utilitaire
- Comme le logiciel se base sur le concept de fenêtre, le fichier WINDOWS.SYS nous propose ses procédures de gestion de fenêtres
- Notre programme de simulation doit présenter des résultats graphiques à l'écran mais les valeurs des couples de points à afficher ne sont connues qu'à l'exécution. Par conséquent, le fichier FINDWRLD.HGH se chargera d'adapter l'échelle des graphiques à l'espace d'affichage (une fenêtre).
- Quand on représente un ensemble de couples de valeurs numériques, il est utile d'afficher des axes gradués qui indiquent la valeur des résultats; DRAWAXIS.HGH s'en charge.
- Enfin, n'oublions pas le principal : le fichier POLYGON.HGH tracera les courbes qui résultent de l'exécution du modèle.

Après ces directives d'inclusion, 2 constantes intéressent directement l'utilisateur puisqu'elles concernent le nombre de points calculés par courbe et le nombre maximum d'équations différentielles. Quand le système comprend plus d'équations que la valeur maximale ne l'autorise, l'utilisateur peut augmenter la valeur de cette constante sous réserve qu'il y ait encore assez de place mémoire au moment de l'exécution. Nous verrons pourquoi ci-dessous. Remarquons qu'aucune question de la première phase ne concerne l'incrément de temps, qui est en fait calculé de la manière suivante : on soustrait du temps de fin de simulation le temps de début de simulation et on divise le tout par le nombre de points à calculer.

A chaque variable d'état, on associe un tableau "Di" qui contient toutes les valeurs successives calculées pour la variable d'état correspondante. C'est évidemment un gros inconvénient : le système est résolu avant que ne commence l'affichage et pour chaque variable d'état, TOUTES ses valeurs sont conservées dans un tableau. Plus le nombre de points calculés est élevé et plus d'équations le modèle contient, plus il faut de tableaux pour stocker les valeurs des variables d'état et donc plus de chances a-t-on de

manquer d'espace mémoire, d'autant plus qu'il y a encore d'autres tableaux de cette dimension, notamment le tableau "temps" des temps intermédiaires. Les valeurs initiales des variables d'état sont elles aussi conservées dans un tableau séparé. Les paramètres "Pi" ($1 \leq i \leq 10$) sont de simples variables de type réel (en fait, ce sont des constantes dans la terminologie actuelle de SimEDO). Les valeurs initiales des variables d'état sont soit définies par le professeur lors de la première phase de création, soit introduites par l'étudiant à l'exécution. Dans les 2 cas, ces valeurs sont enregistrées dans un tableau "QI" dont l'indice correspond à celui de la variable initialisée.

Le premier remplissage du programme Pascal consiste à transférer, pour chaque variable d'état, sa valeur initiale dans le premier élément du tableau "Di" correspondant. Ensuite commence la résolution du modèle : il faut remplir ces tableaux "Di" par la procédure EULER (seule méthode de résolution) en même temps que le tableau des temps intermédiaires.

Pour afficher une courbe à l'aide des procédures du Graphix Toolbox, les abscisses et ordonnées de tous les points à représenter doivent se trouver dans un seul et même tableau à 2 dimensions. Avant d'envoyer le(s) tableau(x) de points à la procédure qui affichera la (les) courbe(s) correspondantes, il faut évidemment remplir ces tableaux à 2 dimensions suivant les expressions que le professeur désire représenter en abscisse et en ordonnée à partir des tableaux de valeurs des variables d'état, des temps intermédiaires et les valeurs des paramètres.

L'exécution d'un noeud simulation

A l'exécution d'une leçon, quand l'interpréteur du fichier de la structure de la leçon rencontre une instruction d'exécution d'un noeud simulation, il regarde d'abord sur l'unité des données s'il trouve un fichier nommé par le nom du modèle à simuler et d'extension "COM". Dans le cas positif, EXECLESS (le premier nom du module d'exécution) mémorise sa situation avant de lancer l'exécution du programme indépendant qui simule le modèle : un fichier contiendra la pile des fenêtres représentées à

l'écran au moment où on doit simuler le modèle. Un autre fichier conservera divers renseignements nécessaires aussi bien pour exécuter le noeud que pour reprendre la leçon par après :

- la liste des numéros de fenêtres utilisées à l'écran,
- le numéro de la fenêtre où doit s'effectuer la représentation des courbes du modèle et ses dimensions.
- le numéro d'ordre de l'instruction d'exécution du noeud simulation dans la structure de la leçon ou du noeud questions-réponses à partir duquel la simulation est appelée,
- les valeurs des variables qui totalisent les nombres d'essais et de points (obtenus ou maximum) aussi bien pour le dernier noeud questions-réponses que pour l'ensemble de la leçon.

Quand le programme séparé de simulation commence son exécution, il doit d'abord rétablir les fenêtres à l'écran à l'aide de la pile sauvegardée des fenêtres et de l'ensemble des numéros de fenêtres utilisées. Avant de représenter les courbes du modèle, la fenêtre du noeud est aisément définie puisque son numéro et ses dimensions sont sauvegardés dans un fichier constitué par EXECLESS. L'exécution d'un modèle consiste d'abord à demander les valeurs initiales des variables d'état, si nécessaire et selon l'exigence du professeur (cfr première phase), puis à représenter la (les) courbe(s) définie(s) par le professeur (pas de sorties numériques). Une fois que la simulation proprement dite arrive à son terme, le programme met à jour la liste des numéros de fenêtres ainsi que le fichier de la pile des fenêtres en ajoutant la fenêtre simulation puis lance l'exécution du programme EXECLESS qui reprend la leçon à l'instruction qui suit celle d'exécution du noeud simulation après avoir réaffiché toutes les fenêtres de la pile. L'étudiant peut uniquement introduire les valeurs initiales des variables d'état du modèle professeur. Pas question de simuler un modèle de l'étudiant en cours de leçon puisque chaque modèle est un programme compilé.

Evolution

Etant donné que les personnes susceptibles d'utiliser ProfCOMP ne disposent pas nécessairement des connaissances du langage Turbo Pascal et du Graphix Toolbox, cette solution se devait d'être repensée complètement pour toucher un éventail le plus large possible d'utilisateurs potentiels.

Lors de notre stage à l'Instituto Superior Tecnico de Lisboa, une conversation avec Monsieur Antunes nous a éclairé sur la façon de construire un simulateur. Les grandes lignes du projet étaient tracées, il ne restait plus qu'à les concrétiser dans SimEDO et ProfCOMP. La première étape nous a conduit à définir le "language" utilisé pour représenter un modèle à simuler (voir création d'un noeud simulation) après quoi nous avons construit les modules de création et d'exécution d'un système d'équations différentielles ordinaires du premier ordre. Le reste du chapitre évoque la solution actuelle de SimEDO.

Section 3 : Les expressions

Le point 3.4. de la section 1 du chapitre précédent nous a appris toutes les étapes de création d'un modèle. En particulier, nous avons vu que pour chaque équation différentielle de variable d'état, le concepteur devait en introduire la partie à droite du signe égal. Il répète la même opération si le modèle contient des paramètres, variables de retard, sorties numériques et/ou graphiques. Bref, nous dirons que l'utilisateur doit introduire des **EXPRESSIONS** qui doivent vérifier une certaine syntaxe. Nous définirons cette syntaxe avant d'aborder la description de toutes les opérandes et opérateurs qu'une expression peut combiner. Nous pourrons alors voir que la syntaxe décrit 2 types d'expressions dont nous détaillerons l'utilisation.

1. Syntaxe des expressions

bl ::= {' '}

exposant ::= **

expression ::= expression1 | expression2

expression1 ::= {bl} opérande {bl} |
    {bl} expression1 {bl} opérateur2 {bl} expression1 {bl} |
    {bl} opérateur1 {bl} expression1 {bl} |
    {bl} ({bl} expression1 {bl}) {bl}

expression2 ::=
    {bl} { expression1 op-relat expression1 } {bl}
    { expression1 } {bl} { expression1 }

expression-retard ::= expression-retard1 |
    expression-retard 2

expression-retard1 ::= {bl} opérande1 {bl} |
    {bl} expression-retard1 {bl} opérateur2 {bl}
    expression-retard1 {bl} |
    {bl} opérateur1 {bl} expression-retard1 {bl} |
    {bl} ({bl} expression-retard1 {bl}) {bl}

expression-retard2 ::=
    {bl} { expression-retard1 op-relat expression-retard1 }
    {bl} { expression-retard1 } {bl} { expression-retard1 }

indice-c ::= 1 .. **nombre de constantes**

indice-d ::= 1 .. **nombre de variables de retard**

indice-p ::= 1 .. **nombre de paramètres**

indice-x ::= 1 .. **nombre d'équations**

opérande ::= opérande1 | opérande2

opérande1 ::= S-X {bl} ({bl} indice-x {bl}) |

S-X {bl} ({bl} S-P {bl} ({bl} indice-p {bl}){bl}) |
S-X {bl} ({bl} S-X {bl} ({bl} indice-x{bl}) {bl}) |
S-P {bl} ({bl} indice-p {bl}) |
S-P {bl} ({bl} S-P {bl} ({bl} indice-p {bl}){bl}) |
S-P {bl} ({bl} S-X {bl} ({bl} indice-x {bl}){bl}) |
S-C {bl} ({bl} indice-c {bl}) |
{bl} **entier** {bl} |{bl} **réel** {bl} |{bl} temps {bl}
{bl} **PI** {bl}

opérateur2 ::= S-D {bl} ({bl} indice-d {bl})

opérateur1 ::= **-** | **abs** | **cos** | **sin** | **arctan** | **exp** | **ln** |
frac | **int** | **sqr** | **sqrt**

opérateur2 ::= **+** | **-** | ***** | **/** | **mod** | **div** | exposant

op-relat ::= **<** | **>** | **=** | **>=** | **<=** | **<>**

S-C ::= **C** | **c**

S-D ::= **D** | **d**

S-P ::= **P** | **p**

S-X ::= **X** | **x**

temps ::= **T** | **t**

Figure 4.1 - Syntaxe des expressions

Nous commencerons l'analyse de la syntaxe décrite ci-dessus en détaillant les composants de telles expressions. Nous poursuivrons par les types d'expressions que l'on peut rencontrer et leur utilisation dans la définition d'un modèle.

2. Les opérandes

2.1. Les variables d'état

Tout modèle d'équations différentielles ordinaires du premier ordre **simulable par SimEDO** contient autant de variables d'état que d'équations différentielles.
Qu'est-ce qu'une variable d'état ? C'est le renseignement minimum à connaître pour définir la solution de manière univoque si on connaît la fonction d'entrée.

Une variable d'état s'identifie par la lettre X (majuscule ou minuscule) suivie de son indice entre parenthèses. L'ensemble des indices forme un ensemble d'entiers strictement croissants compris entre 1 et le nombre de variables d'état du modèle. Le cardinal de cet ensemble vaut le nombre de variables d'état du modèle. Le système autorise un modèle d'au plus 20 équations différentielles.
Règle :

n variables d'état <=> $X(i)_{\#\{i\}=n}^{1\le i\le n}$ $\quad \forall i1,i2\varepsilon\{i\}:i1\neq i2$ $\quad 1\le n\le 20$

Dans la syntaxe, l'opérande variable d'état présente 3 alternatives, à savoir X(indice-x), X(X(indice-x)) et X(P(indice-p)). Au niveau de l'indice, on peut trouver un entier (indice-x) mais aussi la <u>valeur</u> d'une variable d'état X(indice-x) ou d'un paramètre P(indice-p) (nous examinerons l'opérande paramètre ci-dessous). Evidemment, dans ces 2 derniers cas, c'est la VALEUR courante de la variable d'état ou du paramètre qui joue le rôle d'indice et qui doit respecter les propriétés exposées ci-dessus.

2.2. Les paramètres

Un paramètre dépend d'autres opérandes du modèle et est décrit par une équation non différentielle. On l'appelle parfois variable dépendante dans la littérature.

La lettre P (minuscule ou majuscule) qui précède un indice entre parenthèses désigne un paramètre du système. En ce qui concerne les indices, on peut faire les mêmes remarques que pour les variables d'état. C'est pourquoi nous donnerons simplement la règle adaptée aux paramètres sans faire davantage de commentaires :

m paramètres <=> $P(j)_{\#\{j\}=m}^{1\le j\le m}$ $\quad \forall j1,j2\varepsilon\{j\}:j1\neq j2$ $\quad 0\le m\le 20$

Remarquons cependant qu'un système ne comporte pas obligatoirement de paramètre : $0\le m\le 20$.
La syntaxe propose encore 3 représentations pour l'indice : un entier indice-p, la VALEUR courante d'une variable d'état X(indice-x) ou d'un paramètre P(indice-p). Quelle que soit cette représentation, l'indice devra toujours être compris

dans l'ensemble strictement croissant d'entiers compris entre 1 et le nombre de paramètres (quand il y a des paramètres définis dans le système).
Remarque : étant donné que la valeur d'une variable d'état ou d'un paramètre est un nombre réel, SimEDO en prend la partie entière lorsqu'ils interviennent comme indice.

2.3. Les variables de retard

Utilité des variables de retard ?

Dans une équation différentielle où le temps est la variable indépendante, chaque terme de l'équation est évalué au même instant dans le temps. Beaucoup de systèmes physiques peuvent être décrits par des équations de ce type. Cependant, il existe dans certains systèmes des délais dans le temps.
Exemple : la transmission d'un signal sur une ligne accoustique. La ligne de transmission requiert un intervalle de temps fixe pour qu'un signal se propage d'un bout à l'autre de la ligne. Par conséquent, tout système qui incorpore une ligne de transmission doit tenir compte de ce délai en définissant des variables de retard.

On identifie une variable de retard (en anglais delay) par la lettre D et son indice entre parenthèses. Un modèle peut contenir jusqu'à 4 variables de retard. L'indice respecte une règle similaire à celle des variables d'état ou des paramètres :

$$r \text{ var de retard } \Leftrightarrow \underset{\#\{k\}=r}{D(k)} \quad 1\leq k\leq r \quad \underset{0\leq r\leq 4}{\forall k1,k2\in\{k\}:k1\neq k2}$$

Seul changement : l'indice est uniquement un nombre entier compris entre 1 et le nombre de variables de retard du modèle.

2.4. Les constantes

Pourquoi avoir introduit ces composants indicés alors que la syntaxe autorise un nombre entier ou réel comme opérande ?
A l'exécution d'un modèle, l'utilisateur peut à tout moment stopper la simulation, changer les valeurs des constantes (de même d'ailleurs que les conditions initiales des variables d'état), ce qui modifie toutes les expressions où

elles interviennent. Une nouvelle simulation peut alors redémarrer pour observer le changement de comportement.

Les derniers composants indicés sont les constantes représentées par la lettre C.
Règle :

$$q \text{ constantes} <=> C(l)_{\substack{1 \le l \le q \\ \#\{l\}=q}} \quad \forall l1, l2 \in \{l\} : l1 <> l2 \quad 0 \le q \le 20$$

L'indice est uniquement représenté par une valeur entière.

2.5. Le temps

La valeur réelle du temps de l'itération courant est symbolisée par la lettre T ou t.

2.6. Un nombre entier ou réel

Tout nombre entier est compris entre les valeurs -32768 et 32767 tandis que l'intervalle permis pour un réel s'étend de -10^{38} à 10^{38}. On accepte un réel sous 2 formats : mantisse E exposant (évaluation : mantisse * $10^{exposant}$) ou le classique point décimal.

2.7. La constante π

π est représenté par le mot PI ou pi.

3. Les opérateurs

Pour combiner les opérandes et former des expressions complexes, nous avons besoin d'opérateurs, répartis en 3 catégories. Afin d'évaluer correctement une expression, nous avons attaché une priorité à chaque opérateur (nous en reparlerons à la section 4 du présent chapitre.

3.1. Les opérateurs à un seul opérande

L'argument se trouve toujours après l'opérateur. L'usage des parenthèses et des blancs pour séparer opérateur et argument est facultatif mais recommandé pour une meilleure lisibilité. Tous ces opérateurs sont du même niveau de priorité : 2.

Trois opérateurs donnent un résultat de même type que l'argument (entier ou réel) : -, **ABS** et **SQR** renvoient respectivement la valeur opposée, absolue et du carré de l'argument. Tous les autres renvoient un réel comme résultat, que l'opérande soit du type entier ou réel. Les opérateurs trigonométriques **COS** et **SIN** donnent respectivement le cosinus et le sinus de l'argument exprimé en radians tandis qu'**ARCTAN** donne l'angle, exprimé en radians, dont la tangente vaut l'argument. **INT** ne garde que la partie entière de l'argument et **FRAC** n'en conserve que la partie fractionnelle. **LN** propose le logarithme naturel de l'argument strictement positif et **SQRT** la racine carrée. Enfin, **EXP** renvoie l'exponentielle de l'argument.

3.2. Les opérateurs à 2 opérandes

Le logiciel utilise la notation infixée dans ses expressions, ce qui veut dire qu'on trouve le premier opérande avant l'opérateur et le second après. Nous décrirons ces opérateurs par ordre croissant de priorité.
****** (exposant) renvoie une valeur de type réel qui représente le premier argument (entier ou réel) élevé à la puissance du second argument (entier ou réel). Priorité : 3. Quatre opérateurs partagent la priorité 4, à savoir : * et / qui multiplient ou divisent le premier argument par le second. Les opérandes sont de type entier ou réel, le résultat est entier sauf si au moins un opérande est de type réel. **DIV** effectue la division de 2 arguments entiers pour donner un résultat de même type. **MOD** donne le reste de la division du premier argument (entier) par le second (entier). Les opérateurs + et - partagent la priorité 5.

3.3. Les opérateurs relationnels

On utilise ces opérateurs dans un cas bien particulier. Sachez pour l'instant qu'il s'agit d'opérateurs à 2 opérandes de priorité 6. Leur liste : <, <=, >, >=, =, <>.

4. Les types d'expressions

Jusqu'à présent, nous avons énoncé une liste d'opérandes et d'opérateurs. Examinons maintenant les types d'expressions qu'on peut former avec ces éléments et surtout, quand on utilise ces expressions.

La syntaxe définit 2 types d'expressions qui contiennent chacun 2 alternatives. En fait, toutes les expressions qui vérifient syntaxiquement une des alternatives de "expression" vérifie aussi "expression-retard". La seule différence entre les 2, c'est que "expression-retard" admet un opérande de plus, à savoir les variables de retard : "expression" admet les opérandes "opérande1" tandis que "expression-retard" admet "opérande" qui regroupe "opérande1" et "opérande2". Cette distinction empêche donc l'utilisation de variables de retard dans certaines expressions.

Intéressons-nous davantage aux alternatives de "expression-retard".

- "Expression-retard1" désigne ce que nous appellerons par la suite une expression normale où interviennent des opérandes et des opérateurs (non relationnels) dans une combinaison syntaxiquement correcte.

Exemples : X(2)
(1-X(1)**2)*X(2)-X(1)+.5*COS(6.28*T/5.9)
1-D(2)+X(1) mod P(X(1))+LN P(5)

- "Expression-retard2" décrit des expressions plus complexes appelées expressions conditionnelles, composées de 3 sous-expressions, chacune entre accolades. La première sous-expression se décompose en 3 parties : 2 "expression-retard1" séparées par un opérateur relationnel qui permet de définir une condition qui sera évaluée à chaque itération de l'exécution du modèle. Si la condition est évaluée à vrai, la seconde expression de type "expression-retard1" sera évaluée et représentera le résultat de l'expression complète. Dans le cas contraire, c'est la troisième sous-expression qu'on évaluera.

Exemple : On peut interpréter l'expression
{T > 50} {X(1)} {X(2)} comme suit :
SI le temps est supérieur à 50,
ALORS l'expression prend la valeur de X(1)
SINON l'expression prend la valeur de X(2)

On peut donner les mêmes explications pour les alternatives de "expression", avec la restriction qu'on ne peut y trouver de variables de retard comme opérande.

Dans quel cas l'expression introduite doit-elle être conforme à "expression" ou "expression-retard" ?

Dans l'étape de définition d'un modèle, l'utilisateur doit spécifier l'équation différentielle de chaque variable d'état. Le premier membre de cette équation s'affichera à l'écran, il restera à en définir le second par l'expression adéquate.

Exemple : d/dt [X(1)] = ...

Après les variables d'état, c'est le second membre de l'équation de chaque paramètre qu'il faut définir par une expression, du moins si le système en utilise.

Exemple : P(1) = ...

Une sortie graphique demande les expressions dont les valeurs seront représentées sur chaque axe et une sortie numérique interroge sur les expressions intéressantes pour lesquelles l'utilisateur veut conserver les valeurs dans un fichier. **TOUTES CES EXPRESSIONS S'ACCORDENT AVEC LA SYNTAXE DE "expression-retard".**

Reste le seul cas où on ne peut utiliser de variable de retard : D(i) = ... $(1 \leq i \leq 4)$

L'EXPRESSION QUI DEFINIT LE MEMBRE DROIT DE L'EQUATION D'UNE VARIABLE DE RETARD NE PEUT CONTENIR DE VARIABLE DE RETARD et vérifie, par conséquent, la syntaxe de "expression".

Section 4 : Méthode d'analyse des expressions et leur représentation interne

Le membre de droite de chaque équation de variable d'état, de paramètre, de variable de retard ou encore l'expression d'un axe graphique ou d'une sortie numérique sont d'abord représentés par la suite de caractères que l'utilisateur a tapé au clavier. L'étape de création d'une simulation, que ce soit dans PC_CREAT ou SimEDO, vérifie la correction syntaxique chaque fois que l'utilisateur termine l'introduction ou la modification (dans un cas de retour en

arrière ou de reprise d'un modèle existant) d'une expression pour un des éléments du modèle cité ci-dessus. En cas d'incohérence syntaxique, un message d'erreur invite l'utilisateur à modifier son expression pour la rendre correcte. Le processus se répète jusqu'au moment où la correction syntaxique de l'expression est garantie.

En fin d'étape de création, toutes les réponses que l'utilisateur a données forment le modèle de simulation qui doit être sauvé dans un fichier (de l'unité des données) portant le même nom que le modèle créé. En particulier, toutes les expressions du modèle sont sauvées dans ce fichier telles qu'introduites par l'utilisateur.

Ce fichier sert de base à l'exécution du modèle dans PC_EXEC ou SimEDO. Le premier travail, avant d'exécuter le modèle, consiste à relire tous les éléments du fichier constitué à la création. Toutes les expressions sont revérifiées syntaxiquement et transformées sous un format qui les rend facilement évaluables à l'exécution du modèle. Quand au moins une des expressions est syntaxiquement incorrecte, le logiciel ne peut exécuter le modèle. Comment les expressions seraient-elles incorrectes dans le fichier du modèle alors que la fin de création en assure la correction syntaxique ? Un utilisateur pourrait modifier le fichier (de type texte) qui contient une simulation, c'est pourquoi une nouvelle analyse s'impose en début d'exécution. Tout ce travail de vérification de conformité des expressions par rapport à la syntaxe est réalisé par un **analyseur syntaxique** dont le but est également de transformer les chaînes de caractères en représentation adéquate pour en faciliter l'évaluation à l'exécution du modèle.

Dans la suite, nous examinerons le travail acccompli en 2 phases par cet analyseur syntaxique pour chaque expression introduite dans la partie création du modèle ou lue dans le fichier avant l'exécution.

Première phase : Transformation d'une expression sous forme de tableau

La première tâche de l'analyseur syntaxique consiste à :

. enlever les blancs qui figurent dans l'expression, puis

. parcourir la chaîne de caractères à partir de la gauche pour reconnaître les accolades, parenthèses, opérandes et opérateurs qui composent l'expression et,

. construire, parallèlement, un tableau dont l'élément i identifie le ième composant de l'expression, sans oublier

. d'effectuer quelques vérifications syntaxiques.

Chaque élément du tableau peut représenter un opérande, un opérateur, une parenthèse ouvrante ou fermante, c'est le type de l'élément. Pour un opérateur, l'élément du tableau spécifie de quel opérateur il s'agit, quelle est sa priorité et le nombre d'opérandes qui lui sont associés. La représentation d'un opérande est différente; l'élément du tableau mentionne de quel opérande il s'agit mais aussi :

- la valeur entière ou réelle si l'opérande est un entier ou un réel,
- un entier qui accompagne une constante ou variable de retard et qui spécifie la valeur de l'indice,
- pour une variable d'état ou un paramètre, l'indice associé. Le problème ici est qu'on peut trouver plusieurs types d'indices :

  X(i), X(X(i)), X(P(j))

  P(j), P(X(i)), P(P(j))

  avec 1≤i≤nombre de variable(s) d'état du modèle

  1≤j≤nombre de paramètre(s) du modèle

Comment représenter l'indice par un nombre entier ?

La solution adoptée est la suivante :

Soient maxeq et maxpar le nombre maximum de variables d'état et de paramètres qu'autorise SimEDO (=20),

X(i) => indice = i

X(X(i)) => indice = maxeq + i

X(P(j)) => indice = 2 * maxeq + j

Représentons graphiquement la valeur de indice :

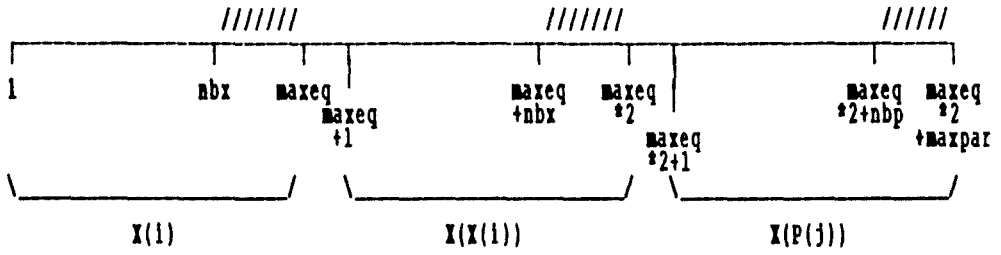


Figure 4.2 - Représentation graphique de la valeur de l'indice d'une variable d'état

P(j) => indice = j
P(X(i)) => indice = maxpar + i
P(P(j)) => indice = maxpar + maxeq + j

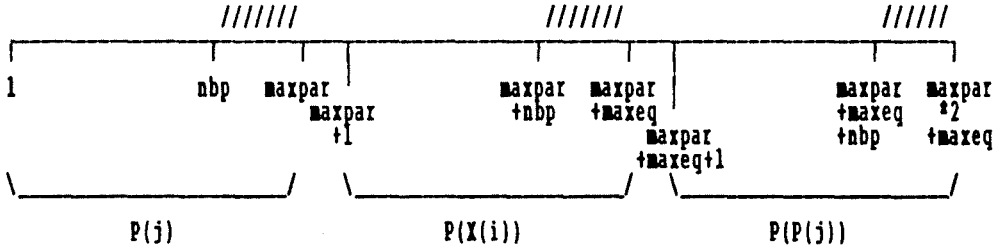


Figure 4.3 - Représentation graphique de la valeur de l'indice d'un paramètre

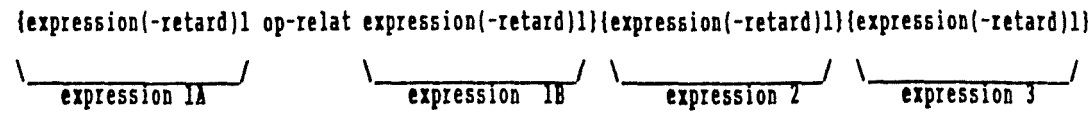
Ce procédé garantit la non ambiguïté au niveau des indices. A partir de la valeur de l'entier associé à l'élément du tableau, on peut aisément retrouver l'indice, qu'il soit i, j, X(i) ou P(j). Si le modèle comprend nbx (<maxeq) variables d'état et nbp (<maxpar) paramètres, les parties /// ne seront jamais utilisées comme valeur d'indice.

Remarques : . Lorsqu'on rencontre π dans une expression, on considère qu'on a affaire à un opérande de type réel auquel on affecte la valeur 3.14159265...

. L'opérande T (temps) ne possède aucun attribut de spécification.

Observons qu'un élément ne peut représenter une accolade, utilisée dans une expression conditionnelle. Comment traite-t-on une expression conditionnelle ?

Après l'étape d'élimination des blancs, une telle expression se présente comme suit :



Dès que le premier caractère est une accolade ouvrante, ça signifie qu'on a affaire à une expression conditionnelle. Le premier travail consiste alors à diviser l'expression en 4 sous-expressions de type "expression1" si l'expression globale est de type "expression" ou de type "expression-retard1" si l'expression globale est de type "expression-retard". On sait que la première paire d'accolades renferme 2 sous-expressions séparées par un opérateur relationnel. Il suffit de localiser cet opérateur et la première paire d'accolades pour délimiter les 2 premières sous-expressions notées 1A et 1B ci-dessus. Les troisième et quatrième sous-expressions sont facilement identifiées en repérant la paire d'accolades qui entourent chacune. En résultat, nous avons 4 chaînes de caractères qui doivent vérifier la syntaxe de expression1 ou expression-retard1, c'est-à-dire des expressions normales. Nous pouvons dès lors leur appliquer le traitement de construction de tableau. Nous obtiendrons 4 tableaux pour une expression conditionnelle et un seul pour une expression normale.
Remarque : la notation expression 1A,... expression 3 sera utilisée par la suite.

Nous allons maintenant analyser la procédure de l'analyseur syntaxique qui construit le tableau des composants d'une expression normale tout en effectuant quelques vérifications syntaxiques. Les 4 chaînes de caractères d'une expression conditionnelle sont traitées les unes après les autres par la procédure.
Arguments :

. une chaîne de caractères censée représenter une expression normale

. un booléen : VRAI si on peut trouver une variable de retard dans l'expression; FAUX sinon.

Résultat : un tableau qui reprend, dans l'ordre où on les a trouvés dans la chaîne de caractères, tous les composants de l'expression normale.
Post-condition : ce tableau n'est valable que si aucune erreur syntaxique n'est découverte.

```
ALGORITHME :
 dernier_caractère_lu := opérateur;
 caractère courant := premier caractère de la chaîne de caractères;
 i:=1;    {indice de l'élément courant du tableau}
TANT QU'on n'est pas à la fin de la chaîne de caractères et QU'on ne détecte pas d'erreur syntaxique
 FAIRE
     SI le caractère courant = '('
     ALORS le type de tableau[i] := parenthèse ouvrante;
          dernier_élément_lu := opérateur;
          caractère courant := le suivant
     SINON SI le caractère courant = ')'
          ALORS le type de tableau[i] = parenthèse fermante;
               dernier_élément_lu := opérande;
               caractère courant := le suivant;
          SINON on essaie de reconnaître un opérateur à partir du caractère courant
               SI on réussit
               ALORS le type de tableau[i] := opérateur;
                    valeur de tableau[i] := opérateur reconnu;
                     SI (dernier_élément_lu = opérateur) ET (l'opérateur reconnu = '-')
                     ALORS priorité de tableau[i] := 2;
                          nombre d'opérande de tableau[i] := 1
                     SINON priorité de tableau[i] := priorité de l'opérateur trouvé;
                                                     (voir description des opérateurs)
                          SI priorité de tableau[i] = 2
                          ALORS nombre d'opérande de tableau[i] := 1;
                          SINON nombre d'opérandes de tableau[i] := 2
                     SI l'opérateur trouvé est relationnel
                     ALORS erreur syntaxique : "OPERATEUR RELATIONNEL NON AUTORISE";
                                               (puisqu'on ne traite que des expressions normales)
                     caractère  courant := celui qui suit immédiatement le dernier caractère de
                     l'opérateur reconnu
               SINON on essaie de reconnaître un opérande à partir du caractère courant en tenant
                     compte du booléen en paramètre
                     SI on réussit
                     ALORS dernier_élément_lu := opérande;
                          le type de tableau[i] := opérande;
                          valeur de tableau[i] := opérande reconnu;
                                                  (I,P,C,D,T,entier ou réel)
                          SI opérande = I,P,C ou D
                          ALORS indice de tableau[i] := indice calculé
                                                        (cfr ci-dessus)
                          SI opérande = entier OU réel
                          ALORS valeur de tableau[i] := valeur de l'entier ou du réel
                          caractère courant := celui qui suit immédiatement le dernier caractère
                          de l'opérande reconnu
```

```
        SINON erreur syntaxique : "SYMBOLE NON RECONNU"
    i := i + 1
FIN TANT QUE
```

Figure 4.4 - Algorithme de construction du tableau d'une expression normale

Commentaires :
On utilise la variable "dernier_élément_lu" pour reconnaître le nombre d'opérandes d'un opérateur de soustraction. C'est en effet le seul qui peut prendre 2 priorités différentes suivant qu'un ou deux opérandes lui sont associés. Quand le dernier élément lu est une parenthèse ouvrante ou un opérateur (dernier_élément_lu = opérateur) et qu'on reconnaît immédiatement après un opérateur moins, cela signifie qu'on a affaire à un opérateur de priorité 2 à un seul opérande.
Exemples : 2 + (-1 * P(4))
P(1) * -1
Par contre, si le dernier élément lu est une parenthèse fermante ou un opérande (dernier_élément_lu = opérande), l'opérateur moins rencontré immédiatement après porte la priorité 5 et exige 2 opérandes.
Exemples : P(2) - X(1)
(1 - X(1)) - D(2)

Cette transformation nous permet de travailler, dans la seconde phase, non plus sur base des caractères mais directement sur les composants qui forment l'expression.

La seconde phase poursuit la vérification syntaxique si aucune erreur n'est apparue lors de la phase initiale et construit, pour chaque expression (normale ou conditionnelle), l'arbre qui permettra de l'évaluer chaque fois que ce sera nécessaire à l'exécution.

Pour éviter toute ambiguïté dans l'évaluation des expressions, nous avons fixé une priorité à chaque opérateur de manière à spécifier univoquement l'ordre d'évaluation des opérateurs.
Exemple : soit l'expression a β1 b β2 c où a, b, c sont les opérandes et β1, β2 les opérateurs. On peut construire 2 arbres différents pour cette expression :

Nous avons adopté la règle suivante :
L'opérateur à la racine de l'arborescence est celui qui a la priorité la plus élevée. Si les priorités de 2 opérateurs sont égales, tout se passe comme si l'opérateur de droite avait une priorité supérieure (règle d'association à droite).

Dans l'exemple ci-dessus, l'arbre (1) représente l'expression si la priorité de β1 est > à la priorité de β2, l'arbre (2) si la priorité de β1 est ≤ à la priorité de β2. Remarquez qu'on trouve un opérateur comme noeud d'arbre et les opérandes aux feuilles.

La poursuite des vérifications syntaxiques et la construction de l'arbre à partir du tableau des composants de l'expression normale issus de la phase 1 s'effectuent par une procédure récursive que nous intitulerons CONSTRUCTION_ARBRE.

Arguments :

. tableau : le tableau des composants construit à l'étape 1,

. binf, bsup : les bornes du tableau entre lesquelles la procédure doit analyser les composants et construire l'arbre correspondant,

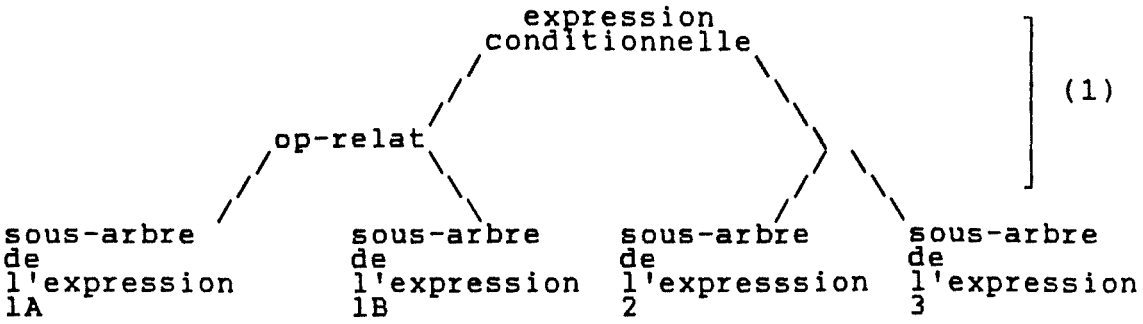
. arbre : adresse de la racine de l'arborescence à construire

Résultat : arbre représente la racine de l'arbre construit

Post-condition : l'arbre est valable si aucune erreur syntaxique n'a été découverte.

Remarque : comme pour la transformation en tableau, la procédure CONSTRUCTION_ARBRE ne travaille qu'à partir de tableaux d'expressions normales. Pour une expression conditionnelle, on désire évidemment avoir un seul arbre pour toute l'expression. Nous savons déjà que l'étape 1 a divisé l'expression en 4 sous-expressions de type "expression1" ou "expression-retard1" pour lesquelles nous possédons la représentation sous forme de tableaux. Nous désirons que l'arbre d'une expression conditionnelle se

présente comme suit :



Pour réaliser cela, il suffit de construire le dessus de l'arbre (1) puis d'appeler 4 fois la procédure CONSTRUIRE_ARBRE en donnant successivement les tableaux des expressions 1A, 1B, 2, 3 et l'adresse du pointeur qui y correspond.

Algorithme :

```
procédure CONSTRUIRE_ARBRE (tableau, binf, bsup, arbre);
Début :
SI binf = bsup
ALORS SI le type de tableau[binf] <> opérande
     ALORS erreur syntaxique : "UNE EXPRESSION D'UN SEUL ELEMENT DOIT ETRE UN OPERATEUR"
     SINON arbre représente le seul opérande (feuille)
SINON position_de_l'opérateur_prioritaire := 0;
     priorité_de_l'opérateur_prioritaire := 0;
     i := binf;   (recherche de l'opérateur prioritaire)
     TANT QUE (i<=bsup) ET qu'on n'a pas rencontré d'erreur
     FAIRE
          SI le type de tableau[i] = parenthèse ouvrante
          ALORS on parcourt le tableau à partir de l'élément i jusqu'au moment où on trouve la
               parenthèse fermante correspondante
               SI on ne la trouve pas
               ALORS erreur syntaxique : "PARENTHESE OUVRANTE SANS PARENTHESE FERMANTE"
               SINON i := indice de l'élément du tableau qui représente la parenthèse fermante
          SINON SI le type de tableau[i] = parenthèse fermante
               ALORS erreur syntaxique : "PARENTHESE FERMANTE SANS PARENTHESE OUVRANTE"
               SINON SI le type de tableau[i] = opérateur
                    ALORS SI la priorité de tableau[i] ≥ priorité_de_l'opérateur_prioritaire
                         ALORS priorité_de_l'opérateur_prioritaire := priorité de tableau[i];
                              position_de_l'opérateur_prioritaire := i;
          i := i+1
     FIN TANT QUE;
SI la position_de_l'opérateur_prioritaire <>0
```

```
ALORS arbre représente l'opérateur prioritaire;
     SI le nombre d'opérandes de tableau[position_de_l'opérateur_prioritaire] = 1
     ALORS SI la position_de_l'opérateur_prioritaire <> binf
          ALORS erreur syntaxique : "OPERATEUR A UN SEUL OPERANDE ET EXPRESSION GAUCHE NON VIDE"
     SINON SI la position_de_l'opérateur_prioritaire = binf
          ALORS erreur syntaxique : "OPERATEUR A 2 OPERANDES : EXPRESSION GAUCHE VIDE"
     SI la position_de_l'opérateur_prioritaire = bsup
     ALORS erreur syntaxique : "OPERATEUR : EXPRESSION DROITE VIDE"
     SI (on n'a pas rencontré d'erreur) ET (le nombre_d'opérandes_de_l'opérateur_prioritaire = 2)
     ALORS CONSTRUIRE_ARBRE(tableau,binf,position_de_l'opérateur_prioritaire - 1,sous-arbre gauche
                            de arbre);
     SI on n'a pas rencontré d'erreur
     ALORS CONSTRUIRE_ARBRE(tableau,position_de_l'opérateur_prioritaire + 1,bsup,sous-arbre droite
                            de arbre);
SINON SI (le type de tableau[binf] = parenthèse ouvrante)
        ET  (le type de tableau[bsup] = parenthèse fermante)
                                                    (pas d'opérateur trouvé)
     ALORS binf := binf + 1;
          bsup := bsup - 1;
          SI bsup <binf
          ALORS erreur syntaxique : "RIEN DANS LES PARENTHESES"
          SINON aller en début
     SINON erreur syntaxique : "EXPRESSION DE PLUSIEURS ELEMENTS SANS OPERATEUR"
```

Figure 4.5 - Algorithme de construction d'arbre à partir de la représentation d'une expression sous forme de tableau

Commentaires :

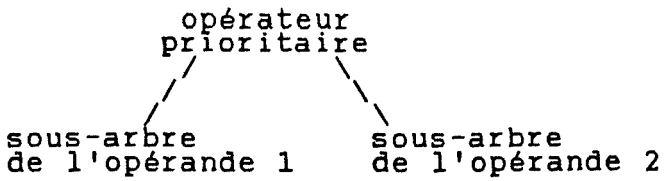
En cas d'absence d'erreur syntaxique :

. l'appel récursif cesse dès que les bornes du tableau à inspecter sont égales et que le type de l'élément unique est un opérande. L'opérande occupe alors une feuille de l'arbre global.

. Si le tableau à inspecter contient plus d'un élément, un appel de procédure CONSTRUIRE_ARBRE consiste d'abord à trouver l'opérateur prioritaire. Trois cas peuvent se présenter :

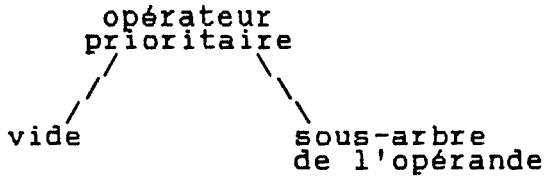
- l'opérateur prioritaire possède 2 opérandes :

Le sommet de l'arbre représente l'opérateur et il suffit d'appeler 2 fois la procédure CONSTRUIRE_ARBRE pour construire le sous-arbre de gauche de l'opérateur (premier opérande) et le sous-arbre de droite (second opérande)



- l'opérateur prioritaire possède un seul opérande

On trouve toujours l'opérateur au sommet de l'arbre. Le sous-arbre de gauche est vide. Un seul appel à CONSTRUIRE_ARBRE est nécessaire pour construire le sous-arbre de droite de l'opérande unique.



- le tableau à inspecter commence par une parenthèse ouvrante et se termine par une parenthèse fermante et pas d'opérateur trouvé

Dans ce cas, on élimine ces 2 parenthèses en faisant progresser la borne inférieure du tableau d'une position et régresser la borne supérieure d'une unité, puis on recommence l'analyse sans lancer d'appel récursif à CONSTRUIRE_ARBRE.

Pendant ces 2 transformations, toutes les vérifications syntaxiques ont pu être opérées. Le lecteur en quête d'exemples d'erreur peut se référer au point 2.3. de l'annexe B.

Toute expression qui arrive au bout du traitement de l'analyseur syntaxique est déclarée correcte. Pour rappel, ce traitement s'effectue aussi bien à la création qu'à l'exécution. L'arbre qui résulte de la transformation d'une expression est directement utilisé pour l'évaluation de celle-ci à l'exécution. Par contre, à la création, une fois que la correction syntaxique de l'expression est garantie, l'arbre n'est d'aucune utilité. C'est pourquoi on l'efface directement pour libérer l'espace mémoire, ressource si rare !!!

Section 5 : Méthodes de résolution et évaluation des expressions

L'exécution d'un modèle de simulation n'est possible que si on dispose d'une méthode de résolution. SimEDO et ProfCOMP en proposent deux que nous allons décrire mathématiquement avant d'énoncer la façon dont elles sont mises en oeuvre sur ordinateur [Conte], [Cra&Dor], [SDA].

Rappelons qu'une équation différentielle s'écrit $y'=f(x,y)$. Géométriquement, cette équation possède comme solutions une famille de courbes dans le plan (x,y). Un point (x_0,y_0) étant donné arbitrairement, l'équation $y'=f(x,y)$ donne le coefficient angulaire de la tangente en ce point, à la courbe qui en est issue :

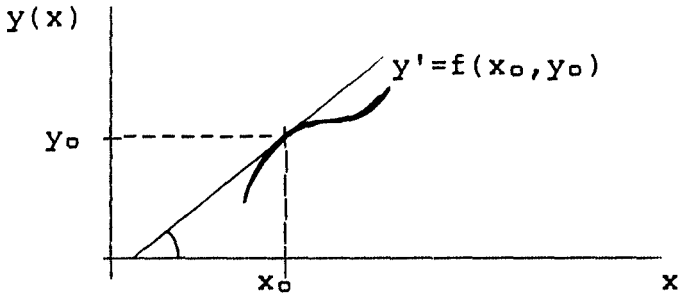


Figure 4.6 - Signification géométrique d'une équation différentielle

De proche en proche, l'équation donne les directions des tangentes successives. Ceci suggère le principe fondamental d'une méthode numérique de résolution :
A partir du point initial (x_0,y_0), on calcule les coordonnées du point (x_1,y_1), avec $x_1 = x_0 + h$, puis de $(x_2 = x_1 + h, y_2)$, et ainsi de suite.

SimEDO et ProfCOMP offrent 2 méthodes de Runge-Kutta dont voici les propriétés essentielles :
1) Ce sont des méthodes à un seul pas : pour trouver y_{m+1}, on n'a besoin d'informations que sur le point précédent (x_m,y_m).
2) Ces méthodes s'accordent avec les séries de Taylor jusqu'au terme en h^p, où p est l'ordre de la méthode.
3) Enfin, l'évaluation des dérivées de f(x,y) n'est pas

requise, contrairement aux méthodes de Taylor. Le prix à payer est d'évaluer la fonction f(x,y) p fois pour chaque pas d'intégration.

Nous nous contenterons de donner le résultat du développement des 2 méthodes :

- EULER AMELIORE (ordre 2)

$$y_{m+1} = y_m + h/2 * [f(x_m,y_m) + f(x_m + h, y_m + h * y_m')]$$

Géométriquement :

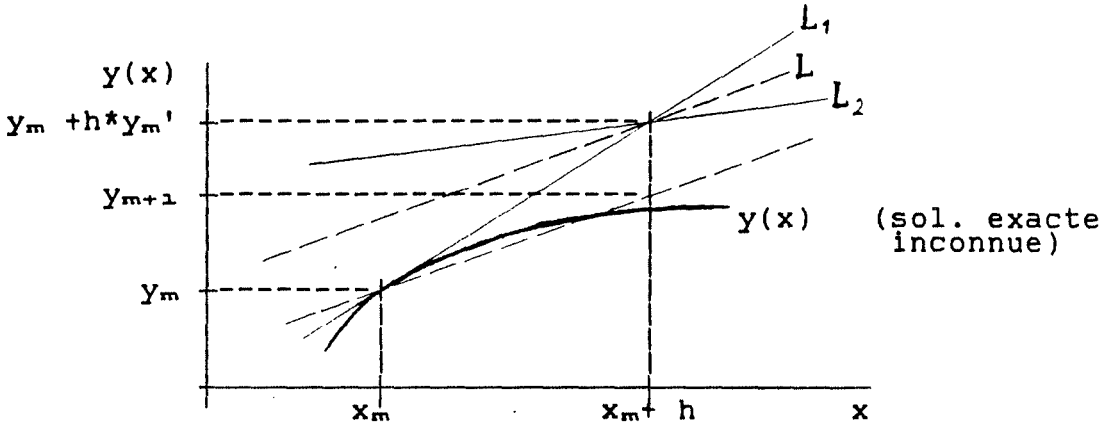


Figure 4.7 - Signification géométrique de la méthode d'Euler amélioré

On part du point (x_m,y_m) et on trace la pente L1 d'équation $y_m'=f(x_m,y_m)$ qui passe par (x_m,y_m). Cette pente L1 coupe la droite $x=x_m+h$ au point (x_m+h, y_m+h*y_m'). La méthode d'Euler choisirait ce point comme (x_{m+1},y_{m+1}). En ce point, on calcule la pente de la courbe $f(x_m+h, y_m+h*y_m')$ et on obtient L2. On fait la moyenne de L1 et L2 et on obtient L. Finalement, on trace une droite parallèle à L passant par (x_m,y_m). Cette droite coupe $x=x_m+h$ et donne le point (x_{m+1},y_{m+1}). Cette méthode requiert 2 fois l'évaluation de la fonction f(x,y), d'où l'ordre 2.

- RUNGE-KUTTA (ordre 4)

Cette méthode est l'une des plus utilisées pour l'intégration d'équations différentielles :

$$y_{m+1} = y_m + 1/6 * [k_1 + 2*k_2 + 2*k_3 + k_4]$$
avec $k_1 = h * f(x_m, y_m)$,
$k_2 = h * f(x_m + h/2, y_m + k_1/2)$,
$k_3 = h * f(x_m + h/2, y_m + k_2/2)$,
$k_4 = h * f(x_m + h, y_m + k_3)$

La méthode exige 4 évaluations de la fonction f(x,y) par pas d'intégration.

Il apparaît que ces 2 méthodes sont largement suffisantes pour avoir une estimation correcte d'un système.
Tableau comparatif des 2 méthodes :

	EULER amélioré	RUNGE-KUTTA 4
Volume des calculs	faible par pas mais beaucoup de pas	4 estimations de f par pas
Erreur de troncature	mauvaise	bonne
Estimation d'une borne d'erreur	difficile	difficile
Stabilité	stable	généralement stable
Possibilité d'un changement de pas	sans aucun problème	sans aucun problème
Domaine d'application	pour des calculs rapides de précision médiocre	bonne méthode assez précise si f(x,y) pas trop compliqué

Figure 4.8 - Tableau comparatif des méthodes de résolution

Dans un modèle de SimEDO, y'=f(x,y) représente l'équation différentielle d'une variable d'état et la variable indépendante x est le temps. L'accroissement h entre 2 valeurs de temps successives est un incrément de temps, souvent désigné δt.

Exemple : d/dt[X(1)]= {P(4)>X(1)}
{P(4)/P(0)-(1/P(0)+1/P(1))*X(1))/P(2)}
{P(5)}

X(1)' = f(t , X(1))

En règle générale, l'équation différentielle d'une variable d'état fait intervenir une fonction f où on trouve la variable d'état définie, le temps mais aussi d'autres variables d'état, des paramètres, des variables de retard et des constantes.

Chaque fonction f est, en fait, ce que nous avons précédemment apppelé une expression, pour laquelle nous avons construit une représentation en arbre.

Outre les équations différentielles des variables d'état, un modèle peut aussi contenir des paramètres et variables de retard également décrits par une équation (non différentielle).

L'évaluation d'une expression

A chaque pas d'intégration, le module d'exécution de SimEDO ou ProfCOMP doit évaluer des expressions de variables d'état, de paramètres, de variables de retard, de sorties numériques ou graphiques. Heureusement, nous disposons, pour chacune, de leur représentation sous forme d'arbre.

Une fois de plus, le traitement diffère selon qu'il s'agit d'une expression normale ou conditionnelle.

Décrivons d'abord la procédure récursive EVALUER_ARBRE_NORMAL qui évalue l'arbre d'une expression normale, et que nous intégrerons ensuite dans la procédure globale d'évaluation d'une expression.
EVALUER_ARBRE_NORMAL admet 2 paramètres :
Argument : l'arbre de l'expression normale à évaluer
Résultat : le résultat de l'évaluation (un réel)
Post-condition : le résultat n'est valable que si on a pu évaluer l'arbre

Algorithme :

```
Procédure EVALUER_ARBRE_NORMAL (arbre,résultat);
variable: résultat1,résultat2 : réel;

SI le sommet de l'arbre représente un opérande
ALORS résultat := valeur courante de l'opérande concerné
SINON SI le sommet de l'arbre représente un opérateur à un opérande
      ALORS EVALUER_ARBRE_NORMAL (sous-arbre droit,résultat1);
            SI résultat1 vérifie les propriétés de l'opérateur concerné
            ALORS résultat := opérateur (résultat1)
            SINON STOP !!! : "EVALUATION IMPOSSIBLE"
      SINON SI le sommet de l'arbre représente un opérateur à 2 opérandes
            ALORS EVALUER_ARBRE_NORMAL (sous-arbre gauche,résultat1);
                  EVALUER_ARBRE_NORMAL (sous-arbre droite,résultat2);
                  SI résultat1 ET résultat2 vérifient les propriétés de l'opérateur concerné
                  ALORS résultat := résultat1 opérateur résultat2
                  SINON STOP !!! : "EVALUATION IMPOSSIBLE"
```

Figure 4.9 - Algorithme d'évaluation d'une expression normale

Quand l'(es) opérande(s) ne vérifie(nt) pas les propriétés requises de l'opérateur, l'évaluation ne peut se poursuivre. Par conséquent, la résolution du système s'arrête et un message d'erreur alerte l'utilisateur de l'incident.

Exemples : une division par zéro,
le logarithme naturel d'un nombre négatif, ...

La liste complète des contrôles sur les valeurs des opérandes se trouve en annexe B point 3.3.

On utilise cette procédure EVALUER_ARBRE_NORMAL dans la procédure non récursive chargée d'évaluer un arbre, qu'il soit normal ou conditionnel : EVALUER_ARBRE. Les paramètres sont identiques à ceux de la procédure précédente mais sans restriction sur l'arbre.

Argument : l'arbre de l'expression (normale ou conditionnelle) à évaluer
Résultat : le résultat (un réel) de l'évaluation
Post-condition : le résultat n'est valable que si on a pu évaluer l'arbre

Algorithme :

```
Procédure EVALUER_ARBRE (arbre,résultat);
variable résultat1, résultat2 : réel;

SI le sommet de l'arbre indique un arbre conditionnel
ALORS EVALUER_ARBRE_NORMAL (sous-arbre de l'expression 1A,résultat1);
      EVALUER_ARBRE_NORMAL (sous-arbre de l'expression 1B,résultat2);
      SI résultat1 op-relat résultat2
      ALORS EVALUER_ARBRE_NORMAL (sous-arbre de l'expression 2,résultat);
      SINON EVALUER_ARBRE_NORMAL (sous-arbre de l'expression 3,résultat);
SINON EVALUER_ARBRE_NORMAL (arbre,résultat);
```

Figure 4.10 - Algorithme d'évaluation d'une expression

Quand l'expression est conditionnelle, on utilise d'abord EVALUER_ARBRE_NORMAL pour évaluer les 2 expressions normales qui entourent l'opérateur relationnel. On compose ensuite ces résultats avec l'opérateur relationnel pour voir si la relation est vraie ou fausse. Si elle est vraie, on évalue l'expression 2 comme résultat global de l'expression. Sinon, c'est l'expression 3 qui fournit le résultat.

Algorithmes des méthodes de résolution

La résolution d'un système consiste principalement à évaluer, pour chaque pas d'intégration, la fonction f de chaque variable d'état, 2 ou 4 fois selon l'ordre de la méthode employée, tout en maintenant à jour la valeur du temps, de chaque paramètre et variable de retard. Il suffit ensuite de calculer la nouvelle valeur de chaque variable d'état d'après les résultats obtenus (cfr l'expression mathématique des méthodes de résolution). Chaque évaluation nécessite un appel à la procédure EVALUER_ARBRE exposée ci-dessus.

Dans la description des algorithmes, nous utiliserons quelques tableaux de réels pour stocker les valeurs intermédiaires :

P : tableau d'autant de réels qu'il y a de paramètres dans le modèle. L'élément P[i] désigne la valeur courante du paramètre i,

X, X_0 et X_1 : tableau d'autant de réels que de variables d'état dans le modèle. X[i] contient la valeur courante de la i-ème variable d'état tandis que X_0[i] et X_1[i]

contiennent des valeurs intermédiaires de la même variable,

D : D[i] renseigne la valeur réelle de la i-ème variable de retard,

K_1, K_2, K_3, K_4 : 4 tableaux qui renferment les valeurs des coefficients du même nom (méthode de Runge-Kutta du quatrième ordre). L'indice correspond à la variable d'état concernée par la valeur (on applique la méthode pour chaque variable d'état; on a donc les 4 coefficients pour chaque équation différentielle).

Remarque : la procédure EVALUER_ARBRE consulte les tableaux X, P et D pour trouver la valeur courante d'une variable d'état, paramètre ou variable de retard. Le temps est une variable de type réel, de même que chaque constante du modèle. L'évaluation d'un de ces éléments consiste simplement à prendre la valeur de la variable correspondante.

Algorithmes :

* Euler amélioré :

```
TANT QUE pas d'erreur ET (le temps ≤ fin de l'intervalle du temps de représentation)
FAIRE
    POUR CHAQUE paramètre j : EVALUER_ARBRE (arbre du paramètre, P[j]);
    POUR CHAQUE variable d'état i : EVALUER_ARBRE (arbre de la variable d'état, X0[i]);
    temps := temps + incrément de temps;
    POUR CHAQUE paramètre j : EVALUER_ARBRE (arbre du paramètre, P[j]);
    POUR CHAQUE variable d'état i : EVALUER_ARBRE (arbre de la variable d'état, X1[i]);
    POUR CHAQUE variable d'état i : X[i] := X[i] + incrément de temps * (X0[i] + X1[i])/2;
    POUR CHAQUE variable de retard k : EVALUER_ARBRE (arbre de la variable de retard, D[k]);
    Opérations des sorties numériques et graphiques
FIN
```

Figure 4.11 - Algorithme de résolution (Euler amélioré)

Remarque : X_0 renferme, pour chaque variable d'état, le résultat de l'évaluation de sa fonction $f(x_m, y_m)$ et X_1 le résultat de $f(x_m + h, y_m + h * y_m')$.

* Runge-Kutta du quatrième ordre :

```
TANT QUE pas d'erreur ET (le temps ≤ fin de l'intervalle du temps de représentation)
FAIRE
    POUR CHAQUE variable d'état i : X[i] := X0[i];
    POUR CHAQUE paramètre j : EVALUER_ARBRE (arbre du paramètre, P[j]);
    POUR CHAQUE variable d'état i : EVALUER_ARBRE (arbre de la variable d'état, K1[i]);
                                    K1[i] := K1[i] * incrément de temps;
    POUR CHAQUE variable d'état i : X[i] := X0[i] + K1[i] / 2;
    temps := temps + incrément de temps / 2;
    POUR CHAQUE variable d'état i : EVALUER_ARBRE (arbre de la variable d'état, K2[i]);
                                    K2[i] := K2[i] * incrément de temps;
    POUR CHAQUE variable d'état i : X[i] := X0[i] + K2[i] / 2;
    POUR CHAQUE variable d'état i : EVALUER_ARBRE (arbre de la variable d'état, K3[i]);
                                    K3[i] := K3[i] * incrément de temps;
    POUR CHAQUE variable d'état i : X[i] := X0[i] + K3[i];
    temps := temps + incrément de temps / 2;
    POUR CHAQUE variable d'état i : EVALUER_ARBRE (arbre de la variable d'état, K4[i]);
                                    K4[i] := K4[i] * incrément de temps;
    POUR CHAQUE variable d'état i : X0[i] := X0[i] + (K1[i] + 2 * K2[i] + 2 * K3[i] + K4[i]) / 6;
    POUR CHAQUE variable de retard k : EVALUER_ARBRE (arbre de la variable de retard, D[k]);
    Opérations des sorties numériques et graphiques
FIN
```

Figure 4.12 - Algorithme de résolution (Runge-Kutta 4)

Les opérations de sorties numériques et graphiques consistent, à l'issue de chaque pas d'intégration, à évaluer toutes les expressions associées à ces sorties et à en représenter les résultats graphiquement à l'écran et/ou à les écrire dans le fichier des sorties numériques.

Remarques : . pour chacune des méthodes de résolution, nous supposons qu'au départ, les tableaux X et P contiennent déjà les valeurs initiales des variables d'état et des paramètres.

. L'évaluation d'une variable d'état qui figure dans une expression graphique ou numérique diffère selon la méthode de résolution :

- Euler : on consulte l'élément i du tableau X pour obtenir la valeur de X(i)
- Runge-Kutta : on consulte l'élément i du tableau X_0 pour obtenir la valeur de X(i)

Section 6 : Un exemple de modèle simulé

La croissance de population [Cunnin]

Sous certaines conditions idéalistes, une colonie d'animaux d'une seule espèce croît de telle sorte que sa population varie selon la relation

$$dx/dt = x' = r * x$$

où x = population à un instant donné,
r = taux de reproduction et
t = le temps.

Si l'espace d'habitation et la quantité de nourriture sont limités, il y a un seuil maximum de population. La relation qui décrit la population s'écrit alors

$$dx/dt = r * x * (1 - (x / x_{\infty})),$$

x_{∞} représente la population qu'on doit atteindre à l'état final.

Les études expérimentales ont montré une variation de population avec le temps. Cependant, au lieu d'atteindre la valeur x_{∞} de manière monotone, on remarque des oscillations importantes autour de cette valeur. On doit donc modifier l'équation de croissance pour tenir compte de ces solutions oscillatoires :

$$dx/dt_{(t)} = r * x_{(t)} * (1 - (x_{(t-\beta)} / x_{\infty}))$$

où $dx/dt_{(t)}$ et $x_{(t)}$ sont évalués au temps t tandis que $x_{(t-\beta)}$ est évalué au temps $t-\beta$. En effet, la population ne réagit pas immédiatement à l'augmentation de son nombre, mais bien après un certain délai β. Le mécanisme de retard s'explique par plusieurs facteurs. Le temps de gestation ou de durée de vie lorsqu'on supprime toute nourriture interviennent notamment dans l'explication de ce retard mais d'autres facteurs peuvent jouer.

Représentation graphique de l'équation de croissance

Soient r = 1, x_{∞} = 20 et l'incrément de temps = 0.5. La valeur initiale de x est 2 (individus de l'espèce) et la période d'observation (le temps) de 10.
Nous voudrions observer x dans le temps pour 2 valeurs différentes de retard β : 1 et 2.

Visiblement, ce modèle ne comporte qu'une seule variable d'état x. Pour être conforme avec le vocabulaire de SimEDO, nous la renommerons X(1). r et x_m sont les constantes C(1) et C(2) du modèle et $x_{(t-\tau)}$ la variable de retard D(1). Etant donné que nous devons observer l'évolution de la population pour 2 valeurs de retard différentes, nous dédoublerons la variable d'état et la variable de retard pour représenter les courbes sur un même graphique : X(1) sera lié au retard associé à D(1) tandis que X(2) le sera au retard associé à D(2).

Le modèle s'écrit comme suit dans SimEDO:

```
d/dt[X(1)] = C(1) * X(1) * (1 - (D(1) / C(2)))
d/dt[X(2)] = C(1) * X(2) * (1 - (D(2) / C(2)))
C(1) = 1
C(2) = 20
D(1) = X(1) avec un retard de 1
D(2) = X(2) avec un retard de 2, et
la valeur initiale de X(1) = 2.
```

Nous représenterons 3 graphiques, avec la valeur du temps (comprise entre 0 et 10) en abcisse et, en ordonnée :

- la valeur de C(2) (= x_m) (sert de point de repère pour observer les oscillations des 2 autres graphiques autour de cette valeur).
- le nombre d'individus si le retard associé est 1 (=X(1))
- le nombre d'individus si le retard associé est 2 (=X(2)).

Résultat simulé :

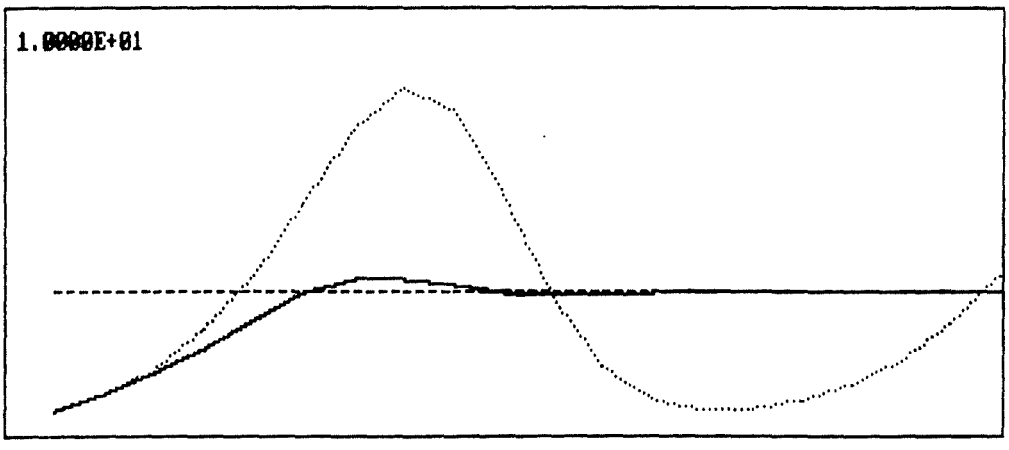


Figure 4.13 - Les 3 courbes demandées

Nous observons que plus le retard est important, plus les oscillations le sont également mais la population tend vers le nombre fixé : 20.

D'autres exemples sont décrits dans [Murphy], [Baccio] et [GTLK].

Section 7

Les limites de SimEDO

Les nombreux tests du logiciel ont révélé certaines lacunes que nous avons, pour la plupart, résolues en cours de développement. Néanmoins, SimEDO accuse toujours quelques points faibles :

- L'inconvénient majeur concerne l'ordre des systèmes d'équations différentielles que SimEDO accepte. Tout système d'ordre supérieur à 1 doit être réduit au premier ordre pour utiliser le logiciel.

- Le nombre de variables d'état, de paramètres ou de constantes se limite à 20, ce qui peut poser problème dans les systèmes compliqués, comme en cardiologie par exemple où le nombre d'équations pousse SimEDO dans ces derniers retranchements. Nous pensons que cette limite peut disparaître ou en tout cas s'atténuer si nous augmentons ce nombre d'équations,... Nous ne l'avons pas encore fait car nous ne disposions pas de modèle de cette taille pour tester le logiciel. Reconnaissons quand même que dans la majorité des systèmes, cette limite actuelle ne pose aucun problème.

- La longueur d'une expression qui définit le membre droit de l'équation différentielle (d'une variable d'état) ou non (paramètre ou variable de retard) se limite à 60 caractères.

- Les sorties numériques ou graphiques ne peuvent représenter les valeurs de d/dt[X(i)] (i=1..nombre de variables d'état du système simulé).

CHAPITRE 5
PROBLEMES RENCONTRES

Section 1
Limites des outils utilisés

Comme nous l'avons déjà mentionné, nous avons employé plusieurs produits de Borland : bien sûr, le langage, le Turbo Pascal 3.0, mais également le Turbo Graphix Toolbox et le Turbo Editor Toolbox. Ce choix n'a pas eu que des avantages.

Le Turbo Pascal (du moins dans la version utilisée) s'avère un peu "juste" pour des projets de cette envergure. Dès l'édition, il ne traite pas les fichiers de plus de 64 K. Or, pour PC_CREAT par exemple, le volume des textes sources s'élève à plus de 400 K. Cela oblige à une gymnastique pour répartir le texte en différents fichiers, les séparations ne pouvant pas toujours avoir une justification logique. Mais cette limite est mineure par rapport aux autres problèmes rencontrés.

L'absence de "linkage" oblige à compiler tout un programme en une seule pièce. Or, code source et code objet occupent environ 700 K. C'est un gros problème lorsqu'on ne dispose pas d'un disque dur ; il faut alors chipoter pour répartir les sources sur deux disquettes avant compilation, en tenant compte du fait que le fichier source principal et tout le code compilé doivent se trouver sur la même disquette. Dans ces conditions, les fichiers .BAK créés automatiquement par l'éditeur du Turbo Pascal deviennent un véritable cauchemar.

Il y a, ensuite, le problème de l'overflow des réels. Evidemment, avec un logiciel de simulation, Nous devions éviter l'affichage du message standard du Turbo Pascal : "Run-time error 01 at PC xxxx", suivi de l'interruption du programme et retour au DOS. Or, en fouillant bien la documentation accompagnant le langage (fichier READ.ME sur la disquette), on découvre que la seule possibilité offerte pour contourner ce problème consiste à écrire soi-même la procédure à exécuter en cas d'overflow. Malheureusement, cela reste insuffisant : au mieux, nous pouvons fermer

quelques fichiers et changer le message à afficher. En effet, si le retour au DOS n'est pas assez rapide à son gré, Turbo Pascal "plante" le programme lui-même. Il fallait donc trouver autre chose. Un "hacker" nous a fourni la solution. Celle-ci nous ramène au programme principal (mais à un endroit choisi par nous), après affichage d'un message plus "gentil" et destruction du stack. Cette opération est obligatoire car il a été complètement faussé par l'overflow et son traitement. Mais cela a quand même l'avantage, pour SimEDO, de ne pas sortir du logiciel (retour au menu principal), et pour ProfCOMP (PC_EXEC) de ne pas interrompre la leçon en cours. En cas de problème dans une simulation, on passera au noeud suivant après avoir averti l'étudiant par un message explicite.

Le problème suivant tient à la conception même du Turbo Pascal 3.0, qui prévoit un maximum de 64 K pour le code et de 64 K pour les données. On ne peut en aucun cas dépasser ces limites. Seule solution, la déclaration de procédures "overlays" : compilées, elles ne seront pas conservées dans le fichier principal ".COM", mais dans un fichier secondaire ".00?" . L'avantage ? Deux procédures se trouvant dans le même fichier secondaire ne seront jamais chargées en mémoire en même temps, l'on ne prévoit donc, dans les 64 K réservés au code, que la place nécessaire pour loger la plus grande d'entre elles. Mais pour utiliser cette possibilité, sans accroc à l'exécution, les différentes procédures qui appartiennent au même fichier d'overlay doivent être complètement indépendantes l'une de l'autre. De plus, on ne choisit pas dans quel fichier on place une procédure : cela se fait automatiquement, avec comme règle que pour que deux procédures se trouvent dans le même fichier, il faut qu'elles soient de même niveau, et que toutes les procédures de ce niveau se trouvant entre les deux dans le code source soient elles aussi déclarées en overlay. Etant donné la taille du projet, nous avons dû recourir abondamment à cette possibilité. Nous nous retrouvons avec 8 fichiers d'overlays pour PC_CREAT. Malgré cela, le moindre changement à l'un des logiciels repose le problème de l'overflow (ici, cela signifie que l'on dépasse la limite des 64 K). Le programme le plus sensible sur ce

point est sans aucun doute PC_EXEC, le plus "excentré", la simulation en occupant une très grosse part. Mais ce problème a d'autres conséquences plus graves : nous avons dû, pour chaque programme, ne garder du Turbo Graphix Toolbox que ce qui était réellement utilisé. Cela nous en donne trois versions différentes, chacune d'elle étant elle-même différente de la version standard. Cela complique les corrections et modifications aux procédures graphiques, que l'on doit répercuter dans trois logiciels différents.

Enfin, toujours dans le même sujet, ce problème nous a obligés à faire deux programmes de la partie création d'une leçon de ProfCOMP : PC_CREAT a dû être doublé par PC_EDIT. Il faut donc aussi gérer le passage de l'un à l'autre (cfr page 30), ce qui est évidemment beaucoup plus complexe qu'un simple appel de procédure. Cela nous oblige également, pour des raisons internes à Turbo Pascal, à préciser à chaque compilation de PC_CREAT, la taille des segments code et données de PC_EDIT. Par conséquent, toute modification apportée à PC_EDIT force à recompiler aussi PC_CREAT.

Les "packages" standards ne sont pas non plus exempts d'erreurs. Beaucoup sont minimes : dans le Turbo Graphix Toolbox, certaines procédures ne répondent pas tout à fait à leurs spécifications, ou encore il faut enchaîner plusieurs d'entre elles, dans un ordre qui n'est pas précisé dans le manuel, pour qu'elles fonctionnent correctement. L'erreur la plus typique, nous l'avons rencontrée dans le Turbo Editor Toolbox. Elle mérite d'être expliquée en détail, car il s'agit en fait d'une mauvaise solution apportée à une erreur. C'est dans la procédure d'affichage du répertoire que nous l'avons trouvée. Nous avons récupéré cette procédure pour tous nos programmes, afin d'afficher le répertoire courant lorsque l'utilisateur le désire (**F5**). Mais nous avons voulu changer l'endroit de l'écran où on affiche ce répertoire. C'est là que l'erreur apparaît : en effet, dès la deuxième ligne, l'affichage débutait de nouveau à l'endroit qu'avait fixé Borland. Voici la procédure originale :

```
overlay procedure getdirectory;
{  This routine gets the user specified directory by making
   calls to MS-DOS functions $4E and $4F.}
Const
  Prompt : String[15] = 'Directory mask:';
  Nodir  : String[14] = 'No entry found';
  Return : String[32] = 'Please press any key to continue';
  cl = 3; ln = 10; wd = 72; hg = 14;
Var
  Entry,St:VarString;
  reg : record
          case integer of
           1 : (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : integer);
           2 : (AL, AH, BL, BH, CL, CH, DL, DH            : byte);
          end;
  ch:Char;
  c,i:Byte;
  zeroflag:boolean;

  procedure promptnext;
  begin
    MenuColor := WLowColor;
    SetMemAddress(cl + 2, ln + 12);
    WriteString(Return);
    ch := ReadChar;
    MenuColor := WNormColor;
  end;

begin { procedure GetDirectory }
  MakeWindow(cl,ln,wd,hg);
  SetMemAddress(cl + 2,ln + 1);
  MenuColor := WLowColor;
  WriteString(Prompt);
  MenuColor := WNormColor;
  CharFilter := ['0'..'9','@'..'Z','*','\','.',':','$','_'];
  St := GetString(cl + 20,ln + 1,40,DirMask,true);
  c := 0;
  SetMemAddress(cl + 2,ln + 2);
  if St <> #27 then
    with reg do
    begin
      If (st[0] = #0) or
         (St[Byte(St[0])] = '\') or
         (st[byte(st[0])] = ':') then st := st + '*.*';
      DirMask := St;
      St[Byte(St[0]) + 1] := #0;
      AH := $1A;
      DS := Seg(Entry);
      DX := Ofs(Entry);
      MSDos(reg);
      AH := $4E;
      CX := 0;
      DS := Seg(St[1]);
      DX := Ofs(St[1]);
      MSDos(Reg);
      If AL <> 0 then
      begin
        SetMemAddress(cl + 5,ln + 5);
        WriteString(Nodir);
      end
                else
      While AL = 0 do
      begin
        ZeroFlag := false;
        for i := 30 to 43 do
        begin
          if Entry[i] = #0 then ZeroFlag := true;
          if ZeroFlag then Entry[i] := ' ';
        end;
        Entry[29] := #12;
        MenuColor := WNormColor;
        WriteString(Entry[29]);
        c := Succ(c);
        memadr := memadr + 4;
        if c mod 5 = 0 then
          SetMemAddress(5,ln + 2 + (c div 5));
        if c = 50 then
        begin
          promptnext;
          St[0] := #68;
          fillchar(st[1],68,' ');
```

```
          for c := ln + 2 to ln + 12 do
          begin
            SetMemAddress(5,c);
            WriteString(st);
          end;
          c := 0;
          SetMemAddress(5, ln + 2);
        end;
        ah := $4F;
        MSDos(reg);
      end;
    end;
  if (st <> #27) and (c <> 50) then promptnext;
  RestoreWindow(cl,ln,wd,hg);
end; { procedure GetDirectory }
```

figure 5.1

On voit que pour modifier l'endroit d'affichage, il faut modifier les constantes CL (colonne) et LN (ligne), du moins dans l'esprit du concepteur. SetMemAddress étant pratiquement l'équivalent de GotoXY, les SetMemAddress(5,...) soulignés ci-dessus auraient dû, en fait, être remplacés par SetmemAddress(CL+2,...). C'est sans doute ce que Borland avait fait dans un premier temps. Mais cela ne pouvait pas marcher non plus : on utilise aussi Reg, qui est un record. Un des éléments de ce record est Reg.CL. L'instruction "With reg do" soulignée ci-dessus entraîne que deux variables différentes sont désignées par le même nom. Malheureusement, dans les SetMemAddress, c'est l'élément du record qui est choisi et pas la constante. Borland s'est donc aperçu de l'erreur, mais n'en a pas compris la cause : il l'a ensuite corrigée d'une façon qui annule totalement l'avantage procuré par l'emploi de constantes. Ceci n'est malheureusement qu'un exemple parmi d'autres de bugs existant dans les "packages" commerciaux.

Section 2
Types de cartes

Nous avons déjà largement abordé ce problème lorsque nous avons développé l'éditeur graphique de ProfCOMP (chapitre 3). Mais il est important : le Turbo Graphix Toolbox est, en fait, constitué de deux toolbox différents, suivant la carte graphique installée (IBM ou Hercules). Cela oblige normalement à travailler en permanence sur deux versions des logiciels, une par type de carte, avec tous les inconvénients que cela comporte. Il est en effet difficile de gérer en permanence des versions différentes, sans utiliser un outil de gestion de projets. Nous avons donc choisi une autre solution : fondre les deux "toolbox" en un seul, avec détection automatique du type de carte (au lieu d'une simple vérification de cohérence entre la carte et le toolbox utilisé). Cela nous a obligé à modifier en profondeur la structure de ce toolbox.

Cette solution n'a malheureusement pas que des avantages : le toolbox compilé occupe plus de place, inconvénient majeur, comme nous l'avons déjà signalé. Mais surtout sa maintenance devient beaucoup plus compliquée. Imaginons en effet que demain, on mette à notre disposition une nouvelle version plus performante du Turbo Graphix Toolbox. Il sera alors très difficile de le réadapter à nos logiciels, ce qui n'aurait pas été le cas si l'on avait conservé une structure plus proche du "package" commercial. Il sera donc éventuellement plus compliqué de faire profiter à SimEDO et ProfCOMP des progrès réalisés par Borland dans le Turbo Graphix Toolbox.

Quid des nouveaux standards apparus sur le marché à l'initiative d'IBM (VGA,...) ? Le problème est le même que pour la carte EGA, dont les capacités sont sous-employées par nos logiciels. En effet, la seule différence acceptée entre CGA et EGA est que pour le deuxième type de carte, en mode graphique, la couleur du fond de l'écran peut être autre que noire. Mais la définition ne varie pas : elle est volontairement limitée par le toolbox à 640 x 200 points. Il n'y a pas de raison qu'il en soit autrement pour les nouvelles cartes. Traiter à leur pleine capacité ces cartes

graphiques semble impossible, certaines ayant une définition de 1024 points sur 750. La mémorisation d'un écran graphique sous la forme choisie par le Turbo Graphix Toolbox (1 bit par pixel) exige alors 96K, tout en restant en monochrome ! Rien que pour exécuter une animation dans PC_EXEC, on pourrait ainsi consommer jusqu'à 480 K de la mémoire centrale. Employer ces cartes à pleine capacité exige une modification en profondeur du Turbo Graphix Toolbox.

Section 3 : "Bugs" de la version laboratoire

Nous allons évoquer ici, outil par outil, les bugs majeurs qui "polluaient" ProfCOMP. Nous considérerons également comme bugs les mauvais choix de conception. Cette liste n'est bien entendu pas exhaustive.

Tout d'abord, le système de menus. Il était quasi identique à celui qui existe encore aujourd'hui. Malheureusement, en voulant trop bien faire, on centrait les différents choix possibles; cela enjolivait la présentation, mais l'effet principal était d'empêcher le choix par l'initiale. L'ensemble de celles-ci n'étant pas aligné, l'utilisateur ne pouvait les visualiser d'un seul coup d'oeil [Ramb].

Autre "bug" général : le manque d'homogénéité. Dans PC_CREAT, il existait au moins deux procédures de saisie d'entier. Dans le même registre, certains noeuds demandaient la taille des fenêtres de façon numérique, alors que d'autres procédaient de façon graphique, comme actuellement. Les touches de fonction n'avaient pas toujours la même signification.

PC_EDIT

Là, la faute principale était apparemment volontaire, et destinée à masquer une erreur du formateur. Lorsqu'on passait à la ligne suivante (par le fait de la frappe kilométrique - ou "Wordwrap") dans l'édition d'un texte, on se retrouvait en colonne 2, et non en colonne 1. Comme nous l'expliquerons par la suite, cela permettait de cacher la perte par le formateur du premier caractère de chaque ligne.

PC_CREAT

Première option du menu, premier problème. On offrait la possibilité de modifier les couleurs, mais on ne sauvait pas les changements effectués. L'utilisateur, qui passait dans PC_EDIT (qui pour lui est le même programme), retrouvait les couleurs originales à son retour dans PC_CREAT !

Passons au formateur. Comme nous l'avons déjà évoqué, une erreur très grave se cachait ici : en effet, la perte du premier caractère de chaque ligne modifie complètement le texte, mais en plus empêche de reconnaître les commandes qui s'y trouvent. Ce premier caractère, que l'on croyait perdu, était affecté au byte 0 de **taban** (cfr page 39), qui représente normalement la longueur de la chaîne de caractères. Celle-ci était donc également fantaisiste.

D'autres erreurs résidaient dans cette partie du programme, qui souvent, mystérieusement, formatait (?) n'importe comment.

L'éditeur graphique ne contenait pas d'erreur grave, mais s'avérait beaucoup trop limité. Il n'offrait pas d'option "restore" : un premier menu permettait de choisir entre dessin normal, dessin d'un symbole prédéfini, ou encore dessin à partir d'une base existante. Les seuls "restore" possibles étaient ceux de symbole prédéfinis, sauf si on se trouvait dans la deuxième option (symboles prédéfinis non récursifs).

Enfin, il n'y avait pas de curseur (le pixel courant s'inscrivait en inverse), on ne pouvait modifier la vitesse de déplacement. Les noeuds animation n'existaient pas encore.

L'éditeur de noeuds questions-réponses posait peu de problèmes. La taille d'un questionnaire étant limitée à 50 questions, une erreur sur la condition de boucle empêchait toutefois de modifier un questionnaire de taille maximum. De plus, il était impossible de modifier la question courante lorsqu'on se trouvait au choix entre l'abandon et l'enregistrement : le retour en arrière grâce à **F1** n'était pas prévu ici.

Pour les noeuds simulation, l'erreur a déjà été évoquée dans la section 2 du chapitre 4, qui rappelle son historique : c'est le choix d'un programme séparé par modèle qui était à remettre en cause.

L'éditeur d'enchaînement n'offrait pas la possibilité de se déplacer dans les menus, et interdisait l'interruption de l'édition de la ligne courante. Le compilateur était très valable, mais considérait comme erreurs graves des problèmes qui ne posaient pas de difficulté à l'exécution de la leçon, et qui n'auraient donc du provoquer que des avertissements.

PC_EXEC

Nous avons déjà expliqué le problème de PC_EXEC : c'était bien entendu le retour au programme après exécution d'une simulation. Ce retour nécessitait la reconstruction du "stack", et de revenir dans le programme juste après l'appel à la simulation. Un autre erreur pouvait provenir d'une division par zéro, dans le cas par exemple d'un test sur le rapport entre le nombre de points obtenus par l'étudiant et le maximum possible, alors qu'aucune cotation n'a encore été effectuée.

Utilitaires

Aucune installation des programmes n'était prévue : il était impératif d'utiliser un système pourvu de deux lecteurs de disquettes. Les fichiers messages utilisés par le logiciel devaient avoir l'extension ".MES" : l'utilisateur devait donc utiliser les commandes MS-DOS rename, copy,... pour changer la langue du programme. En plus, aucun changement définitif des couleurs n'était possible.

La traduction des messages elle-même se faisait en deux passes : tout d'abord via un éditeur de texte classique, ensuite par l'utilisation d'un petit programme qui transformait ce texte dans le format exigé par ProfCOMP.

Enfin, aucun utilitaire ne remplaçait les commandes MS-DOS copy, delete et rename. Or, comme celles-ci ne sont pas suffisantes (le fichier WINDOWS.DIM n'est pas mis à jour - cfr annexe A, point 8), il était impossible de copier un noeud d'une disquette vers une autre.

CHAPITRE 6
DOMAINES D'UTILISATION

Section 1 : ProfCOMP

Au départ, il était clairement établi que ProfCOMP était un logiciel d'EAO, conçu pour créer et exécuter des leçons dans des domaines les plus diversifiés possibles. Les noeuds texte, graphique ou animation sont valables quel que soit le type de cours. La simulation intégrée est plus spécifique à certains domaines, tels que électricité, circuits électroniques, mais aussi physiologie (il est par exemple possible de simuler un modèle du coeur sous forme d'équations différentielles), ... La vraie restriction viendrait plutôt des noeuds questions-réponses. La vérification des réponses est en effet trop limitée pour certaines branches plus littéraires. Pour l'adapter à de tels domaines, il faudrait presque développer un système expert de compréhension du langage naturel (valable pour toutes les langues, puisque le logiciel est "traduisible"). Cela tient de la gageure. Comme nous l'avons vu, il est possible de contourner ce problème en n'introduisant aucune réponse lors de la création du questionnaire. Cela élimine toute vérification sur la réponse, et interdit l'adaptation du reste de la leçon à la réponse fournie.

On peut trouver un deuxième domaine d'application, qui se rapproche très fort de l'EAO : la réalisation de "tutorials". PC_EXEC semble très bien adapté à l'apprentissage d'autres logiciels ; dans ce cas, la ou les leçons créées pour composer ce tutorial peuvent avoir une diffusion beaucoup plus large qu'avec l'EAO classique où l'enseignant composera souvent des cours uniquement pour ses propres classes.

Un dernier domaine très important : les tests de sélection. Ici, les questionnaires ne constituent pas une restriction, les QCM étant autant utilisées que les questions ouvertes, qu'elles soient vérifiées ou non. En effet, le rapport de l'exécution de la leçon, avec les différents choix effectués par le testé (notamment dans les questions-menu), ses réponses aux questions, et le temps pris pour assimiler la leçon ou effectuer les tests,

permettent de porter un jugement très précis sur les différents candidats. Avec l'avantage d'observer des candidats placés face à une machine qui leur est peut-être inconnue, l'ordinateur.

Il ne faut cependant pas oublier que quelle que soit l'utilisation que l'on fait du logiciel, son efficacité dépendra principalement de la valeur de la leçon créée, qu'elle le soit par un enseignant ou par un "testeur".

Section 2 : SimEDO

Bien sûr, SimEDO intéresse uniquement les personnes qui travaillent dans des domaines où l'on peut modéliser la réalité sous forme de systèmes d'équations différentielles. Mais il peut aussi bien se révéler outil d'expérimentation pour du personnel scientifique, qu'outil d'enseignement dans un cadre scolaire.

En effet, pour un étudiant en électronique, le meilleur moyen de comprendre un circuit à peu de frais n'est-il pas de le modéliser, puis de jouer sur ce modèle et ses paramètres afin d'observer les variations de résultats ? A partir du moment où il réussit à écrire correctement le modèle, sa compréhension du circuit étudié est bonne. En jouant sur les conditions initiales, il va assimiler le champ des valeurs possibles pour chacun des coefficients intervennant dans le modèle, leur influence, et discerner plus aisément les valeurs réalistes.

Mais outil d'enseignement, SimEDO peut l'être aussi tout simplement dans un cours sur les systèmes d'équations différentielles. C'est la contre-partie idéale d'un cours théorique sur cette matière ; ce serait beaucoup plus amusant, plus captivant pour les étudiants que les habituels exercices sur papier.

CHAPITRE 7
JUGEMENT CRITIQUE

ProfCOMP et SimEDO ont été conçus par prototypage : définition de grands principes généraux, et des fonctionnalités désirées, mais pas de fonctionnalités détaillées. Dans un projet de ce genre, il semblait difficile de faire autrement. Imaginer dès le départ les spécifications des logiciels tels qu'ils existent maintenant semble tout à fait impossible. La méthode par prototypage a l'avantage que l'on obtient rapidement quelque chose de montrable, mais il faut, pour qu'elle réussisse, que la programmation tienne compte du fait qu'une fois la première version réalisée, elle subira d'incessantes modifications. Le simple fait que nous disposions aujourd'hui d'une version "correcte" prouve que nos logiciels ont bien été conçus dans un tel esprit.

Mais la méthode a les inconvénients de ses avantages : impossibilité de prévoir le travail à répartir, de le planifier, de savoir où l'on va exactement. De plus elle laisse de nombreux choix de conception pour l'étape de programmation, qui n'en est pas facilitée.

Ces logiciels ont l'énorme avantage d'être facilement extensibles : ProfCOMP peut recevoir de nouveaux types de noeuds, propres à un domaine d'enseignement particulier ; on peut aussi ajouter de nouvelles instructions à son langage d'enchaînement, de nouvelles commandes au formateur, de nouvelles fonctions à l'éditeur graphique. Du point de vue simulation, on peut aisément programmer de nouvelles méthodes de résolution, tant pour ProfCOMP que pour SimEDO. La simulation a d'ailleurs elle-même déjà fortement évolué, puisqu'on y a déjà intégré, depuis sa création sous cette forme, les constantes et les variables de retard. C'est dire si les possibilités d'extension sont réelles.

Mais des inconvénients existent aussi, il ne faut pas s'en cacher. La rapidité d'exécution n'est pas excellente, surtout si l'on utilise un système sans disque dur. Cela est dû en partie au système d'overlays, expliqués dans la première section du chapitre 5. Comme nous les avons multipliés à l'excès pour arriver à faire tenir tout le code

en 64 K, nous avons même déclaré ainsi certaines procédures d'utilisation très fréquente. Tant qu'il n'y en a qu'une par fichier, les temps peuvent rester très raisonnables, car si elle est vraiment utilisée très fréquemment, elle restera quasi en permanence en mémoire centrale. Dans le cas contraire, si à chaque utilisation d'une procédure il faut d'abord aller la lire sur fichier, les performances deviennent vraiment très mauvaises. On comprend donc aisément que, bien que fonctionnant très correctement sur un système muni uniquement de deux lecteurs de disquette, l'utilisation d'un disque dur augmentera sensiblement le confort d'utilisation de nos logiciels.

Ne mettons pas tout sur le compte des overlays : les messages du logiciel interviennent également comme source de lenteur. Il est bien sûr beaucoup plus agréable de pouvoir traduire les messages d'un programme, pour l'adapter à la langue maternelle de celui qui l'utilise, mais cela implique l'obligation de les placer dans des fichiers séparés. Nous avons essayé de limiter ces pertes de temps, en placant les messages dans leur ordre d'utilisation, quitte même à en dédoubler certains. Cette technique a pour but de diminuer le nombre de défauts de page lors de la lecture de ces messages.

Enfin, il y a un problème de place occupée sur mémoire secondaire. En effet si, en général, les différents types de noeuds, ou encore les leçons, ne prennent que très peu de place, ce n'est pas le cas des graphiques. Ceux-ci peuvent prendre de 15 K maximum pour une carte IBM à plus de 31 K pour une carte Hercules. Cela malgré le fait qu'on ne travaille qu'en monochrome lorsqu'on est en mode graphique. C'est dire que lorsqu'on utilise un système sans disque dur, et que l'on veut consacrer une disquette aux données, il ne faut pas mettre trop de graphiques dans la leçon si l'on veut éviter les problèmes.

Par rapport à la transparence voulue, nous estimons avoir rencontré notre objectif : en effet, l'enseignant visualise directement le noeud qu'il crée. En ce qui concerne les textes, après formatage, le noeud s'exécute pour lui exactement de la même façon que pour l'étudiant.

S'il s'aperçoit d'un problème, il peut alors soit retourner à l'éditeur, soit (par **F1**), redéfinir la taille de la fenêtre dans laquelle s'exécutera le noeud.

Pour les graphiques, il a en permanence sous les yeux ce que verra l'étudiant. Seules les animations ne sont pas vraiment "testables" par l'enseignant. Les questionnaires, eux, sont réellement conçus en deux parties : la fenêtre que verra l'étudiant s'affiche en permanence à l'écran.

La simulation pose problème : il est en effet difficile de permettre à l'enseignant de la tester tout en restant dans PC_CREAT, sans risquer de transformer notre logiciel en monstre. SimEDO permet de réduire cet inconvénient. Logiciel indépendant, il peut aussi servir à la mise au point de simulations pour ProfCOMP.

Malheureusement, l'enchaînement de la leçon elle-même ne peut être visualisé, même en ne dessinant que les cadres des fenêtres à l'endroit où elles seront placées pour l'étudiant. C'est sans aucun doute l'accroc le plus grave à la transparence.

Pour conclure, il faut bien dire que la taille même des logiciels cause une certaine lourdeur d'utilisation, impose un certain apprentissage. Bien qu'étant guidé au maximum, il faudra en effet perdre un certain temps en tâtonnements avant d'utiliser efficacement SimEDO et ProfCOMP. C'est d'ailleurs pourquoi nous avons voulu réaliser des manuels d'utilisation les plus complets possibles (cfr annexes A et B).

CHAPITRE 8
EVOLUTION DESIREE

Bien que la version actuelle soit utilisable, nous pourrions lui appliquer une foule d'idées pour l'améliorer encore. Certaines sont très simples à réaliser, mais d'autres exigent des modifications plus profondes.

Certaines de ces idées sont très générales, et la première découle d'ailleurs de ce que nous avons développé au chapitre précédent. Il serait intéressant d'augmenter sensiblement la vitesse d'exécution de nos logiciels. Ce n'est pas simple, car cela passe sans doute par l'utilisation du Turbo Pascal 4.0, qui n'est plus limité aux 64 K, et permet le linkage d'unités compilées. Cela exige donc un changement en profondeur de la structure actuelle des logiciels.

Tests

Autre désir irréalisable dans le contexte actuel : tester toutes les entrées-sorties, et toutes les demandes dynamiques de place mémoire, comme cela se fait dans PC_EDIT. Cela paraît simple, mais c'est tout à fait impossible pour le moment : nous frôlons en effet de trop près la limite de 64 K pour le code (Cfr chapitre 5, section 1) que pour surcharger encore le logiciel par ces nombreux tests. Nous avons dû nous contenter de tester les lectures sur fichiers, les écritures devant normalement, elles, se dérouler sans accroc s'il y a assez de place sur la mémoire secondaire. Pour les besoins dynamiques en mémoire centrale, nous nous sommes assurés qu'il n'y ait aucun problème possible sur un système disposant de 640 K.

Répertoire

L'affichage du répertoire se produit lorsque, devant introduire un nom de fichier, l'utilisateur appuie sur **F5**. Nous voudrions offrir la possibilité de choix direct dans la liste des noms affichés, c'est-à-dire que l'utilisateur pourrait se déplacer, grâce aux flèches, vers le fichier de son choix, et le sélectionner ensuite grâce à la touche **RETURN**.

Enregistrement des noeuds

Chose très simple à mettre en oeuvre : avant de sauver un noeud, redemander son nom définitif, en proposant comme valeur par défaut le nom donné en début de définition de noeud. Au cas où les deux noms sont identiques, le sauvetage se ferait alors directement. Par contre, s'ils sont différents, un test se fera pour vérifier si ce noeud existe déjà, afin d'éviter son effacement accidentel. S'il existe, on demande donc à l'enseignant de confirmer le nom donné, ou d'en choisir un autre. Ce système existe déjà pour l'éditeur d'enchaînement et, dans une version légèrement différente, pour l'éditeur graphique.

Help

Autre idée très intéressante, l'intégration à tous les niveaux d'un **"help on-line"**. Cette aide devrait donc expliquer le point où l'utilisateur se trouve, mais à partir de là, l'accès à l'entièreté du texte devrait être possible. A chaque page d'explication, l'utilisateur aurait le choix entre plusieurs possibilités, ce qui lui permettrait de parcourir toute l'aide à son gré. Ceci est quasi impossible pour l'instant, car cela alourdirait trop le texte du programme que pour pouvoir encore faire tenir le code compilé dans les 64 K évoqués au chapitre 5. En effet, une telle procédure appelable de partout n'est indépendante d'aucune autre partie du programme et ne peut être déclarée en overlay. En se contentant toutefois de généraliser le système d'aide qui existe dans l'éditeur graphique, nous pourrions l'envisager.

Souris

Il nous a été impossible d'intégrer correctement la souris à toutes les parties de nos logiciels. Une intégration valable de cet outil doit répondre à notre avis à plusieurs exigences : entrée des données indifféremment par le clavier ou la souris, et si on utilise les deux alternativement, conservation de l'ordre d'introduction des données ; signification changeante, suivant le contexte, des boutons de la souris ; enfin, la structure actuelle des programmes ne doit pas être modifiée : pas question par exemple que l'éditeur graphique se soucie de la provenance du caractère qu'il va utiliser.

PC_EDIT

Une autre idée, qui est difficilement applicable, consiste à fusionner l'éditeur et le formateur de texte. Cela ne semble en effet qu'un voeu pieux, sauf à réécrire entièrement un tel outil, qui se rapprocherait alors très fort d'un traitement de texte.

Enchaînements

On peut avancer la même idée à propos de la création des enchaînements : pourquoi ne pas fusionner l'éditeur et le compilateur ? Cela simplifierait fortement l'apprentissage de cette partie par l'enseignant. Mais la difficulté est ici qu'on perdrait soit la garantie qu'une leçon compilée ne contient aucune erreur grave, soit la possibilité de conserver un enchaînement non encore totalement terminé.

Mais il est toutefois possible d'améliorer l'éditeur d'enchaînement, en permettant de revenir un pas en arrière dans l'écriture d'une instruction, en prévoyant la possibilité de corriger une instruction existante, en facilitant le déplacement à l'intérieur de l'enchaînement. Et, pourquoi pas, prévoir que l'enseignant écrive l'enchaînement à l'aide d'un éditeur de texte classique plutôt que d'utiliser cet outil où il est beaucoup plus guidé. Cela entraînerait la création d'un second compilateur qui fonctionnerait, lui, à partir d'un fichier texte.

Toujours dans ce domaine, il est possible d'étoffer le jeu d'instructions disponibles. C'est ainsi qu'il semble très intéressant de doubler l'instruction **EXECUTE NODE** d'un **EXECUTE LESSON**. Très simple à réaliser, cela permettrait à l'enseignant d'adopter une démarche plus modulaire dans la conception de son cours. Une telle modification ne pose aucun problème majeur de réalisation.

Autre instruction possible, le déplacement d'une fenêtre existante. Mais cela pose toujours un problème : le professeur ne peut se rendre compte de ce que va donner sa répartition des fenêtres, lors de l'exécution de sa leçon. Le déroulement de celle-ci n'étant pas linéaire (**IF**, **GOTO**), il est en effet impossible de le lui montrer lors de l'édition de l'enchaînement.

Enfin, pour en terminer avec cet outil, on peut regretter que celui-ci ne soit basé que sur un système de cotation lors des questionnaires. Il serait sans doute intéressant de remplacer le tout par un petit système expert qui déciderait du déroulement de la leçon lors de son exécution. Outre la difficulté de réalisation, cela pose toutefois le problème de l'enseignant non informaticien. Comment maîtriseait-il un tel outil ? Il semble beaucoup plus simple d'intégrer un système expert pour la vérification des réponses en langage naturel, en limitant par là même ProfCOMP au français, que d'en concevoir un pour le déroulement même de la leçon.

Questionnaire

Dans les questionnaires, il serait évidemment très intéressant d'intégrer un tel système expert pour la vérification des réponses en langage naturel [Athéna]. Dans l'état actuel des recherches, c'est quasi-impossible si l'on ne se limite pas à une seule langue, ce qui est gênant dans un pays comme la Belgique.

Mais nous regrettons toutefois que la structure d'un questionnaire soit trop statique : impossibilité de supprimer une question, ou de modifier l'ordre de ces questions. Cela est dû au peu de primitives sur fichiers offertes par Turbo Pascal 3.0, et sauf à ajouter un utilitaire de tri et de gestion de fichiers (par exemple celui du Turbo Access Toolbox de Borland), il semble très difficile de corriger ce défaut.

Simulation

Dans tous les programmes où il est possible de définir un modèle de simulation, il serait bon d'améliorer l'interface, par exemple en affichant toujours l'entièreté d'une page d'écran. Ainsi, lorsqu'on saisit quatre équations de variables d'état, on afficherait directement :

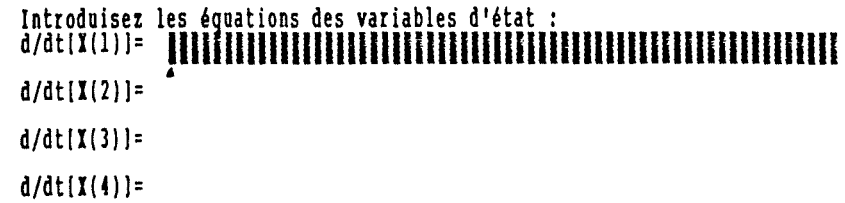


figure 8.1

alors que pour le moment, on n'affiche cela que ligne par ligne, au fur et à mesure que l'enseignant avance dans la définition du système. Un essai en ce sens avait été tenté, mais s'est une fois de plus heurté à la fameuse limite... de 64K.

Pour améliorer la rapidité d'exécution, il serait utile de bufferiser la sortie numérique. Actuellement, toutes les valeurs sont écrites une à une dans le fichier. Cette sortie numérique vers fichier pourrait être doublée d'une sortie à l'écran, au fur et à mesure (scrolling du resultat) s'il n'y a pas de sortie graphique, ou en bloc à la fin dans le cas contraire. L'affichage se ferait alors page par page.

Enfin, même s'il est tout à fait impossible de modifier les expressions d'un système lors de son exécution, il serait bon de pouvoir visualiser celles-ci. Que l'étudiant puisse "lire" le système défini par son professeur (ProfCOMP), ou le scientifique vérifier au beau milieu de la simulation quelles sont les expressions qu'il a écrites (SimEDO), voilà qui serait sans nul doute un plus pour nos logiciels.

PC_EXEC

Comme nous l'avons déjà signalé, il est difficile pour l'enseignant de s'imaginer ce que va donner sa répartition des fenêtres. On pourrait donc permettre à l'étudiant de la corriger, en lui donnant toutes les primitives de manipulation offertes à son professeur. Et pourquoi pas, lui permettre de revenir en arrière dans le cours, en "pointant" la fenêtre à laquelle il veut retourner ?

Dernière observation : pendant l'exécution d'une leçon, si le noeud suivant n'existe pas, un message en informe l'étudiant, et on passe directement à la suite. Cela peut être gênant, surtout dans le cas d'un système où les données figurent sur disquette. Une amélioration serait, dans le cas d'un système sans disque dur, de proposer à l'étudiant de mettre la bonne disquette dans le drive. Si on ne trouve toujours pas le bon noeud, alors seulement on passerait à la suite.

BIBLIOGRAPHIE

[Bar&Yat] Barker and Yates
Introducing Computer Assisted Learning,
Prentice-Hall International

[Simon] Jean-Claude Simon
L'éducation et l'informatisation de la société,
Rapport du Président de la République,
La Documentation Française, 1980

[Péd] L'Informatique à l'école Une voie à suivre ?,
Documents Pédagogiques,
Ministère de l'Education Nationale, 1984

[Nie&cie] J. Nievergeld, A. Ventura and H. Hinterberger
Interactive Computer Programs for Education,
Addison-Wesley, Reading, Mass.

[Kears] G. Kearsley
"Authoring Systems in Computer Based Education",
Communications of the ACM,
July 1982 Volume 25 Number 7

[Ramb] G.K. Rambally and R.S. Rambally
"Human factors in CAI design",
Comput. Educ. Vol. 11, N°2, pp 149-153, 1987

[Faraz] Z. Farazinc
"Computers in support of conceptual learning",
HPSA, Geneva

[Boyd] G.M. Boyd
"Four ways of providing Computer Assisted Learning and their probable impacts",
Comput. Educ. Vol. 6, pp 305-310, 1982

[EA01] "Une vue générale de l'application de l'informatique dans l'enseignement",
Informatique et Enseignement N°3

[EA02] "Enseignement et Informatique : des points de vue",
Informatique et Enseignement N°5

[Freed] R.S. Freedman, Polytechnic University
"Expert Systems that teach",
IEEE Expert, Summer 1987

[SDA] S.D. Antunes
Modelizaçao, Simulaçao e controlo de um processo biológico,
Ph D dissertation,
Universidade Técnica de Lisboa - IST - Portugal

[Cunnin] W.J. Cunningham
Introduction to NonLinear Analysis,
Mc Graw-Hill Book Company, 1958

[Murphy] G.J. Murphy
Control Engeneering,
The Van Nostrand Series in Electronics and Communications 1959

[Cra&Dor] D. Mc Cracken and W.S. Dorn
Numerical Methods and Fortran Programming,
John Wiley and Sons, 1965

[Conte] S.D. Conte
Elementary Numerical Analysis,
Mc Graw-Hill, 1965

[Baccio] A. Bacciotti, P. Boieri and P. Moroni
"A computer approach to nonlinear planar systems of ODE's",
Comput. Educ. Vol. 11, N°4, pp 253-265, 1987

[GTLK] Gillis, Thomason, Lefèvre and Kretsinger
"Pavalbumins and muscle relaxation :
a computer simulation study",
Journal of Muscle Research and Cell Mobility, 1982

[Leibl] M.D. Leiblum
"A model for describing CAL Authoring Systems applied to TAIGA",
Comput. Educ. Vol. 12, N°1, pp 141-149, 1988

[LFBN1] J.Lefèvre, R.Fabri, J.Barreto,M.Noirhomme-Fraiture
"The coupling of simulation, CAI and expert systems : a prospective study"

[LFBN2] J.Lefèvre, R.Fabri, J.Barreto,M.Noirhomme-Fraiture "An explanatory module for an ICAI-system using simulation of lumped models"

[JBJLMN] J. Barreto, J. Lefèvre, M.Noirhomme-Fraiture "Simulation and AI in Computer Assisted Learning", Informaticasecusu'86 Rio De Janeiro

[Pilot] "Pilot, un langage auteur aux aspects multiples", *Informatique et Enseignement N°5*

[Tutor] "Private Tutor : système auteur pour l'EAO", *Informatique et Enseignement*, 20/1/87

[SEF] "Un nouveau système auteur : Self Education Facility", *Informatique et Enseignement N°4*

[Madau] Madaule, Barril, De La Passardière, Le Calvez, Poc et Urtasun "Systèmes d'EAO : étude comparative", *Technique et science informatiques* , Vol.6, N°1, 1987

[Athéna] J. Solvay "L'Intelligence artificielle à l'aide de l'EAO", *Revue Athéna*, mai-juin 1988

LISTE DES FIGURES

Facultés Universitaires Notre-Dame de la Paix - Namur
Institut d'informatique

Etude et réalisation d'un système auteur avec simulateur de systèmes d'équations différentielles intégré :

ProfCOMP et SimEDO

ANNEXES

Mémoire présenté pour
l'obtention du titre de
licencié et maître en informatique

par Claude Léonard et
Olivier Marchand

Promoteur : Madame Monique Noirhomme-Fraiture
Directeur : Monsieur Jorge Muniz Barreto

Année académique 1987 - 1988

Contenu des annexes

Annexe A : Manuel d'utilisation de ProfCOMP

Annexe B : Manuel d'utilisation de SimEDO

Annexe C : SimEDO : An ordinary differential equations simulator, papier présenté au congrès de la Société Portugaise de physique Coimbra (septembre 88)

Annexe D : ProfCOMP Integrated author-system/simulation package, papier présenté au Melecon'89 (Avril 1989)

Annexe E : Historique d'un noeud simulation

Mea culpa !!!

Les manuels de ProfCOMP et SimEDO qui figurent dans ces annexes contiennent eux-mêmes des annexes. Nous nous excusons de ce procédé récursif peu élégant. Quand nous référençons une annexe dans le mémoire, il s'agit du premier niveau.

ANNEXE A

Le manuel de

ProfCOMP

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

INSTITUTO SUPERIOR TÉCNICO
1096 LISBOA CODEX

Manuel de

ProfCOMP

Système auteur - Professor COMPatible

S.D. Antunes, Instituto Superior Técnico,
Av. Rovisco Pais, 1096 Lisboa Codex
PORTUGAL

J.M. Barreto, C. Léonard et O. Marchand,
FNDP - Institut d'Informatique,
Rue Grandgagnage 21, 5000 Namur, BELGIQUE

Namur 1988

TABLE DES MATIERES

1. INTRODUCTION ET PHILOSOPHIE GENERALE

Bienvenue à vous, cher enseignant ou assimilé, le logiciel que vous découvrez maintenant a été élaboré par des étudiants, n'entendant pas grand chose à l'art de l'enseignement. Si vous pensez qu'il faut garder à chacun ses spécialités mais que, malgré tout, l'informatique peut être dans certains cas un auxiliaire intéressant pour vous, nous trouverons certainement dans ce logiciel un terrain d'entente. Si, par contre, vous pensez à l'instar d'Alain que l'enseignement doit être résolument retardataire, notre logiciel possède une merveilleuse fonction de "sortie" qui sûrement vous intéressera... Loin de nous l'intention de vous prendre à tout prix dans nos filets ! Tout le monde sait bien que "l'enseignement de l'araignée n'est pas pour la mouche" ...(Henri Michaux).

NE JAMAIS SE LAISSER IMPRESSIONNER...

Ce que nous vous proposons ici est un ensemble d'outils intégrés vous permettant d'élaborer le plus rapidement et le plus aisément possible, une leçon dite "Assistée par Ordinateur". Certains de ces outils (Simulation) sont spécialement destinés à des personnes possédant quelques connaissances en mathématique tandis que les autres, plus classiques, sont accessibles à tous et ne nécessitent aucun prérequis.

Si dans l'expression "Enseignement Assisté par Ordinateur" (EAO) c'est le mot "Ordinateur" qui retient le plus votre attention, il est grand temps de vous réveiller de ce rêve (ou ce cauchemar) d'un enseignement automatisé, réduisant le rôle de l'enseignant à celui d'un simple opérateur. Nous nous permettrons d'insister plutôt sur le mot "Assisté", qui dit bien ce qu'il veut dire...

Comme vous le comprendrez rapidement, une telle leçon vous demandera bien plus de préparation qu'une leçon classique, même si nous avons eu à coeur de réduire ce temps de préparation, en vous mâchant le travail au maximum !

A vous dès lors, chers enseignants, de discriminer les cas où pareil enseignement se révèle intéressant, utile ou ... irremplaçable, des cas où une leçon d'EAO serait une

aberration pédagogique ! En ce qui nous concerne, nous ne rentrerons pas davantage dans le débat.

Si vous êtes prêts et si la chimère que vous avez devant vous et qui tient à la fois de la télévision, de la machine à écrire et du "tourne-disque"ne vous fait pas trop peur, entrons dans le vif du sujet !

LA PHILOSOPHIE GENERALE...

Assumant notre faible connaissance du phénomène de l'enseignement, nous nous sommes toutefois efforcés de satisfaire un certain nombre d'objectifs généraux qui sont autant de facilités et d'agréments toujours appréciés d'un utilisateur.

Avant tout la TRANSPARENCE nous a préoccupés. Celle-ci vous permet, en cours d'élaboration de la leçon, de voir ce que l'étudiant percevra effectivement lors de son exécution. Ceci afin d'éviter les mauvaises surprises, qui imposent de lourdes modifications, au moment où l'on croit le travail enfin terminé.

Le second objectif a été de limiter le nombre de données que vous devez entrer, en prévoyant un système de valeurs par défaut, comblant tout manque d'imagination. Dans la mesure du possible, nous avons préconisé un système de choix d'options par flèches et touches de fonction, vous évitant ainsi l'utilisation du clavier, si vous n'êtes pas un fanatique de la machine à écrire...

Le troisième, c'est la prise en compte des nombreuses erreurs d'utilisation. Bien que nous ayons veillé à une homogénéité au niveau de la signification des touches de commande, vous pouvez très bien enfoncer une touche ne correspondant pas à l'option que vous désiriez vraiment. A de tels instants, vous apprécierez beaucoup de pouvoir "revenir en arrière" ! Les erreurs fatales et les messages d'injure qui s'ensuivent toujours dans de telles situations, seront donc en nombre limité...

Si vous disposez d'une version de ce logiciel dans votre langue maternelle, vous ne serez peut-être pas très sensible au quatrième : la facilité déconcertante avec laquelle tout message peut être traduit.

2. PRESENTATION GENERALE DES OUTILS

Avant de vous expliquer comment réaliser pratiquement une leçon, il est bon de vous présenter les collègues qui vous assisteront :

- **Un EDITEUR DE TEXTE** vous permet de taper un texte quelconque, en utilisant, suivant votre curiosité, les fonctions classiques d'une machine à écrire ou des fonctions beaucoup plus puissantes. Dans un second temps, vous pourrez l'enregistrer, ainsi que lui apporter toutes les modifications que vous désirez.

- **Un FORMATEUR DE TEXTE** associé à cet éditeur s'occupera de la mise en page de votre texte, et ce dans le cadre d'une fenêtre dont vous définirez la taille.

- **Un EDITEUR DE QUESTIONNAIRES** vous permet de construire un questionnaire comprennant des "Questions à Choix Multiples" ou des "Questions Ouvertes". Toute la gestion associée à de tels questionnaires se fera automatiquement. Cet outil vous offre une possibilité de tester les connaissances de base ou la compréhension d'un étudiant, au début ou en cours de leçon, et d'adapter en conséquence la suite de votre enseignement.

- **Un EDITEUR GRAPHIQUE** vous prouvera à vous-mêmes que vous êtes doués pour le dessin. Quand vous dessinerez dans une fenêtre (dont vous aurez préalablement déterminé la taille), vous disposerez d'un crayon, d'un compas, d'une règle, d'une gomme,...

- **L'outil de CREATION D'UNE SIMULATION** vous permet de simuler des systèmes d'équations différentielles ordinaires du premier ordre et de laisser la possibilité à l'étudiant de créer ses propres modèles.

3. COMMENT CREER UNE LECON ?

Vous devez avant tout mettre très clairement sur papier le plan de la leçon. Ce plan ne sera pas une suite d'opérations à effectuer les unes à la suite des autres, car une leçon se doit d'être adaptée à l'élève... (c'est là sa qualité principale !). On représentera plutôt ce plan sous forme d'un graphe, chacun des noeuds de ce graphe pouvant

être réalisé à l'aide d'un des outils offerts.

Les composantes de la leçon seront dès lors des NOEUDS TEXTE, GRAPHIQUE, ANIMATION, QUESTIONS-REPONSES et SIMULATION.

Le passage d'un noeud à un autre se fera, soit de façon inconditionnelle, soit en fonction d'un "score" (à préciser) obtenu lors d'un questionnaire. On utilisera pour cela un dernier outil que nous n'avions pas encore présenté : l'outil de CREATION D'UN ENCHAINEMENT.

Prenons un exemple : pour débuter votre leçon, vous commencez par un aperçu de la matière vue précédemment. On peut supposer que ce bref rappel soit obligatoirement dispensé à l'élève, quel que soit son niveau (pour ce faire l'enseignant créera un noeud texte).

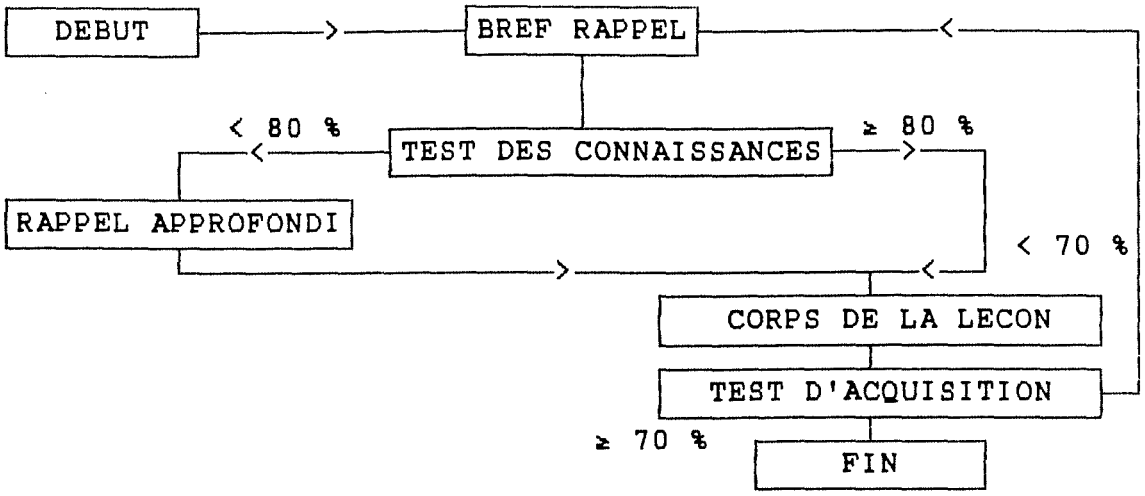
Avant de continuer plus loin la leçon, vous voulez être sûr que l'étudiant possède la base nécessaire. Pour satisfaire cet objectif, vous préparez un petit questionnaire (ce qui correspond à la création d'un noeud questions-réponses).

Si les points sont supérieurs à 80%, on peut passer à la suite des opérations , à savoir :

- exposé de la nouvelle matière au moyen de textes, de graphiques et d'animations (création de noeuds texte et de noeuds graphique).
- possibilité pour l'étudiant d'effectuer une simulation après avoir vu celle du professeur (création d'un noeud SIMULATION).
- petit texte récapitulatif (création d'un noeud texte).
- petit questionnaire pour tester la compréhension (création d'un noeud questions-réponses).

A l'issue de ce dernier questionnaire, on verra si l'étudiant ne doit pas "subir" une seconde fois la leçon...

Si les points sont inférieurs à 80% ,un rappel beaucoup plus détaillé de la matière précédente sera fourni à l'élève, avant de passer à la suite (et ce, éventuellement, tant que ses points ne seront pas satisfaisants).



Ce qui, sur le graphe est intitulé "corps de la leçon" correspond en fait à un sous-graphe.

On remarque entre "bref rappel" et "test des connaissances" un enchaînement inconditionnel; entre "test des connaissances" et "rappel approfondi", un enchaînement conditionnel portant sur les points obtenus lors de "test des connaissances".

A vous maintenant d'alterner le texte et le graphisme,
les simulations et les questionnaires,
d'ajouter une pincée d'imagination
et surtout,
de ne jamais oublier l'humour...
qui fait avaler tant de choses !

4. ECRANS ET FENETRES ...

Par la suite, ce sont deux vocables que nous allons souvent utiliser et il est bon de définir clairement ces deux concepts avant d'aller plus loin.

Devant vous, un écran, que nous appelerons "écran physique". Sa surface étant limitée, nous allons "faire comme si" celui-ci avait une troisième dimension, correspondant à une suite d'écrans "logiques".

Pour bien comprendre le mécanisme des fenêtres, considérons ensemble l'exemple suivant :

Vous êtes assis à votre bureau et vous avez devant vous une surface de travail. Sur celle-ci sont empilés un certain nombre de "papiers" de différentes tailles que nous désignerons par le terme général de FEUILLETS. Chacun de ces feuillets est une surface de travail dans la mesure où il constitue un support de lecture et/ou d'écriture.

A un instant donné, vous ne pouvez voir qu'un seul feuillet, celui du dessus du paquet, à moins que deux (ou plusieurs) feuillets disjoints n'occupent le haut du tas ! Pour voir un feuillet particulier, il faut "dépiler" jusqu'à ce que celui-ci occupe le dessus de la pile.

Pour nous, la surface de travail, c'est l'écran, et les feuillets, ce sont les fenêtres (de petits écrans donc...). De la même façon que superposer un feuillet sur un autre n'efface pas ce dernier, superposer une fenêtre sur une autre empêche seulement momentanément, de voir l'entièreté de cette fenêtre. (La superposition peut être partielle ou totale, comme sur un tas de papiers !)

Comme vous le comprendrez vite, à l'exécution de la leçon, il sera souvent intéressant de travailler avec plusieurs fenêtres, l'une contenant un graphique, l'autre un texte explicatif, une troisième exécutant un questionnaire, etc...

C'est pourquoi la création d'un noeud (quel que soit son type) sera toujours réalisée dans le cadre d'une fenêtre, dont vous déterminerez les dimensions en début de

création. La position de la fenêtre quant à elle sera définie lors de l'ENCHAINEMENT (cfr infra)

1 NOEUD <---> 1 FENETRE

Si vous désirez un affichage simultané de plusieurs fenêtres, à vous d'en spécifier les dimensions de façon à ce qu'il soit possible de les arranger à l'écran sans trop de recouvrement, voire, si possible de façon disjointe.

5. L'INTERFACE ProfCOMP - UTILISATEUR

L'introduction d'un entier, d'un réel ou d'une chaîne de caractères se présente toujours de la même manière. Le logiciel a standardisé toutes ces opérations.

Un message renseigne la donnée à définir puis s'affiche un rectangle en vidéo inversé. Si une valeur par défaut doit être mentionnée pour la donnée, celle-ci s'affiche dans la partie gauche de ce rectangle.

Exemple :

Donnée: valeur par défaut

Si aucune valeur par défaut n'est associée à "donnée", on aura l'affichage :

Donnée:

La longueur du rectangle inversé détermine le nombre de caractères que l'utilisateur peut encore taper avant d'atteindre la longueur maximale imposée par le logiciel. Un triangle pointé vers le haut symbolise le curseur en mode "surfrappe" tandis qu'un pique le représente en mode insertion. Quand l'utilisateur doit introduire un nombre entier ou réel, le curseur se positionne sur le premier caractère de la valeur par défaut (si elle existe). Dans tous les autres cas, le curseur se place sur la première position à la gauche du rectangle.

Chaque fois que l'utilisateur frappe un caractère, celui-ci s'inscrit à l'endroit désigné par le curseur, le rectangle se rétrécit de la longueur du caractère et le curseur passe à la position suivante.

Les flèches GAUCHE et DROITE permettent le déplacement du curseur dans les caractères introduits (ou dans la valeur

par défaut). Si le triangle pointe un caractère et que l'utilisateur enfonce la touche d'un autre caractère, celui-ci remplacera l'ancien : c'est le mode surfrappe ("overwrite"). La même opération en mode insertion déplacera tous les caractères, à partir de la position courante du curseur, d'une position vers la droite et insèrera le caractère frappé à l'endroit du curseur. Enfin, le curseur avancera d'une position vers la droite.

Le passage du mode insertion à la surfrappe (et réciproquement) se réalise par pression de la touche **INS**.

La touche **DEL** permet d'effacer le caractère sous le curseur tandis que la touche |<- efface le caractère à gauche du curseur. Dans les 2 cas d'effacement, le rectangle en vidéo inversé s'aggrandit d'une position à gauche puisqu'il récupère la place du caractère effacé.

Un bip sonore signale toute erreur de manipulation.

Certaines touches du clavier acquièrent une signification particulière lors de la définition de données. Voici, en résumé, la liste des touches dont il faut connaître la signification pour entrer les données le plus efficacement possible :

INS : passer du mode insertion à la surfrappe et réciproquement,

DEL : efface le caractère au-dessus du curseur,

|<- : efface le caractère à gauche du curseur,

->,<- : déplacement du curseur dans les caractères introduits,

F1 : un pas en arrière dans la définition des données, c'est-à-dire que l'utilisateur revient à la définition de la dernière donnée introduite,

F2 : abandon de l'introduction des données et retour au menu principal,

F5 : affiche le répertoire des noeuds concernés par l'outil utilisé.

Remarque : l'effet d'une touche spécifique peut être inhibé à certains moments. Par exemple, la touche **F5** n'a d'effet que lorsqu'on définit le nom d'un noeud.

Maintenant que la démarche générale commence tout doucement à vous être familière, nous allons vous présenter chacun des outils, de façon détaillée, avec leurs objectifs, leurs limites et... leurs commandes.

6. PRESENTATION DETAILLEE DES OUTILS (PC_CREAT)

6.1. Le système de menu

Après une page de présentation, apparaîtra le premier menu : LE MENU PRINCIPAL . Celui-ci vous propose différentes options.

6.1.1. Comment sélectionner une des options ?

Par défaut, c'est toujours la dernière option qui est choisie (elle apparaît dans une couleur différente). Pour sélectionner une autre option, on peut :

- utiliser les flèches BAS et HAUT
- préciser directement l'option choisie par son rang : on utilisera alors le chiffre ou la touche de fonction correspondant à ce choix. (**1** ou **F1** pour la première option, **2** ou **F2** pour la seconde, etc...)
- également préciser la première option en pressant la touche **HOME** et la dernière en pressant la touche **END**.

Comme vous le verrez très vite, utiliser la flèche HAUT sur la première option provoque un saut vers la dernière et utiliser la flèche BAS sur la dernière provoque un saut vers la première, comme si les options formaient un cercle.

6.1.2. Comment fixer votre choix ?

En pressant la touche <—┘, ou la touche étiquetée **ENTER** sur le pavé numérique.

6.2. Choix des couleurs

Il s'agit ici de choisir les couleurs dans lesquelles vous allez travailler et non des couleurs dans lesquelles la leçon sera exécutée (changement définitif).

L'écran affichera toutes les combinaisons possibles de couleur et d'arrière-fond et un numéro identifiera chaque possibilité. Vous devrez choisir successivement deux couleurs pour le texte, une pour les cadres, une pour les commandes, une pour les erreurs (qu'on souhaite voir apparaître rarement), et la dernière pour le mode graphique. A vous d'introduire les combinaisons idéales (couleur, arrière-fond) pour chacune de ces couleurs. Si vous ne désirez pas modifier une couleur existante, réintroduisez son identifiant. Attention pour la couleur en mode graphique : si vous ne disposez pas d'une carte graphique EGA, l'arrière-fond restera noir.

6.3. Edition d'un texte (PC EDIT)

Nous n'avons pas l'intention d'expliquer ici, en quelques lignes, le fonctionnement d'un éditeur de texte. Ce serait en effet une entreprise vouée à l'échec. L'éditeur utilisé ici est un sous-ensemble de l'éditeur MICROSTAR fourni dans le package "Turbo Editor Toolbox" par Borland International Corporation. Bornons-nous ici à décrire quelques particularités ainsi qu'à donner une liste des commandes utilisables. Globalement, on peut dire que les utilisateurs du traitement de texte WORDSTAR ne seront pas dépaysés. (WordStar est une marque déposée de MICROPRO INT. CORP.)

6.3.1. Menus déroulants et commandes au clavier

Un premier message vous demande le nom du fichier à éditer. Si vous voulez modifier un fichier, introduisez son nom. Si vous créez un nouveau fichier, vous pouvez définir son nom dès le début ou au moment du sauvetage. Une fois entré dans l'éditeur, on peut lire le message "F10 for pulldown menu". Si vous pressez **F10**, vous pouvez alors vous déplacer grâce aux flèches GAUCHE/DROITE sur la "barre de menus" afin de choisir le menu qui vous intéresse. Une fois celui-ci choisi, vous affinerez votre choix en faisant

descendre un petit menu (<—┘). Vous choisirez une option (avec les flèches BAS et HAUT) et vous fixerez votre choix (<—┘). Une ou deux pressions sur **ESC**, suivant le niveau du menu où vous vous trouvez, vous feront sortir de ces menus déroulants.

Cette façon de procéder, très intéressante pour l'utilisateur néophyte, peut se révéler extrêmement lourde pour l'utilisateur chevronné, c'est pourquoi, parallèlement au système de "pulldown menu", coexiste un système de commandes au clavier.
Remarque : l'éditeur peut se trouver sur un répertoire différente de celui qui contient le programme de création (voir 8.1. l'utilitaire INST_EAO).

6.3.2. Commandes de l'éditeur

Pour ces quelques commandes, ^a signifie 'Control A' et s'obtient en pressant la touche **CTRL** et en la maintenant enfoncée pendant que vous pressez la touche a.

* **Déplacement du curseur :**
  caractère : ^S gauche ^D droite (ou flèches gauche-droite)
  mot : ^A gauche ^F droite
  (ou **CTRL** flèches gauche-droite)
  ligne : ^E monte ^X descend (ou flèches haut - bas)
  jusqu'à la colonne spécifiée : ^O^I
  jusqu'à la ligne spécifiée : ^O^N
* **Défilement vertical**
  ligne : ^Z monte ^W descend
  écran : ^C monte ^R descend (ou **PgUp - PgDn**)
* **Nouvelle ligne : RETURN**
* **Insérer une ligne sans déplacer le curseur** : ^N
* **Effacement :**
  caractère : ^G ou **DEL**
  mot : ^T
  ligne : ^Y
  caractère à gauche : **<-DEL**
  jusqu'à la fin de la ligne : ^Q^Y
* **Mode Insertion ON/OFF** : ^V (ou **INS**)
* **Défaire le dernier changement (UNDO)** : **ESC**
* **Etablir le nombre maximum de UNDO (ESC)** : ^O^S
* **Passage des minuscules aux majuscules** : ^O^K

* **Activer les "pulldown" menus : F10**
* **Aide : ^Q^H**
* **Insérer un caractère par son code ASCII : ^P**
  (équivaut au passage par le menu impression dans WordStar)
* **Codes rapides :**
  à gauche de la ligne : ^Q^S (ou **HOME**)
  à droite de la ligne : ^Q^D (ou **END**)
  en début de fichier : ^Q^R
  en fin de fichier : ^Q^C
  en début d'un boc : ^Q^B
  en fin d'un bloc : ^Q^K
  auto indentation mode : ^Q^I
  début/fin de ligne : ^J
* **Rechercher une occurence de ... : ^Q^F**
  Options : u ignore les majuscules et minuscules
  w ne considère que les occurrences entières
  b effectue la recherche vers le début du fichier
* **Rechercher + Remplacer par ... : ^Q^A**
  Mêmes options que la commande de recherche avec en plus :
  g recherche et remplace dans tout le fichier
  n remplace automatiquement sans demander confirmation
  x, où x est un entier représentant le nombre de recherches-remplacements à effectuer
* **Rechercher l'occurence suivante : ^L**
* **Commandes de fin**
  sauver et ouvrir un nouveau fichier : ^K^D
  sauver et continuer modifications : ^K^S
  abandonner un fichier et en ouvrir un nouveau : ^K^Q
  retour à PC_CREAT : ^K^X
* **Marqueurs**
  poser le marqueur 1 : ^K 1
  poser le marqueur 2 : ^K 2
  ...
  poser le marqueur 9 : ^K 9
  poser un marqueur : ^K^M
  sauter vers le marqueur 1 : ^Q 1
  sauter vers le marqueur 2 : ^Q 2
  ...
  sauter vers le marqueur 9 : ^Q 9
  sauter vers le marqueur : ^Q^J

*** Blocs**

Début de bloc : ^K^B
Fin d'un bloc : ^K^K
Cacher les repères : ^K^H
Copier un bloc : ^K^C
Déplacer un bloc : ^K^V
Détruire un bloc : ^K^Y
Ecrire un bloc : ^K^W
Lire un bloc : ^K^R

*** Tabulations**

Définir la largeur de tabulation : ^K^T
Tabulation suivante : ^I (ou **TAB**)

* Certaines commandes ne sont accessibles que via les menus. C'est le cas notamment de l'aide concernant le formatage d'un texte (cfr point 6.4), de certaines options sur fichiers (dont l'affichage du répertoire, la restauration d'un fichier .BAK, renommer un fichier, ...). Une de ces options mérite un peu plus d'explications, l'impression d'un fichier (option Print du menu File).

*** Print** : Il ne s'agit pas uniquement d'imprimer un fichier quelconque. En effet, l'éditeur permet d'insérer dans le texte des caractères de contrôle (par ^p, suivi d'un caractère; par exemple ^pa pour insérer ^a dans le texte). Ces caractères de contrôle seront transformés en code pour l'imprimante, afin d'utiliser pleinement toutes les possibilités de celle-ci. Voici toutes les options prévues, ainsi que la séquence réellement envoyée à l'imprimante :

^a : sélectionne le mode condensé (SI ou ^o)
^b : sélectionne ou relâche le mode gras (ESC E - ESC F)
^c : sélectionne ou relâche la qualité courrier
(ESC X 1 - ESC X 0)
^d : sélectionne ou relâche la double impression
(ESC G - ESC H)
^h : recule d'un caractère la tête d'impression (BS ou ^h)
^i : sélectionne ou relâche le mode italique
(si l'imprimante le permet) (ESC 4 - ESC 5)
^l : avance en début de page suivante (FF, FormFeed)
^n : relâche le mode condensé (DC2 ou ^r)
^s : sélectionne ou relâche le mode souligné
(ESC '-' 1 - ESC '-' 0)

^t : sélectionne ou relâche le mode indices supérieurs
(ESC S 0 - ESC T)
^v : sélectionne ou relâche le mode indices inférieurs
(ESC S 1 - ESC T)
^w : sélectionne ou relâche le mode double largeur
(ESC W 1 - ESC W 0)

Tous ces caractères sont 'éliminés' par le formatage expliqué ci-dessous, c'est-à-dire qu'ils n'apparaissent pas lors des exécutions de leçons.

6.4. Formatage d'un texte

L'éditeur est déjà en soi un outil très puissant, mais quand un formateur de texte lui est associé, de nouveaux horizons s'ouvrent...

En fait, le formatage d'un texte est une opération obligatoire ici, dans la mesure ou le texte créé doit pouvoir défiler dans une fenêtre de taille quelconque ! A partir de cet instant, le nombre de caractères par ligne ou le nombre de lignes par page n'est plus quelque chose d'absolu !

Notre formateur de texte permet de règler un certain nombre de problèmes tels
- Le choix de la couleur du texte à l'exécution de la leçon
- La numérotation des pages
- La vitesse de défilement
- Les sauts de page
- La justification gauche-droite, ...

ET SURTOUT ! La mise en forme d'un texte édité avec PC_EDIT dans une fenêtre de taille quelconque !

Le nom du noeud et la taille de la fenêtre vous seront demandés avant le formatage.

SEUL UN TEXTE FORMATE AVEC NOTRE FORMATEUR POURRA CONSTITUER UN NOEUD TEXTE!

6.4.1. Comment utiliser le formateur ?

En insérant, lors de l'édition du texte, des commandes en début de ligne destinées uniquement à être interprétées par le formateur, et qui n'apparaîtront pas à l'écran, une fois le texte traité.

6.4.2. Règles :

- Pour qu'une commande soit reconnue, il faut qu'elle figure obligatoirement en début de paragraphe.
- Toute commande commence par le caractère "^" (accent circonflexe) suivi du nom de la commande (3 caractères maximum). Pour certaines commandes on ajoutera à la suite le caractère '.' (point) suivi d'un nombre entier (l'argument de la commande).
- Toute commande doit être obligatoirement suivie d'un blanc.

Exemple de commande sans argument : ^j␣ (␣ symbolisant un blanc)

Exemple de commande avec argument : ^d.3␣

- Une commande prend toujours effet à partir de (et y compris) la ligne où elle figure.
- Certaines commandes vont par deux. Ainsi, si on veut ne faire la justification gauche-droite que pour une partie du texte, la ligne de commande :^j␣ devra être insérée en début, et la ligne de commande : ^fj␣ en fin de cette partie de texte.

6.4.3. Commandes du formateur de texte

(␣ symbolise un blanc et NOMBRE représente l'argument entier d'un chiffre de la commande)

* ^c.NOMBRE␣ : change la couleur du texte jusqu'à rencontre de ^fc␣ en lui attribuant la couleur de n° NOMBRE (1 à 127), l'argument ayant la même signification que lors du choix de la couleur graphique (cfr 8.1 INST_EAO).
* ^fc␣ : fin de la partie du texte à écrire dans la couleur spécifiée dans la commande précédente ^c.NOMBRE.
* ^n.NOMBRE␣ : numérote les pages à partir de cet endroit, en commençant la numérotation à NOMBRE. (La numérotation se fait par défaut si l'utilisateur ne précise rien, et commence à la page 1)
* ^fn␣ : arrête la numérotation des pages.
* ^p␣ : provoque un saut de page.
* ^d.NOMBRE␣ : commande qui sert lors de l'exécution de la leçon et qui fixe le délai d'affichage pour le défilement des pages.

délai ≈ NOMBRE * 1 seconde.

* ^j▒ : justification gauche-droite du texte (celle-ci se fait par défaut si l'utilisateur ne précise rien).
La justification gauche-droite consiste à aligner sur une même colonne tous les premiers caractères de chaque ligne, et sur une même colonne, tous les derniers caractères de chaque ligne.
Exemple : Le texte que vous lisez maintenant n'est justifié qu'à gauche tandis que la majeure partie du texte restant est justifié à gauche et à droite.
Remarque : la dernière ligne non vide d'un paragraphe sera alignée à gauche et pas à droite.
* ^fj▒ :termine la justification du texte.
* ^l▒ : laisser le texte tel quel jusqu'à rencontre de ^fl, sauf si la longueur de la ligne est trop grande. Elle est alors coupée et une justification gauche-droite est opérée (sauf si ^fj est précisé). C'est la fonction littéral
* ^fl▒ : stoppe l'effet de la commande ^l▒.
* ^mg.NOMBRE▒ : établit la marge gauche à NOMBRE : chaque ligne du texte formaté commencera par NOMBRE-1 blancs.
* ^md.NOMBRE▒ : établit la marge droite à NOMBRE : chaque ligne se termine par au moins NOMBRE - 1 blancs. La signification de la marge droite est donc symétrique à celle de la marge gauche.
* ^m▒ : chaque ligne est centrée entre les marges, jusqu'à la commande ^fm.
* ^fm▒ : fin de la commande de centrage des lignes.
* ^jd▒ : le texte sera justifié à droite : les derniers caractères de chaque ligne seront alignés sur la colonne marge droite.
* ^fjd▒ : termine la justification droite du texte.
* ^a▒ : définit un alinéa en début de paragraphe en décallant de quelques positions, par rapport à la colonne marge gauche, la première lettre de ce paragraphe.
* ^fa▒ : fin d'alinéa. Les débuts de paragraphe commenceront sur la colonne marge gauche.

6.4.4. Fichier des erreurs

A l'issue du formatage, la question vous sera posée de consulter ou non le fichier des erreurs, s'il y en a. Celui-ci reprend pour chaque erreur, la ligne où elle est intervenue ainsi qu'une description de l'erreur. Le vocable

"erreur" est assez maladroit, "avertissement" serait préférable dans la mesure où il s'agit davantage de vous informer de certaines incohérences rencontrées lors du formatage.

6.4.5. Affichage du texte formaté

Le résultat du formatage vous sera proposé dans une fenêtre identique à celle que l'étudiant percevra au moment de l'exécution de la leçon, seule sa position sera différente. Vous pourrez faire défiler les pages pour voir si l'opération de formatage vous satisfait. Si vous désirez changer la taille de la fenêtre, reformatez ce texte (**F1**). Si d'autres modifications sont nécessaires, retournez dans l'éditeur pour les accomplir.

6.5. Création de graphiques

Comme le dit la sagesse populaire : bien souvent, "un petit dessin vaut mieux qu'un grand discours". L'utilité de petits dessins parlants appuyant un texte, l'expliquant ou le rendant moins rébarbatif nous a semblée évidente. L'outil que nous vous fournissons ici allie simplicité et puissance.

6.5.1. Création d'une animation

Cet éditeur graphique permet de créer des noeuds graphiques mais aussi les composants d'une animation. Comment constituer une animation ? En affichant successivement plusieurs noeuds graphiques de même dimension. C'est en quelque sorte la technique des dessins animés mais le logiciel ne permet qu'un maximum de cinq graphiques par animation. Vous spécifierez le délai d'affichage de chaque planche et le nombre de graphiques qui composent l'animation à la définition de l'enchaînement des noeuds. La technique de mise au point des planches d'une animation est la suivante : créez la première planche, sauvez-la avec un nom dont le dernier caractère est "1". Repartez de cette planche (c'est le seul moyen de garantir que tous les composants auront la même dimension), apportez-y les modifications nécessaires pour qu'elle constitue la planche 2 et sauvez-la avec le même nom mais où vous aurez remplacé le dernier caractère "1" par "2". Procédez de même pour toutes les autres en repartant chaque fois de la dernière planche

terminée.

6.5.2. Commandes

La première information qu'on vous demande d'introduire est le nom d'un dessin. Si vous voulez modifier un dessin, tapez son nom; sinon, tapez **RETURN** pour signaler qu'il s'agit d'une nouvelle création. Dans ce dernier cas, il vous sera demandé de régler la taille de la fenêtre dans laquelle vous désirez présenter vos graphiques. Pour celà vous utiliserez les flèches et vous presserez **RETURN** pour confirmer. La fenêtre graphique apparaîtra ensuite à l'écran. Le curseur se présentera sous la forme d'une flèche orientée vers le haut, indiquant le point courant.

Liste des commandes :

A élabore une figure fermée. La première pression de A mémorise la position du curseur. Chaque pression ultérieure de A tracera une droite entre la position courante du curseur et l'endroit où vous avez frappé A l'avant-dernière fois. Si vous frappez **ENTER** ou **F2**, la commande se termine par la fermeture de la figure : deux droites sont dessinées; La première relie le point où vous avez frappé A la dernière fois et la position courante du curseur, la seconde relie la position courante du curseur à l'endroit où vous avez débuté la commande (*).

B déplace le curseur sur le bord gauche du graphique (Backward).

C reproduit une circonférence. La position du curseur correspond au centre de cette circonférence. Déplacez le curseur horizontalement à l'extrémité du rayon et pressez la touche C pour dessiner la circonférence (*) (Circle).

D positionne le curseur en bas de la fenêtre (Down).

E simule l'effet d'une gomme. Le curseur disparaît pour ne laisser qu'un seul point : le curseur d'effacement, ou la gomme si vous préférez. Choisissez la taille de ce curseur avec les flèches HAUT (pour l'augmenter) ou BAS (pour la diminuer) et tapez **ENTER** pour confirmer la taille. Ensuite, utilisez les flèches pour promener la gomme aux endroits à effacer. Terminez l'opération par **ENTER**(*).

F envoit le curseur à droite de la fenêtre (**F**orward).

G change le type de lignes utilisé pour tracer des droites, pour remplir des rectangles,... Vous avez le choix entre le trait continu ou divers pointillés.

H aide l'utilisateur dans l'exécution d'une commande ou présente un texte reprenant toutes les commandes et les mouvements du curseur (**H**elp).

I sauve le chef-d'oeuvre que vous réalisez dans un fichier de l'unité données. Introduisez le nom du fichier sans extension. Pressez **F5** pour consulter la liste des graphiques existants. Si le nom du fichier désigne un graphique existant dans le répertoire des données, un message vous demande si vous désirez effacer l'ancien contenu pour le remplacer par le nouveau. L'opération se termine sans sauvetage quand vous ne voulez pas effacer l'ancien contenu.

J sauve une partie du graphique dans un fichier. Cette commande fait apparaître un rectangle visible par ces quatre coins. C'est le contenu de ce rectangle qui sera sauvé. La première opération consiste à faire coïncider le coin supérieur gauche du rectangle avec le coin supérieur gauche de la zone à sauver, en utilisant les flèches. Pressez **ENTER** pour fixer l'endroit. Ensuite, choisissez la taille de la zone avec les flèches et fixer votre choix en confirmant par **ENTER**. Tapez enfin le nom du fichier. La touche **F5** vous affichera le répertoire des graphiques. Un nom de fichier existant peut écraser l'ancien contenu ou quitter la commande sans sauvetage. Cette commande est très utile, par exemple, lorsque vous désirez constituer une petite bibliothèque de symboles couramment utilisés dans votre domaine d'enseignement (voir commandes R pour restaurer ces symboles).

K copie une partie de la fenêtre dans une zone de la même fenêtre, avec une multiplication de taille éventuelle. Placez le coin supérieur gauche du rectangle sur le coin supérieur gauche de la zone à copier, choisissez la taille de la zone avec les flèches. Déplacez ensuite le rectangle vers l'endroit de la copie. Multipliez la taille de la

copie en utilisant les flèches. Chaque opération se termine par **ENTER** (*).

L trace une droite. La position du curseur au moment de la première pression de L correspond à une extrémité de la droite. Déplacez alors le curseur à l'autre extrémité et pressez encore L pour visualiser cette droite (*) (**L**ine).

M déplace un morceau de la fenêtre avec une éventuelle multiplication de taille. Déterminez d'abord la zone à déplacer en plaçant le coin supérieur gauche du rectangle sur le coin supérieur gauche de l'endroit à déplacer, puis choisissez la taille du rectangle. Placez le rectangle à l'endroit voulu et multipliez la taille de la zone déplacée (en longueur et/ou en largeur). Pressez **ENTER** entre deux opérations (*) (**M**ove).

N inverse une zone de la fenêtre. Posez le rectangle sur la zone à inverser par le procédé habituel et confirmez par **ENTER** (*).

O remplit la figure fermée dans laquelle se trouve le curseur. Interrompez la commande en enfonçant une touche quelconque (*).

P imprime la fenêtre graphique et vous offre la possibilité de placer un titre d'impression sur la première ligne. L'imprimante doit être configurée en mode EPSON (**P**rint).

R rétablit un graphique créé antérieurement. La position du curseur correspond au coin supérieur gauche de la fenêtre restaurée. Introduisez le nom de ce graphique, pressez **F5** pour obtenir la liste des noms qui existent déjà. Si la taille du graphique ne permet pas une restauration complète du dessin restauré, seule la partie comprise dans le rectangle dont deux coins opposés sont déterminés par la position du curseur et le coin inférieur droit de la fenêtre graphique sera rétablie. Le graphique rétabli sera superposé au graphique qui existait avant cette commande (*) (**R**estore).

S dessine un rectangle. La position du rectangle correspond à un coin du rectangle. Déplacez le curseur dans le coin opposé du rectangle et frappez encore S. Vous pouvez

remplir ce rectangle avec le type de ligne courante (*) (Square).

T écrit une ligne de texte. Introduisez le message et sélectionnez le type de caractères : les caractères normaux sont de taille 8X8 pour une carte EGA ou CGA et 9X14 pour une carte Hercules tandis que les caractères graphiques sont de taille 4X6 pour les deux types de cartes. Choisissez l'endroit pour inscrire le texte et déterminez la taille du message que vous pouvez multiplier aussi bien en longueur qu'en largeur (*) (Text).

U positionne le curseur en haut de la fenêtre (Up).

V change la vitesse de déplacement du curseur. Les vitesses horizontale et verticale s'expriment chacune par un nombre entier compris entre 1 (vitesse la plus lente) et 15 (la plus rapide). Toute pression des flèches HAUT et BAS augmente ou diminue la vitesse horizontale ou verticale d'une unité tandis que les flèches GAUCHE et DROITE permettent de passer de la vitesse horizontale à la verticale et réciproquement. **ENTER** termine la commande.

W dessine tout en suivant le curseur. C'est en quelque sorte la trace du curseur. Pressez W pour finir la commande (*).

X efface le magnifique dessin que vous étiez en train de parachever. Prudence donc quant à l'utilisation de cette touche si vous n'avez pas sauvé le dessin. Il ne vous reste qu'un ultime recours : (*).

Z vous propose une vue agrandie d'une partie de la fenêtre. Déterminez d'abord l'endroit à agrandir en déplaçant le rectangle à cet endroit, puis frappez **ENTER**. La fenêtre graphique disparaît pour laisser place à la vue grossie. Vous pouvez inverser la couleur du point où se trouve le curseur en tapant sur la barre d'espacement et déplacer le curseur avec les flèches. Pressez **ENTER** pour terminer la commande. Vous observerez que la partie agrandie a subi les modifications que vous avez apportées (*) (Zoom).

F1 permet de revenir à l'étape antérieure dans les commandes qui demandent plusieurs opérations.

F2 abandonne l'exécution de commandes.

(*) **Y** tapé après cette commande annule l'effet de celle-ci et rétablit le graphique dans l'état où il se trouvait avant l'exécution de la commande. Si une nouvelle commande a été entamée (ou terminée) depuis, il est trop tard pour en annuler l'effet.

6.6. Création de questionnaires

6.6.1. Objectifs et utilisation de cet outil

Cet outil vous permet de créer deux types de questions :

- Des questions de type QCM (Question à Choix Multiples), composée ainsi :

  Enoncé de la question

  Enoncé des différentes réponses possibles (maximum 5 ici) L'étudiant doit alors choisir parmi ces réponses celle qui convient (A,B,C,D ou E).

- Des questions dites "ouvertes" où seul l'énoncé est fourni à l'étudiant.

Notre outil traite indistinctement ces deux types de questions et en permet même un mélange au sein d'un même questionnaire. Si vous donnez au plus une réponse, nous en déduisons qu'il s'agit d'une question ouverte, et seul l'énoncé de cette question apparaîtra à l'écran lors de l'exécution de la leçon.

Une leçon est par définition quelque chose d'interactif et nous ne pouvions nous contenter de donner des outils permettant seulement un "exposé" ! Un noeud questions-réponses a pour but de tester les connaissances de l'élève à un moment donné de la leçon, et en fonction des résultats, d'orienter la suite du déroulement vers l'un ou l'autre scénario adapté à l'élève faible, fort ou moyen.

Parfois, le déroulement de la leçon peut être laissé à la maîtrise de l'élève. Dans ce cas, c'est lui qui oriente le choix du noeud suivant en optant pour l'une ou l'autre des options présentées dans un menu. Or, un menu n'est pour nous ici, qu'un cas particulier de questionnaire, où les réponses possibles sont des OPTIONS et où certains concepts comme "LA bonne réponse" ou la cotation n'ont pas de sens.

On le voit bien, il est difficile de parler de cet outil sans faire référence à l'outil d'enchaînement et inversément... Les enchaînements "conditionnels" se feront toujours sur base de données ou de quantités observables lors de l'exécution d'un questionnaire.

* Edition ou consultation ?

C'est le premier choix que vous aurez à faire. La consultation d'un questionnaire élaboré auparavant vous permet de visualiser, aussi bien les questions telles qu'elles apparaîtront aux yeux de l'élève, que les options relatives à ces questions (nombre de points par réponse correcte, etc...)

L'édition d'un questionnaire vous permet aussi bien de créer un nouveau questionnaire que de modifier un ancien.

* Généralités

Bien que nous ayons privilégié ici un système de menus, un certain nombre de commandes étendent leurs effets à l'entièreté de l'outil (1) :

F2 Pour quitter cet outil. Si vous êtes en "création /modification", le contenu de la question courante ou, les modifications apportées à celle-ci ne seront pas "sauvées". Une question n'est en effet sauvée que lorsque son édition est totalement terminée.

F3 Comme cet outil travaille (tant en édition qu'en consultation) avec deux fenêtres, il est courant que l'une recouvre partiellement l'autre. Lors de la frappe de **F3**, la fenêtre du dessus s'efface momentanément au profit de l'autre, et ne réapparaît qu'après la frappe d'une touche quelconque.

END Pour spécifier la fin d'édition (du texte de la question ou des réponses possibles à celle-ci)

UP, DOWN Lors de l'édition du texte de la question, des réponses ou options, pour passer une ligne/question/option au dessus/en dessous.

---

(1) Si ces commandes ne sont pas pertinentes, à un instant donné, elles seront sans effet (bip sonore)

Le système de choix par menus existe encore ici. De plus, dans la mesure où il s'agit toujours d'un choix entre DEUX options, une commande supplémentaire a été ajoutée, pour une simple raison de facilité d'utilisation :

SPACE pour changer d'option choisie.

6.6.2. Explication de certains "concepts" relatifs à l'édition de questionnaires

- Création ou Modification ?

L'édition d'un nouveau questionnaire, ou la modification d'un ancien sont traitées indistinctement. Quand vous préciserez le nom d'un questionnaire existant déjà, question vous sera posée de savoir si vous désirez écraser ou modifier le contenu de l'ancien fichier. Si vous choisissez l'option d'écrasement, vous recréez un nouveau questionnaire et perdez le contenu de l'ancien.

Si vous choisissez l'autre option, vous démarrez l'édition avec le contenu de l'ancien fichier.

Si vous précisez le nom d'un fichier n'existant pas encore, notre outil conclut qu'il s'agit d'une création, et vous demandera de définir la taille de la fenêtre dans laquelle vous voulez que ce questionnaire apparaisse.

- Le système de valeurs par défaut

Afin de vous éviter au maximum l'introduction répétée d'informations identiques, une valeur dite par défaut a été prévue lors de (pratiquement) toute introduction de données.

Soit vous accepter ces valeurs et vous confirmez par **RETURN**, soit vous les modifiez et les nouvelles valeurs (ou options) introduites deviennent les valeurs (ou options) par défaut.

- La question courante.

Le numéro de la question courante apparaît toujours dans le bas de la fenêtre. C'est toujours cette question que l'on édite. Bien sûr, il est toujours possible de redéfinir une question, et cette possibilité vous sera offerte au début de chaque question.

Remarque : on ne peut REDEFINIR qu'une question qui a précé-

demment été définie en donnant son numéro.

Redéfinir une question correspond à la modifier et non à la recréer entièrement. Toutes vos erreurs d'introduction sont donc récupérables à peu de frais !

- Fenêtre d'édition et fenêtre de résultat.

Comme nous l'avons dit plus haut, l'objectif principal qui nous a guidé a été d'assurer la TRANSPARENCE. Sans aller à l'encontre de cet objectif, nous avons malgré tout choisi d'introduire les données dans une fenêtre à part, appelée fenêtre d'édition. Un fois les données introduites contrôlées, elles sont reportées sur la fenêtre réelle (voir figure 1).

Si la fenêtre d'édition recouvre des informations de la fenêtre réelle dont vous avez besoin, utilisez **F3**.

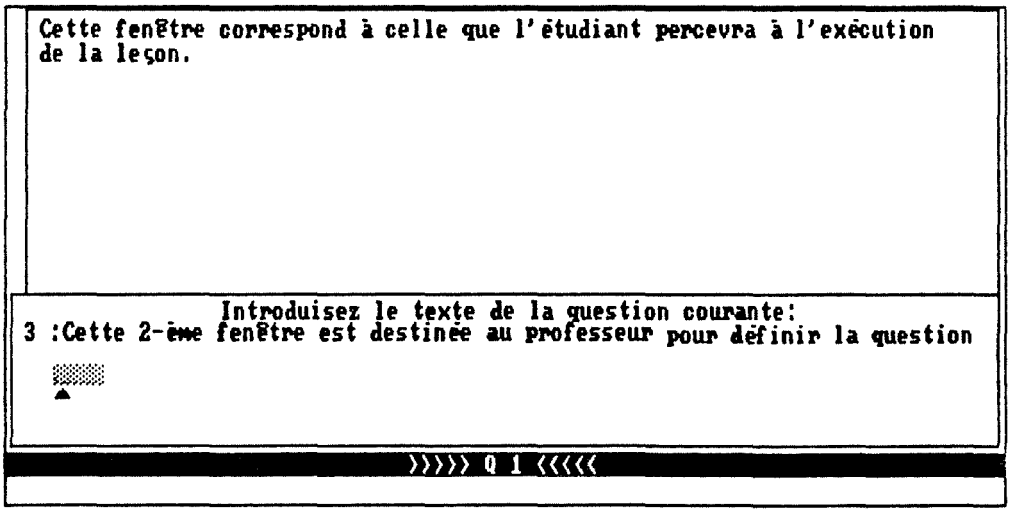


Figure 1 - Ecran de définition d'une question avec ses 2 fenêtres

- Edition de l'énoncé de la question

Celle-ci se fait ligne par ligne. Pour passer d'une ligne à l'autre, utilisez les flèches HAUT et BAS. Pour spécifier la fin de l'édition, pressez **END**.
Remarque : l'énoncé d'une question-menu correspond à un titre et des explications éventuelles.

- Question SIMPLE ou question de type MENU.

Comme nous l'avons dit précédemment, un menu n'est pour nous qu'un cas particulier de questionnaire, dans lequel un certain nombre de concepts n'ont pas de sens. la distinction n'est introduite ici qu'afin de vous éviter une introduction de données inutiles.

Comme une question de type MENU sert à effectuer un branchement vers un autre noeud, elle sera nécessairement la dernière du questionnaire. Ne vous étonnez donc pas de sortir automatiquement après l'édition d'une telle question.

- Edition des différentes REPONSES/OPTIONS possibles.

Cette introduction se déroule de manière similaire à l'édition de l'énoncé de la question. S'il s'agit d'une question de type MENU, vous devez au moins définir 2 options pour que cela ait un sens. Par contre, pour une simple question, vous avez trois possibilités :

+ Vous pouvez ne pas introduire de réponses. Dans ce cas, l'étudiant disposera du reste de la fenêtre pour introduire sa réponse, le professeur y aura accès en consultant le fichier "mouchard" de l'exécution.

+ Introduire une seule réponse. A l'exécution, seul l'énoncé de la question apparaîtra dans la fenêtre.

+ Introduire plusieurs réponses qui seront affichées à l'exécution. C'est le cas pour une question à choix multiples. Vous indiquerez ensuite quelle réponse est la bonne (seul cas possible si question-menu).

- Simulation relative à la question ou non ?

Votre réponse dépendra de l'existence ou non d'une simulation en rapport avec la question. Si c'était le cas, le fait de spécifier SIMULATION permettra à l'élève, s'il en éprouve le besoin, d'examiner le comportement d'une simulation créée par le professeur et si ce dernier lui en laisse la possibilité, de créer son propre modèle et de le tester.

Remarques : * La simulation dont vous préciserez le nom peut être seulement définie par après. Dans ce cas, un message vous avertira que "la simulation n'est pas encore définie".

* Cette question ne vous sera pas posée si la question est de type menu.

Quand ProfCOMP connaît au moins une réponse pour la question courante, il vous demandera en plus :

- Question susceptible de cotation ou non ?

Si vous déclarez que la question doit être cotée, deux autres questions auxiliaires vous seront posées afin de déterminer le nombre de points à attribuer en cas de réponse correcte ou erronée.

- Nombre d'essais

Vous avez le choix entre un nombre d'essais standard (3) ou un autre nombre quelconque (fixé à 2 par défaut). Si vous n'accordez qu'un seul essai, et uniquement dans ce cas, la question suivante vous sera posée :
"Voulez-vous que l'on confirme , à l'issue de cet essai, si la réponse introduite par l'étudiant est correcte ?"
Suivant qu'il s'agisse d'un examen ou d'un apprentissage, la réponse sera différente...

- Correction ?

Si l'élève a épuisé ses essais, et que sa réponse est toujours aussi mauvaise, faut-il ou non lui donner la réponse correcte ?
Cette question ne sera pas posée dans le cas d'une question avec un seul essai, vu la redondance avec la "confirmation".

6.7. Création d'une simulation

6.7.1. Introduction

L'étape de définition d'un modèle de simulation constitue la partie la plus importante puisque c'est elle qui guidera l'exécution du même modèle. L'utilisateur doit donc être attentif et comprendre parfaitement ce que ProfCOMP lui demande. ProfCOMP simule tous les modèles de la classe des systèmes d'équations différentielles ordinaires du premier ordre (1). La présentation mathématique d'un tel modèle est la suivante :

---

(1) Un système d'ordre supérieur à un peut également être simulé lorsqu'on peut le ramener à un système d'ordre un.

```
d/dt [X(i)] = f1(t,X(m),P(n),D(q),C(r))      (a)
P(j) = f2(t,X(m),P(n),D(q),C(r))             (b)
D(k) = fk(t,X(m),P(n),C(r))                  (c)
C(l) = réel                                  (d)
avec
    i,m = 1..nombre de variables d'état
    j,n = 1..nombre de paramètres
    k,q = 1..nombre de variables de retard
    l   = 1..nombre de constantes du modèle
et
    1 ≤ nombre de variables d'état ≤ 8
    0 ≤ nombre de paramètres ≤ 8
    0 ≤ nombre de variables de retard ≤ 4
    0 ≤ nombre de constantes ≤ 10
```

Les seules équations différentielles sont celles des variables d'état (a) mais ProfCOMP autorise également la définition de paramètre(s) (b) et/ou de variable(s) de retard (c) par une équation (non différentielle). Enfin, des constantes (d) peuvent faire partie du modèle. Rassurez-vous, chacun de ces éléments sera expliqué plus en détail par la suite.

Chaque fonction f, qui définit le membre de droite de l'équation différentielle de chaque variable d'état ou encore l'équation d'un paramètre ou d'une variable de retard, doit vérifier une certaine syntaxe. En fait, cette fonction f est une expression dans laquelle interviennent les paramètres de la fonction. Par la suite, nous emploierons souvent le terme EXPRESSION en lieu et place de fonction f.

Avant d'être en mesure de créer un modèle de simulation, nous détaillerons tous les composants d'un modèle en analysant la syntaxe des expressions.

6.7.2. Les expressions

La syntaxe des expressions fait l'objet de l'annexe A. Nous pratiquerons une analyse ascendante de cette syntaxe en détaillant d'abord les composants. Nous verrons alors

comment les combiner pour former les différents types d'expression dont nous décrirons l'utilisation précise. Nous terminerons par l'explication des messages d'erreur que ProfCOMP affiche lors de la détection d'incorrection syntaxique.

6.7.2.1. Les composants d'une expression

La plupart des composants qu'on peut trouver dans une expression figurent déjà dans le modèle mathématique exposé ci-dessus, en paramètres des fonctions f. Nous allons néanmoins les reprendre un par un pour les détailler davantage. Nous examinerons ensuite les opérateurs que l'on peut combiner avec ces opérandes pour former des expressions complexes.

a) Les opérandes

Les variables d'état

Tout modèle d'équations différentielles ordinaires contient autant de variables d'état que d'équations différentielles. Une variable d'état s'identifie par la lettre X (majuscule ou minuscule) suivie de son indice entre parenthèses. L'ensemble des indices forme un ensemble d'entiers strictement croissants compris entre 1 et le nombre de variables d'état du modèle. Le cardinal de cet ensemble vaut le nombre de variables d'état du modèle. Le système autorise un modèle d'au plus 8 équations différentielles.
Règle :

$$n \text{ variables d'état} \iff \underset{\#\{i\}=n}{X(i)} \quad 1 \le i \le n \quad \underset{1 \le n \le 8}{\forall i1, i2 \in \{i\} : i1 \neq i2}$$

Dans la syntaxe, l'opérande variable d'état présente 3 alternatives, à savoir X(indice-x), X(X(indice-x)) et X(P(indice-p)). Au niveau de l'indice, on peut trouver un entier (indice-x) mais aussi la valeur de la variable d'état X(indice-x) ou du paramètre P(indice-p) (nous examinerons l'opérande-paramètre ci-dessous). Evidemment, dans ces 2 derniers cas, c'est la VALEUR dynamique de la variable d'état ou du paramètre qui joue le rôle d'indice et qui doit respecter les propriétés exposées ci-dessus.

Les paramètres

La lettre P (minuscule ou majuscule) qui précède un indice entre parenthèses désigne un paramètre du système. En ce qui concerne les indices, on peut faire les mêmes remarques que pour les variables d'état. C'est pourquoi nous donnerons simplement la règle adaptée aux paramètres sans faire davantage de commentaires :

$$m \text{ paramètres} <=> P(j)_{\#\{j\}=m}^{1\le j\le m} \quad \forall j1,j2\in\{j\}: j1<>j2 \quad 0\le m\le 8$$

Remarquez cependant qu'un système ne comporte pas obligatoirement de paramètre : $0\le m\le 8$.
La syntaxe propose encore 3 représentations pour l'indice : un entier indice-p, la VALEUR dynamique d'une variable d'état X(indice-x) ou d'un paramètre P(indice-p). Quelle que soit cette représentation, l'indice devra toujours être compris dans l'ensemble strictement croissant d'entiers compris entre 1 et le nombre de paramètres (quand il y a des paramètres définis dans le système).
Remarque : étant donné que la valeur d'une variable d'état ou d'un paramètre est un nombre réel, ProfCOMP en prend la partie entière lorsqu'ils interviennent comme indice.

Les variables de retard

On identifie une variable de retard (en anglais delay) par la lettre D et son indice entre parenthèses. Un modèle peut contenir jusqu'à 4 variables de retard. L'indice respecte une règle similaire à celle des variables d'état ou des paramètres :

$$r \text{ var de retard} <=> D(k)_{\#\{k\}=r}^{1\le k\le r} \quad \forall k1,k2\in\{k\}: k1<>k2 \quad 0\le r\le 4$$

Seul changement : l'indice est uniquement un nombre entier compris entre 1 et le nombre de variables de retard du modèle.
Qu'est-ce qu'une variable de retard ?

Dans le modèle mathématique, on découvre qu'une variable de retard est définie par une équation dont l'expression à droite du signe égal lui donne sa valeur, mais avec un certain retard, exprimé comme multiple de

l'incrément de temps entre 2 pas d'intégration. La valeur à l'itération courante de cette variable vaut la valeur qu'avait l'expression qui la définit au temps courant MOINS le retard. Tant que la valeur du temps n'a pas atteint le retard à la résolution, la variable a une valeur nulle.

Les constantes

Les derniers composants indicés sont les constantes représentées par la lettre C.
Règle :

$$q \text{ constantes} \Leftrightarrow C(l)_{\#\{l\}=q}^{1\le l\le q} \quad {}^{\forall l1,l2\in\{l\}:l1\neq l2}_{0\le q\le 10}$$

L'indice est uniquement représenté par une valeur entière.
Pourquoi avoir introduit ces composants indicés alors que la syntaxe autorise un nombre entier ou réel comme opérande ?
A l'exécution d'un modèle, l'étudiant peut à tout moment stopper la simulation et, si le professeur l'y a autorisé, changer les valeurs des constantes (de même d'ailleurs que les conditions initiales des variables d'état), ce qui a pour effet de modifier toutes les expressions où elles interviennent. Une nouvelle simulation peut alors redémarrer pour observer le changement de comportement.

Le temps

La valeur réelle du temps de l'itération courante est symbolisé par la lettre T ou t.

Un nombre entier ou réel

Tout nombre entier est compris entre les valeurs -32768 et 32767 tandis que l'intervalle permis pour un réel s'étend de $-(10^{38})$ à 10^{38}.

La constante π

π est représenté par le mot PI ou pi.

b) Les opérateurs

Pour combiner les opérandes et former des expressions complexes, nous avons besoin d'autre chose : les opérateurs.

2 types d'opérateurs

- Les opérateurs à un argument

L'argument se trouve toujours après l'opérateur.

- Les opérateurs à deux arguments

Dans ce cas, le premier argument se trouve avant l'opérateur et le second après.

ProfCOMP utilise donc la notation infixée.
L'usage des parenthèses et des blancs est facultatif mais recommandé pour une meilleure lisibilité des expressions.

Les opérateurs et leur priorité

A chaque opérateur, on associe une priorité, exprimée par un nombre entier, afin d'évaluer correctement les expressions.

+ : Priorité : 5
- : Priorité : 5 ou 2 si l'opérateur n'est associé qu'à un seul opérande.
***** : Priorité : 4
/ : Priorité : 4
ABS : renvoie la valeur absolue de l'argument. Le résultat est du même type que l'argument (entier ou réel). Priorité : 2
ARCTAN : renvoie l'angle, exprimé en radians, dont la tangente est l'argument. L'argument est de type entier ou réel et le résultat est de type réel. Priorité : 2
COS : renvoie le cosinus de l'argument, exprimé en radians et dont le type est entier ou réel. Résultat de type réel. Priorité : 2
SIN : renvoie le sinus de l'argument, exprimé en radians et dont le type est entier ou réel. Résultat de type réel. Priorité : 2
EXP : donne l'exponentielle de son argument, entier ou réel. Résultat réel. Priorité : 2
FRAC : extrait la partie fractionnelle de l'argument (entier ou réel). Résultat de type réel. Priorité : 2
LN : renvoie le logarithme naturel de l'argument (entier ou réel). Résultat réel. Priorité : 2
INT : renvoie la partie entière de l'argument, c'est-à-dire le plus grand entier inférieur ou égal à l'argument si

l'argument est positif ou nul, ou le plus petit entier supérieur ou égal à l'argument si l'argument est négatif. Résultat réel. Priorité : 2

SQR : élève l'argument (entier ou réel) au carré. Résultat de même type que l'argument. Priorité : 2

SQRT : donne la racine carrée de l'argument (entier ou réel). Résultat de type réel. Priorité : 2

DIV : effectue la division de deux arguments entiers. Résultat de type entier. Priorité : 4

MOD : renvoie le reste de la division du premier argument (entier) par le second (entier). Résultat de type réel. Priorité : 4

****** : renvoie la valeur du premier argument (entier ou réel) avec le deuxième argument comme exposant (entier ou réel). Priorité : 3

Les opérateurs relationnels sont les suivants: **<**, **<=**, **>**, **>=**, **=**, **<>**. Leur priorité est 6. Nous en examinerons l'utilité ci-dessous.

6.7.2.2. Les types d'expressions et leur utilisation

Jusqu'à présent, nous avons énoncé une liste d'opérandes et d'opérateurs. Examinons maintenant les types d'expressions qu'on peut former avec ces éléments et surtout, quand on utilise ces expressions.

La syntaxe définit 2 types d'expressions qui contiennent chacune 2 alternatives. En fait, toutes les expressions qui vérifient syntaxiquement une des alternatives de "expression" vérifient aussi "expression-retard". La seule différence entre les 2, c'est que "expression-retard" admet un opérand de plus, à savoir les variables de retard : "expression" admet les opérandes "opérande1" tandis que "expression-retard" admet "opérande" qui regroupe "opérande1" et "opérande2". Cette distinction empêche donc l'utilisation de variables de retard dans certaines expressions.

Intéressons-nous davantage aux alternatives de "expression-retard".

"Expression-retard1" désigne ce que nous appellerons par la suite une expression normale où interviennent des opérandes et des opérateurs (non relationnels) dans une combinaison syntaxiquement correcte.

Exemples : X(2)

(1-X(1)**2)*X(2)-X(1)+.5*COS(6.28*T/5.9)

1-D(2)+X(1) mod P(X(1))+LN P(5)

"Expression-retard2" décrit des expressions plus complexes appelées expressions conditionnelles, composées de 3 sous-expressions, chacune entre accolades. La première sous-expression se décompose en 3 parties : 2 "expression-retard1" séparées par un opérateur relationnel qui permet de définir une condition qui sera évaluée à chaque itération de l'exécution du modèle. Si la condition est évaluée à vrai, la seconde expression, de type "expression-retard1", sera évaluée et représentera le résultat de l'expression complète. Dans le cas contraire, c'est la troisième sous-expression qu'on évaluera.

Exemple : L'expression {T > 50} {X(1)} {X(2)} peut être interprétée comme suit :

SI le temps est supérieur à 50,
ALORS l'expression prend la valeur de X(1)
SINON l'expression prend la valeur de X(2)

On peut donner les mêmes explications pour les alternatives de "expression", avec la restriction qu'on ne peut y trouver de variables de retard comme opérande.

Dans quel cas l'expression introduite doit-elle être conforme à "expression" ou "expression-retard1" ?

Dans l'étape de définition d'un modèle, vous devez spécifier l'équation différentielle de chaque variable d'état. Le premier membre de cette équation s'affichera à l'écran, il restera à en définir le second par l'expression adéquate.

Exemple : d/dt [X(1)] = ...

Après les variables d'état, c'est le second membre de l'équation de chaque paramètre, si le système en utilise, qu'il faut définir par une expression.

Exemple : P(1) = ...

Enfin, une sortie graphique demande les expressions dont les valeurs seront représentées sur chaque axe et une sortie numérique interroge sur les expressions intéressantes pour lesquelles l'utilisateur veut conserver les valeurs dans un fichier.

TOUTES CES EXPRESSIONS S'ACCORDENT AVEC LA SYNTAXE DE "expression-retard".

Reste le seul cas où on ne peut utiliser de variable de retard : D(i) = ... (1≤i≤4)

L'EXPRESSION QUI DEFINIT LE MEMBRE DROIT DE L'EQUATION D'UNE VARIABLE DE RETARD NE PEUT CONTENIR DE VARIABLE DE RETARD et vérifie, par conséquent la syntaxe de "expression".

6.7.3. Explication des messages d'erreur concernant la syntaxe d'une expression

ProfCOMP réagit dès que vous commettez une erreur lors de l'introduction d'une expression. Voici tout ce qu'il peut vous dire :

* UNE EXPRESSION D'UN SEUL ELEMENT DOIT ETRE UN OPERANDE

Une expression qui ne contiendrait qu'un opérateur n'a aucun sens.

Exemple : 4+MOD
?...

l'expression MOD devrait être un opérande.

* PARENTHESE OUVRANTE SANS PARENTHESE FERMANTE

Dans une expression, le nombre de parenthèses ouvrantes doit toujours être égal au nombre de parenthèses fermantes sinon, l'expression ne peut être évaluée.

Exemples : P(1+4*C(2) MOD 3
?...

(4*P(1)+2**P(2)-X(1)
?...

* PARENTHESE FERMANTE SANS PARENTHESE OUVRANTE

Même remarque que pour l'erreur précédente.

Exemple : P(1)+2**X(4)) MOD C(1)
...?

* OPERATEUR A UN OPERANDE, EXPRESSION GAUCHE NON VIDE

L'opérande d'un tel opérateur se trouve toujours à droite de l'opérateur. Si un élément se trouve à gauche de cet

opérateur, ce ne peut être qu'un autre opérateur sinon ce serait un opérateur à deux opérandes.

Exemple : 4SIN(2*P(1)*PI)
?...

* EXPRESSION DROITE VIDE

C'est le cas d'un opérateur, qu'il soit à un ou deux opérandes, pour lequel l'opérande à droite est indéfini.

Exemples : 2+
?...
2+3 MOD
?...

* OPERATEUR A EXPRESSION GAUCHE VIDE

C'est aussi le cas d'un opérateur à deux opérandes, mais pour lequel le premier opérande est indéfini.

Exemple : *3
...?

* RIEN DANS LES PARENTHESES

L'usage des parenthèses est facultatif (sauf pour les variables indicées ou pour changer la priorité d'un opérateur) mais il ne faut pas abuser de leur usage dans un sens opposé.

Exemple : 3*()
?...

* EXPRESSION DE PLUSIEURS ELEMENTS SANS OPERATEUR

Une telle expression contient plusieurs opérandes contigus or, entre deux opérandes, il faut toujours au moins un opérateur à deux opérandes.

Exemple : P(1)C(2)X(3)
?... ?...

* SECOND OPERATEUR RELATIONNEL LU OU NON AUTORISE

Une expression conditionnelle ne peut contenir qu'un seul opérateur relationnel situé dans la première sous-expression entre accolades. Les autres sous-expressions ne peuvent avoir d'opérateur relationnel.

Exemple : {0<=X(1)<=5} {t} {-t}
?...

* STRUCTURE D'UNE EXPRESSION CONDITIONNELLE INCORRECTE

Une expression conditionnelle peut être incorrecte pour plusieurs raisons : il manque une sous-expression, un nombre impair d'accolades ...

Exemple : {X(1)>10} {t}
?...

* EXPRESSION VIDE

6.7.4. Les étapes de la création d'une simulation

La création d'un modèle de simulation constitue toujours la première étape d'utilisation du logiciel. La définition des exigences de l'utilisateur s'effectuera par une série de questions ou de demandes d'introduction d'éléments bien précis. Nous allons passer en revue toutes les opérations nécessaires à la création d'un modèle complet. Quand il s'agit d'introduire un nombre entier ou réel, le logiciel compare toujours cette valeur aux bornes qu'il admet avant de l'accepter ou de la refuser. Nous indiquerons les bornes admises de la manière suivante pour un entier :

valeur minimale ≤ entier ≤ valeur maximale

Quand il s'agit d'un réel, le mot réel remplacera le mot entier mais ce sera la même représentation.

On vous demandera d'introduire, dans l'ordre:

* Le nom sous lequel les informations de la simulation seront sauvées. Si vous introduisez un nom de simulation déjà existant, vous pourrez soit modifier soit redéfinir le modèle déjà créé. Pressez la touche **F5** pour obtenir la liste de toutes les simulations déjà créées.
* La taille de la fenêtre graphique qui sera utilisée à l'exécution pour représenter le(s) graphique(s) désiré(s). La longueur (X) et la largeur (Y) de la fenêtre courante sont affichées dans le coin supérieur gauche de l'écran et la fenêtre qui correspond à ces dimensions apparaît également à l'écran. Utilisez les flèches pour en faire varier la taille. En cas de modification d'un modèle existant, vous pouvez accepter les dimensions définies précédemment ou en redéfinir de nouvelles.
* Après l'exécution de cette simulation, l'étudiant peut-il simuler son propre système ? Si oui, il pourra introduire tous les éléments nécessaires pour modéliser un système et ses résultats attendus, c'est-à-dire tout ce qui est décrit dans cette section, excepté les questions destinées spécialement au professeur. Exemple : la question décrite ci-dessus.

* La méthode de résolution du système. ProfCOMP propose 2 méthodes de résolution numériques : Euler amélioré (deuxième ordre) ou Runge-Kutta (quatrième ordre).
* Le nombre de variable(s) d'état. Tout système comprend au moins une variable d'état et au plus 8 variables d'état.
* Le nombre de paramètres du système ($0 \leq entier \leq 8$).
* Le nombre de constantes du système ($0 \leq entier \leq 10$).
* Le nombre de variables de retard ($0 \leq entier \leq 4$).
* Le temps du début de simulation, nécessaire pour calculer la première itération du système ($10^{-5} \leq réel \leq 10^{5}$).
* L'incrément de temps entre deux itérations successives ($10^{-20} \leq réel \leq 10^{4}$).
* Les équations différentielles des variables d'état, puis les équations des paramètres, s'il y en a, par ordre croissant d'indice (voir le paragraphe sur les expressions).
* L'étudiant peut-il modifier les valeurs initiales des variables d'état et les valeurs des constantes du modèle professeur et réexécuter ce système pour en observer le nouveau comportement ?
* Les valeurs des constantes, s'il y en a, par ordre croissant d'indice ($-10^{20} \leq réel \leq 10^{20}$)
* Pour chaque variable de retard, l'expression qui la définit et le retard associé (incrément de temps$\leq$réel$\leq$50 * incrément de temps). ProfCOMP accepte donc un retard maximum de 50 pas d'intégration.
* Les valeurs initiales des variables d'état ($-10^{20} \leq réel \leq 10^{20}$).
* Les valeurs initiales des paramètres (s'il y en a) dont l'équation fait appel à des indices de paramètres supérieurs au leur ($-10^{20} \leq réel \leq 10^{20}$).

Exemple : P(1)= X(2)+P(5) et P(4)= P(2)+P(3) ; L'équation de P(1) fait appel au paramètre P(5). P(5) est défini après P(1) et donc une valeur initiale est requise pour P(1). Par contre, la définition de P(4) fait intervenir les paramètres d'indice 2 et 3, définis juste avant. Il est donc inutile de demander une valeur initiale pour P(4) puisqu'on connait les valeurs initiales de P(2) et P(3).
* Le type de résultat désiré à l'exécution:
 - une sortie graphique qui représentera une ou plusieurs courbes, selon vos souhaits.

- une sortie numérique qui reprendra dans un fichier de type texte les valeurs d'expression(s) qui vous paraisse(nt) intéressantes.

-Les deux types de sorties décrites précédemment.

* Si une sortie graphique a été demandée, on vous demande:

- si vous voulez relier tous les points successifs d'une courbe ou les afficher tout simplement comme des points isolés.

- le nombre de courbes à dessiner à l'exécution.

- la fréquence de représentation des points à l'affichage ($1 \leq entier \leq 100$). Si vous voulez une fréquence de représentation de N, un point sur N sera représenté graphiquement.

- pour chaque courbe,

. l'expression dont les résultats de l'évaluation seront représentées sur l'axe des abcisses (expression).

. l'expression dont les résultats de l'évaluation seront représentés sur l'axe des ordonnées.

. les valeurs minimale ($-10^{35} \leq réel \leq 10^{34}$) et maximale (valeur minimale + $10^{-3} \leq réel \leq 10^{35}$) de l'intervalle de représentation des points en abcisse, puis les mêmes valeurs pour l'axe des ordonnées.

* Si une sortie numérique a été demandée :

- le nombre d'expression(s) dont on doit conserver les valeurs dans le fichier texte ($1 \leq entier \leq 3$), puis

- la définition des expressions dont les valeurs seront écrites dans le fichier des résultats numériques.

- la fréquence d'écriture des valeurs des équations dans le fichier.

* L'intervalle de temps au cours duquel une représentation graphique ou numérique est demandée.

6.8. Enchaînement des noeuds

6.8.1. Principe

C'est grâce à ce dernier outil que nous allons enfin créer ce qui mérite d'être appelé une LECON assistée par ordinateur.

1 ENCHAINEMENT <--> 1 LECON

Pour bien comprendre cet outil, faisons une comparaison avec

le phénomène du compagnonnage.

Jadis, lorsque l'on voulait devenir artisan, il n'y avait pas UNE école, mais bien des dizaines de centres d'apprentissage. L'aprenti assimilait les connaissances et le savoir-faire nécessaires à l'exercice de la profession, en allant de ville en ville, de village en village...

A chaque étape, un "maître-artisan" lui faisait partager un peu de son doigté dans les domaines où il excellait.

Une fois que l'apprenti était parvenu à réaliser une "pièce-type" témoignant d'une certaine maîtrise qu'il avait acquise, il se dirigeait vers un autre maître susceptible de lui faire partager d'autres spécialités...

L'enseignement a bien changé depuis lors; mais nous pouvons tout de même comparer l'étudiant à l'apprenti, les noeuds texte, graphique, animation et simulation aux villes et les noeuds questions-réponses à la réalisation de "pièces-types". L'enchaînement des noeuds peut dès lors être comparé aux voies ferrées reliant les différentes villes et aux aiguillages régissant ce réseau ferroviaire.

Si l'on assume le fait que vous remplissiez le rôle des différents maîtres, vous comprendrez aisément qu'il vous faudra remplir le travail de l'aiguilleur !

Afin de mener à bien cette mission, vous allez disposer d'un LANGAGE, basé sur un nombre limité de termes anglais.

Un pilote vous assistera pendant l'édition de l'enchaînement, vous évitant toute erreur d'orthographe ou de syntaxe. Le pilote vous présente à tout moment (dans une fenêtre d'édition) les différentes options possibles à ce moment. Il vous suffira de taper le symbole correspondant à l'option désirée (**1,2,3,... F1,F2,...**) ou de déplacer le rectangle en vidéo inversé sur l'option désirée avant de confirmer par **RETURN**. Seules certains paramètres devront être introduits entièrement (exemple : le nom d'un noeud).

Malgré le contrôle attentif du pilote, des erreurs d'ordre sémantique/syntaxique peuvent subsister, c'est

pourquoi nous avons prévu un traitement supplémentaire à la fin de l'édition : la compilation d'un enchaînement, révélant deux types d'erreurs :

- erreurs fatales : Des erreurs telles que l'oubli d'une étiquette rendraient toute interprétation ultérieure de ce texte impossible. Vous DEVEZ corriger ces erreurs !
- avertissements : Si, par exemple, vous spécifier le nom d'un noeud inexistant, c'est peut être parce que vous avez l'intention de le définir par après... il ne s'agit donc pas à proprement parler d'une erreur. Les avertissements sont là pour vous conscientiser, ils ne nécessitent pas toujours de corrections.

6.8.2. Sémantique des instructions

La syntaxe du langage fait l'objet de l'annexe B.

Plusieurs instructions sont auto-explicatives. Il suffit en effet de se rappeler le principe du multi-fenêtrage abordé plus haut:

- **Clear all the screen** efface tout ce qui se trouve à l'écran;
- **Erase window x** enlève de l'écran la fenêtre identifiée par l'entier x;
- **Inverse the two last windows** ramène l'avant-dernière fenêtre définie, à l'avant-plan sur l'écran.
- **Get window x up** ramène à l'avant-plan la fenêtre identifiée par l'entier x.

Nous allons par contre détailler quelque peu les autres instructions :

- **EXECUTE nom.ext (n,x,y [,z,d]) [WITH SIMULATION (a,b,c)]** exécute le noeud "nom" de type "ext" dans la fenêtre "n" positionnée en (x,y). Les éléments entre crochets seront expliqués ci-dessous.

Le premier choix consiste à définir le type du noeud exécuté :

- animation (ANI),
- graphique (GRF),
- question-réponse (QCM),
- simulation (SIM) ou
- texte (TXT).

L'extension du noeud correspond à son type (cfr ci-dessus entre parenthèses). Le nom du noeud exécuté est celui défini à sa création, sauf pour une animation, composée de plusieurs graphiques dont le dernier caractère du nom est un entier identifiant l'ordre d'affichage. Le nom à donner pour une animation est le radical commun à tous ces composants, obtenu en enlevant le dernier caractère.
Exemple : GRAPH1.GRF, GRAPH2.GRF, GRAPH3.GRF sont les trois composants d'une animation. Le radical commun à ces noms est "GRAPH". On aura donc une instruction du type :

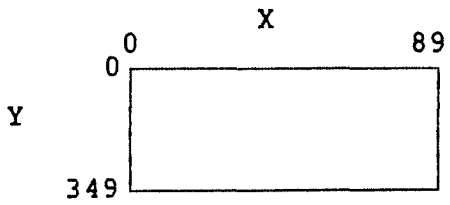
EXECUTE graph.ANI (...);

Comme nous vous l'avons fait remarquer plus d'une fois, un noeud sera toujours exécuté dans le cadre d'une fenêtre que vous avez définie lors de la création. Après avoir défini le type et le nom du noeud exécuté, nous vous demandons maintenent de donner un numéro de fenêtre n (de 1 à 7). Ce numéro n'a pas beaucoup de sens pour vous, il n'est là que pour vous empêcher de travailler simultanément avec plus de sept fenêtres. Si vous avez déjà utilisé le numéro de fenêtre x et que dans la suite de l'enchaînement vous le réutilisez, le contenu précédant de cette fenêtre est perdu.

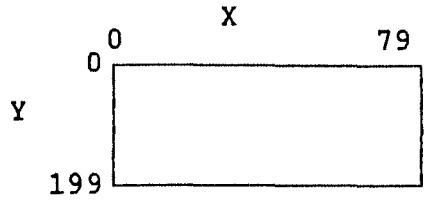
Nous vous demandons également la position à laquelle vous voulez voir apparaître votre fenêtre (x,y). Pour cela, vous donnerez les coordonnées du coin supérieur gauche de votre fenêtre, et nous essayerons de vous satisfaire dans la mesure du possible. Le logiciel peut s'exécuter indifféremment sur une carte HERCULES ou IBM (EGA ou CGA) mais la définition d'un écran est différente suivant qu'on travaille en mode HERCULES ou IBM. De plus, on distingue deux types de coordonnées : les coordonnéees graphiques ou textes. Voici ces types de coordonnées pour chacune des cartes :

- Les coordonnées graphiques pour l'exécution des noeuds SIMULATION, GRAPHIQUE ou ANIMATION :

En mode HERCULES, un écran graphique est défini comme suit :

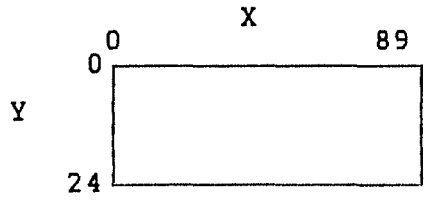


tandis qu'en mode CGA, on a :

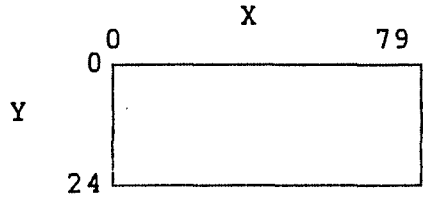


- Les coordonnées de type texte pour l'exécution d'un noeud de type QCM ou TEXTE :

En mode HERCULES :



En mode IBM :



A vous de positionner correctement vos fenêtres par rapport aux modèles d'écran ci-dessus en tenant compte du type d'écran.

Pour une animation, on trouve encore deux chiffres :

- z correspond au nombre de composants de l'animation,
- d représente le délai entre l'affichage de deux composants successifs.

L'exécution d'un noeud QCM peut être couplée à l'exécution d'un noeud SIMULATION. Il faut dans ce cas spécifier le numéro de la fenêtre (a) et les coordonnées (b,c) du coin supérieur gauche de la fenêtre SIMULATION.

- **GOTO x** : En principe les lignes de l'enchaînement seront interprétées séquentiellement, les unes à la suite des autres. Vous pouvez déroger à ce principe en forçant l'interpréteur à ALLER VERS une certaine partie de l'enchaînement. Comme il faut spécifier quelle est cette partie, vous allez l'étiqueter x (étiquette).

Exemple :

```
EXECUTE TOTO.TXT (1,5,10) ;
EXECUTE TATA.GRF (3,39,56)';
GOTO JMB ;
EXECUTE TUTU.SIM (5,50,50);
JMB :
EXECUTE TITI.QCM (1,12,18) WITH SIMULATION (2,40,40);
...
```

Cet exemple est stupide car TUTU.SIM ne sera jamais exécuté. Le saut INCONDITIONNEL provoqué par GOTO JMB a pour conséquence le schéma d'exécution suivant :

exécution de toto
exécution de tata
exécution de titi
...

Remarquons tout de même dans cet exemple que titi réutilise la fenêtre 1 ce qui nous fait supposer que son contenu était devenu inutile !

-**IF cond** : Cette instruction est un peu plus compliquée et ne prend tout son sens que couplée à l'instruction **GOTO**. Donnons d'abord ici sa sémantique individuelle : lors de l'interprétation de l'enchaînement, si la condition exprimée à la suite du **IF** est réalisée, l'interprétation suit son cours normalement, à la ligne d'instruction suivante. Sinon, elle saute la ligne d'instruction suivante et continue à celle qui se trouve juste après.
Remarquons que nous parlons ici de ligne d'instruction, les lignes comportant les étiquettes ne sont donc pas prises en compte ici !

La condition cond peut être de deux types :

- après l'exécution d'une question de type MENU, on doit tester le résultat de l'élève pour aiguiller la leçon. La condition exprime alors une réponse possible de l'élève.
Exemple : IF MENU CHOICE A sera évalué à VRAI si l'élève a choisi la première option du menu.
- Dans d'autres cas, il s'agira de comparer la valeur d'une des six variables prédéfinies par rapport à un nombre entier. Les opérateurs possibles : <, <=, >, >=, =, <>. Les variables prédéfinies sont les suivantes :

* NOMBRE D'ESSAIS QCM
* NOMBRE DE POINTS QCM

* (NOMBRE TOTAL DE POINTS / NOMBRE DE POINTS QCM POSSIBLE)
* NOMBRE TOTAL D'ESSAIS
* NOMBRE TOTAL DE POINTS
* (NOMBRE TOTAL DE POINTS/NOMBRE TOTAL DE POINTS POSSIBLE)

Exemple :

```
EXECUTE TOTO.TXT (1,5,10);
EXECUTE TATA.QCM (2,50,15);
IF QCM POINTS NUMBER <= 30
EXECUTE TUTU.TXT (1,5,10);
EXECUTE TITI.GRF (4,40,100);
EXECUTE TETE.ANI (3,15,32,4,2);
...
```

Voici les deux schémas d'exécution possible,

Si la condition est réalisée :	Si elle ne l'est pas :
exécution de toto	exécution de toto
execution de tata	exécution de tata
exécution de tutu	exécution de titi
exécution de titi	exécution de tete
exécution de tete	

Remarquons d'abord que préalablement à l'instruction **IF**, nous avons exécuté un noeud QCM ! Sinon, les valeurs intervenant dans le **IF** n'auraient pas de sens !

Remarquons ensuite que l'instruction **IF** permet, sur base de l'évaluation d'une certaine condition d'exécuter UNE ET UNE SEULE INSTRUCTION dans un cas et pas dans l'autre !
L'exemple suivant utilise une **combinaison intelligente de IF et de GOTO** afin de réaliser deux parties de leçon différentes sur base d'un test.
Supposons que vous vouliez dispenser un rappel bref ou un rappel approfondi suivant que l'étudiant possède déjà bien la matière ou pas....

TEST.QCM : le noeud QCM correspondant au test de connaissances de base

RAPPEL1.TXT est le noeud texte correspondant au rappel bref

Le rappel approfondi sera constitué de trois noeuds :

RAPPEL2.TXT : texte d'explications approfondies ...
DESSIN.GRF : graphique appuyant ce premier texte
RAPPEL3.TXT : suite des explications

La suite de la leçon sera identique pour les deux "types" d'étudiants et consistera (pour simplifier) en un grand noeud texte : COMMUN.TXT

```
EXECUTE TEST.QCM (1,1,1);
IF QCM POINTS NUMBER < 60
GOTO FAIBLE;
EXECUTE RAPPEL1.TXT (2,10,10);      ]1
GOTO SUITE;
FAIBLE :
EXECUTE RAPPEL2.TXT (2,10,10);      ]
EXECUTE DESSIN.GRF  (3,50,100);     ]2
EXECUTE RAPPEL3.TXT (4,40,20);      ]
SUITE :
EXECUTE COMMUN.TXT (7,12,14);
```

Remarquons que la partie 1 (qui, en toute généralité pourrait être plus grande, pourvu qu'elle se termine par **GOTO** suite) n'est exécutée que si la condition n'est PAS réalisée, et que la partie 2 (qui, en toute généralité, pourrait être plus grande, pourvu qu'elle soit comprise entre les étiquettes "faible" et "suite") n'est exécutée que si la condition est réalisée !
Remarquons également que si "rappel1" est exécuté lors d'une leçon, rappel2 ne le sera pas, et inversément ! Il serait donc inutile d'utiliser des numéros de fenêtre différents.

Un dernier exemple va vous montrer comment on peut réaliser des conditions complexes du genre : " Si le nombre total de points est > 100 ET que le nombre total d'essais est < ou = 10 ..."
Supposons que dans l'affirmative, on exécute COURS1.TXT et dans le cas contraire COURS2.TXT et que la suite du cours soit commune et corresponde au noeud SIMULAT.SIM.

```
EXECUTE QUEST1.QCM (1,1,1);
...
EXECUTE QUEST2.QCM (7,12,12) WITH SIMULATION (2,50,100);
...
IF TOTAL POINTS NUMBER <= 100
GOTO TRAITEMENT2;
IF TOTAL TRIALS NUMBER > 10
GOTO TRAITEMENT2 :
EXECUTE COURS1.TXT (3,12,4);
GOTO SUITE;
TRAITEMENT2 :
EXECUTE COURS2.TXT (5,1,2);
SUITE :
EXECUTE SIMULAT.SIM (6,50,100);
...
```

Remarquons que les tests se feront sur le total des points et des essais au cours de la leçon, c'est-à-dire au cours de l'exécution des noeuds QUEST1.QCM et QUEST2.QCM.
Remarquons également que si une condition avait été exprimée avec un OU :
"Si le nombre de points du <u>dernier</u> test > 100 ou que le nombre d'essais du <u>dernier</u> test < 10 ..." ceci devient :

```
...
EXECUTE QUEST1.QCM (1,1,1);
...
EXECUTE QUEST2.QCM (7,12,12) WITH SIMULATION (2,50,100);
...
IF QCM POINTS NUMBER > 100
GOTO TRAITEMENT1;
IF QCM TRIALS NUMBER < 10
GOTO TRAITEMENT1;
EXECUTE COURS2.TXT1 (3,12,4);
GOTO SUITE;
TRAITEMENT1 :
EXECUTE COURS1.TXT1 (3,12,4);
SUITE :
EXECUTE SIMULAT.SIM (6,50,100);
...
```

Les tests se font maintenant sur les points remportés lors du dernier questionnaire, à savoir QUEST2.QCM ainsi que sur le nombre d'essais lors de ce même questionnaire.

Il vous est également possible de raisonner en terme de pourcentage, en utilisant une condition du type :

IF (POSSIBLE TOTAL POINTS NUMBER/TOTAL POINTS NUMBER)<... ou
IF (POSSIBLE TOTAL POINTS NUMBER / QCM POINTS NUMBER) ...

7. EXECUTION D'UNE LECON (PC_EXEC)

Avant d'entrer dans l'exécution du cours proprement dit, l'étudiant doit introduire le nom de la leçon (défini par le professeur au moment de l'enchaînement des noeuds) et un code qui l'identifie (son nom par exemple). La dernière ligne de l'écran oriente l'étudiant en lui indiquant toutes les opérations permises à l'instant courant.

Le fichier "mouchard", rapport d'exécution, portera le même nom que le code identifiant l'étudiant avec l'extension ".REL". Ce fichier, destiné au professeur, contient un rapport complet d'exécution et est visualisable par la simple commande MS-DOS "type". On y retrouve le cheminement d'exécution si on relève le nom de tous les noeuds exécutés.
Exemple :

Si l'étudiant s'appelle Jorge, pour consulter le "mouchard" de la dernière leçon effectuée par cet étudiant, vous introduirez, au niveau DOS, la commande type JORGE.REL|MORE, qui affichera ce fichier page par page.

Les noeuds questions-réponses y sont très détaillés puisqu'on trouve, pour chaque question, son énoncé, la (les) réponse(s) fournie(s) et, si cotation il y a, les scores obtenus.

Quand un changement de valeurs initiales des variables d'état et des valeurs des constantes est autorisé dans une simulation créée par le professeur, on les retrouve mentionnées lorsque l'étudiant utilise cette possibilité. Pour chaque simulation d'un modèle de l'étudiant, les équations des variables d'état, des paramètres et les valeurs des constantes sont indiquées.

Ce fichier se termine par le temps d'exécution global de la leçon et par la cotation méritée pour l'élève.
ATTENTION !!! ce fichier mouchard n'est pas protégé. Veuillez donc à ce que l'étudiant n'aille pas le trafiquer pour augmenter ses cotes. De plus, si l'étudiant exécute deux leçons de suite, il doit veiller à donner deux codes différents.

8. LES UTILITAIRES

8.1. INST_EAO

Ce programme permet l'installation des trois programmes principaux qui constituent ce logiciel : PC_CREAT.COM, PC_EDIT.COM et PC_EXEC.COM. Le premier menu vous interroge sur le nom du programme à installer. Ensuite, vous devez définir le chemin d'accès pour accéder à ce programme à installer, si celui-ci ne se trouve pas sur le drive ou le répertoire courant (pressez **RETURN** dans le cas contraire). Si le fichier à installer ne se trouve pas dans le répertoire indiqué, le premier menu se réaffiche. Pour chacun des 3 programmes installables, un menu vous indique les options que vous pouvez modifier :

- Cinq unités sont définies pour l'ensemble du logiciel. A chaque installation, ces cinq unités vous serons demandées même si pour l'outil installé, certaines n'ont pas de sens. Pour plus de précision sur le contenu de ces unités, référez-vous à l'annexe C. Voici ces unités :
  * L'unité "programmes". A l'installation de l'éditeur, spécifiez le répertoire du programme PC_CREAT.COM (car l'éditeur est appelé à partir de ce programme et PC_EDIT lance PC_CREAT en fin d'utilisation de l'éditeur). Doit bien sûr aussi être précisée pour PC_CREAT.
  * L'unité des "données"
  * L'unité "éditeur" : l'éditeur étant un programme distinct de PC_CREAT.COM, il peut se trouver sur un répertoire différent. Cette unité doit être précisée pour PC_EDIT et PC_CREAT.
  * L'unité exécution (pas de sens pour les programmes PC_CREAT.COM et PC_EDIT.COM)
  * L'unité "commune"

- La touche 2 vous permet d'installer les couleurs du programme. Il y en a par exemple six pour PC_CREAT : deux couleurs pour l'affichage de texte, une pour les cadres, une pour les commandes, une pour les erreurs et la dernière utilisée lorsqu'on se trouve en mode graphique. L'écran affichera toutes les combinaisons possibles de couleur et d'arrière-fond et un numéro identifiera chaque possibilité. Choisissez la combinaison idéale (couleur, arrière-fond)

pour chacune des couleurs en tapant l'identifiant correspondant. Attention pour la couleur "graphique" : si vous ne disposez pas d'une carte EGA, l'arrière-fond restera noir et ne prendra pas la couleur que vous avez choisie.

- Tous les fichiers de messages ou d'aide peuvent être traduits dans la langue maternelle de l'utilisateur par l'intermédiaire de l'utilitaire TRANS_ME.COM. Pour chacun des 3 programmes du logiciel, la (les) version(s) "locale(s)" de son (ses) fichier(s) de messages est (sont) caractérisée(s) par une extension unique. C'est cette extension qu'on vous demande de définir ici.
Exemple : L'extension des versions anglaises des différents fichiers de messages et d'aide est ".ENG". Si vous voulez utiliser ce logiciel en langue portugaise, utilisez TRANS_ME.COM pour traduire les différents fichiers (voir la liste ci-dessous), donnez leur l'extension ".POR" et installez l'extension ".POR".

- L'option 4 vous permet de sauver les modifications d'installation dans le programme concerné.

- Quitter l'installation de ce programme en pressant 0.

8.2. TRANS_ME

Tous les messages qui apparaissent aussi bien à la création qu'à l'exécution d'une leçon peuvent être modifiés ou traduits par l'intermédiaire de TRANS_ME. Chaque message fait partie d'un enregistrement d'un fichier de messages dont l'extension caractérise la langue utilisée. Voici, l'ensemble des fichiers que vous pouvez modifier ou traduire :

- pour la création d'une leçon :
  + Messages.??? : tous les messages des outils de création à l'exception de ceux de l'éditeur de texte.
  + Graphix.??? : l'aide de l'éditeur graphique.
  + Editerr.??? : les messages de l'éditeur de texte ainsi que l'aide pour ce même éditeur et pour le formateur.
  + Read.??? : le manuel du logiciel
- pour l'exécution d'une leçon :
  + Exec.??? contient tous les messages d'exécution.

Exemple : Traduction des messages en portugais à partir de la version anglaise Messages.ENG. L'utilitaire se trouve sur le drive A (répertoire courant). Le fichier Messages.ENG est localisé dans le sous-répertoire C:\Creation\Messages. Le nouveau fichier créé, Messages.POR se situe dans le même répertoire (figure 2).

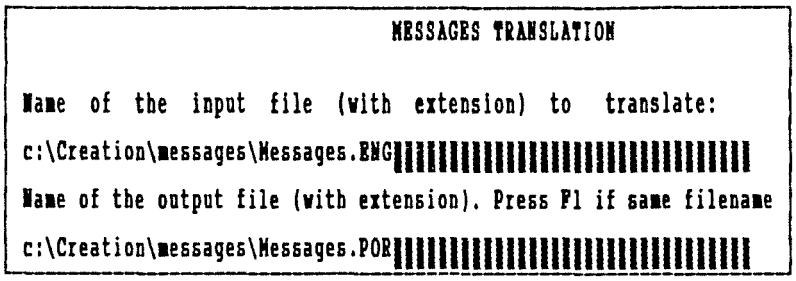


Figure 2 - Première page de TRANS_ME

Les messages s'afficheront par page de 5 à l'écran. La ligne inversée qui suit un message est destinée à recevoir les modifications de ce message (voir figure 3). La traduction ou la modification d'un message se déroule conformément aux opérations décrites au point 5 de ce manuel.

Utilisez les flèches UP, DOWN ou les touches **PGUP**, **PGDN** pour passer d'un message à l'autre. La touche **F10** charge le contenu du message sur la ligne des modifications. Il suffit de déplacer le curseur sur cette ligne pour apporter les corrections au message. Dans un cas de traduction de messages, introduisez directement le nouveau contenu. La touche **F1** vous permet de sauver toutes les modifications. Les messages non modifiés seront recopiés tels quels dans le fichier mis à jour ou traduit. La touche **END** quitte cet utilitaire sans sauver quoi que ce soit. Prudence donc si vous ne voulez pas annihiler des heures de traduction de messages !!!

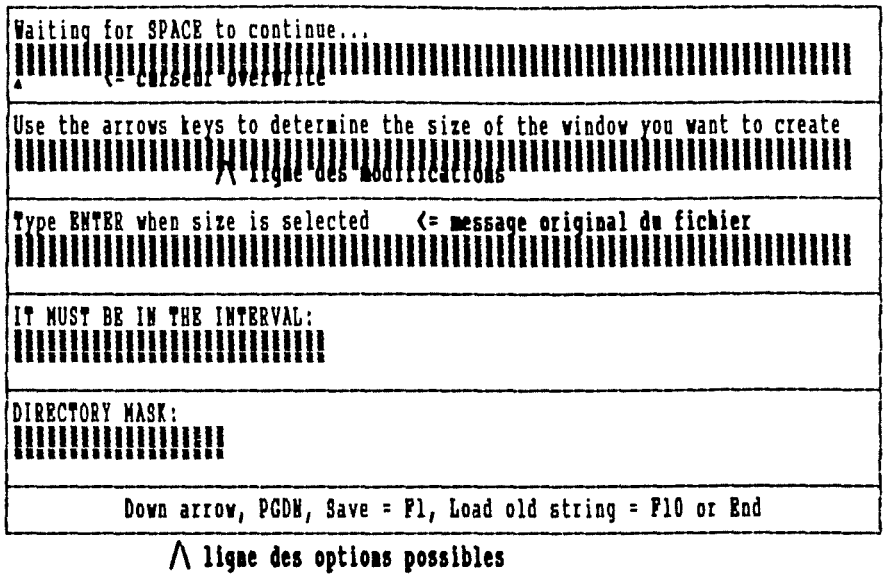


Figure 3 - Page de messages de TRANS_ME

8.3. DEL NODE, COPYNODE, REN NODE

Pour effacer, copier ou renommer un noeud. Pourquoi utiliser ces trois programmes alors que les commandes du MS-DOS permettent d'effacer, de copier ou de renommer un fichier? Tout simplement parce qu'il faut faire autre chose que d'exécuter ces commandes MS-DOS. Le fichier WINDOWS.DIM (sur l'unité "Données") contient les dimensions des fenêtres correspondant aux noeuds. Le but principal de ces utilitaires est d'actualiser ce fichier en effaçant, copiant ou renommant une entrée mais aussi d'effectuer ces mêmes opérations sur les fichiers contenant les noeuds concernés.

Ces trois utilitaires se ressemblent très fort quant aux opérations à effectuer. C'est pourquoi nous les décrirons en même temps. Tout d'abord, vous devez introduire le répertoire du noeud concerné par l'opération. S'il s'agit d'une opération de copie de noeud, vous spécifierez en plus le répertoire vers lequel s'effectuera la copie (figure 4). Donnez ensuite le nom du noeud sans extension. Si vous renommez un noeud, tapez ensuite le nouveau nom (figure 5). Terminez l'opération en tapant le chiffre correspondant au type du noeud concerné.

F1 to go back --- F2 to abort

Source directory of the copy

Destination directory of the copy

Name of the node to copy:

1. Graphic node
2. Simulation node
3. Text node
4. Question node

Your choice ?

Figure 4 - Ecran de COPYNODE

F1 to go back --- F2 to abort

Directory of the node to rename:

Name of the node to rename:

Give now the new name:

1. Graphic node
2. Simulation node
3. Text node
4. Question node

Your choice ?

Figure 5 - Ecran de REN_NODE

F1 to go back --- F2 to abort

Directory of the node to erase :

Name of the node to erase:

1. Graphic node
2. Simulation node
3. Text node
4. Question node

Your choice ?

Figure 6 - Ecran de DEL_NODE

ANNEXE A - Syntaxe des expressions

Nous allons définir ici la syntaxe des expressions utilisées pour décrire le système à simuler aussi bien que les résultats que l'on désire obtenir (sorties graphiques et numériques). Pour ce faire, nous utiliserons le formalisme "Backus-Naur", dont voici les quelques méta-symboles:

::= signifie "est défini comme".
| signifie "ou".
{} la partie incluse peut-être répétée 0 ou plusieurs fois.
.. désigne une des valeurs entières comprises dans l'intervalle défini.

Nous soulignerons les constructions syntaxiques. Les parties prédéfinies seront mises en caractères gras (par exemple **X** ou **entier**).

bl ::= {' '}

exposant ::= **

expression ::= expression1 | expression2

expression1 ::= {bl} opérande {bl} |
{bl} expression1 {bl} opérateur2 {bl} expression1 {bl} |
{bl} opérateur1 {bl} expression1 {bl} |
{bl} ({bl} expression1 {bl}) {bl}

expression2 ::=
{bl} { expression1 op-relat expression1 } {bl}
{ expression1 } {bl} { expression1 }

expression-retard ::= expression-retard1 |
expression-retard 2

expression-retard1 ::= {bl} opérande1 {bl} |
{bl} expression-retard1 {bl} opérateur2 {bl}
expression-retard1 {bl} |
{bl} opérateur1 {bl} expression-retard1 {bl} |
{bl} ({bl} expression-retard1 {bl}) {bl}

expression-retard2 ::=
    {bl} { expression-retard1 op-relat expression-retard1 }
    {bl} { expression-retard1 } {bl} { expression-retard1 }

indice-c ::= **1 .. nombre de constantes**

indice-d ::= **1 .. nombre de variables de retard**

indice-p ::= **1 .. nombre de paramètres**

indice-x ::= **1 .. nombre d'équations**

opérande ::= opérande1 | opérande2

opérande1 ::= S-X {bl} ({bl} indice-x {bl}) |
    S-X {bl} ({bl} S-P {bl} ({bl} indice-p {bl}){bl}) |
    S-X {bl} ({bl} S-X {bl} ({bl} indice-x{bl}) {bl}) |
    S-P {bl} ({bl} indice-p {bl}) |
    S-P {bl} ({bl} S-P {bl} ({bl} indice-p {bl}){bl}) |
    S-P {bl} ({bl} S-X {bl} ({bl} indice-x {bl}){bl}) |
    S-C {bl} ({bl} indice-c {bl}) |
    {bl} **entier** {bl} |{bl} **réel** {bl} |{bl} temps {bl}
    {bl} **PI** {bl}

opérateur2 ::= S-D {bl} ({bl} indice-d {bl})

opérateur1 ::= **-** | **abs** | **cos** | **sin** | **arctan** | **exp** | **ln** |
    frac | **int** | **sqr** | **sqrt**

opérateur2 ::= **+** | **-** | ***** | **/** | **mod** | **div** | exposant

op-relat ::= **<** | **>** | **=** | **>=** | **<=** | **<>**

S-C ::= **C** | **c**

S-D ::= **D** | **d**

S-P ::= **P** | **p**

S-X ::= **X** | **x**

temps ::= **T** | **t**

ANNEXE B - Syntaxe du langage d'enchaînement

Nous allons exprimer celle-ci dans un formalisme tel que l'interprétation de cette "grammaire" de notre langage soit univoque. Le formalisme BNF (Backus-Naur Form) utilise un certain nombre de méta-symboles ne faisant pas partie du langage d'enchaînement :

::= signifie "est défini comme",
| indique une alternative,
{} indique la répétition possible (1 ou n fois),
[] précise l'aspect facultatif.
.. désigne une des valeurs comprises dans l'intervalle défini par le nombre qui précède les 2 points et celui qui suit ces 2 points.
... désigne une des lettres comprises entre les 2 bornes.

Nous soulignerons les constructions syntaxiques. Les mots du langage seront mis en caractères gras.

Syntaxe du language :

LECON ::= enchaînement
amener-fenêtre-au-dessus ::= **GET WINDOW** identificateur-de-fenêtre **UP ;**
coin-sup-gauche-X ::= **0..79** (carte IBM)
| **0..89** (carte HERCULES)
coin-sup-gauche-Y-GRF ::= **0..191** (carte IBM)
| **0..335** (carte HERCULES)
coin-sup-gauche-Y-TXT ::= **0..23**
déclaration-étiquette ::= identificateur :
délai-affichage ::= **1..30**
effacement-écran ::= **CLEAR ALL THE SCREEN ;**
effacement-fenêtre ::= **ERASE WINDOW** identificateur-de-fenêtre **;**
enchaînement ::= {ligne}
étiquette ::= identificateur
exécution-noeud ::= exécution-noeud-animation |
exécution-noeud-graphe |
exécution-noeud-qcm |
exécution-noeud-simulation |
exécution-noeud-texte

exécution-noeud-animation ::= **EXECUTE** nom-noeud.**ANI**
fenêtre-animation ;

exécution-noeud-graphe ::= **EXECUTE** nom-noeud.**GRF**
fenêtre-graphe ;

exécution-noeud-qcm ::= **EXECUTE** nom-noeud.**QCM** fenêtre-texte
[**WITH SIMULATION** fenêtre-graphe] ;

exécution-noeud-simulation ::= **EXECUTE** nom-noeud.**SIM**
fenêtre-graphe ;

exécution-noeud-texte ::= **EXECUTE** nom-noeud.**TXT**
fenêtre-texte ;

fenêtre-animation ::= (identificateur-de-fenêtre,
coin-sup-gauche-X,
coin-sup-gauche-Y-GRF,
nombre-graphe-animation,
délai-affichage)

fenêtre-graphe ::= (identificateur-de-fenêtre,
coin-sup-gauche-X,
coin-sup-gauche-Y-GRF)

fenêtre-texte ::= (identificateur-de-fenêtre,
coin-sup-gauche-X,
coin-sup-gauche-Y-TXT)

identificateur ::= une suite de 1 à 15 caractères dont le code ASCII est compris entre 32 et 126 ou entre 128 et 254

identificateur-de-fenêtre ::= **1** | **2** | **3** | **4** | **5** | **6** | **7**

identificateur-de-fichier-sans-extension ::= une suite de 1 à 8 caractères choisis parmi **a...b**, **A...Z**, **0..9** et **_**

instruction ::= amener-fenêtre-au-dessus |
effacement-écran |
effacement-fenêtre |
exécution-noeud |
instruction-conditionnelle |
instruction-de-branchement |
inversion-deux-dernières-fenêtres

instruction-conditionnelle ::=
instruction-conditionnelle-menu
| instruction-conditionnelle-simple

instruction-conditionnelle-menu ::= **IF MENU CHOICE** option

instruction-conditionnelle-simple ::=
IF premier-membre opérateur second-membre

instruction-de-branchement ::= **GOTO** étiquette ;

inversion-deux-dernières-fenêtres ::=
INVERT THE 2 LAST WINDOWS ;

ligne ::= instruction | déclaration-étiquette

nombre-entier ::= **-32768..32767**

nombre-graphe-animation ::= **2..5**

nom-noeud ::= identificateur-de-fichier-sans-extension

opérateur ::= **<** | **<=** | **>** | **>=** | **=** | **<>**

option ::= **A** | **B** | **C** | **D** | **E**

premier-membre ::= **MCQ POINTS** | **MCQ TRIALS** |
TOTAL POINTS | **TOTAL TRIALS** |
(MCQ POINTS/POSSIBLE MCQ POINTS) |
(TOTAL POINTS/POSSIBLE TOTAL POINTS)

second-membre ::= nombre-entier

ANNEXE C - Répertoire des fichiers

1. A la création

Sur l'unité "programmes" :

PC-CREAT.COM	Le programme principal et ses 8
PC_CREAT.000	fichiers overlays
...	
PC_CREAT.007	
MESSAGES.???	Le fichier des messages du programme, où ??? représente l'extension installée identifiant la langue.
GRAPHIX.???	Le fichier d'aide de l'éditeur graphique ayant la même extension que le fichier des messsages précédents.
READ.???	Le fichier d'aide général de même extension que le fichier Messages.???

Sur l'unité "données" :

- Tous les noeuds créés :
  - *.GRF : les noeuds graphiques et les composants d'une animation
  - *.TXT : les noeuds de type texte
  - *.FRM : les noeuds texte formatés
  - *.QCM : les noeuds questions-réponses
  - *.DTA : les noeuds simulation
- le fichier contenant la dimension de fenêtre de chacun des noeuds de la leçon : Windows.DIM
  - *.ENC : les enchaînements avant compilation
  - *.LES : les leçons compilées prêtes à l'exécution

Sur l'unité "éditeur" :

PC_EDIT.COM	Le fichier principal de l'éditeur et son
PC_EDIT.000	fichier overlay
EDITERR.???	Le fichier des messages et des aides éditeur et formateur

Sur l'unité "commune" :

8X8.FON	Les fichiers contenant les polices de
14X9.FON	caractères correspondant aux différentes
	cartes graphiques
4X6ASCII.FON	

2. A l'exécution

Sur l'unité "exécution" :

PC_EXEC.COM	Le programme principal et ses 7 fichiers
PC_EXEC.000	overlays
...	
PC_EXEC.005	
8X8.FON	Les fichiers contenant les polices de
14X9.FON	caractères correspondant aux différen-
	tes cartes graphiques
EXEC.???	Le fichier des messages où ??? repré-
	sente l'extension installée identifiant
	la langue.

Sur l'unité "données" :

- Tous les noeuds nécessaires à l'exécution de la leçon (voir l'unité données de la partie création ci-dessus) ainsi que le fichier WINDOWS.DIM

- Le fichier "mouchard", boîte noire de l'exécution se trouvera sur cette unité après exécution de la leçon.

- Toutes les sorties numériques des simulations seront dans cette unité après l'exécution. Elles auront le même nom que la simulation à laquelle elles correspondent, et l'extension ".TCH" lorsque c'est la sortie de la simulation que vous avez définie (**TeaCHer**), et ".STD" pour la simulation définie par l'étudiant (**STuDent**).

ANNEXE B

Le manuel de SimEDO

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

INSTITUTO SUPERIOR TÉCNICO
1096 LISBOA CODEX

Manuel de

SiMEDO

Simulateur de systèmes D'Equations Différentielles Ordinaires

S.D. Antunes, Instituto Superior Técnico,
Av. Rovisco Pais, 1096 Lisboa Codex
PORTUGAL

J.M. Barreto, C. Léonard et O. Marchand,
FNDP - Institut d'Informatique,
Rue Grandgagnage 21, 5000 Namur, BELGIQUE

Namur 1988

TABLE DES MATIERES

1. INTRODUCTION

SimEDO offre un moyen souple et rapide pour mettre au point puis tester des systèmes d'équations différentielles ordinaires du premier ordre. Le logiciel propose 3 options à l'utilisateur qui peut :

- créer (ou modifier) une simulation et en définir les sorties graphiques et/ou numériques attendues,

- simuler un modèle par une méthode de résolution numérique du second ou du quatrième ordre, et enfin

- imprimer un rapport complet des caractéristiques du modèle simulé.

Signalons déjà que ce manuel s'adresse aux personnes familières avec la théorie des systèmes d'équations différentielles.

SimEDO, logiciel indépendant, est aussi une pièce maîtresse du logiciel d'EAO ProfCOMP. Il permet au professeur de créer des modèles qu'il pourra intégrer sans problèmes comme noeud simulation dans une leçon. Réciproquement, tout noeud simulation de ProfCOMP peut être testé et modifié dans SimEDO.

SimEDO vous propose un confort d'utilisation : vous pouvez personnaliser, traduire, modifier,... tous les messages du logiciel. Avantage non négligeable, ne fût-ce que pour adapter le vocabulaire du logiciel à la terminologie exacte que vous employez dans le domaine.

Enfin, du côté des utilitaires, vous pouvez installer SimEDO, manipuler les noeuds simulation entre SimEDO et ProfCOMP ou encore créer votre propre version des messages.

2. DEFINITION D'UN MODELE

2.1. Introduction

L'étape de définition d'un modèle de simulation constitue la partie la plus importante puisque c'est elle qui guidera l'exécution du même modèle. L'utilisateur doit donc être attentif et comprendre parfaitement ce que SimEDO lui demande. Ce logiciel simule tous les modèles de la classe des systèmes d'équations différentielles ordinaires du premier ordre (1). La présentation mathématique d'un tel modèle est la suivante :

$$d/dt\ [X(i)] = f_1(t,X(m),P(n),D(q),C(r)) \quad (a)$$

$$P(j) = f_2(t,X(m),P(n),D(q),C(r)) \quad (b)$$

$$D(k) = f_3(t,X(m),P(n),C(r)) \quad (c)$$

$$C(l) = \text{réel} \quad (d)$$

avec

$i,m = 1..$nombre de variables d'état

$j,n = 1..$nombre de paramètres

$k,q = 1..$nombre de variables de retard

$l = 1..$nombre de constantes du modèle

et

$1 \leq$ nombre de variables d'état ≤ 20

$0 \leq$ nombre de paramètres ≤ 20

$0 \leq$ nombre de variables de retard ≤ 4

$0 \leq$ nombre de constantes ≤ 20

Les seules équations différentielles sont celles des variables d'état (a) mais SimEDO autorise également la définition de paramètre(s) (b) et/ou de variable(s) de retard (c) par une équation (non différentielle). Enfin, des constantes (d) peuvent faire partie du modèle. Rassurez-vous, chacun de ces éléments sera expliqué plus en détail dans la suite de ce chapitre.

---

(1) Un système d'ordre supérieur à un peut également être simulé si on peut toujours le ramener à un système d'ordre un.

Chaque fonction f, qui définit le membre de droite de l'équation différentielle de chaque variable d'état ou encore l'équation d'un paramètre ou d'une variable de retard, doit vérifier une certaine syntaxe. En fait, cette fonction f est une expression dans laquelle interviennent les éléments entre les parenthèses de la fonction. Par la suite, nous emploie-rons souvent le terme EXPRESSION en lieu et place de fonction f.

Avant d'être en mesure de créer un modèle de simulation, nous détaillerons tous les composants d'un modèle en analysant la syntaxe des expressions.

2.2. Les expressions

La syntaxe des expressions fait l'objet de l'annexe A. Nous pratiquerons une analyse ascendante de cette syntaxe en détaillant d'abord les composants. Nous verrons alors comment les assembler pour former les différents types d'expression dont nous décrirons l'utilisation précise. Nous terminerons par l'explication des messages d'erreur que SimEDO génère lors de la détection d'incorrection syntaxique.

2.2.1. Les composants d'une expression

La plupart des composants qu'on peut trouver dans une expression figurent déjà dans le modèle mathématique exposé ci-dessus, en paramètres des fonctions f. Nous allons néanmoins les reprendre un par un pour les détailler davantage. Nous examinerons ensuite les opérateurs que l'on peut combiner avec ces opérandes pour former des expressions complexes.

2.2.1.1. Les opérandes

Les variables d'état

Tout modèle d'équations différentielles ordinaires simulable dans SimEDO contient autant de variables d'état que d'équations différentielles. Une variable d'état s'identifie par la lettre X (majuscule ou minuscule) suivie de son indice entre parenthèses. L'ensemble des indices forme un ensemble d'entiers strictement croissants compris entre 1 et le nombre de variables d'état du modèle. Le

cardinal de cet ensemble vaut le nombre de variables d'état du modèle. Le système autorise un modèle d'au plus 20 équations différentielles.
Règle :

n variables d'état <=> $X(i)_{\#\{i\}=n}^{1\le i\le n}$ $\quad \forall i1,i2\in\{i\}: i1<>i2$ $\quad 1\le n\le 20$

Dans la syntaxe, l'opérande variable d'état présente 3 alternatives, à savoir X(indice-x), X(X(indice-x)) et X(P(indice-p)). Au niveau de l'indice, on peut trouver un entier (indice-x) mais aussi la <u>valeur</u> de la variable d'état X(indice-x) ou du paramètre P(indice-p) (nous examinerons l'opérande paramètre ci-dessous). Evidemment, dans ces 2 derniers cas, c'est la VALEUR dynamique de la variable d'état ou du paramètre qui joue le rôle d'indice et qui doit respecter les propriétés exposées ci-dessus.

Les paramètres

La lettre P (minuscule ou majuscule) qui précède un indice entre parenthèses désigne un paramètre du système. En ce qui concerne les indices, on peut faire les mêmes remarques que pour les variables d'état. C'est pourquoi nous donnerons simplement la règle adaptée aux paramètres sans faire davantage de commentaires :

m paramètres <=> $P(j)_{\#\{j\}=m}^{1\le j\le m}$ $\quad \forall j1,j2\in\{j\}: j1<>j2$ $\quad 0\le m\le 20$

Remarquez cependant qu'un système ne comporte pas obligatoirement de paramètre : 0≤m≤20.
La syntaxe propose encore 3 représentations pour l'indice : un entier indice-p, la VALEUR dynamique d'une variable d'état X(indice-x) ou d'un paramètre P(indice-p). Quelle que soit cette représentation, l'indice devra toujours être compris dans l'ensemble strictement croissant d'entiers compris entre 1 et le nombre de paramètres (quand il y a des paramètres définis dans le système).
<u>Remarque</u> : étant donné que la valeur d'une variable d'état ou d'un paramètre est un nombre réel, SimEDO en prend la partie entière lorsqu'ils interviennent comme indice.

Les variables de retard

On identifie une variable de retard (en anglais delay) par la lettre D et son indice entre parenthèses. Un modèle peut contenir jusqu'à 4 variables de retard. L'indice respecte une règle similaire à celle des variables d'état ou des paramètres :

$$r \text{ var de retard} \Leftrightarrow D(k) \quad 1 \le k \le r \quad \#\{k\}=r \qquad \forall k1,k2 \in \{k\} : k1 \neq k2 \quad 0 \le r \le 4$$

Seul changement : l'indice est uniquement un nombre entier compris entre 1 et le nombre de variables de retard du modèle.

Qu'est-ce qu'une variable de retard ?

Dans le modèle mathématique, on découvre qu'une variable de retard est définie par une équation dont l'expression à droite du signe égal lui donne sa valeur, mais avec un certain retard, exprimé comme multiple de l'incrément de temps entre 2 pas d'intégration. La valeur à l'itération courante de cette variable vaut la valeur qu'avait l'expression qui la définit au temps courant MOINS le retard. Tant que la valeur du temps n'a pas atteint le retard à la résolution, la variable a une valeur nulle.

Les constantes

Les derniers composants indicés sont les constantes représentées par la lettre C.
Règle :

$$q \text{ constantes} \Leftrightarrow C(l) \quad 1 \le l \le q \quad \#\{l\}=q \qquad \forall l1,l2 \in \{l\} : l1 \neq l2 \quad 0 \le q \le 20$$

L'indice est uniquement représenté par une valeur entière.
Pourquoi avoir introduit ces composants indicés alors que la syntaxe autorise un nombre entier ou réel comme opérande ?
A l'exécution d'un modèle, l'utilisateur peut à tout moment stopper la simulation, changer les valeurs des constantes (de même d'ailleurs que les conditions initiales des variables d'état), ce qui a pour effet de modifier toutes les expressions où elles interviennent. Une nouvelle simulation peut alors redémarrer pour observer le changement

de comportement.

Le temps

La valeur réelle du temps de l'itération courante est symbolisé par la lettre T ou t.

Un nombre entier ou réel

Tout nombre entier est compris entre les valeurs -32768 et 32767 tandis que l'intervalle permis pour un réel s'étend de -10^{38} à 10^{38} avec une mantisse d'au plus 11 chiffres significatifs.

La constante π

π est représenté par le mot PI ou pi.

2.2.1.2. Les opérateurs

Pour combiner les opérandes et former des expressions complexes, nous avons besoin d'autre chose : les opérateurs.

2 types d'opérateurs

- Les opérateurs à un argument

  L'argument se trouve toujours après l'opérateur.

- Les opérateurs à deux arguments

  Dans ce cas, le premier argument se trouve avant l'opérateur et le second après.

SimEDO utilise donc la notation infixée.
L'usage des parenthèses et des blancs est facultatif mais recommandé pour une meilleure lisibilité des expressions.

Les opérateurs et leur priorité

A chaque opérateur, on associe une priorité, exprimée par un nombre entier afin d'évaluer correctement les expressions. Nous reparlerons de l'évaluation d'une expression dans la suite de ce chapitre.

+ : Priorité : 5
- : Priorité : 5 ou 2 si l'opérateur n'est associé qu'à un seul opérande.
* : Priorité : 4
/ : Priorité : 4

ABS : donne la valeur absolue de l'argument. Le résultat est du même type que l'argument (entier ou réel). Priorité : 2
ARCTAN : renvoie l'angle, exprimé en radians, dont la tangente est l'argument. L'argument est de type entier ou réel et le résultat est de type réel. Priorité : 2
COS : renvoie le cosinus de l'argument, exprimé en radians et dont le type est entier ou réel. Résultat de type réel. Priorité : 2
SIN : renvoie le sinus de l'argument, exprimé en radians et dont le type est entier ou réel. Résultat de type réel. Priorité : 2
EXP : donne l'exponentielle de son argument, entier ou réel. Résultat réel. Priorité : 2
FRAC : extrait la partie fractionnelle de l'argument (entier ou réel). Résultat de type réel. Priorité : 2
LN : renvoie le logarithme naturel de l'argument (entier ou réel). Résultat réel. Priorité : 2
INT : renvoie la partie entière de l'argument, c'est-à-dire le plus grand entier inférieur ou égal à l'argument si l'argument est positif ou nul, ou le plus petit entier supérieur ou égal à l'argument si l'argument est négatif. Résultat réel. Priorité : 2
SQR : élève l'argument (entier ou réel) au carré. Résultat de même type que l'argument. Priorité : 2
SQRT : donne la racine carrée de l'argument (entier ou réel). Résultat de type réel. Priorité : 2
DIV : effectue la division de deux arguments entiers. Résultat de type entier. Priorité : 4
MOD : renvoie le reste de la division du premier argument (entier) par le second (entier). Résultat de type réel. Priorité : 4
****** : renvoie la valeur du premier argument (entier ou réel) avec le deuxième argument comme exposant (entier ou réel). Priorité : 3

Les opérateurs relationnels sont les suivants: **<**, **<=**, **>**, **>=**, **=**, **<>**. Leur priorité est 6. Nous en examinerons l'utilité ci-dessous.

2.2.2. Les types d'expressions et leur utilisation

Jusqu'à présent, nous avons énoncé une liste d'opérandes et d'opérateurs. Examinons maintenant les types d'expressions qu'on peut former avec ces éléments et surtout, quand on utilise ces expressions.

La syntaxe définit 2 types d'expressions qui contiennent chacune 2 alternatives. En fait, toutes les expressions qui vérifient syntaxiquement une des alternatives de "expression" vérifient aussi "expression-retard". La seule différence entre les 2, c'est que "expression-retard" admet un opérande de plus, à savoir les variables de retard : "expression" admet les opérandes "opérande1" tandis que "expression-retard" admet "opérande" qui regroupe "opérande1" et "opérande2". Cette distinction empêche donc l'utilisation de variables de retard dans certaines expressions.

Intéressons-nous davantage aux alternatives de "expression-retard" :

- "Expression-retard1" désigne ce que nous appellerons par la suite une expression normale où interviennent des opérandes et des opérateurs (non relationnels) dans une combinaison syntaxiquement correcte.

Exemples : X(2)
(1-X(1)**2)*X(2)-X(1)+.5*COS(6.28*T/5.9)
1-D(2)+X(1) mod P(X(1))+LN P(5)

- "Expression-retard2" décrit des expressions plus complexes appelées expressions conditionnelles, composées de 3 sous-expressions, chacune entre accolades. La première sous-expression se décompose en 3 parties : 2 "expression-retard1" séparées par un opérateur relationnel qui permet de définir une condition qui sera évaluée à chaque itération de l'exécution du modèle. Si la condition est évaluée à vrai, la seconde expression de type "expression-retard1" sera évaluée et représentera le résultat de l'expression complète. Dans le cas contraire, c'est la troisième sous-expression qu'on évaluera.

Exemple : L'expression {T > 50} {X(1)} {X(2)} peut être interprétée comme suit :

SI le temps est supérieur à 50,
ALORS l'expression prend la valeur de X(1)
SINON l'expression prend la valeur de X(2)

On peut donner les mêmes explications pour les alternatives de "expression", avec la restriction qu'on ne peut y trouver de variables de retard comme opérande.

Dans quel cas l'expression introduite doit-elle être conforme à "expression" ou "expression-retard1" ?

Dans l'étape de définition d'un modèle, vous devez spécifier l'équation différentielle de chaque variable d'état. Le premier membre de cette équation s'affichera à l'écran, il restera à en définir le second par l'expression adéquate.
Exemple : d/dt [X(1)] = ...

Après les variables d'état, c'est le second membre de l'équation de chaque paramètre qu'il faut définir par une expression, du moins si le système en utilise.
Exemple : P(1) = ...

Enfin, une sortie graphique demande les expressions dont les valeurs seront représentées sur chaque axe et une sortie numérique interroge sur les expressions intéressantes pour lesquelles l'utilisateur veut conserver les valeurs dans un fichier.

TOUTES CES EXPRESSIONS S'ACCORDENT AVEC LA SYNTAXE DE "expression-retard".

Reste le seul cas où on ne peut utiliser de variable de retard : D(i) = ... $(1 \leq i \leq 4)$
L'EXPRESSION QUI DEFINIT LE MEMBRE DROIT DE L'EQUATION D'UNE VARIABLE DE RETARD NE PEUT CONTENIR DE VARIABLE DE RETARD et vérifie, par conséquent la syntaxe de "expression".

2.3. Explication des messages d'erreur concernant la syntaxe d'une expression

SimEDO réagit dès que vous commettez une erreur lors de l'introduction d'une expression. Voici tout ce qu'il peut vous dire :

* UNE EXPRESSION D'UN SEUL ELEMENT DOIT ETRE UN OPERANDE

Une expression qui ne contiendrait qu'un opérateur n'a aucun sens.

Exemple : 4+MOD
?...

l'expression MOD devrait être un opérande.

* PARENTHESE OUVRANTE SANS PARENTHESE FERMANTE

Dans une expression, le nombre de parenthèses ouvrantes doit toujours être égal au nombre de parenthèses fermantes sinon, l'expression ne peut être évaluée.

Exemples : P(1+4*C(2) MOD 3
?...

(4*P(1)+2**P(2)-X(1)
?...

* PARENTHESE FERMANTE SANS PARENTHESE OUVRANTE

Même remarque que pour l'erreur précédente.

Exemple : P(1)+2**X(4)) MOD C(1)
...?

* OPERATEUR A UN OPERANDE, EXPRESSION GAUCHE NON VIDE

L'opérande d'un tel opérateur se trouve toujours à droite de l'opérateur. Si un élément se trouve à gauche de cet opérateur, ce ne peut être qu'un autre opérateur sinon ce serait un opérateur à deux opérandes.

Exemple : 4SIN(2*P(1)*PI)
?...

* EXPRESSION DROITE VIDE

C'est le cas d'un opérateur qu'il soit à un ou deux opérandes pour lequel le deuxième opérande est indéfini.

Exemples : 2+
?...
2+3 MOD
?...

* OPERATEUR A EXPRESSION GAUCHE VIDE

C'est aussi le cas d'un opérateur à deux opérandes, mais pour lequel le premier opérande est indéfini.

Exemple : *3
...?

* RIEN DANS LES PARENTHESES

L'usage des parenthèses est facultatif (sauf pour les variables indicées ou pour changer la priorité d'un opérateur) mais il ne faut pas abuser de leur usage dans un sens opposé.

Exemple : 3*()
?...

* EXPRESSION DE PLUSIEURS ELEMENTS SANS OPERATEUR

Une telle expression contient plusieurs opérandes contigus or, entre deux opérandes, il faut toujours au moins un opérateur à deux opérandes.

Exemple : P(1) C(2) X(3)
?... ?...

* SECOND OPERATEUR RELATIONNEL LU OU NON AUTORISE

Une expression conditionnelle ne peut contenir qu'un seul opérateur relationnel situé dans la première sous-expression entre accolades. Les autres sous-expressions ne peuvent avoir d'opérateur relationnel.

Exemple : {0<=X(1)<=5} {t} {-t}
?...

* STRUCTURE D'UNE EXPRESSION CONDITIONNELLE INCORRECTE

Une expression conditionnelle peut être incorrecte pour plusieurs raisons: il manque une sous-expression, un nombre impair d'accolades ...

Exemple : {X(1)>10} {t}
?...

* EXPRESSION VIDE

2.4. L'interface SimEDO - utilisateur

Tout ce que SimEDO demande à l'utilisateur, c'est soit d'introduire manuellement une chaîne de caractères, la valeur d'un entier, d'un réel, soit de choisir parmi plusieurs options proposées. Le logiciel a standardisé toutes ces opérations.

Introduction manuelle

La demande d'introduction d'un entier, d'un réel ou d'une chaîne de caractères se présente toujours de la même manière. Un message renseigne la donnée à définir puis s'affiche un rectangle en vidéo inversé. Si une valeur par défaut doit être mentionnée pour la donnée, celle-ci s'affiche dans la partie gauche du rectangle qui se réduit d'une longueur égale à celle que la valeur par défaut occupe.

Exemple :

Donnée: valeur par défaut

Si aucune valeur par défaut n'est associée à "donnée", on aura l'affichage :

Donnée:

La longueur du rectangle inversé détermine le nombre de caractères que l'utilisateur peut encore taper avant d'atteindre la longueur maximale imposée par le logiciel. Un triangle pointé vers le haut symbolise le curseur en mode "overwrite" (surfrappe) tandis qu'un pique le représente en mode insertion. Quand l'utilisateur doit introduire un nombre entier ou réel, le curseur se positionne sur le premier caractère de la valeur par défaut (si elle existe). Dans tous les autres cas, le curseur se place sur la première position à la gauche du rectangle.

Chaque fois que l'utilisateur frappe un caractère, celui-ci s'inscrit à l'endroit désigné par le curseur, le rectangle se rétrécit de la longueur du caractère et le curseur passe à la position suivante.

Les flèches **GAUCHE** et **DROITE** permettent le déplacement du curseur dans les caractères introduits (ou de la valeur par défaut). Si le triangle pointe un caractère et que l'utilisateur enfonce la touche d'un autre caractère, celui-ci remplacera l'ancien : c'est le mode surfrappe. La même opération en mode insertion déplacera tous les caractères, à partir de la position courante du curseur, d'une position vers la droite et insèrera le caractère frappé à l'endroit du curseur. Enfin, le curseur avancera d'une position vers la droite.

Le passage du mode insertion à surfrappe (et réciproquement) se réalise par pression de la touche **INS**.

La touche **DEL** permet d'effacer le caractère sous le curseur tandis que la touche |<- efface le caractère à gauche du curseur. Dans les 2 cas d'effacement, le rectangle en vidéo inversé s'aggrandit d'une position à gauche puisqu'il récupère la place du caractère effacé.

Un bip sonore signale toute erreur de manipulation.

Certaines touches du clavier acquièrent une signification particulière lors de la définition de données. Voici, en résumé, la liste des touches dont il faut connaître l'utilisation pour entrer les données le plus efficacement possible :

INS : passer du mode insertion à surfrappe et réciproquement

DEL : efface le caractère au-dessus du curseur

|<- : efface le caractère à gauche du curseur

->,<- : déplacement du curseur dans les caractères introduits.

F1 : un pas en arrière dans la définition des données, c'est-à-dire que l'utilisateur revient à la définition de la dernière donnée introduite.

F2 : abandon de l'introduction des données et retour au menu principal

F5 : affiche le répertoire des modèles de simulation lors de la définition du nom de la simulation.

END : termine l'introduction des données et effectue un sauvetage si elles ont été toutes définies. Toute pression de la touche END alors que les données du modèle n'ont pas toutes été définies équivaut à l'effet de **F2**.

Remarque : l'effet d'une touche spécifique peut être inhibé à certains moments. Par exemple, la touche **F5** n'a d'effet que lorsqu'on définit le nom d'un modèle.

Choix parmi plusieurs options

Quand l'utilisateur a le choix parmi plusieurs options, SimEDO les présente sous forme de menu. Un rectangle en vidéo inversé indique l'option courante. Pour sélectionner l'option désirée, il suffit de déplacer le rectangle sur celle-ci à l'aide des flèches (ou de la barre d'espacement lorsqu'il n'y a que 2 options possibles) et de confirmer par **RETURN**.

2.5. Les étapes de la création d'une simulation

La création d'un modèle de simulation constitue toujours la première étape d'utilisation du logiciel. La définition des exigences de l'utilisateur s'effectuera par une série de questions ou de demandes d'introduction d'éléments bien précis. Nous allons passer en revue toutes les opérations nécessaires à la création d'un modèle complet. Quand il s'agit d'introduire un nombre entier ou réel, le logiciel compare toujours cette valeur aux bornes qu'il admet avant de l'accepter ou de la refuser. Nous indiquerons les bornes admises de la manière suivante pour un entier :

valeur minimale ≤ entier ≤ valeur maximale

Quand il s'agit d'un réel, le mot réel remplacera le mot entier mais ce sera la même représentation.

On vous demandera d'introduire, dans l'ordre:

* Le nom sous lequel les informations de la simulation seront sauvées. Si vous introduisez un nom de simulation déjà existant, vous pourrez soit modifier soit redéfinir le modèle déjà créé. Pressez la touche **F5** pour obtenir la liste de toutes les simulations déjà créées.

* La taille de la fenêtre graphique qui sera utilisée à l'exécution pour représenter le(s) graphique(s) désiré(s). La longueur (X) et la largeur (Y) de la fenêtre courante sont affichées dans le coin supérieur gauche de l'écran et la fenêtre qui correspond à ces dimensions apparaît également à l'écran. Utilisez les flèches pour en faire varier la taille. En cas de modification d'un modèle existant, vous pouvez accepter les dimensions définies précédemment ou en redéfinir de nouvelles.

* La méthode de résolution du système. SimEDO propose 2 méthodes de résolution numériques : Euler amélioré (deuxième ordre) ou Runge-Kutta (quatrième ordre).

* Le nombre de variable(s) d'état. Tout système comprend au moins une variable d'état et au plus 20 variables d'état (!!! 1 à 8 pour ProfCOMP !!!).

* Le nombre de paramètres du système (0≤entier≤20) (!!! 0 à 8 pour ProfCOMP!!!).

* Le nombre de constantes du système (0≤entier≤20) (!!! 0 à 10 pour ProfCOMP!!!).

* Le nombre de variables de retard (0≤entier≤4).

* Le temps du début de simulation, nécessaire pour calculer la première itération du système ($-10^5 \leq$ réel $\leq 10^5$).

* L'incrément de temps entre deux itérations successives ($10^{-20} \leq$ réel $\leq 10^4$).

* Les équations différentielles des variables d'état, puis les équations des paramètres, s'il y en a, par ordre croissant d'indice (voir le paragraphe sur les expressions).

* Les valeurs des constantes, s'il y en a, par ordre croissant d'indice ($-10^{20} \leq$ réel $\leq 10^{20}$)

* Pour chaque variable de retard, l'expression qui la définit et le retard associé (incrément de temps≤réel≤50 * incrément de temps). SimEDO accepte donc un retard maximum de 50 pas d'intégration.

* Les valeurs initiales des variables d'état ($-10^{20} \leq$ réel $\leq 10^{20}$).

* Les valeurs initiales des paramètres (s'il y en a) dont l'équation fait appel à des indices de paramètres supérieurs au leur ($-10^{20} \leq$ réel $\leq 10^{20}$).

Exemple : P(1)= X(2)+P(5) et P(4)= P(2)+P(3) ; L'équation de P(1) fait appel au paramètre P(5). P(5) est défini après P(1) et donc une valeur initiale est requise pour P(1). Par contre, la définition de P(4) fait intervenir les paramètres d'indice 2 et 3, définis juste avant. Il est donc inutile de demander une valeur initiale pour P(4) puisqu'on connaît les valeurs initiales de P(2) et P(3).

* Le type de résultat désiré à l'exécution:
- une sortie graphique qui représentera une ou plusieurs courbes, selon vos souhaits.
- une sortie numérique qui reprendra dans un fichier de type texte les valeurs d'expression(s) qui vous parais-

se(nt) intéressantes.

- Les deux types de sorties décrites précédemment.

* Si une sortie graphique a été demandée, on vous demande:

- si vous voulez relier tous les points successifs d'une courbe ou les afficher tout simplement comme des points isolés.
- le nombre de courbes à dessiner à l'exécution.
- la fréquence de représentation des points à l'affichage ($1 \leq entier \leq 100$). Si vous voulez une fréquence de représentation de N, un point sur N sera représenté graphiquement.
- pour chaque courbe,
    + l'expression dont les résultats de l'évaluation seront représentées sur l'axe des abscisses (expression).
    + l'expression dont les résultats de l'évaluation seront représentés sur l'axe des ordonnées.
    + les valeurs minimale ($-10^{35} \leq réel \leq 10^{34}$) et maximale (valeur minimale + $10^{-3} \leq réel \leq 10^{35}$) de l'intervalle de représentation des points en abscisse, puis les mêmes valeurs pour l'axe des ordonnées.

* Si une sortie numérique a été demandée :

-le nombre d'expression(s) dont on doit conserver les valeurs dans le fichier texte ($1 \leq entier \leq 3$), puis
- la définition des expressions dont les valeurs seront écrites dans le fichier des résultats numériques.
- la fréquence d'écriture des valeurs des équations dans le fichier.

* L'intervalle de temps au cours duquel une représentation graphique ou numérique est demandée.

3. L'EXECUTION D'UN MODELE

3.1. Introduction

Pour exécuter une simulation existante, il vous suffit d'introduire son nom. La touche **F5** visualise toutes les simulations que vous pouvez exécuter. Si une ou plusieurs courbes ont été demandées, celles-ci s'afficheront à l'écran; au fur et à mesure que la résolution évolue, le temps de l'itération courante apparaîtra dans le coin supérieur gauche de la fenêtre graphique, sous la forme d'un réel. Si une sortie numérique a été sélectionnée, un fichier de même nom que la simulation, avec l'extension "DAT" sera créé sur l'unité "données" et contiendra les valeurs exigées par l'utilisateur. Vous pouvez interrompre une simulation en cours pour imprimer le graphique **F8** (votre imprimante devra être configurée OBLIGATOIREMENT en mode **EPSON**), changer les valeurs initiales des variables et/ou des constantes **F6**, effacer les courbes déjà tracées **F9**, reprendre la simulation **F7** ou retourner au menu principal (toute autre touche).

3.2. Les méthodes de résolution et évaluation d'une expression

Pour résoudre un système d'équations différentielles du premier ordre, SimEDO propose 2 méthodes de résolution numérique. Soit l'équation d/dt[X(1)] = f(t,X(1)) à résoudre et δt l'incrément de temps entre 2 pas d'intégration. On dispose de la valeur de X(1) au temps t, notée X_m, et on cherche la valeur de X(1) au temps t+δt, notée X_{m+1}. Voici comment les 2 méthodes du logiciel calculeront cette nouvelle valeur :

- **EULER AMELIORE** (ordre 2)

$$X_{m+1} = X_m + \delta t/2 * f(t,X_m) + f(t + \delta t, X_m + \delta t * X_m')$$

Cette méthode est recommandée pour des calculs rapides mais l'erreur de troncature est mauvaise.

- RUNGE-KUTTA DU QUATRIEME ORDRE

$$X_{m+1} = X_m + 1/6 * [k_1 + 2*k_2 + 2*k_3 + k_4]$$

avec $k_1 = \delta t * f(t, X_m)$

$k_2 = \delta t * f(t + \delta t/2 , X_m + k_1/2)$

$k_3 = \delta t * f(t + \delta t/2 , X_m + k_2/2)$

$k_4 = \delta t * f(t + \delta t , X_m + k_3)$

Cette méthode est l'une des plus utilisées pour la résolution numérique des systèmes d'équations différentielles et est souvent suffisante pour obtenir un résultat correct. C'est la raison pour laquelle SimEDO se contente de ces 2 méthodes.

Quelle que soit la méthode de résolution choisie pour simuler un modèle, SimEDO doit évaluer la fonction f plusieurs fois pour une seul pas d'intégration. C'est donc l'expression qui figure à droite du signe égal de l'équation différentielle de chaque variable d'état qu'il faut évaluer.

Remarque : quand le modèle contient des paramètres, des variables de retard, il faut en évaluer l'expression à chaque pas d'intégration, de même d'ailleurs que les expressions d'une sorties numériques et/ou graphiques.

f peut être une expression normale ou conditionnelle. Envisageons les 2 cas d'évaluation :

- f est une expression normale

L'évaluation tient compte de la priorité des opérateurs. La règle adoptée est la suivante :

L'opérateur le plus prioritaire est celui qui a la priorité la plus élevée. Si 2 opérateurs sont de niveau de priorité identique, tout se passe comme si l'opérateur le plus à droite avait une priorité supérieure.

Exemple : l'expression X(1) + ARCTAN(P(1)) / C(2) MOD D(1) sera interprétée comme suit :

X(1) + (((ARCTAN(P(1)) / C(2)) MOD D(1))

Le "+" est l'opérateur le plus prioritaire. "MOD" et "/" sont de priorité identique mais "MOD" le plus à droite, est considéré comme prioritaire par rapport à "/" (règle d'association à droite).

- f est une expression conditionnelle

Une expression conditionnelle se compose de 3 sous-expressions. La première comporte un opérateur relationnel qui est le plus prioritaire dans l'expression. On peut évaluer la condition qu'il définit d'après la valeur courante de chaque opérande qui y figure. Si la condition est vérifiée, on évalue la sous-expression normale (cfr ci-dessus) qui se trouve à l'intérieur de la deuxième paire d'accolades pour fournir le résultat global de l'expression. Dans le cas contraire, c'est la troisième sous-expression qui joue ce rôle.

3.3. Contrôles sur les opérateurs à l'exécution

/: si le deuxième argument de la division est nul, un message d'erreur vous signale la division par zéro et la simulation est arrêtée.

DIV, MOD: si la valeur d'un argument de la division entière ou de l'opération MODULO n'est pas comprise dans l'intervalle des valeurs permises pour un entier (32767<=entier<=32767) ou si la valeur du deuxième argument est nulle, un message d'erreur apparaît et la simulation est arrêtée. Si un argument est de type réel, sa valeur est tronquée à l'entier correspondant.

******: 1. Si le deuxième argument est de type réel, sa valeur est tronquée à l'entier correspondant.

2. Si la valeur de ce deuxième argument n'est pas comprise dans l'intervalle des valeurs permises pour un entier, un message d'erreur apparaît et la simulation est arrêtée.

3. Si le deuxième argument (l'exposant) est de signe négatif, on effectue le calcul comme si l'exposant était positif. Si ce résultat est nul, on se retrouve dans le cas d'une division par zéro, un message d'erreur apparaît et la simulation s'arrête, sinon le résultat est inversé.

Exemple : (a>0) (b>0)

```
a**-b --> calcul de a**b
  si a**b=0 --> erreur
  sinon résultat=1/a**b
```

3.4. Contrôles sur les indices de variables d'état et de paramètres à l'exécution

Si la valeur d'indice d'une variable d'état X, d'un paramètre P ou d'une constante désigne une variable, un paramètre ou une constante non défini, un message d'erreur apparaît et la simulation est arrêtée.

Exemple : Le système comporte 2 équations; 2 paramètres sont définis.

P(1)=3

On demande, dans une équation, d'évaluer X(P(1))=X(3).

X(3) n'étant pas défini, il n'est pas possible d'évaluer sa valeur.

3.5. Problème d'overflow

La valeur maximale qu'un réel puisse avoir est environ 10^{38}. Dès que cette valeur est dépassée, un message d'erreur apparaît à l'écran et la simulation est arrêtée. Dans ce cas, vous êtes priés de revoir votre simulation puisque vous êtes en présence d'un système instable !!!

4. RAPPORT DES CARACTERISTIQUES DU SYSTEME SIMULE

L'option 3 du menu principal imprime le rapport complet d'un système simulé. Vous devez seulement introduire le nom de la simulation dont vous souhaitez avoir le rapport, allumer votre imprimante et SimEDO se charge du reste. Pour plus de détails sur la présentation de ce rapport, lisez l'annexe D du manuel.

5. LES UTILITAIRES

Le premier utilitaire est propre à SimEDO puisqu'il s'agit de son installation. Les suivants sont communs à SimEDO et ProfCOMP pour manipuler les noeuds simulation ou traduire les messages du logiciel.

5.1. INST_SIM : le programme d'installation de SimEDO.

Au début, vous devez introduire le chemin d'accès au programme SimEDO, s'il ne se situe pas sur le drive et/ou le répertoire courant. Dans le cas contraire, pressez **ENTER**. Si le fichier SimEDO.COM n'est pas localisé à l'endroit désigné, l'installation se termine immédiatement. Sinon, le menu suivant s'affiche (figure 2):

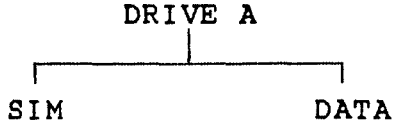
```
SimEDO - INSTALLATION MENU

         1 - Drive unit
         2 - Color
         3 - Messages file extension
         4 - Save
         0 - Quit
```

Figure 2 - Menu de INST_SIM

* Si vous pressez la touche 1, vous pouvez installer les unités "programmes" et "données", c'est-à-dire donner le chemin d'accès pour que SimEDO puisse atteindre les fichiers auxiliaires nécessaires à une bonne exécution (c'est l'unité "programmes") et les fichiers de données (l'unité "données"). Pour plus de détails sur le contenu de ces 2 unités, référez-vous à l'annexe C.

Exemple : Vous disposez du drive A pour exécuter SimEDO. Les fichiers de l'unité "programmes" se trouvent dans un répertoire SIM. Les données se trouvent dans un autre répertoire DATA.



Dans ce cas, introduisez "A:\SIM\" pour l'unité "programmes" et "A:\DATA\" pour l'unité "données". N'oubliez pas de terminer par "\".

* Le point 2 du menu vous permet de choisir la couleur et l'arrière-fond du programme. L'écran affichera toutes les combinaisons possibles de couleurs et d'arrière-fonds. Un nombre entier identifie chaque possibilité. La première ligne de l'écran donne l'identifiant de la couleur courante. A vous de choisir la couleur idéale en introduisant le numéro correspondant. Attention, certaines combinaisons n'auront pas le même effet à l'exécution car certaines cartes ne changent pas le fond en mode graphique.

* Tous les messages qui apparaissent aussi bien à la création qu'à l'exécution d'une simulation se trouvent dans un fichier nommé SimEDO. Grâce à l'utilitaire TRANS_ME, vous pouvez corriger ou traduire les messages à votre guise. C'est l'extension du fichier des messages qui caractérise la langue utilisée. L'option 3 du menu vous permet de définir cette extension.
Exemple : Le nom de la version anglaise du fichier des messages: SimEDO.ENG; l'extension de la version française: "FRA". Si vous voulez converser avec SimEDO en langue française, installer l'extension "FRA". SimEDO.FRA devra alors se trouver sur l'unité "programmes".

* L'option 4 met à jour le fichier SimEDO.COM d'après les modifications que vous avez introduites lors de l'installation.

* Pressez la touche 0 pour quitter INST_SIM.

5.2. TRANS_ME

Tous les messages qui apparaissent aussi bien à la création qu'à l'exécution d'une simulation peuvent être modifiés ou traduits par l'intermédiaire de TRANS_ME. Chaque message fait partie d'un enregistrement du fichier SimEDO, dont l'extension caractérise la langue utilisée.
Exemple : Traduction des messages en portugais à partir de la version anglaise SimEDO.ENG. L'utilitaire se trouve sur le drive A (répertoire courant). Le fichier SimEDO.ENG est localisé dans le sous-répertoire "C:\SimEDO\Messages". Le nouveau fichier créé, SimEDO.POR se situe dans le même

répertoire (figure 3).

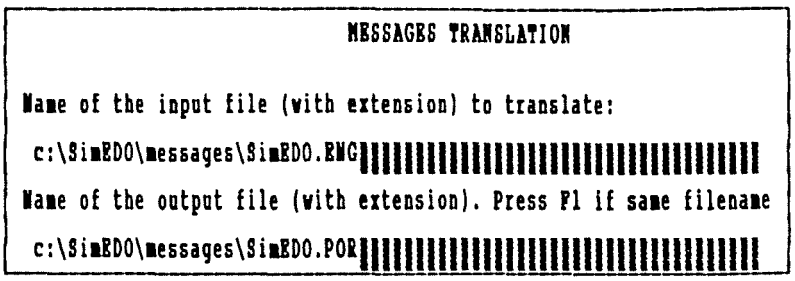


Figure 3 - Première page de TRANS_ME

Les messages s'afficheront par page de 5 à l'écran. La ligne inversée qui suit un message est destinée à recevoir les modifications de ce message (voir figure 4). La traduction ou la modification d'un message se déroule conformément aux opérations décrites au point 2.4. de ce manuel.
Utilisez les flèches **UP**, **DOWN** ou les touches **PGUP**, **PGDN** pour passer d'un message à l'autre. La touche **F10** charge le contenu du message sur la ligne des modifications. Il suffit de déplacer le curseur sur cette ligne pour apporter les corrections au message. Dans un cas de traduction de messages, introduisez directement le nouveau contenu. La touche **F1** vous permet de sauver toutes les modifications. Les messages non modifiés seront recopiés tels quels dans le fichier mis à jour ou traduit. La touche **END** quitte cet utilitaire sans sauver quoi que ce soit. Prudence donc si vous ne voulez pas annihiler des heures de traduction de messages !!!

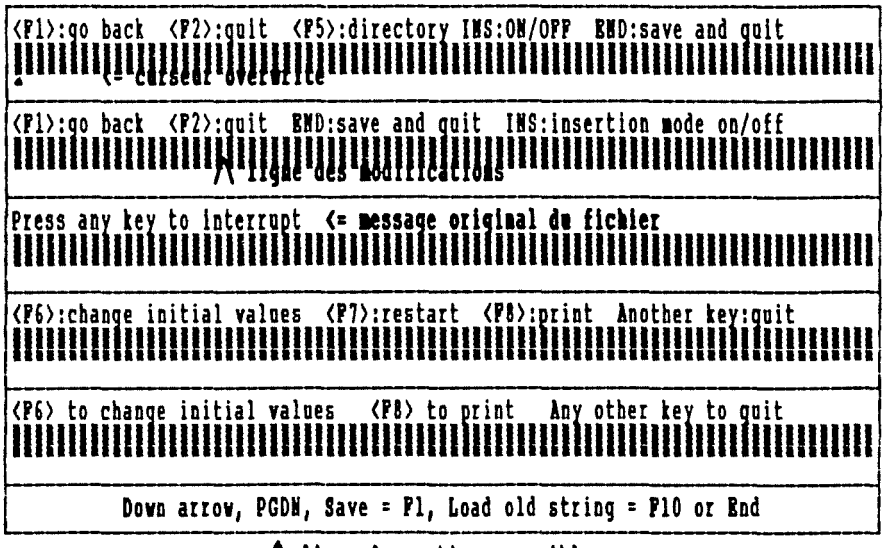


Figure 4 - Page de messages de TRANS_ME

5.3. DEL_NODE, COPYNODE, REN_NODE

Pour effacer, copier ou renommer un noeud. Pourquoi utiliser ces trois programmes alors que les commandes du MS-DOS permettent d'effacer, de copier ou de renommer un fichier? Tout simplement parce qu'il faut faire autre chose que d'exécuter ces commandes MS-DOS. Le fichier WINDOWS.DIM (sur l'unité "Données") contient les dimensions de la fenêtre de chaque noeud. Le but principal de ces utilitaires est d'actualiser ce fichier en effaçant, copiant ou renommant une entrée mais aussi de d'effectuer ces mêmes opérations sur les fichiers qui contiennent les noeuds concernés.

Ces trois utilitaires se ressemblent très fort quant aux opérations à effectuer, c'est pourquoi nous les décrirons en même temps. Tout d'abord, vous devez introduire le répertoire du noeud concerné par l'opération. S'il s'agit d'une opération de copie de noeud, vous spécifierez le répertoire vers lequel s'effectuera la copie (figure 5). Donnez ensuite le nom du noeud sans extension. Si vous renommez une simulation, tapez le nouveau nom (figure 6). Comme ces utilitaires peuvent également être utilisés avec le logiciel d'EAO, vous devez choisir le type de noeud, simulation en l'occurence dans SimEDO.

```
F1 to go back  ---  F2 to abort
Source directory of the copy
Destination directory of the copy
Name of the node to copy:

        1. Graphic node
        2. Simulation node
        3. Text node
        4. Question node
    Your choice ?
```

Figure 5 - Page de COPYNODE

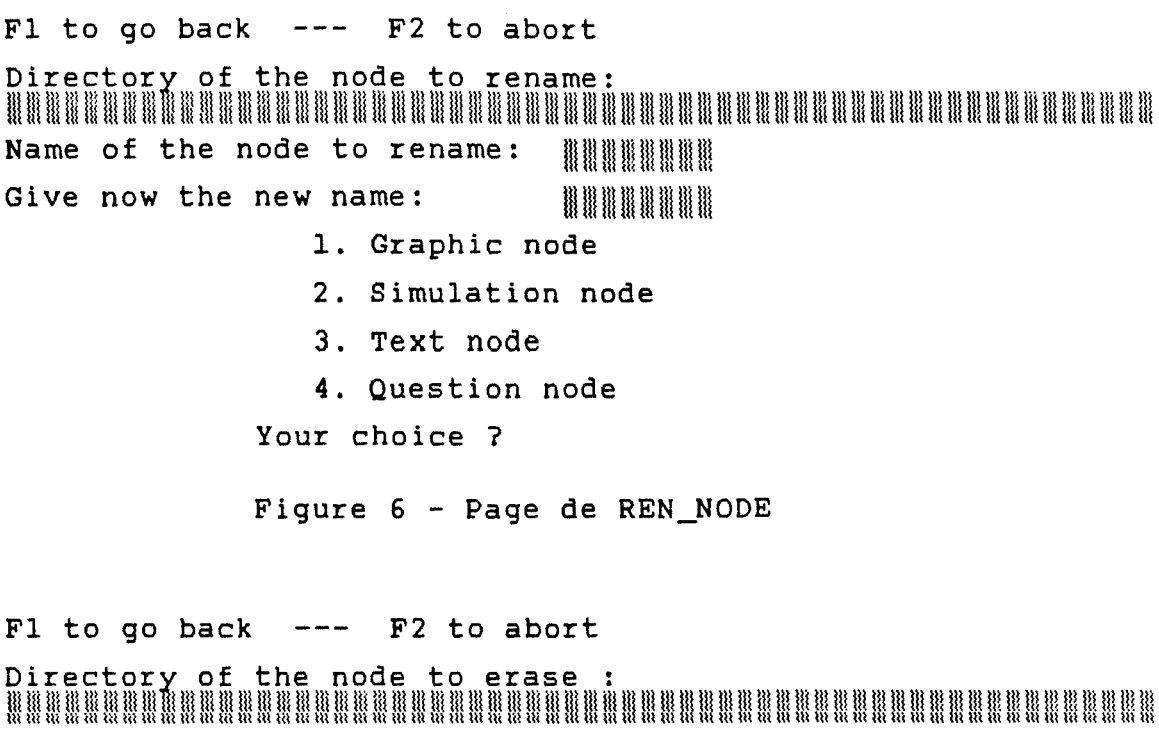


Figure 6 - Page de REN_NODE

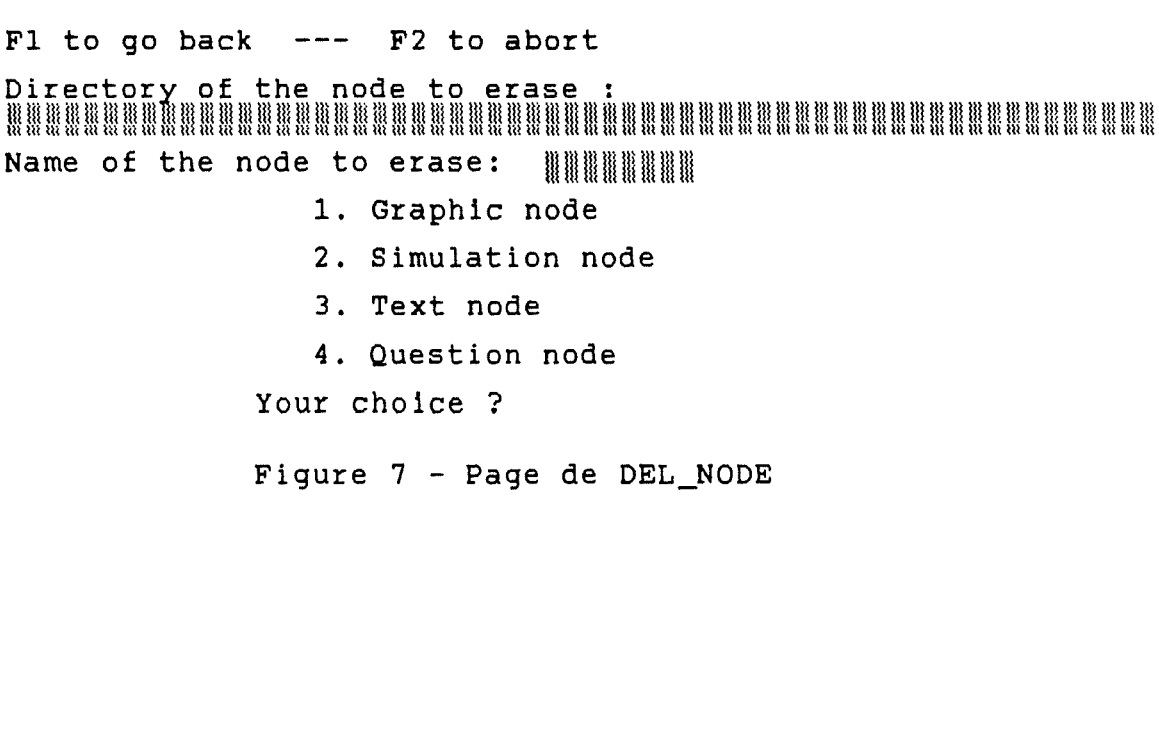


Figure 7 - Page de DEL_NODE

ANNEXE A - Syntaxe des expressions

Nous allons définir ici la syntaxe des expressions utilisées pour décrire le système à simuler aussi bien que les résultats que l'on désire obtenir (sorties graphiques et numériques). Pour ce faire, nous utiliserons le formalisme "Backus-Naur", dont voici les quelques méta-symboles:

- ::= signifie "est défini comme".
- | signifie "ou".
- {} la partie incluse peut-être répétée 0 ou plusieurs fois.
- .. désigne une des valeurs entières comprises dans l'intervalle défini.

Nous soulignerons les constructions syntaxiques. Les parties prédéfinies seront mises en caractères gras (par exemple **X** ou **entier**).

bl ::= {' '}

exposant ::= **

expression ::= expression1 | expression2

expression1 ::= {bl} opérande {bl} |
{bl} expression1 {bl} opérateur2 {bl} expression1 {bl} |
{bl} opérateur1 {bl} expression1 {bl} |
{bl} ({bl} expression1 {bl}) {bl}

expression2 ::=
{bl} { expression1 op-relat expression1 } {bl}
{ expression1 } {bl} { expression1 }

expression-retard ::= expression-retard1 |
expression-retard 2

expression-retard1 ::= {bl} opérande1 {bl} |
{bl} expression-retard1 {bl} opérateur2 {bl}
expression-retard1 {bl} |
{bl} opérateur1 {bl} expression-retard1 {bl} |
{bl} ({bl} expression-retard1 {bl}) {bl}

expression-retard2 ::=
    {bl} { expression-retard1 op-relat expression-retard1 }
    {bl} { expression-retard1 } {bl} { expression-retard1 }

indice-c ::= **1 .. nombre de constantes**

indice-d ::= **1 .. nombre de variables de retard**

indice-p ::= **1 .. nombre de paramètres**

indice-x ::= **1 .. nombre d'équations**

opérande ::= opérande1 | opérande2

opérande1 ::= S-X {bl} ({bl} indice-x {bl}) |
    S-X {bl} ({bl} S-P {bl} ({bl} indice-p {bl}){bl}) |
    S-X {bl} ({bl} S-X {bl} ({bl} indice-x{bl}) {bl}) |
    S-P {bl} ({bl} indice-p {bl}) |
    S-P {bl} ({bl} S-P {bl} ({bl} indice-p {bl}){bl}) |
    S-P {bl} ({bl} S-X {bl} ({bl} indice-x {bl}){bl}) |
    S-C {bl} ({bl} indice-c {bl}) |
    {bl} **entier** {bl} |{bl} **réel** {bl} |{bl} temps {bl}
    {bl} **PI** {bl}

opérateur2 ::= S-D {bl} ({bl} indice-d {bl})

opérateur1 ::= **-** | **abs** | **cos** | **sin** | **arctan** | **exp** | **ln** |
    frac | **int** | **sqr** | **sqrt**

opérateur2 ::= **+** | **-** | ***** | **/** | **mod** | **div** | exposant

op-relat ::= **<** | **>** | **=** | **>=** | **<=** | **<>**

S-C ::= **C** | **c**

S-D ::= **D** | **d**

S-P ::= **P** | **p**

S-X ::= **X** | **x**

temps ::= **T** | **t**

ANNEXE B - Description des fichiers

Vous trouverez la description d'un exemple complet à l'annexe D, reprenant notamment ces deux types de fichiers.

1) Le fichier descriptif d'un modèle de simulation, constitué à la création d'un modèle.

La description d'un modèle de simulation est sauvée dans un fichier de type texte, où chaque donnée occupe une ligne du fichier. Ces fichiers auront toujours la même extension : DTA. Voici, dans l'ordre, les valeurs ou chaînes de caractères que vous pourriez trouver dans un fichier qui résume l'étape de définition d'un modèle complet :

* 1 ou 2: 1= possibilité pour l'étudiant de tester sa propre simulation, 2= l'étudiant ne peut pas introduire de simulation.
Cette valeur n'est reprise dans la structure du fichier que pour maintenir la compatibilité avec le logiciel d'EAO. Elle n'a aucune utilité dans SimEDO. Valeur par défaut=2.

* 1 ou 2: indique la méthode de résolution choisie. 1= méthode d'Euler amélioré, 2= méthode de Runge-Kutta.

* X; le nombre de variables d'état du système. Pour SimEDO, il s'agit d'une valeur comprise entre 1 et 20. Accroc léger à la compatibilité avec ProfCOMP, celui-ci admet un maximum de 8 variables d'état. On ne peut donc d'exécuter avec ProfCOMP une simulation créée avec SimEDO et qui aurait plus de 8 variables d'état.

* Y; correspond au plus grand indice de paramètre. Si aucun paramètre ne figure dans le système, Y=0.
Comme pour le nombre de variables d'état, le nombre maximum de paramètres de SimEDO (20) est supérieur à celui de notre logiciel d'enseignement (8).

* Z; le nombre de constantes, c'est-à-dire un nombre compris entre 0 et 20. Dernier accroc à la transparence de SimEDO et du programme d'enseignement puisque ce dernier n'autorise que 10 constantes.

* R; le nombre de variables de retard compris entre 0 et 4.

* un réel. C'est la valeur du temps au début de la simulation.

* un réel. L'incrément de temps pour la résolution du modèle.

* Puis figurent (sur X lignes) les équations des variables d'état du système, sous la forme de chaînes de caractères.

* Ensuite viennent (sur Y lignes) les équations des paramètres, également sous la forme de chaînes de caractères.

* Pour chaque constante, sa valeur (Z lignes).

* Pour chaque variable de retard, son équation (chaîne de caractères) et son retard (réel) sur la ligne suivante (2*R lignes).

* 1 ou 2: 1= possibilité pour l'étudiant de changer les valeurs initiales des variables d'état ou les valeurs des constantes du modèle professeur. 2= l'étudiant ne peut pas changer les valeurs initiales.
C'est la seconde valeur qui est inscrite dans le fichier pour maintenir la compatibilité avec ProfCOMP. La valeur par défaut est 2. SimEDO ne tient pas compte de cette valeur.

* Pour chacune des variables d'état, sa valeur initiale (X lignes).

* Pour chacun des paramètres du système, faut-il ou non lui affecter une valeur initiale ? (Y lignes)
1= nécessaire.
0= inutile.

* Pour chacun des paramètres pour lesquels elle est nécessaire, sa valeur initiale. (réel)

* Vient ensuite un entier qui indique le choix de la représentation: 1= sortie graphique, 2= sortie numérique, 3= sortie graphique et numérique.

* S'il y a une sortie graphique :

- 1 ou 2: 1= deux points successifs d'un graphique seront reliés par une droite, 2= seuls les points calculés sont représentés.

- U; le nombre de graphiques demandés.
- un entier indiquant la fréquence à laquelle il faut représenter graphiquement les points calculés.
- Pour chacun des graphiques (U fois) :
  - l'expression correspondant à l'axe X.
  - l'expression correspondant à l'axe Y.
  - la valeur minimale pour laquelle il peut y avoir représentation en X (réel).
  - la valeur maximale pour laquelle il peut y avoir représentation en X (réel).
  - la valeur minimale pour laquelle il peut y avoir représentation en Y (réel).
  - et enfin la valeur maximale pour laquelle il peut y avoir représentation en Y (réel).

* un réel, le temps auquel commence la représentation des points calculés.

* un réel, la valeur maximale du temps pour les représentations. C'est aussi le temps à la fin de la simulation.

* S'il y a une sortie numérique :
  - V; le nombre de sorties numériques demandées.
  - Pour chacune de ces sorties (V fois), l'expression correspondante.
  - un entier, la fréquence à laquelle les résultats numériques sont inscrits dans le fichier.

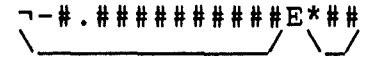
2) Le fichier des sorties numériques (constitué à l'exécution du modèle)

Il s'agit là encore d'un fichier de type texte. On trouve d'abord les valeurs initiales des variables d'état puis les valeurs des constantes (s'il y en a). Ensuite, après une ligne de blancs, on trouve, en en-tête des colonnes, les expressions correspondant à chacune des sorties numériques. Chaque ligne suivante reprend le résultat de l'évaluation de chacune des expressions à un moment particulier de l'exécution du modèle. Il y a donc sur chaque ligne autant de réels que de sorties numériques demandées. Pour respecter les colonnes, chaque réel est suivi de 14 blancs.

Chaque fois qu'il y a une interruption de la simulation pour changer soit les valeurs initiales des variables d'état, soit les valeurs de constantes (s'il y en a), on retrouve cet ensemble de lignes dans le fichier. Chacun des réels est représenté en virgule flottante sur 18 caractères.

Le format d'un réel:

- Positif: ¬¬#.##########E*##
- Négatif: ¬-#.##########E*##



mantisse exposant

où ¬ représente un blanc, # un chiffre et * le signe de l'exposant (+ ou -).

ANNEXE C : Fichiers nécessaires à l'exécution

1) Fichiers nécessaires sur l'unité programmes

SimEDO.COM	Le programme lui-même
SimEDO.000	L'overlay du programme
SimEDO.???	Le fichier des messages du programme, où ??? est l'extension installée identifiant la langue
8X8.FON	
14X9.FON	Les fichiers contenant les caractères correspondant aux différentes cartes graphiques.
IST.GRF	Le graphique servant de titre au programme

2) Les fichiers de l'unité données

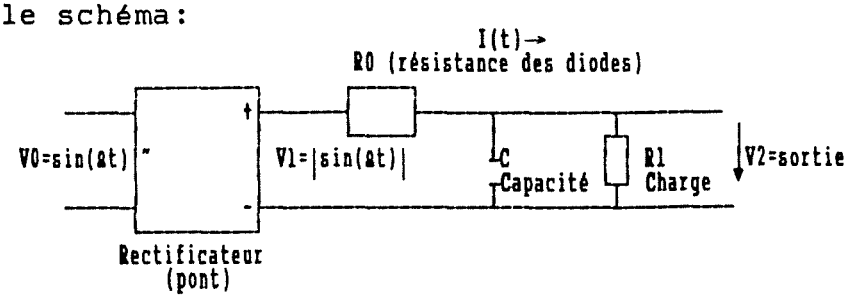
*.DTA	Les descriptifs de modèles de simulation.
*.DAT	Les sorties numériques des simulations.
WINDOWS.DIM	Fichier contenant pour chaque simulation la dimension de la fenêtre de résolution.

3) Les programmes utilitaires

INST_SIM.COM	Pour installer le programme
TRANS_ME.COM	Permet de traduire aisément les messages.
COPYNODE.COM	Copier une simulation d'une unité de disque sur une autre.
DEL_NODE.COM	Effacer une simulation.
REN_NODE.COM	Renomme une simulation

ANNEXE D : Exemple : La rectification d'une onde complète

Soit le schéma:



Le courant alternatif en entrée traverse un pont rectificateur (ou pont redresseur) à 4 diodes. A la sortie de ce pont, le courant est unidirectionnel comme l'indiquent les polarités sur le schéma. La résistance en série R0 limite l'intensité du courant et, avec la capacité en parallèle C, fait partie de la cellule de filtrage; la dite capacité C se charge et se maintient à charge : les fluctuations sont donc éliminées. Enfin, on trouve une résistance de charge R1 qui peut représenter une lampe,...

Nous voudrions simuler ce modèle et voir comment évolueront les tensions d'entrée (V0), intermédiaire (V1) et de sortie (V2) en fonction du temps.

Une analyse détaillée de ce système fait apparaître 8 paramètres et une variable d'état:

P(1) = R0 = 10 Ω V2(t) = X(1) = variable d'état
P(2) = R1 = 1000 Ω
P(3) = C = 0.001 F

Remarque: les valeurs de ces trois paramètres ont été choisies à titre d'exemple.

P(4) = sin(Ωt) = sin t (car Ω = 1)
P(5) = V1(t) = |sin t|
P(6) = -V2(t)/(R1*C)
P(7) = (V1-V2)/R0 = courant R0 si positif ou nul
P(8) = courant sur R0 (courant de charge de C et pour R1) = I(t)

$$d/dt[X(1)] = \begin{cases} ((V1/R0)-((1/R0)+(1/R1))*V2)/C & \text{Si } V1>V2 \\ -V2/(R1*C) & \text{Si } V1\leq V2 \end{cases}$$

$$I(t) = P(8) = \begin{cases} (V1/V2)/R0 & \text{Si } V1>V2 \\ 0 & \text{Si } V1<V2 \end{cases}$$

Il apparaît que les trois premiers paramètres sont des constantes. Il peut être intéressant de changer les valeurs de ces paramètres à l'exécution afin d'observer les changements qui en résultent. A l'exécution, on peut modifier les valeurs initiales des variables d'état et des constantes. Il suffit donc de faire passer les trois premiers paramètres en constantes. Le système est le suivant:

C(1) = 10 Ω
C(2) = 1000 Ω
C(3) = 0.001 F
P(1) = sin(Ωt) = sin t
P(2) = V1(t) = |sin t|
P(3) = -V2(t)/(R1*C)
P(4) = (V1-V2)/R0

$$P(5) = I(t) = \begin{cases} (V1/V2)/R0 & \text{Si } V1>V2 \\ 0 & \text{Si } V1<V2 \end{cases}$$

$$d/dt[X(1)] = \begin{cases} ((V1/R0)-((1/R0)+(1/R1))*V2)/C & \text{Si } V1>V2 \\ -V2/(R1*C) & \text{Si } V1\leq V2 \end{cases}$$

La dernière étape consiste à définir le système à créer dans SimEDO. Nous devons supprimer les V1, V2, R0, R1, C en les remplaçant respectivement par P(2), X(1), C(1), C(2) et C(3). Nous obtenons le système:

```
C(1) = 10
C(2) = 1000
C(3) = 0.001
P(1) = sin t
P(2) = {P(1)>0} {P(1)} {-P(1)}
P(3) = -X(1)/(C(2)*C(3))
P(4) = (P(2)-X(1))/C(1)
P(5) = {P(4)>0} {P(4)} {0}
d/dt[X(1)] = {P(2)>X(1)}
             {(P(2)/C(1)-(1/C(1)+1/C(2))*X(1)/C(3)}
             {P(3)}
```

Supposons que vous désirez afficher les trois courbes, avec pour chacune le temps en abscisse. Les ordonnées seront les suivantes:

* Pour la première courbe: le paramètre P(2) (=V1(t))

* Pour la deuxième: La variable d'état X(1)
* Pour la dernière: Le paramètre P(5) (= le courant sur R0)

Pour que ces différents graphiques ne se chevauchent pas les uns les autres à l'exécution, nous pouvons définir les bornes des intervalles de représentation en abscisse et en ordonnée. Pour les abscisses, nous choisirons un même intervalle de temps pour les trois graphiques, par exemple [0,20]. Pour les ordonnées, voici les intervalles choisis:

* Pour le premier graphique: [-4.5,1.5] de façon à placer cette courbe sur le dessus de la fenêtre. En effet, P(5)=|sin t| ne prendra jamais de valeur négative puisqu'il s'agit d'une valeur absolue. Le reste de l'intervalle ([-4.5,0]) sera utilisé pour représenter les 2 autres graphiques.
* Nous situerons le deuxième graphique au centre de la fenêtre, en choisissant l'intervalle [-1.5,4.5]
* Intervalle du dernier graphique: [-0.00001,0.01].

Nous désirons aussi obtenir le fichier des valeurs de X(1), P(2) et P(5). Nous donnerons une fréquence d'écriture dans le fichier de 5, c'est-à-dire que les valeurs de X(1), P(2) et P(5) seront inscrites dans le fichier une fois tous les 5 pas d'intégration.

Quand vous aurez créé cette simulation, l'option 3 du menu principal de SimEDO vous permet d'imprimer un rapport complet de votre simulation. Nous avons appelé cette simulation DC.DTA (voir figure 8).

```
Nom de la simulation: DC.DTA

Méthode de résolution: RUNGE KUTTA FOURTH ORDER
Nombre d'équation(s): 1
Nombre de paramètre(s): 5
Nombre de constante(s): 3
Nombre de variable(s) de retard: 0
Temps du début de simulation:   0.0000000000E+00
Incrément de temps:   1.0000000000E-02
Equations des variables d'état:
  d/dt[X(1)]= {P(2)>X(1)} {(P(2)/C(1)-(1/C(1)+1/C(2))*X(1))/C(3)} {P(3)}
Equations des paramètres:
  P(1)= sin t
  P(2)= {P(1)>0} {P(1)} {-P(1)}
  P(3)= -X(1) / (C(2) * C(3))
  P(4)= (P(2) - X(1)) / C(1)
  P(5)= {P(4) > 0} {P(4)} {0}
Valeurs des constantes:
  C(1)=   1.0000000000E+01
  C(2)=   1.0000000000E+03
  C(3)=   1.0000000000E-03
Valeurs initiales des variables:
  X(1)=   0.0000000000E+00
SORTIES GRAPHIQUES
Les points seront reliés:YES
Nombre de graphiques: 3
Fréquence de représentation des points: 1/2
Graphique 1   ****************
  Equation de l'axe X: t
  Equation de l'axe Y: P(2)
  Intervalle de représentation de l'axe X:
    [  0.0000000000E00,  2.0000000000E+01]
  Intervalle de représentation de l'axe Y:
    [ -4.5000000000E+00, 1.5000000000E00]
Graphique 2  *   *   *   *
  Equation de l'axe X: t
  Equation de l'axe Y: X(1)
  Intervalle de représentation de l'axe X:
    [  0.0000000000E+00,  2.0000000000E+01]
  Intervalle de représentation de l'axe Y:
    [ -1.5000000000E+00,  4.5000000000E+00]
Graphique 3   *****   *****
  Equation de l'axe X: t
  Equation de l'axe Y: P(5)
  Intervalle de représentation de l'axe X:
    [  0.0000000000E+00,  2.0000000000E+01]
  Intervalle de représentation de l'axe Y:
    [  0.0000000000E+00,  2.0000000000E+01]
SORTIES NUMERIQUES
Nombre de sorties numériques: 3
  Sortie 1: X(1)
  Sortie 2: P(2)
  Sortie 3: P(5)
Préquence d'écriture des points: 1/5
Intervalle de temps des deux sorties:
  [  0.0000000000E+00,  2.0000000000E+01]
```

Figure 8 - Impression du rapport

Toutes les données du modèle DC sont conservées dans le fichier DC.DTA de l'unité des données. Voici le contenu de ce fichier (figure 9). L'explication du contenu de chaque ligne, en gras, ne figure évidemment pas dans ce fichier.

```
1                 L'étudiant peut introduire sa propre simulation
2                 Méthode de Runge Kutta
1                 Nombre de variable d'état
4                 Le plus grand indice de paramètre
3                 Nombre de constantes
0                 Nombre de variables de retard
0.0000000000E+00  Temps au début de la simulation
1.0000000000E-02  Incrément de temps
{p(2)>x(1)}{(p(2)/c(1)-(1/c(1)+1/c(2))*x(1))/c(3)}{p(3)}=d/dt[X(1)]
sint              = P(1)
{p(1)>0}{p(1)}{0} = P(2)
-x(1)/(c(2)*c(3)) = P(3)
(p(2)-x(1))/c(1)  = P(4)
{p(4)>0}{p(4)}{0} = P(5)
1.0000000000E+01  = C(1)
1.0000000000E+03  = C(2)
1.0000000000E-03  = C(3)
2   L'étudiant ne peut changer les cond. init. de la sim. prof.
0.0000000000E+00  La valeur initiale de X(1)
0                 Pas de valeur
0                  initiale
0                  pour les
0                  paramètres
0                  P(1) à P(5)
3                 Sortie graphique et numérique
1                 Les points des graphiques seront reliés
3                 Nombre de sorties graphiques
2                 Fréquence de représentation des points
t                 Graphique 1 - Axe des abscisses
p(2)              Graphique 1 - Axe des ordonnées
0.0000000000E+00  Intervalle de représentation
2.0000000000E+01   sur l'axe X
-4.5000000000E+00 Intervalle de représentation
1.5000000000E+00   sur l'axe Y
t                 Graphique 2 - Axe des abscisses
x(1)              Graphique 2 - Axe des ordonnées
0.0000000000E+00  Intervalle de représentation
2.0000000000E+01   sur l'axe X
-1.5000000000E+00 Intervalle de représentation
4.5000000000E+00   sur l'axe Y
t                 Graphique 3 - Axe des abscisses
p(5)              Graphique 3 - Axe des ordonnées
0.0000000000E+00  Intervalle de représentation
2.0000000000E+01   sur l'axe X
-1.0000000000E-05 Intervalle de représentation
1.0000000000E-02   sur l'axe Y
0.0000000000E+00  Intervalle de temps
2.0000000000E+01   des sorties graphiques et numériques
3                 Nombre de sorties numériques
X(1)              Sortie numérique 1
P(2)              Sortie numérique 2
P(5)              Sortie numérique 3
5                 Fréquence d'écriture des points
```

Figure 9 - Le fichier DC.DTA

Le début du fichier DC.DAT (figure 10) reprend les valeurs des trois sorties numériques demandées. On trouve d'abord les valeurs initiales de la variable d'état X(1) puis les valeurs des constantes C(1), C(2), C(3) de ce modèle. Ensuite, on trouve, après une ligne de blancs, les expressions des sorties, en l'occurence X(1), P(2) et P(5). Les lignes ultérieures contiennent les valeurs de ces expressions. Puisque le fréquence d'écriture dans le fichier est de 5, la première ligne des valeurs représente les valeurs de la cinquième itération, la troisième celles de la dizième itération, etc... Nous remarquons également qu'il y a eu une interruption dans la simulation. L'utilisateur a

changé les valeurs de la variable X(1) et des constantes C(1), C(2) et C(3). C'est pourquoi nous retrouvons 4 lignes semblables aux 4 premières du fichier. SimEDO recommence alors une nouvelle simulation avec les nouvelles valeurs.

```
X(1)= 0.0000E+00
C(1)= 1.0000E+01
C(2)= 1.0000E+03
C(3)= 1.0000E-03

X(1)                    P(2)                    P(5)
  0.0000000000E+00        0.0000000000E+00        0.0000000000E+00
  4.3554732562E-02        4.9979169271E-02        1.6447833261E-03
  9.3010763619E-02        9.9833416647E-02        1.6684801983E-03
  1.4215822044E-01        1.4943813247E-01        1.7084795588E-03
  1.9094981589E-01        1.9866933079E-01        1.7443538520E-03
  2.3926413242E-01        2.4740395925E-01        1.7758691995E-03
  2.8698041319E-01        2.9552020666E-01        1.8029458060E-03
  3.3397939239E-01        3.4289780745E-01        1.8255159870E-03
  3.8014359703E-01        3.8941834230E-01        1.8435233289E-03
  4.2535764066E-01        4.3496553410E-01        1.8569228226E-03
  4.6950851172E-01        4.7942553859E-01        1.8656809762E-03
  5.1248585600E-01        5.2268722891E-01        1.8697758990E-03
  5.5418225254E-01        5.6464247337E-01        1.8691973558E-03
  5.9449348206E-01        6.0518640570E-01        1.8639467926E-03
  6.3331878749E-01        6.4421768720E-01        1.8540373334E-03
  6.7056112577E-01        6.8163875998E-01        1.8394937461E-03
  7.0612741046E-01        7.1735609085E-01        1.8203523827E-03
  7.3992874437E-01        7.5128040509E-01        1.7966610862E-03
  7.7188064177E-01        7.8332690958E-01        1.7684790728E-03
  8.0190323954E-01        8.1341550474E-01        1.7358767827E-03
  8.2992149685E-01        8.4147098475E-01        1.6989357046E-03

X(1)= 5.0000E-01
C(1)= 1.5000E+01
C(2)= 1.5000E+03
C(3)= 1.5000E-03

X(1)                    P(2)                    P(5)
  4.9777777778E-01        0.0000000000E+00        0.0000000000E+00
  4.8680423770E-01        4.9979169271E-02        0.0000000000E+00
  4.7608209819E-01        9.9833416647E-02        0.0000000000E+00
  4.6559611988E-01        1.4943813247E-01        0.0000000000E+00
  4.5534110120E-01        1.9866933079E-01        0.0000000000E+00
  4.4531195512E-01        2.4740395925E-01        0.0000000000E+00
  4.3550370667E-01        2.9552020666E-01        0.0000000000E+00
  4.2591149046E-01        3.4289780745E-01        0.0000000000E+00
  4.1653054826E-01        3.8941834230E-01        0.0000000000E+00
  4.2131696041E-01        4.3496553410E-01        1.2625486925E-03
  4.5984806187E-01        4.7942553859E-01        1.8649980333E-03
  5.0247932491E-01        5.2268722891E-01        1.9146552072E-03
  5.4441164712E-01        5.6464247337E-01        1.9013680580E-03
  5.8504232304E-01        6.0518640570E-01        1.8771576282E-03
  6.2421696849E-01        6.4421768720E-01        1.8476005321E-03
  6.6183206194E-01        6.8163875998E-01        1.8133559254E-03
  6.9779299051E-01        7.1735609085E-01        1.7745715021E-03
  7.3200980750E-01        7.5128040509E-01        1.7313507922E-03
  7.6439698199E-01        7.8332690958E-01        1.6838025239E-03
  7.9487356220E-01        8.1341550474E-01        1.6320456175E-03
  8.2336337248E-01        8.4147098475E-01        1.5762094459E-03
  8.4979520312E-01        8.6742322554E-01        1.5164335715E-03
  8.7410298833E-01        8.9120736001E-01        1.4528674030E-03
  8.9622597129E-01        9.1276394021E-01        1.3856698227E-03
      ...                     ...                     ...
```

Figure 10 - Début du fichier des sorties numériques DC.DAT

Il ne reste plus qu'à essayer l'exécution de DC pour confirmer ou rejeter vos hypothèses de convergence ou de divergence, de forme de courbes,... Heureusement, dans notre cas, DC se comporte admirablement bien (figure 11).

A vous d'en faire autant avec vos simulations !!!

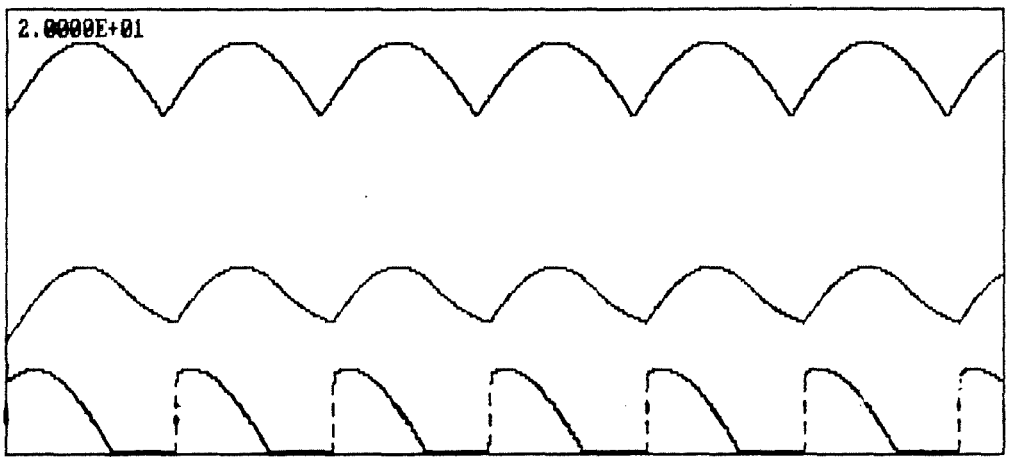


Figure 11 - Les graphiques de l'exécution du modèle de rectification d'une onde complète

ANNEXE C

Papier Proposé et accepté au Congrès de la Société Portugaise de Physique

COIMBRA

(28 et 29 septembre 1988)

SOCIEDADE PORTUGUESA DE FISICA
Delegaçao Regional de Coimbra
Departamento de fisica
Universidade
3000 Coimbra - PORTUGAL

SimEDO : An ordinary differential equations simulator

S.D. Antunes, Instituto Superior Tecnico,
Av. Rovisco Pais, 1096 Lisboa Codex,
Portugal

J.M. Barreto, C. Léonard and O. Marchand,
FNDP - Institut d'Informatique,
Rue Grandgagnage 21, 5000 Namur, Belgique

To make understandable the dynamic behaviour of a complex system, the teacher may not always organise experiences. In this case, the teacher may rely on simulations, which is an attractive courses support to illustrate a theory or make forecastings starting with a theorical model.

The aim of SimEDO consists precisely to provide users with an easy way to create, test and modify nonlinear ordinary differential equations system. This software, running on IBM personal computers and compatibles, has been written with the Turbo Pascal 3.0 language. The user doesn't need to have some programming skills to simulate an equations system. The model's creation step consists in a prompting system : the user is guided through all this stage by messages asking him to introduce the different elements of his model. A system may contain up to 20 state's variables, 20 parameters, 20 constants and 4 delay variables. Each state's variable may be conditional, i.e. composed of 3 subexpressions, each being enclosed within braces. The first subexpression expresses a condition which should be evaluated at each iteration. If this subexpression is true evaluated, then the second subexpression is evaluated as the iteration value of the state's variable else the third subexpression gives the state's variable's value. By example, if we represent the state's variables by X(i) (i being the indice) and time by letter t, the following equation d/dt[X(1)]= { t > 50 } { X(2) } { -X(2) } may be interpreted as :

IF time is greater than 50
THEN d/dt[X(1)] takes the value of X(2),
ELSE d/dt[x(1)] = - X(2).

A parameter is defined either by a not conditional expression or, like a constant, by a real value.

All equations are syntactically verified at edition time and the user is immediately informed of errors he makes. Two types of results are available : graphic or numerical outputs. The user may combine up to 5 curves on the same screen, defining for each one the expressions he wants to see the values on both axes. The plotting interval on the vertical and horizontal axes is also determined. Of course, the axes should not be graduated at execution time because the scales may be totally different for curves. Numerical outputs consists in recording in a file some intermediate values of interesting expressions, choosen by the user. SimEDO offers 2 methods of integration : Euler modified and Runge Kutta fourth order.

When the model is defined, it can be executed i.e. the curves appears on screen and files of numerical values is fulfilled. An execution may be stopped at any moment either to abort the simulation, either to print the curves or to change the initial values of state's variables and/or constant values (if any) before to begin the simulation again.

If the curves are not what was expected, the designer may go in the creation step in order to correct the model.

Finally, a complete report retaining all elements of the model may be printed.

An utility is provided to translate or modify all messages appearing in SimEDO, following the user's language.

ANNEXE D

Papier proposé au

MELECON 89

IEEE Mediterranean ELEctrotechnical CONference
Lisbon, PORTUGAL - Forum PICOAS
April, 11, 12, 13, 1989

ProfCOMP Integrated author-system/simulation package

S.D. Antunes, Instituto Superior Tecnico,
Av. Rovisco Pais, 1096 Lisboa Codex,
Portugal

J.M. Barreto, C. Léonard and O. Marchand,
FNDP - Institut d'Informatique,
Rue Grandgagnage 21, 5000 Namur, Belgique

Area of this paper : D1. Curriculum development or
D4. Continuing education

The computer software ProfCOMP, running on IBM personal computers, is an author system which integrates an ordinary differential equation simulator (SimEDO).

This software provides professors with an easy way to design computer assisted lessons without having to know anything about programming. A computer assisted lesson consists in graphics, animations, texts, simulations and question-answer nodes displayed in windows.

Two main programs compose this system : the creation and the execution part.

The creation part consists in a set of tools related to the types of nodes we may encounter in a lesson.

A graphic editor helps the professor to create graphics and animation components. You don't need to be a good drawer to use this editor : many commands, easily employed, make this task very simple. An animation is like a motion-portion cartoon but with a limited number of drawings which show the evolution.

To display text windows, 2 tools are provided : an editor to introduce the text and one formatter. The formatter offers commands which are inserted into the editing text in order to present text in pages compatible with the window choosen by the teacher.

The big originality of this system is the integration of a differential equations simulator. The teacher may define interactively models which can be tested by the student at execution time. The student may also introduce his system and tests them to compare it with the professor's system. The simulator system may contain:

- up to 8 state's variable,
- up to 8 parameters,
- up to 10 constants,
- up to 4 delay variables, and
- the TIME variable.

Operators are +, -, *, ** , /, MOD, ABS, COS, SIN, ARCTAN,... State's variable equation may be conditional i.e. composed of 3 parts : the first one is a condition on a state's variable, parameter or time; the second one is the evaluated expresssion if the condition is true evaluated and the third one is the expression evaluated if it's false evaluated. The user defines the graphics output he wants to see. It's also possible to create a file which contain all intermediate values of defined expressions.

Last but not least, the teacher may question and classify his student. Three types of questions are available :

- Multiple choice questions : the student has to choose the correct answer in a list of potential answers.
- Simple questions: in this case the answer is not presented and the student must introduce the complete answer on the keyboard; and,
- Questions where the professor hasn't defined the response. In this case, the student has to answer with a text.

Before to answer a question, the student may perform a simulation experiment on the system under study if necessary.

At creation time, the teacher sees the content of the created node in the same window as those seen by students at execution time.

When all nodes are created, the teacher has to chain them using a syntactical chaining editor which proposes instructions as "EXECUTE NODE...", "ERASE THE SCREEN",... The execution scheme is not entirely fixed by professor because the teacher may forsee conditions which may orient lesson in some directions. For example: "if the student has not obtained 50 % of the points, begin again lesson with more explicitations". Professor may propose menu to orient the lesson.

The execution part is the program usable by the student to execute a lesson. After had given his name and the name of the lesson, he executes the lesson as forseen by professor. The result of the lesson is contained in a file where we can find in it the learning way of the student. A question node is detailed and we may find the question, the given responses, and cotation. Simulation nodes are also detailed. This file ends with the time spent on the lesson and total quotation.

ANNEXE E

L'historique d'un noeud

Simulation

(Exemple de programme que l'enseignant devait compléter à l'issue du laboratoire pour simuler un modèle d'équations différentielles ordinaires du premier ordre)

```
program EXECUTION_SIMULATION;

{***********************************************************************}
{* MODIFICATIONS: - MaxplotGlb (TYPEDEF.SYS) Nouvelle valeur: 500     *}
{*                - FINDWRLD.HGH                                       *}
{***********************************************************************}

{$i b:typedef.sys}
{$i b:graphix.sys}
{$i b:kernel.sys}
{$i b:windows.sys}
{$i b:findwrld.hgh}
{$i b:axis.hgh}
{$i b:polygon.hgh}

{*************************************************************************}
{*                                                                       *}
{*  La simulation représentée dans cet exemple  concerne le cuircuit     *}
{*  suivant :                         R                                  *}
{*                      -----------/\/\/\-----------                     *}
{*                      |          résistance      |                     *}
{*                      |                          |                     *}
{*                     ---                        ---                    *}
{*                 E    -  batterie            C  ---  condensateur      *}
{*                      |                          |                     *}
{*                      |                          |                     *}
{*                      ----------------------------                     *}
{*                                                                       *}
{* L'équation différentielle correspondante est la suivante:             *}
{*      i = dQ / dT = (E / R) - Q / (R * C)                              *}
{*  Pour pouvoir représenter cette équation, on doit transformer les E, R, *}
{*  Q et C en variables ou paramètres de ce programme.                   *}
{*  * Q est la condition initiale de l'unique équation et sera donc      *}
{*    représentée par QI[1]                                              *}
{*  * R correspondra à la variable P1                                    *}
{*  * C correspondra à la variable P2                                    *}
{*  * E correspondra à la variable P3                                    *}
{*  L'équation à introduire dans le programme sera donc la suivante :    *}
{*     (P3 / P2 ) - QI[1] / (P1 * P2)                                    *}
{*  L'ensemble des valeurs de Q étant représentées dans le tableau       *}
{*  tableau_1, on remplace encore QI[1] par tableau_1[i-1]               *}
{*                                                                       *}
{*************************************************************************}

const texte=9; { numéro de la fenêtre affichant le texte avant la saisie d'un paramètre à l'exécution }
      nbrepoints=200; { nombre de points calculés pour chaque simulation }
      maxeq=3; { nombre maximum d'équations admises }

type demiplotarray=array[1..nbrepoints] of real;
     str5=string[5];
     str8=string[8];
     str78=string[78];
     tab_QI=array[1..maxeq] of real;

var mon_nom_fichier:str8;
    var_bidon:str8;
    D1,D2,D3:demiplotarray; { tableaux de points calculés par la fonction EULER }
    temps:demiplotarray; { tableau des valeurs de temps }
    QI:tab_QI; { tableau des conditions initiales }
    i,j:integer; { compteurs }
    delta_t:real; { différence de temps entre 2 points successifs }
    temps_depart,temps_fin:real; { temps de départ et temps de fin pour la simulation }
    P1,P2,P3,P4,P5,P6,P7,P8,P9,P10:real; { variables intervenant dans les équations }
    nbrequations,nbrevar:integer; { nombre d'équations et de variables }
    ligne:str78;
    valeur:str5;
    f_dta:text;
    f:file of str8;
    f_com:file;
    tableau_1,tableau_2,tableau_3:plotarray; { tableaux des points à représenter sur la (les) courbe(s) }
    num_fen,mon_numero,gauche,haut,droit,bas:integer;
    f_int:file of integer;

function EULER(var a:real;b:real):real;
begin
    EULER:=a+delta_t*b;
end;
```

```
procedure MUSIQUE;
begin
     sound(1000);delay(100);nosound
end;

{PROCEDURES D'E/S}

procedure LIRENTIER(x,y:integer;var z:integer;binf,bsup,nbunderline:integer);
var ch:str5;
    erreur:boolean;
    code,i:integer;
begin
     erreur:=false;
     repeat
           z:=0;
           gotoxy(x,y);
           readln(ch);
           val(ch,z,code);
           if (code<>0)or(z<binf)or(z>bsup)
             then begin erreur:=true;
                        gotoxy(x,y);
                        for i:=1 to nbunderline do write('_');
                        musique
                  end
             else erreur:=false
     until erreur=false;
end;

procedure LIREEL(x,y:integer;var z:real;binf,bsup:real;nbunderline:integer);
type str15=string[15];
var ch:str15;
    erreur:boolean;
    code,i:integer;
begin
     erreur:=false;
     repeat
           z:=0;
           gotoxy(x,y);
           readln(ch);
           val(ch,z,code);
           if (code<>0)or(z<binf)or(z>bsup)
            then begin erreur:=true;
                       gotoxy(x,y);
                       for i:=1 to nbunderline do write('_');
                       musique
                 end
            else erreur:=false
     until erreur=false;
end;

procedure LECTURE_TEMP_INT; {lecture du fichier constitué par Execless}
var ch:char;
    x1,y1:integer;
begin
     assign(f_int,'B:TEMP.INT');
     reset(f_int);
     read(f_int,num_fen);
     if num_fen<>0 then loadwindowstack('b:stackw');
     while num_fen<>0 do
     begin
          restorewindow(num_fen,0,0);
          read(f_int,num_fen);
     end;
     read(f_int,mon_numero);
     read(f_int,gauche);
     read(f_int,haut);
     read(f_int,droit);
     read(f_int,bas);
     close(f_int)
end;
```

```
{Lecture du fichier des questions constitué par Création}

procedure LECTURE_DONNEES_QI;
begin
     for i:=1 to nbrequations do
     begin
          readln(f_dta,valeur);
          if valeur='FALSE' then readln(f_dta,QI[i])
                            else begin
                                 copyscreen;
                                 selectwindow(texte);
                                 setbackground(0);
                                 drawborder;
                                 for j:=1 to 5 do begin
                                                  readln(f_dta,ligne);
                                                  gotoxy(2,18+j);
                                                  write(ligne)
                                                end;
                                 gotoxy(2,24);write('________');
                                 drawborder;
                                 LIREEL(2,24,QI[i],-1E10,1E10,10);
                                 swapscreen
                                end
     end
end;

procedure LECTURE_DONNEES_PI;
begin
     for i:=1 to nbrevar do
     begin
          readln(f_dta,valeur);
          if valeur='FALSE' then case i of
                                 1:readln(f_dta,P1);
                                 2:readln(f_dta,P2);
                                 3:readln(f_dta,P3);
                                 4:readln(f_dta,P4);
                                 5:readln(f_dta,P5);
                                 6:readln(f_dta,P6);
                                 7:readln(f_dta,P7);
                                 8:readln(f_dta,P8);
                                 9:readln(f_dta,P9);
                                 10:readln(f_dta,P10)
                                end
                            else begin
               copyscreen;
               selectwindow(texte);drawborder;
               setbackground(0);
               for j:=1 to 5 do begin
                                readln(f_dta,ligne);
                                gotoxy(2,18+j);
                                write(ligne)
                                end;
               gotoxy(2,24);write('________');
               drawborder;
               case i of
                1:LIREEL(2,24,P1,-1E10,1E10,10);
                2:LIREEL(2,24,P2,-1E10,1E10,10);
                3:LIREEL(2,24,P3,-1E10,1E10,10);
                4:LIREEL(2,24,P4,-1E10,1E10,10);
                5:LIREEL(2,24,P5,-1E10,1E10,10);
                6:LIREEL(2,24,P6,-1E10,1E10,10);
                7:LIREEL(2,24,P7,-1E10,1E10,10);
                8:LIREEL(2,24,P8,-1E10,1E10,10);
                9:LIREEL(2,24,P9,-1E10,1E10,10);
                10:LIREEL(2,24,P10,-1E10,1E10,10)
               end;
               swapscreen
            end
     end
end;
```

```
{------------------------------------------------------------------------}
{                 DEBUT DU PROGRAMME PRINCIPAL                           }
{------------------------------------------------------------------------}

begin
     initgraphic;
     assign(f,'B:TEMP.PIC');
     reset(f);
     read(f,mon_nom_fichier);
     read(f,var_bidon);
     close(f);
     assign(f_dta,'B:'+mon_nom_fichier+'.DTA');
     assign(f_com,'A:EXECLESS.COM');
     definetextwindow(texte,1,18,79,24,1);
     LECTURE_TEMP_INT;
     reset(f_dta);
     readln(f_dta,nbrequations);
     readln(f_dta,nbrevar);
     readln(f_dta,temps_depart);
     temps[1]:=temps_depart;
     readln(f_dta,temps_fin);
     delta_t:=(temps_fin-temps_depart)/nbrepoints;
     LECTURE_DONNEES_CI;
     LECTURE_DONNEES_PI;
     close(f_dta);
     definewindow(mon_numero,gauche,haut,droit,bas);
     selectwindow(mon_numero);
     setbackground(0);
     drawborder;
{ REMPLISSAGE DU PREMIER ELEMENT DE CHAQUE TABLEAU Di }
     D1[1]:=CI[1];

{ FIN }
{CALCUL DES POINTS SELON LA FONCTION EULER}
     for i:=2 to nbrepoints do
     begin
          temps[i]:=temps[i-1]+delta_t;
          D1[i]:=EULER(D1[i-1],(P3/PI)-(D1[i-1]/(P1*P2)));
     end;
{REMPLISSAGE DU/DES TABLEAUX DE POINTS A REPRESENTER SUR COURBES}
     for i:=1 to nbrepoints do
     begin
          tableau_1[j,1]:=temps[i];
          tableau_1[j,2]:=D1[i];
     end;
     findworld(mon_numero,tableau_1,nbrepoints,1,1);
     drawborder;
     setlinestyle(0);
     drawaxis(8,-8,0,0,0,1,0,0,true);
{TRACAGE DE CES COURBES}
     drawpolygon(tableau_1,1,j-1,0,2,0);
     read(kbd,ch);
     redefinewindow(mon_numero,gauche,haut,droit,bas);
     storewindow(mon_numero);
     savewindowstack('B:STACKW');
     leavegraphic;
     execute(execution_lecon)
end.
```