MASTER EN SCIENCES INFORMATIQUES

Méthode de conception d'une gestion des erreurs de l'utilisateur et application à un logiciel de tests sur la mémoire humaine

Schepman, Marie-Hélène

Award date:
1989

Awarding institution:
Universite de Namur

INSTITUT D'INFORMATIQUE

FACULTES UNIVERSITAIRES
NOTRE-DAME DE LA PAIX
NAMUR

METHODE DE CONCEPTION D'UNE GESTION DES ERREURS DE L'UTILISATEUR ET APPLICATION A UN LOGICIEL DE TESTS SUR LA MEMOIRE HUMAINE.

par Marie-Hélène SCHEPMAN

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

Promoteur :

Professeur C.CHERTON

Année académique 1988-1989

La robustesse d'un logiciel est un des aspects essentiels de sa convivialité. Un logiciel robuste est un logiciel qui réagit de façon adéquate aux fautes de l'utilisateur.
Si les méthodes et les outils sont nombreux en matière de gestion des erreurs de programmation, il n'en est pas de même pour la gestion des erreurs de l'utilisateur.
Le but de ce mémoire est de proposer une méthode de conception d'une gestion systématique des fautes de l'utilisateur. Cette méthode sera indépendante de l'application et du langage de programmation utilisé.
Nous illustrerons la description de cette méthode par son application à un logiciel particulier : un logiciel de tests psychologiques sur la mémoire humaine.

The robustness of a software is one essential aspect of its user-friendliness. A sofware is said to be correct when reacting in an appropriate way to the user's errors.
If there are many methods and tools for the programming errors, the problem is quite different concerning user's errors.
The object of this thesis is to suggest a systematic user's errors handle design method. This method will be application and language independant.
We illustrate the decribed method with its application to a specific software : a software of psychologic tests on the human memory.

Je suis heureuse que l'occasion me soit offerte aujourd'hui d'exprimer toute ma gratitude mais surtout mes vifs remerciements à tous ceux qui ont contribué à la réalisation et à l'aboutissement de ce mémoire.

Merci tout d'abord à Monsieur le Professeur C.Cherton, promoteur de ce mémoire, qui, par ses conseils, ses réflexions, ses suggestions et son inlassable patience, m'a guidée et m'a aidée à poser un regard parfois neuf sur ce travail.

Je remercie également l'ensemble des Professeurs des Facultés pour l'enseignement dispensé durant ces cinq années.

J'adresse encore mes plus vifs remerciements à Monsieur C.Durez et ses collaborateurs de la société Inter System Tournai pour leur acceuil et pour le matériel mis à ma disposition.

Enfin, j'adresse mes sentiments de gratitude à tous ceux qui ont oeuvré dans l'ombre et m'ont soutenue, durant cette année mais également tout au long de mes études; Qu'ils se reconnaissent ici.

TABLE DES MATIERES

PARTIE II : APPLICATION DE LA METHODE A UN LOGICIEL : CAS D'UN LOGICIEL DE TESTS DE LA MEMOIRE HUMAINE

INTRODUCTION

Le point de départ du présent mémoire est le mémoire précédemment réalisé par Fabienne Rinclin ("Etude de certains comportements de la mémoire humaine" F.RINCLIN Année académique 1985-1986). Celui-ci avait pour idée principale de "fournir un outil informatique aux pédagogues dans le cadre de leurs études sur le comportement de la mémoire humaine".

Nous allons tout d'abord rappeler de façon précise mais succincte le contenu du mémoire précédent afin de disposer de tous les éléments nécessaires pour pouvoir poursuivre le travail entamé.

Ensuite, sur base de la part de travail déjà réalisée, nous envisagerons toute une série de prolongements éventuels à réaliser dans le cadre du présent mémoire. Nous ferons la critique tout à tour de ces solutions et ébaucherons enfin les objectifs de la solution retenue.

Enfin, nous présenterons brièvement la logique qui guidera le développement du présent mémoire.

DESCRIPTION DE L'ETUDE PRECEDENTE (mémoire de F.Rinclin)

L'analyse de tests existants en psychologie a permis de constater qu'il était possible de créer un logiciel générateur de tests psychologiques à propos de la mémoire. Le mémoire de F.Rinclin, qui avait pour but de réaliser ce logiciel, s'est limité à prendre en compte trois types de tests particuliers qui sont :

1) la représentation du contenu d'un texte par un réseau sémantique, c'est-à-dire sous forme d'un graphe orienté dans lequel les sommets sont des concepts qui représentent les idées principales du texte et dans lequel les arcs sont des relations entre ces concepts;

2) la hiérarchisation des paragraphes d'un texte, c'est-à-dire un ordonnancement des paragraphes qui puisse mettre en évidence les idées principales du texte et les idées secondaires qui s'y rapportent;

3) la création de liaisons entre les paragraphes d'un texte représentées par un graphe orienté dont les sommets sont les identifiants des paragraphes et les arcs les liens entre ces paragraphes.

Le point commun entre ces trois types de tests est le fait qu'ils utilisent comme matériau de base un texte de travail et qu'ils cherchent à tester le sujet quant à la connaissance de ce texte.

Pour connaître avec précision les exigences des pédagogues vis-à-vis du logiciel, F.Rinclin s'est basée sur un document, une "Grille d'Analyse de didacticiels" réalisée par le centre OSE de Namur (l'Ordinateur au Service de l'Education). L'étude des tests existants en psychologie permet de penser qu'il est possible et envisageable de réaliser un logiciel global permettant l'introduction et l'exécution de tests psychologiques sur la mémoire humaine.

En accord avec les desiderata des psycho-pédagogues, ce logiciel a pour but de permettre l'exécution de quatre phases distinctes qui constituent un test proprement dit :

1ère phase :

Le pédagogue doit au moins pouvoir introduire un nouveau test parmi les tests existants. Ce nouveau test sera défini par un type (réseau, liaison, hiérarchisation) et un identifiant. Ensuite, il lui faudra introduire les divers éléments constitutifs du test, c'est-à-dire :

- des éléments explicatifs (facultatifs) qui indiqueront au sujet testé ce qu'il doit faire :

  . soit un texte explicatif seul, sous une forme libre;

  . soit un texte explicatif suivi d'un exemple dont la forme varie en fonction du type de test considéré.

- des éléments qui forment le matériel du test :

  - le texte de travail;

  - des matériaux propres à chaque type de test qui aident à construire la réponse :

    . pour un test de type réseau, la signification des liaisons et des concepts;
    . pour un test de type liaison de paragraphes, la signification des liaisons;

  - une réponse-type du pédagogue qui présente à la personne testée une solution possible. Cette réponse peut lui être présentée progressivement, entrecoupée de textes explicatifs, ou bien en un bloc c'est-à-dire sans les textes explicatifs;

2ème phase :

Une fois le test introduit par le pédagogue, il peut être exécuté par les personnes à tester. Pour exécuter un test, la personne doit donner son nom et indiquer le test souhaité. Ensuite, se succèdent les différentes parties qui forment le test :

- la présentation des éléments explicatifs, s'ils ont été introduits par le pédagogue

  . soit un texte explicatif seul;

  . soit un texte explicatif suivi d'un exemple dont la présentation peut se faire progressivement ou en un seul bloc;

- la présentation des éléments qui forment les données du test :

  . le texte de travail, présenté une première fois en mode consultation uniquement;

  . les matériaux propres à chaque type de test;

- la réalisation de la réponse du sujet testé qui se fera sur un écran vierge au départ. Au cours de cette phase, il aura la possibilité d'accéder :

  . au texte de travail

    en le consultant pour un test de type réseau;

    en le modifiant pour les deux autres types de tests (le sujet testé doit attribuer un nom identifiant à chaque paragraphe);

  . aux identifiants des concepts et des liaisons pour un test de type réseau;

  . aux identifiants des liaisons pour un test de type liaison de paragraphes.

Au cours de la création d'une réponse, le sujet testé a également la possibilité de justifier chacun des éléments figurant dans la réponse par un texte (texte réponse).

L'annexe 1 présente un exemple illustrant ce dont l'élève dispose pour chaque type de test et ce qu'il doit réaliser.

3ème phase :

Lorsque la personne testée a terminé le test, une réponse-type définie par le pédagogue peut lui être présentée pour comparaison. Ici aussi, la réponse-type peut faire l'objet d'une présentation en un bloc ou d'une présentation progressive, accompagnée d'explications.

4ème phase :

Toutes les réponses reçues peuvent être conservées par le logiciel, ce qui permet aux pédagogues d'aller à tout moment consulter un réponse d'une personne à un test donné. Elles peuvent être soit affichées à l'écran, soit imprimées sur papier.

La consultation peut porter sur divers éléments selon le type de test :

si le test est de type réseau, on peut consulter :

- la liste chronologique des concepts et liaisons choisis pour constituer la réponse;
- la réponse sous forme d'un graphe orienté qui représente le réseau sémantique;
- le(s) texte(s) explicatif(s) accompagnant la réponse.

si le test est de type hiérarchisation de paragraphes ou de type liaison de paragraphes :

- le texte de travail avec l'identifiant associé à chaque paragraphe;
- la réponse telle qu'elle a été réalisée, c'est-à-dire sous forme de hiérarchie de paragraphes ou de graphe orienté;
- la liste chronologique des identifiants de paragraphes, et le cas échéant, des identifiants de liaisons, choisis pour réaliser la réponse;
- le(s) texte(s) explicatif(s) accompagnant la réponse;

Le pédagogue peut aussi supprimer une ou plusieurs réponses existantes : supprimer les réponses de tous les sujets testés à un test déterminé, ou une réponse d'une certaine personne à un certain test, ou les réponses d'une personne à tous les tests qu'elle a réalisés.
Le logiciel permet aussi de supprimer un test tout entier, c'est-à-dire ses données et toutes les réponses associées.

Dans son mémoire, Fabienne Rinclin a regroupé les divers traitements exposés ci-dessus pour former des modules logiques de différents niveaux et a spécifié de façon complète tous ces modules. Mais les modules spécifiés dans le cadre du mémoire de F.Rinclin n'ont pas été tous implémentés, par manque de temps.

Les modules implémentés entièrement concernent essentiellement la gestion des écrans et la gestion des fichiers contenant la liste des tests, des noms de pédagogues et des noms de sujets testés, les textes, les matériaux et les réponses aux tests.
D'autres modules ont été implémentés, mais de façon partielle.

Cette implémentation a été réalisée en langage Pascal, sur Apple MacIntosh. Le choix de ce type de machine relève de diverses raisons dont les principales sont la convivialité d'utilisation d'une application MacIntosh du point de vue de l'utilisateur ainsi que le fait que ce matériel est disponible au centre OSE de Namur (l'Ordinateur au Service de l'Education).

Cependant, l'implémentation de ces modules n'a pas été réalisée en utilisant les outils proposés par les "ToolBox" du MacIntosh, par manque de temps également.

OBJECTIFS DU PRESENT MEMOIRE

Le but final poursuivi dans le cadre du présent mémoire est d'aboutir à un logiciel opérationnel, c'est-à-dire un logiciel qui offre les qualités suivantes : la conformité et la complétude par rapport aux spécifications élaborées par F.Rinclin, l'extensibilité aisée (c'est-à-dire l'ajout aisé de nouvelles fonctionnalités) et une certaine facilité d'utilisation. Cependant, obtenir un logiciel opérationnel ne signifie pas pour autant que celui-ci pourrait être diffusé tel quel : il faudrait encore apporter certaines améliorations dans le but d'obtenir une application tout à fait conviviale : on pourrait, dans un premier temps, rendre le logiciel utilisable par des non-spécialistes et par la suite, après une période d'essai, tenir compte des remarques des utilisateurs pour modifier et améliorer le logiciel.

Etant donné ce but final, une des étapes à réaliser obligatoirement est de terminer l'implémentation des modules spécifiés mais non implémentés par F.Rinclin . Il s'agit, en gros, des modules qui permettent la partie "Réalisation de tests" et la partie "Analyse".
Il serait également indispensable de reprendre les modules déjà implémentés et de les modifier en utilisant les "ToolBox", afin de profiter de toutes les potentialités offertes par le matériel choisi.

Mais la réalisation d'une application conviviale constitue un problème important à analyser en tant que tel mais également de façon plus générale que dans le cadre limité de cette application. Une part importante de ce mémoire sera donc consacrée à l'étude de certains problèmes soulevés par la réalisation d'un logiciel convivial.
A la suite de cette étude plus théorique, nous serons alors à même de mettre en oeuvre dans l'application choisie les principes dégagés.

Il est tout d'abord important de définir avec précision les objectifs à atteindre, de sorte qu'ils ne soient ni trop restrictifs, ni trop ambitieux, et ce afin de ne pas devoir en abandonner une partie.

Quelles sont les voies possibles qui s'offrent dans le prolongement du mémoire de F.RINCLIN si l'on tient compte du temps imparti pour la réalisation du projet et la rédaction du mémoire, du matériel choisi, des possibilités techniques offertes mais surtout des critiques et des souhaits émis par les pédagogues ?

La première possibilité envisagée est d'étoffer le logiciel actuel, en lui ajoutant d'autres fonctionnalités pour obtenir un logiciel plus complet :

F. Rinclin a retenu principalement trois types de tests et les a précisés dans la spécification informatique du problème.

Cependant, les pédagogues avaient émis d'autres souhaits comme la possibilité de tests sous forme de questionnaires à choix multiples ou de questionnaires dont les réponses se trouvent dans un ensemble d'éléments ou bien encore, la possibilité de réaliser des dessins....

Le questionnaire à choix multiples (QCM) pourrait actuellement être intégré sans problèmes : en effet, tout en conservant le même matériau de travail que pour les trois types de tests spécifiés, à savoir un texte, le QCM aurait pour but de tester la mémorisation, par exemple, de certains détails. On pourrait ainsi observer les progrès d'une personne testé au fur et à mesure des lectures successives.

Le gros avantage ici, serait que ce type de test formerait un tout avec les trois autres types de tests considérés puisqu'il utiliserait un même texte comme matériau de base.

Le questionnaire dont les réponses se trouvent dans un ensemble d'éléments pourrait, lui aussi, être utilisé dans le même contexte que le QCM. Mais, tout en ayant l'avantage de laisser une plus grande liberté d'expression à la personne testée, il présente l'inconvénient pour le pédagogue de rendre la correction et l'analyse des réponses plus complexes que dans le cas du QCM.

Cependant ce style de questionnaire peut avoir son utilité pour tester d'autres éléments que le QCM tout en conservant le texte de travail comme matériau de base.

La réalisation de dessins trouverait, quant à elle, moins bien sa place dans cet ensemble qui forme un tout centré du concept "texte de travail".

Cependant, avant de songer à accroître les possibilités du logiciel actuel, il est sans aucun doute plus important de s'attacher d'abord à rendre le sous système utilisable pratiquement par des non-spécialistes en informatique. Ainsi, après une période d'essai sur le terrain, on pourra tenir compte des remarques des psycho-pédagogues afin de modifier ce qui existe et d'y ajouter d'autres fonctionnalités.

Dans l'optique de perfectionner ce qui existe avant d'accroître les possibilités, on peut penser à améliorer la convivialité du sous-système afin que même un utilisateur sans expérience informatique puisse l'utiliser facilement. Actuellement, bon nombre de chercheurs s'intéressent aux problèmes posés par l'utilisation des ordinateurs, et en particulier les questions relatives à la conception d'interfaces homme-machine qui soient le plus agréables possibles. A partir de résultats d'études et d'observations, certains spécialistes ont émis des suggestions et des recommandations qui peuvent être un guide lors de la conception d'un logiciel, et plus particulièrement de son interface avec l'utilisateur.

Une voie intéressante pourrait être de mettre en oeuvre et d'appliquer ces conseils afin d'obtenir un logiciel le plus agréable possible à utiliser. De plus, le type d'ordinateur choisi, l'Apple MacIntosh, se prête particulièrement bien à une telle réalisation.

Mais la principale amélioration du logiciel concerne sa robustesse par rapport à l'utilisateur :

- après la mise au point du logiciel, ne plus rencontrer d'erreurs qui font sortir de l'exécution du programme et retourner au niveau système ou qui laissent l'utilisateur devant une machine 'morte';

- rendre le logiciel plus tolérant aux erreurs de l'utilisateur;

- réduire les occasions propices aux erreurs;

- augmenter la capacité de l'utilisateur à détecter ses propres erreurs;

- fournir à l'utilisateur des moyens aisés de correction pour les erreurs qu'il pourrait encore faire.

Pour réduire les occasions d'erreurs, il faut veiller à indiquer clairement à l'utilisateur ce qu'il doit faire, les données qu'il doit introduire, la syntaxe de ces données, les manipulations à effectuer, les possibilités offertes... On peut aussi proposer une aide "on-line" afin que l'utilisateur ne doive retenir qu'un minimum de choses pour se servir du logiciel.

Augmenter la capacité de l'utilisateur à détecter ses propres erreurs peut se faire en lui proposant des messages d'erreurs clairs et spécifiques, qui lui expliquent, de façon précise, la cause et la nature de l'erreur commise et lui indique aussi ce qu'il doit faire.

Faciliter la correction des erreurs peut se faire en permettant des marche arrière faciles à tout moment; il faut entre autre que l'utilisateur puisse remettre le logiciel dans l'état où il se trouvait avant la commande qu'il vient de réaliser et qu'il puisse sans difficulté revenir au menu principal. Mais tout cela doit cependant se faire sans que des dialogues longs et fastidieux soient inutilement imposés à l'utilisateur.

On peut classer les erreurs en deux grands types :

- d'une part, il y a les erreurs qui surviennent suite à une action de l'utilisateur, comme par exemple, le choix d'une option inexistante, l'introduction d'un chiffre au lieu d'une lettre..

- d'autre part, il y a les événements imprévus qui peuvent arriver à n'importe quel moment et mettre l'utilisateur dans une situation embarrassante. C'est le cas entre autres des erreurs 'système', comme par exemple une erreur d'écriture ou de lecture , ou le manque de place sur un disque, ou encore un problème à l'imprimante...

Les erreurs dues à une action de l'utilisateur sont plus aisément détectables : il existe des méthodes qui sont assez lourdes mais qui permettent néanmoins de détecter les erreurs de manière systématique.

Par contre, les erreurs 'système' sont plus difficiles à prévoir et à traiter puisqu'elles peuvent survenir à tout moment et qu'elles dépendent de l'environnement machine.

Mais avant d'être capable d'intégrer une détection et une gestion des cas d'erreurs, il faut tout d'abord mettre au point une méthode de conception d'un logiciel comportant un tel traitement.

Une première partie de ce mémoire s'attachera donc à la recherche et à la description d'une telle méthode qui devra être générale c'est-à-dire à la fois indépendante de l'application considérée mais aussi du langage de programmation utilisé.

Ensuite, dans une deuxième partie, nous appliquerons au cas du logiciel qui nous intéresse les principes de la méthode décrite.

PARTIE I

CONCEPTION D'UNE GESTION D'ERREURS DANS UN PROGRAMME

CHAPITRE 1

PRESENTATION DU PROBLEME

1.1 INTRODUCTION

Lors du processus de développement d'une application, la complexité de conception fait qu'il est préférable de concevoir d'abord le logiciel en fonction des cas normaux : on suppose que l'utilisateur obéit à la lettre aux directives indiquées et que la machine fonctionne parfaitement, sans panne et sans défaillance.
Le logiciel réalisé sous cette hypothèse d'un monde idéal répondra aux spécifications de l'analyse fonctionnelle mais sera peu agréable à l'utilisation.

Malheureusement ni l'utilisateur, ni l'environnement ne répondent ni parfaitement, ni en tous temps à ces critères, et la conception d'un logiciel doit protéger l'utilisateur contre lui-même mais aussi contre le système. En effet, quelque soit son niveau d'expérience avec le logiciel, il arrivera que l'utilisateur commette des erreurs : l'utilisateur débutant, par exemple, au cours de sa découverte du logiciel, poussé par sa curiosité ou une mauvaise interprétation, fera souvent des fautes, tandis que les erreurs de l'utilisateur expérimenté seront plutôt dues à la distraction, à la fatigue ou à la précipitation.

Le logiciel doit être conçu pour que les erreurs commises n'aient pas de conséquences catastrophiques et qu'elles puissent être corrigées aisément.

Le traitement des exceptions est une question intéressante à laquelle on accorde actuellement de plus en plus d'importance, dans le cadre plus général de ce que l'on appelle la convivialité des logiciels interactifs.

Ce problème sera examiné sous deux angles différents :

On peut tout d'abord envisager le traitement des erreurs à la façon d'un spécialiste en ergonomie, et considérer la tolérance aux fautes comme un critère de convivialité que l'on veut respecter.

On peut aussi se placer du point de vue du concepteur et voir comment intégrer une bonne gestion d'erreurs dans la conception d'un logiciel.
La dernière partie de ce premier chapitre consistera avant tout en un aperçu succinct du problème tel qu'on peut le trouver dans la littérature parlant de ce sujet qui sera développé en détails au cours du deuxième chapitre.

1.2 LA GESTION DES ERREURS ET LA CONVIVIALITE

A.La convivialité : un problème actuel

Depuis quelques années, les spécialistes en ergonomie se penchent de plus en plus attentivement sur les problèmes liés à la communication entre un homme et un ordinateur.

Cet intérêt récent est dû à plusieurs facteurs qui ont changé au cours du temps :

1°) Tout d'abord, la technologie a évolué et rend maintenant possible, sur le plan technique, la création d'interfaces plus conviviales. Les progrès les plus remarquables se manifestent surtout en ce qui concerne les appareils d'entrées-sorties : les claviers ont évolué, les écrans couleurs ont fait leur apparition, la souris et les autres mécanismes de désignation directe prennent de plus en plus d'importance.
La convivialité est ainsi devenue technologiquement possible.

2°) Non seulement, la convivialité est devenue réalisable sur le plan technique, mais elle est aussi devenue nécessaire parce que les utilisateurs et les conditions d'utilisation ont changé :

L'informatique n'est plus maintenant l'apanage d'un petit nombre d'initiés et de spécialistes mais elle se répand de plus en plus dans la vie quotidienne, sous différentes formes. Elle doit se mettre à la portée du grand public. En outre, beaucoup de logiciels s'adressent à des enfants ou des handicapés, c'est-à-dire des personnes ayant des besoins et des exigences très spécifiques.

Au début de l'apparition des ordinateurs, ceux qui travaillaient sur ces machines étaient des spécialistes, et bien souvent, ceux qui utilisaient des logiciels étaient aussi ceux qui les réalisaient. Ils savaient comment fonctionnaient leurs programmes et étaient peu exigeants vis-à-vis de l'interface. Comme peu de personnes, à part eux, utilisaient les ordinateurs, les spécialistes imposaient leurs standards à un public souvent limité et localisé. Maintenant le phénomène s'inverse : l'informatique se diffuse, tant en profondeur, à un même endroit, que de par le monde (un même produit hardware ou software est maintenant utilisé par des personnes de langues, de formations et de cultures différentes) et il faut alors distinguer les concepteurs et les utilisateurs. Les utilisateurs ont un poids de plus en plus important et se sont eux qui dictent leur loi et réclament des logiciels qui soient d'un usage aisé et agréable.
De plus, les interfaces graphiques permettent de résoudre le problème des langues : en effet, les icones et autres objets graphiques sont un langage pictural universel, compréhensible de tous.

L'informatique apparaît également dans les lieux de travail. L'introduction de ces technologies nouvelles suscite parfois chez les futurs utilisateurs des sentiments de crainte ou de refus. Cela peut aboutir à une résistance face au changement ou à une démotivation face au travail. Il faut lutter contre ce phénomène en offrant de meilleures conditions de travail; cela peut se faire par exemple en proposant une meilleure formation, ou bien, en rendant l'outil de travail plus agréable à utiliser.

Beaucoup de livres ont ainsi été publiés dans ce domaine afin d'aider les concepteurs de logiciels à se rendre mieux compte des difficultés que peut rencontrer l'utilisateur face à un logiciel et à faire correspondre leur logiciel avec les processus typiquement humains. Cette prise de conscience est plus critique encore dans le cas d'utilisateurs inexpérimentés.

Certains de ces livres sont des guides qui se basent sur la connaissance des processus cognitifs typiquement humains et sur des résultats expérimentaux et présentent ensuite toute une série de recommandations pratiques à l'usage des concepteurs.
Ces recommandations ne sont cependant pas des règles absolues, car elles ne peuvent s'appliquer qu'en tenant compte de la tâche et de l'utilisateur auquel s'adresse le logiciel.

3°) L'ensemble des ces facteurs technologiques, humains et commerciaux ont entraîné la définition implicite ou réfléchie de nouveaux standards : étant donné l'intérêt commercial présenté par le nombre de plus en plus grand d'utilisateurs, la concurrence commence à jouer entre les firmes : chacune tente de faire mieux que les autres et de mettre le plus d'atouts possible de son côté en proposant des logiciels qui soient agréables à utiliser, en sachant que l'utilisateur choisit en fonction de ce qui lui semble le mieux , et comme il n'est pas spécialiste, il juge en fonction de ce qu'il voit au premier abord, c'est-à-dire, l'interface.

Peut-on rêver d'une machine fantastique, universelle et conviviale?

La technologie évolue sans cesse, les utilisateurs eux aussi progressent et exigent de plus en plus et enfin, les interfaces évoluent : ces trois paramètres sont liés et condamnés à évoluer ensemble. Cette évolution est semblable à la destinée humaine, à la recherche de l'infiniment grand et de l'infiniment petit, on y tend de plus en plus sans jamais l'atteindre, en laissant toujours la place à de nouvelles recherches.

B.L'ergonomie : une solution aux problèmes de conception d'interfaces homme-machine

1°)Les déficiences caractérisant la conception des logiciels interactifs

Les problèmes de programmation posés par la première génération d'ordinateurs étaient liés aux contraintes imposées par le matériel. Entre temps, le progrès technologique a permis de lever ces limites, mais les problèmes se posent aujourd'hui en termes de problèmes relatifs à la conception de logiciels interactifs, et donc en termes de méthodologie adaptée.
Les déficiences des méthodologies existantes en matière de conception et de programmation des applications interactives ont donné naissance à un phénomène appelé "crise du logiciel" : cette crise se manifeste par une augmentation du coût de conception et du coût de maintenance des logiciels. Ces mêmes déficiences se font ressentir plus encore lorsqu'il s'agit de réaliser une application qui communique avec l'utilisateur.

Souvent, les concepteurs manquent de connaissances précises relatives aux tâches qu'ils ont à informatiser et relatives aux utilisateurs auxquels ils s'adressent. A cause de cela, ils conçoivent leurs applications en fonction de critères de performance du système plutôt qu'en fonction des critères liés aux objectifs des utilisateurs et aux contraintes dictées par la tâche.

Un autre problème est dû au fait que, la plupart du temps, les logiciels sont conçus par plusieurs personnes, à des moments, voire même en des endroits différents. Cette situation peut éventuellement provoquer une absence d'homogénéité dans la conception et, par la même occasion, des incohérences dans le produit final et dans sa présentation.
Cependant, cet inconvénient, causé par le travail à plusieurs et en parallèle, peut être transformé, et même devenir bénéfique, si il existe une bonne coordination entre les diverses personnes qui travaillent à la conception du logiciel.

Le phénomène le plus inquiétant est l'attitude même des concepteurs qui ont tendance à vouloir à tout prix adapter l'utilisateur à leur logiciel, plutôt que d'essayer, comme cela devrait normalement être le cas, de concevoir un logiciel qui soit adapté à l'utilisateur.
Dans le même ordre d'idées, les concepteurs ne parviennent pas toujours à prévoir les erreurs humaines et à traiter comme il se doit ces cas d'exceptions. Cela provient du fait que les concepteurs ont du mal à se mettre dans la peau d'un utilisateur qui ne connaît rien à leur logiciel.

Enfin, de manière générale, il n'existe pas de méthodologie de conception orientée de façon spécifique vers les problèmes d'interfaces homme-machine.

2°) Conséquences de ces déficiences

Ces défauts dans la conception des logiciels, en général, et dans la conception des logiciels interactifs, en particulier, donnent lieu à des conséquences importantes au niveau de l'utilisation du logiciel et de ses performances.

En effet, si les utilisateurs peuvent parfois, momentanément, compenser les effets d'une mauvaise conception par des efforts supplémentaires d'adaptation, il y a cependant des limites à l'adaptation à une mauvaise interface. De plus, les traces laissées par une expérience négative avec un logiciel ne seront que très difficilement effacées par des expériences positives.

Une interface ou un logiciel mal conçu ou mal adapté peut conduire à des dysfonctionnements du système, à des performances dégradées ou à des plaintes et des rejets de la part des utilisateurs. De manière plus précise, on pourra constater une diminution de l'utilisation du logiciel, lorsque celle-ci est facultative, accompagnée d'un retour aux procédures manuelles. Lorsque l'utilisation est obligatoire, les utilisateurs exploiteront alors une partie seulement des possibilités du logiciel afin d'éviter les difficultés rencontrées au cours de la réalisation d'une opération précise.

Les premiers symptômes occasionnés par ces défauts sont l'augmentation du taux d'erreur et la diminution des performances. Les frustrations, les rejets et les abandons sont alors monnaie courante; et, dans les cas les plus graves, on peut même assister à des boycottages ou des sabotages.

La seule solution pour résoudre les problèmes qui découlent de ces déficiences est alors la modification du logiciel.
Or modifier un logiciel coûte cher : d'après certaines études, le coût de maintenance d'un projet informatique représente 67% du coût total du projet. De plus, la modification d'un logiciel provoque bien souvent l'apparition de nouvelles erreurs qu'il faut ensuite corriger, et on entre alors dans la ronde infernale des corrections et des erreurs...

En conclusion, les défauts dans la conception de logiciels interactifs donnent lieu à des pertes de temps, à des diminutions de la productivité, à des efforts et des investissements perdus, et, ce qui est plus inquiétant, à l'échec vis-à-vis des avantages que pourrait procurer une informatisation qui se déroulerait sans problème.

3°) Une solution : le recours à l'ergonomie

La conception des interfaces homme-machine ne devrait plus reposer, comme c'est actuellement le cas, sur des opinions ou des jugements personnels, mais plutôt sur l'application systématique de connaissances.

L'ergonomie peut apporter cette connaissance manquante en termes de méthodes ou de résultats.

Mais l'ergonomie est inutile et, en tout cas, plus coûteuse si elle n'est appliquée qu'à posteriori sur base des critiques des utilisateurs émises une fois le logiciel réalisé et totalement opérationnel.

Pour atteindre l'objectif souhaité, des solutions ergonomiques doivent être appliquées à chaque étape de la conception du logiciel, le plus tôt possible, et non pas à posteriori lors d'une étape supplémentaire qui serait une évaluation extérieure non intégré à la conception.

C.Des principes généraux à respecter pour une interface conviviale

B.Shneiderman, dans son livre "Designing the user interface : strategies for effective human computer interaction", donne une liste de huit "règles d'or" à suivre pour concevoir une interface homme-machine la plus conviviale possible.

Ces règles sont :

- la lutte pour la cohérence aussi bien dans la présentation que dans le déroulement des séquences d'opérations à exécuter;

- la possibilité d'utiliser des raccourcis, ce qui peut être intéressant lorsque l'utilisateur acquiert une certaine maîtrise de l'utilisation du logiciel;

- la nécessité de "feed-back" après chaque action, ce qui permet à l'utilisateur de contrôler la bonne exécution des opérations;

- la structuration du système en séquences de commandes par blocs pour que l'utilisateur puisse constater sa progression dans la suite des opérations qu'il doit effectuer;

- le pilotage des opérations par l'utilisateur afin qu'il se sente plus en confiance puisque c'est lui qui dirige le système;

- un traitement simple des erreurs : il faut concevoir le système de telle sorte que l'utilisateur ne puisse jamais commettre une erreur qui soit sérieuse sur le plan de ses conséquences. Si malgré tout une erreur survenait, le système devrait la détecter et offrir des moyens simples, non seulement, pour la corriger mais aussi en vue de rendre l'utilisateur capable de détecter par lui-même ses erreurs;

- l'existence d'une fonction "marche arrière" facile et accessible à tout moment : toutes les actions doivent autant que possible être réversibles, afin de réduire l'anxiété de l'utilisateur puisqu'il sait qu'il peut faire marche arrière à tout moment. Cela lui permettra également de découvrir par lui-même les potentialités du système;

- la réduction de la charge de mémorisation requise de l'utilisateur, puisque la probabilité d'erreur humaine augmente lorsque la charge informationnelle est élevée;

A la lecture de ces règles, on remarque l'importance accordée, par les spécialistes en ergonomie des interfaces homme-machine, au problème des erreurs humaines et à la nécessité d'un bon traitement de ces erreurs commises par l'utilisateur.

De plus, d'après des études rapportées dans ce même livre de B.Shneiderman, il ressort ceci :
en moyenne sur différents types de logiciels, les utilisateurs novices font jusqu'à 19% d'erreurs dans leurs commandes; les utilisateurs expérimentés environ 10% de fautes alors que les utilisateurs classés comme les professionnels commettent eux encore 7% d'erreurs.

Cela prouve qu'encore beaucoup de logiciels ne sont pas assez conviviaux ou tolérants aux erreurs. La prise en compte de ces erreurs et leur traitement sont importants non seulement pour les débutants, parce que les effets sont plus proéminents, mais aussi pour les autres utilisateurs, car eux aussi peuvent être moins expérimentés dans certaines parties du logiciel.

Pour toutes ces raisons, nous allons nous pencher un peu plus attentivement sur le problème de la détection et du traitement des erreurs et examiner quels sont les principes ergonomiques plus spécifiques à appliquer à ce problème.

D.Trois principes pour gérer les erreurs

Les principes énoncés ci-dessous sont tirés des livres de B.Shneiderman et de D.L.Scapin dont les références sont indiquées dans la bibliographie.

1°) Premier principe : fournir de bons messages d'erreur

Une façon assez simple de réduire les erreurs est d'améliorer les indications données par le logiciel, c'est-à-dire de présenter des messages plus spécifiques, plus constructifs et de les exprimer sur un ton plus positif.
Les expériences des utilisateurs face aux messages d'aide et d'erreur jouent un grand rôle dans l'acceptation ou la non-acceptation du logiciel : la formulation de ces messages doit être pensée avec soin car les messages sont une guidance et un moyen d'apprentissage pour l'utilisateur débutant, mais, d'autre part, ils ne doivent pas sembler trop lourds ou inutiles pour les utilisateurs expérimentés.
Une solution pour résoudre le problème posé par la formulation des messages d'erreurs est la suivante : on peut imaginer de concevoir une application comportant plusieurs niveaux de messages, adaptés aux différents utilisateurs selon leur expérience avec le logiciel.

Voici quelques recommandations pour aider à obtenir des messages d'erreur efficaces :

a) Spécificité et précision

Des messages d'erreurs trop généraux ou trop simples ne sont d'aucune aide pour l'utilisateur car ils rendent difficiles la découverte et la compréhension de l'erreur commise.
D'autre part, les messages servent d'aide-mémoire pour l'utilisateur expérimenté : la formulation doit être suffisamment précise et univoque pour constituer un rappel.

Certains messages d'erreurs se composent uniquement d'un code qui fait référence à une explication plus détaillée donnée dans le manuel d'utilisation. Un tel procédé est à éviter car, si il permet de réduire la longueur des messages affichés à l'écran, il peut ne pas être d'une grande utilité : le manuel d'utilisation n'est pas toujours à la disposition de l'utilisateur à ce moment précis et sa consultation prend du temps et interrompt l'utilisateur dans la tâche qu'il était occupé à réaliser.
Cependant, le manuel d'utilisation doit répertorier tous les messages d'erreurs en les accompagnant d'une explication ou d'un renvoi à une autre partie de manuel.

b) Guidance constructive et ton positif

Les messages, en plus de rapporter simplement l'erreur, devraient indiquer ce que l'utilisateur peut faire pour corriger son erreur. Mais, il n'est pas toujours facile pour le concepteur de deviner les intentions de l'utilisateur et de parvenir à formuler des messages constructifs.

Certains concepteurs optent pour la correction automatique des erreurs commises par l'utilisateur; ce choix peut être intéressant lorsque les erreurs sont dues à la distraction ou à la précipitation, mais il comporte un désavantage pour un utilisateur débutant : il n'apprendra pas la syntaxe ou la manipulation du logiciel en tirant profit de ses erreurs et de leur correction.
Une autre possibilité est d'offrir différentes solutions de correction à l'utilisateur et de le laisser décider quelle est celle qui lui convient le mieux.

La formulation des messages doit se centrer sur l'utilisateur et encourager l'utilisateur plutôt que de condamner ses erreurs.

c) Contrôle du système par l'utilisateur

C'est l'utilisateur qui doit piloter le système, et non pas le contraire. L'utilisateur doit pouvoir choisir le type d'information qui lui est fournie : en effet, si pour un utilisateur expérimenté, la concision est un avantage, elle est parfois mal adaptée au cas d'un utilisateur débutant.

On peut, pour répondre aux besoins des deux types d'utilisateurs, présenter un message d'erreur standard qui soit le plus bref possible, tout en restant précis. Puis, s'il le souhaite, l'utilisateur peut, à la demande, obtenir des explications supplémentaires plus détaillées.

d) Présentation à l'écran

La localisation des messages d'erreurs sur l'écran a donné lieu à bien des discussions. Certains pensent qu'il faut placer ce message au centre de l'écran pour être sûr que l'utilisateur s'aperçoive qu'il a commis une erreur. D'autres préfèrent les disposer tout en bas de l'écran pour ne pas interrompre l'exécution de la tâche. Quoiqu'il en soit, la localisation des messages doit être cohérente à travers tout le logiciel.

La plupart des utilisateurs préfèrent lire des messages contenant à la fois des majuscules et des minuscules; On réservera les messages en majuscules pour signaler les erreurs graves.

Certains logiciels utilisent un signal sonore lorsqu'une erreur survient : ce "beep" peut être utile lorsque l'utilisateur pourrait ne pas remarquer l'erreur, mais bien souvent ce signal représente une source de distraction et une gêne pour l'opérateur et son entourage. Dans ce cas aussi, l'utilisation du signal sonore doit être soumise au contrôle de l'opérateur.

e) Développer des messages efficaces

Si le logiciel s'adresse à des utilisateurs débutants, une attention toute particulière sera portée à la conception, à l'implémentation et au test de l'interface. Les messages d'erreurs doivent faire partie de la phase de conception; ils doivent être revus au cours du développement et évalués ensuite par différents groupes d'utilisateurs.

Il faut aussi rassembler des données concernant la fréquence de chaque message, et donc aussi de chaque erreur. Ainsi, si une erreur s'avérait trop fréquente, il faudrait non pas partir du principe que c'est l'utilisateur qui est en cause, mais bien le logiciel lui-même qu'il faut reconsidérer : il convient alors de modifier le logiciel, de revoir le traitement des erreurs, d'améliorer la formation des utilisateurs ou le manuel d'utilisation, selon les cas.

Il faut aussi revoir les messages au cours du temps afin qu'ils soient toujours en adéquation avec l'environnement.

Bien entendu, améliorer les messages d'erreurs ne transforme pas un logiciel mal conçu ou non adapté en un logiciel parfait. Mais les messages d'erreurs qui sont une des voies importantes de communication entre la machine et l'opérateur, jouent un rôle significatif dans les performances et l'attitude de l'utilisateur.

2°) Second principe : empêcher les erreurs courantes

Améliorer les messages d'erreurs est nécessaire mais insuffisant. Ce n'est qu'un remède, il faut aider l'utilisateur à ne pas commettre d'erreurs et supprimer ainsi le plus possible la nécessité des messages d'erreurs.

Dans bien des cas, le concepteur pourrait lui-même faciliter la tâche de l'utilisateur :

Parfois, pour mener à bien une action, l'utilisateur doit effectuer plusieurs étapes successives, et si une seule est oubliée, l'action ne sera pas exécutée correctement. Si le concepteur essaie le plus possible d'offrir des actions qui ne nécessitent qu'une seule commande, on évitera de tels inconvénients.

De même, il y a des commandes qui utilisent des éléments qui doivent obligatoirement aller par paires, et si l'utilisateur omet un des deux éléments, le logiciel signalera l'erreur. C'est le cas des parenthèses ou des guillemets, par exemple.

Une solution est d'afficher un message dès que l'utilisateur a introduit un premier élément et de ne le faire disparaître que lorsque le second élément a été introduit.
Mais le message peut encombrer l'écran et perturber l'utilisateur.

Une autre solution serait que, lorsque l'utilisateur a introduit le premier des deux éléments, le logiciel place directement l'autre élément et positionne le curseur entre les deux éléments.
Cette solution a le désavantage de ne pas pousser l'utilisateur à apprendre la syntaxe.

La plupart des erreurs se font lorsque l'utilisateur introduit des informations via le clavier, et ces erreurs sont souvent dues à la distraction ou à des fautes de frappe ou d'orthographe. Dans d'autres cas, l'utilisateur ne donne pas les informations sous la forme attendue par le logiciel, par exemple, parce qu'il n'a pas compris ce qu'il devait faire.

Pour éviter les erreurs dues à la distraction, le logiciel doit tolérer les erreurs de frappe ou les fautes d'orthographe, être capable de compléter une commande dont on ne donne que les premières lettres, et avant tout, interpréter les majuscules et les minuscules comme ayant la même valeur.

L'entrée de chiffres "0" en début de nombre doit être optionnelle et il ne doit pas y avoir de distinction entre un et plusieurs blancs dans une chaîne de caractères. De plus, l'emploi des séparateurs dans les commandes devrait être standardisé.

En règle générale, on peut essayer de réduire le nombre de commandes qui se font par frappe au clavier en utilisant le curseur ou la souris pour pointer une commande sur l'écran. Mais, certains utilisateurs n'aiment pas employer la souris, et il faudrait alors permettre les deux solutions (frappe au clavier ou souris), au choix de l'utilisateur.

Pour éviter les erreurs dues au fait que l'utilisateur ne comprend pas ce qu'on lui demande, ou bien ne comprend pas une erreur ou encore a oublié la signification d'une commande, il doit y avoir une aide à la demande.
La meilleure solution serait de lui fournir une aide, automatiquement, chaque fois que le logiciel pourra déterminer que l'utilisateur est en difficulté.
Ici aussi, il faut laisser à l'utilisateur le contrôle de la présence et du niveau de l'aide qu'il souhaite selon son niveau d'expérience.

3°) Troisième principe : rendre difficiles les actions irréversibles et fournir des actions réversibles

a) Rendre difficiles les actions irréversibles

Lorsque l'utilisateur doit entrer des données, il faut lui offrir la possibilité de revoir et de modifier ces données avant l'action explicite d'introduction.
L'utilisateur évitera ainsi une perte de temps due à l'affichage d'un message d'erreur, à la correction et à l'introduction de la donnée corrigée.

Une commande qui peut avoir des effets destructeurs doit solliciter une confirmation de l'utilisateur avant son exécution; l'aide pour une telle confirmation doit être claire et précise. Mais le recours à la confirmation ne doit se faire que dans les cas vraiment nécessaires : si on demande à l'utilisateur des confirmations trop nombreuses, on risque de provoquer une lassitude ou une habitude et l'effet obtenu est l'inverse de l'objectif visé.

Lorsque l'utilisateur termine une session ou quitte le logiciel avec un risque de perte de données, il doit y avoir un message signalant le danger et demandant confirmation à l'utilisateur.

b) Fournir des actions réversibles

Il faut prévoir dans le logiciel des moyens faciles de revenir en arrière et d'annuler les effets de toute opération.
Mais ce principe doit être appliqué avec discernement : parfois, en effet, offrir une fonction marche arrière pour toute opération peut complexifier le logiciel et embrouiller l'utilisateur. Il faut alors faire un choix entre la réversibilité et la simplicité.

A un niveau plus restreint, lorsqu'une donnée ou une commande introduite par l'utilisateur est erronée, le logiciel, après avoir détecté et signalé l'erreur, doit offrir la possibilité de corriger l'élément inapproprié sans devoir entrer à nouveau la donnée ou la commande complète.
L'édition se fera de préférence par remplacement des anciennes données par les nouvelles.

E.L'implication des utilisateurs est nécessaire

Pour le concepteur déjà familiarisé avec l'utilisation de l'ordinateur, il n'est pas toujours aisé de se mettre à la place d'un utilisateur de son logiciel. Cela est plus vrai encore lorsque l'utilisateur découvre non seulement le logiciel mais aussi la manipulation d'un ordinateur.
Même en faisant l'effort de se rappeler ou de s'imaginer les difficultés rencontrées lors de la première approche d'un logiciel, le concepteur peut ne pas partager l'avis d'autres utilisateurs quant à ce qui est simple et ce qui ne l'est pas.

De plus, le concepteur ne connaît pas la tâche à exécuter aussi bien que l'utilisateur, souvent habitué à ce travail mais avec une procédure manuelle.

C'est pourquoi le concepteur doit travailler en collaboration avec les futurs utilisateurs. Cette participation des utilisateurs ne doit pas se restreindre à l'étape d'élaboration du cahier des charges : une validation de la part des utilisateurs doit avoir lieu à la suite de chaque étape du processus de développement.
En particulier, lorsque le logiciel ou une de ses parties peut être utilisé, il faut soumettre ce prototype aux critiques des utilisateurs, en d'autres termes, réaliser des tests d'acceptation.
On y observera les réactions des utilisateurs, les difficultés qu'ils rencontrent, leurs commentaires, en un mot, les raisons de leur satisfaction ou de leur insatisfaction.

Pour les concepteurs, ces critiques apportent beaucoup : elles permettent de modifier une partie du logiciel qui serait inadéquate, d'améliorer la présentation ou le traitement des erreurs, de veiller à une meilleure formation ou à un manuel d'utilisation mieux adapté.
Ce qui est intéressant, c'est que ces modifications sont plus aisées et moins coûteuses si elles sont faites avant que le logiciel ne soit entièrement réalisé et voire même lancé sur le marché.

Cependant, il ne faut pas oublier que le concepteur peut souvent imaginer plus aisément que l'utilisateur des façons nouvelles de résoudre certains problèmes, façons que l'utilisateur de l'outil informatique autorise alors qu'elles étaient impensables dans le contexte manuel. Il est donc indispensable que l'informaticien cherche le plus possible à comprendre les besoins de l'utilisateur et ne se contente pas de se conformer à ses desiderata.

F.CONCLUSION

Cette première partie avait pour objet d'exposer le traitement des erreurs sous son aspect externe, c'est-à-dire vu sous l'angle de l'utilisateur ou du spécialiste en ergonomie.

La convivialité des logiciels est un problème qui suscite de plus en plus d'intérêt étant donné que les ordinateurs se répandent dans la vie quotidienne et sur les lieux de travail, au lieu d'être le privilège de spécialistes.

Les logiciels interactifs sont de plus en plus nombreux mais les méthodes traditionnelles de conception de logiciels présentent des faiblesses face à la conception d'interfaces conviviales. Ces déficiences sont lourdes de conséquences en termes de refus de la part des utilisateurs, de perte de temps et d'investissements inutiles.
L'ergonomie ne peut pallier les défauts de conception que si elle s'applique le plus tôt possible et à chaque étape plutôt qu'à posteriori, lorsque le logiciel est totalement implémenté.

De nombreux spécialistes en ergonomie ont publié des livres qui sont des guides à l'intention des concepteurs. Toutefois, ces recommandations doivent être utilisées avec discernement, en tenant compte de l'utilisateur et de la tâche à accomplir.
Ces spécialistes ont proposé une liste de conseils pour obtenir une interface conviviale. Ces recommandations découlent de résultats expérimentaux et d'études.
Une d'entre elles concerne le traitement des erreurs dont on voit ainsi l'importance dans la conception d'une interface homme-machine.

On peut donner des principes plus spécifiques au traitement des erreurs. Les principes exposés concernant la gestion des erreurs sont au nombre de trois :

Le premier principe exprime la nécessité d'accorder un soin particulier aux messages d'erreurs, parce qu'ils sont un moyen de communication avec l'utilisateur. Les messages doivent être spécifiques et constructifs. Le contrôle doit être laissé aux mains de l'utilisateur et leur présentation doit être cohérente à travers tout le logiciel.
Pour parvenir à des messages possédant toutes ces caractéristiques, il faut travailler en collaboration avec les utilisateurs.

Le second principe vise à empêcher l'utilisateur de commettre des fautes en lui offrant une guidance mieux appropriée, en lui évitant des actions inutiles, par exemple.

Le troisième principe a pour objet la réversibilité des opérations : il faut offrir des possibilités faciles d'annulation et rendre difficiles les opérations irréversibles.

Enfin, la participation des utilisateurs est une nécessité pour que le concepteur puisse se rendre compte des défauts éventuels de son logiciel.

1.3 LE TRAITEMENT DES ERREURS ET LA CONCEPTION DE LOGICIELS

A.Les qualités d'un bon logiciel

La première qualité d'un bon logiciel est la fiabilité, c'est-à-dire le fait que le logiciel réalise la fonction qui lui a été assignée, dans un environnement donné et pour une période de temps donnée.

Pour être fiable, un logiciel doit être non seulement correct, mais aussi robuste.

Un logiciel est dit correct si il obéit à sa spécification. Le fait qu'un logiciel soit correct ne signifie cependant pas forcément que la spécification corresponde aux intentions de l'utilisateur.
Un logiciel dont la spécification correspond aux desiderata de l'utilisateur est appelé un logiciel conforme.

La robustesse d'un logiciel est sa capacité à gérer les différents types d'erreurs qui peuvent survenir.

La deuxième qualité d'un logiciel est sa maintenabilité, c'est-à-dire le fait que le système puisse redevenir rapidement opérationnel après la découverte d'une erreur ou suite à une modification ou une extension demandée par l'utilisateur.

Le logiciel est dit efficace si il exécute la tâche qui lui est assignée de la façon la plus performante possible.

Le logiciel possède la qualité de portabilité si il peut sans trop de difficultés être adapté à un environnement différent.

Enfin, une autre qualité dont on a déjà parlé longuement est la convivialité.

La robustesse d'un logiciel est donc une caractéristique essentielle si on souhaite obtenir un système hautement fiable.

Dans un environnement réel, les erreurs dans les informations introduites par l'utilisateur, les pannes du matériel et les erreurs dans le logiciel sont toujours possibles et c'est durant la conception du logiciel que des mécanismes devront être développés pour détecter ces erreurs et réduire leur impact sur l'application.

B.Définitions

La langue française contient un grand nombre de mots différents pour désigner "une déviation entre l'état ou le comportement supposé d'un système et son état ou son comportement réel".

Nous allons, avant tout, essayer de définir le sens particulier que l'on donne à ces termes dans le jargon informatique :

"Erreur" est un terme général pour désigner une déviation entre un état ou un comportement attendu et la réalité.

Une "défaillance" est définie comme une erreur causée par le logiciel.

Une "panne" est une erreur due au hardware.

Une "faute" est une erreur commise par l'opérateur, donc une erreur humaine.

Une erreur quelle qu'elle soit, ne peut exister que lorsqu'une norme, un état ou un comportement attendu a été défini. Le terme "erreur" est donc un terme relatif qui perd sa signification en l'absence d'une norme.
Pour un logiciel, la norme est définie par les spécifications du système c'est-à-dire par rapport aux intentions ou aux attentes humaines.

Définir avec précision l'erreur humaine est difficile car ces erreurs ne sont souvent identifiées qu'après coup.
De plus, une faute de l'utilisateur provoque parfois une défaillance du programme, et il devient alors difficile de distinguer les deux types d'erreurs.

Rendre un logiciel tolérant aux fautes, c'est justement essayer d'éviter qu'une erreur commise par l'utilisateur ne provoque une défaillance du programme.
Mais la robustesse, cela peut être plus encore : on peut également permettre à l'utilisateur de récupérer, sans trop de difficultés, une opération fautive dans le sens où elle ne correspond pas à ses attentes, même si cette opération est valide pour le logiciel.

C.La détection automatique des erreurs

Dans ce paragraphe, nous exposerons et nous commenterons de façon critique la technique proposée par H.Kopetz dans son livre dont les références sont mentionnées dans la bibliographie.

En principe, il existe deux techniques pour obtenir un logiciel tolérant aux erreurs de l'utilisateur, c'est-à-dire un logiciel qui soit capable de reconnaître une erreur et d'exécuter une action adéquate : soit on s'assure que chaque composant du logiciel est tout à fait fiable; soit on utilise un système redondant dont les composants sont capables de détecter et de gérer les fautes.

Un système est dit redondant si il contient plus de ressources qu'il n'en aurait besoin pour exécuter correctement sa tâche.

Comme il est plus économique de construire un système redondant plutôt que de se battre avec la fiabilité des composants, quoique cela soit possible techniquement, on préférera la solution de la redondance.

1°) Analyse des mécanismes de détection d'erreurs

La détection d'erreurs est basée sur un critère d'acceptation qu'il faut définir pour chaque étape du processus de détection : si le résultat de l'étape considérée vérifie le critère, alors, il est considéré comme étant correct.
Le système est redondant puisque les ressources utilisées sont plus nombreuses que les ressources nécessaires.

2°) Méthodes de détection d'erreurs

L'exécution d'un module du logiciel peut être décrite comme un processus qui transforme des données appartenant à un domaine d'entrée en des résultats appartenant à un domaine de sortie. Ces domaines sont restreints par des contraintes imposées par les caractéristiques de l'application ou par la structure du programme.
En vérifiant si les données et les résultats de chaque processus élémentaire obéissent aux contraintes, on pourra détecter un certain nombre d'erreurs.
L'exécution d'une instruction de transformation de données est considérée comme un processus élémentaire. A chaque processus élémentaire, on associe un domaine d'entrée et un domaine de sortie.

Nous allons d'abord nous occuper des contraintes déterminées par la structure du programme.
Ensuite, nous traiterons des contraintes imposées par l'application elle-même.

a) Détection des erreurs dans les données

Les contraintes imposées par la structure du programme sont déterminées, d'une part, par les algorithmes utilisés pour réaliser le programme (détection partielle) et, d'autre part, par chacun des processus élémentaires appelés par chacun des algorithmes (détection complète).

* Détection partielle des erreurs dans les données

Appelons "Contraintes intrinsèques" les contraintes imposées par les caractéristiques de l'algorithme.

On peut alors définir la détection partielle des erreurs dans les données comme ceci :

Un module contient une détection partielle d'erreurs dans les données si toutes ses valeurs d'entrée sont vérifiées par rapport à toutes les contraintes intrinsèques imposées par le module.

* Détection complète des erreurs dans les données

La détection partielle est insuffisante pour repérer toutes les erreurs possibles dans les entrées. Des erreurs dans les données des processus élémentaires appelés par un module peuvent aussi survenir; et, si les processus élémentaires ne contiennent pas leur propre détection d'erreurs, de telles erreurs peuvent provoquer des résultats incorrects ou une interruption du système.

Un module contient une détection complète des erreurs dans les données si toutes les valeurs d'entrée sont vérifiées non seulement par rapport aux contraintes intrinsèques imposées par le module, mais aussi par rapport aux contraintes imposées au domaine d'entrée par les processus élémentaires appelés dans le module.

L'implémentation d'une détection complète des erreurs dans les données pour un module particulier nécessite une section de programme supplémentaire. Lorsque la détection complète d'erreurs est implémentée pour chaque module, les sections de programme supplémentaires peuvent entraînent les problèmes suivants :

- Fiabilité :
  L'éventualité d'erreurs dans le processus même de détection d'erreurs ne doit pas être oubliée; Comme la détection d'erreurs s'exécute en série avec les modules qui réalisent le processus normal, l'ajout d'une longue section de code peut provoquer une diminution de la fiabilité du système.

- Performance :
  L'exécution du code supplémentaire implémentant la détection complète d'erreurs prend du temps et diminue la rapidité d'exécution du logiciel.

- Maintenabilité :
  Si les contraintes sur un domaine d'entrée d'un processus élémentaire sont modifiées, il faudra modifier tous les modules qui appellent ce processus élémentaire afin de conserver un mécanisme de détection complète.

- Effort de développement :
  L'implémentation d'une détection complète augmente considérablement l'effort de développement.

A cause de tous ces problèmes, la détection complète d'erreurs n'a pratiquement jamais été introduite en pratique.
Mais, du point de vue du logiciel complet, on peut néanmoins implémenter une détection d'erreurs, si chaque processus élémentaire contient une détection partielle d'erreurs.

Le traitement général des erreurs se fait alors de la façon suivante :

Une erreur qui survient au cours d'un processus élémentaire est signalée au processus appelant. Ce processus appelant peut soit gérer lui-même cette erreur, s'il possède tous les éléments qui le lui permettent, soit faire remonter l'erreur au processus appelant de niveau supérieur. (voir schéma)

Malheureusement, très peu de langages offrent les instructions nécessaires pour implémenter cette technique de détection et de gestion des erreurs de façon simple, claire et efficace.

b) Vérification de plausibilité dans le logiciel

Les techniques décrites auparavant sont indépendantes de l'application particulière implémentée. En plus de ces techniques, il peut être intéressant de vérifier les résultats intermédiaires et les résultats finals en fonction de leur plausibilité déterminée par l'application.

Des données ou des résultats sont dits plausibles si ils correspondent aux attentes de l'utilisateur et au domaine d'utilisation du logiciel.

Les vérifications de plausibilité confronteront donc les résultats attendus aux résultats obtenus et détecteront, d'une part, la complétude des données, et, d'autre part, la conformité physique par rapport au monde réel.

Cette vérification par rapport à la plausibilité se base non seulement sur la spécification du programme mais aussi sur la définition du problème. Ainsi, les erreurs dues à une mauvaise analyse seront également repérées.

Ces vérifications sont tout à fait spécifiques à l'application et doivent être considérées comme une acceptation provisoire des résultats intermédiaires et des résultats finals produits par le logiciel.

3°) Interfaces de gestion d'erreurs

Nous avons traité la détection d'erreurs de façon séparée par rapport au processus normal. Cette séparation doit être maintenue le plus longtemps possible durant la conception du système, pour les raisons suivantes :

- Maintenabilité :
  Si une modification est nécessaire, il faut distinguer la modification elle-même de sa vérification. De cette façon, les erreurs qui seraient dues à la modification sont détectées plus facilement.

- Testabilité :
  Le test du mécanisme de détection d'erreurs se fait plus facilement si il est réalisé dans une section séparée par rapport à la section qui contient le processus normal d'exécution.

- Algorithmes différents :
  Les objectifs différents des deux sections impliquent l'utilisation d'algorithmes différents : dans le section du processus normal, la question qui se pose est "Que reste-t-il à faire?" alors que dans la section de détection d'erreurs, on se pose la question suivante "Qu'est-ce qui n'est pas autorisé?".

-Manipulations différentes des données :
  Dans l'exécution normale, les données sont modifiées; dans la détection des erreurs, elles sont simplement utilisées. Les privilèges sont différents pour les deux sections.

Si on décide de rassembler toutes les gestions d'erreurs en un seul module, il faut alors décider du nombre et de la position des interfaces entre le processus d'exécution normale et la détection d'erreurs.
Le choix du nombre et de la position des interfaces dépend d'abord des objectifs de l'application, c'est-à-dire des erreurs les plus critiques , mais dépend aussi de l'influence de la gestion d'erreurs sur la fiabilité du système dans son ensemble.

Pour déterminer la localisation des interfaces, il faut considérer les points suivants :

- Arguments :
Etant donné la plus forte probabilité d'erreurs dans les données introduites par l'utilisateur, il doit y avoir une vérification après chaque introduction d'informations. Il faut donc prévoir une interface après chaque introduction de données.

- Propagation d'erreurs :
La position des interfaces détermine jusqu'où il faut remonter le report de l'erreur avant qu'elle ne puisse être traitée. Il est nécessaire de bien analyser les conséquences de la propagation de l'erreur avant de décider où placer l'interface.

- Interchangeabilité des composants :
La taille des composants qui doivent être remplacés en cas de correction ou de modification joue aussi un rôle important dans le choix de la localisation de l'interface.

4°) Commentaires

Lorsque l'on parle des spécifications de l'application, il est utile de préciser ce que cela signifie dans le contexte d'une gestion d'erreurs : la spécification d'une application contenant une gestion d'erreurs est plus complète que la spécification de la même application sans gestion d'erreurs. En effet, dans un certain nombre de circonstances, l'application sans gestion d'erreurs a un comportement indéterminé, alors que cette indétermination est levée par la spécification des gestions d'erreurs.
On peut alors distinguer deux sortes de fautes par rapport à l'utilisateur : d'une part, les fautes dont le logiciel peut se rendre compte et, d'autre part, les fautes dues à la non-correspondance par rapport à l'opération que l'utilisateur voulait exécuter. Si le logiciel peut gérer au cas par cas les erreurs du premier type, les erreurs du second type ne peuvent être corrigées que par une possibilité de marche arrière facile, disponible à tout moment.

Il semble également qu'il y ait, dans la technique exposée ci-dessus, une confusion entre ce qui est détecté, c'est-à-dire la nature de l'erreur, et le moment de la détection : du point de vue de la nature des erreurs détectées, on peut effectivement considérer que l'ensemble de toutes les détections partielles permet d'obtenir une détection complète. Cependant, du point de vue du moment de la détection, cette affirmation n'est plus du tout justifiable.

Enfin, la notion même de contrainte intrinsèque est discutable : on peut tout aussi bien considérer que les contraintes relatives à un module sont celles résultant des contraintes propres à l'algorithme utilisé dans le module et de celles de tous les processus élémentaires invoqués par ce module.

D.CONCLUSION

Après avoir considéré le problème du traitement des erreurs du point de vue de l'utilisateur, nous nous sommes penchés sur ce même problème, mais en nous plaçant du côté du concepteur, sur le plan interne par rapport au programme. La question qui nous préoccupe est la suivante : Comment concevoir et implémenter un logiciel qui soit le plus tolérant possible aux erreurs?

Un bon logiciel doit présenter certaines qualités : il doit être fiable (c'est-à-dire à la fois correct et robuste), facile à maintenir, convivial et portable.
La tolérance aux erreurs est un facteur important car des fautes sont fréquemment commises par l'utilisateur.

Le mot "erreur" est un mot vague et général. Avant toute chose, il faut d'abord distinguer et définir clairement le sens que l'on donne au terme "erreur" dans le jargon informatique : on différencie la faute, commise par l'opérateur, la défaillance, provoquée par le logiciel, et la panne, causée par le matériel.

Puis, on peut s'intéresser aux diverses techniques qui permettent de détecter systématiquement les erreurs et de rendre ainsi le logiciel plus tolérant aux fautes, en effectuant une action adaptée à l'erreur découverte.

La solution la moins coûteuse globalement est d'ajouter un module chargé spécialement de détecter les erreurs.
Pour détecter les erreurs, on vérifie que les données de chaque processus élémentaire obéissent aux contraintes imposées par la structure du programme (détection partielle et détection complète des erreurs dans les données), mais aussi aux contraintes dictées par l'application elle-même (vérification de la plausibilité).
Pour diverses raisons, la détection complète est peu envisageable en pratique. Mais il existe une autre solution qui consiste à faire remonter l'erreur de proche en proche à la procédure appelante jusqu'à ce qu'elle puisse être gérée comme il se doit.

La séparation entre le processus normal d'exécution et la gestion d'erreurs est justifiée par la complexité de la conception, par la maintenabilité et la testabilité plus aisées, mais aussi par les algorithmes et les manipulations de données qui sont différentes.

Pour faire la liaison entre le processus normal d'exécution et la gestion d'erreurs, il faut des interfaces dont le nombre et la localisation sont laissés à la décision du concepteur en fonction de l'application.

Cette partie consistait avant tout en un aperçu succinct et théorique du problème : nous y avons en effet présenté un relevé de ce qui existait dans la littérature à propos du problème de la conception de logiciels comportant une gestion des erreurs de l'utilisateur.
Dans la suite, nous nous attacherons à trouver et à décrire avec précision une technique réalisable en pratique et à implémenter ces mécanismes dans le langage cible choisi dans le cadre du logiciel retenu dans le cadre de ce mémoire : le langage Pascal.

CHAPITRE 2

METHODE DE CONCEPTION D'UN SYSTEME DE GESTION DES ERREURS ET DES EXCEPTIONS DANS UN PROGRAMME

2.1 INTRODUCTION

Après avoir examiné ce que disait la littérature à propos de la nécessité d'une bonne gestion des erreurs en vue de rendre un logiciel convivial, nous allons à présent nous pencher plus attentivement sur les problèmes rencontrés par un concepteur pour obtenir une telle application.

Mais avant de décrire plus précisément des techniques de conception d'un système de gestion des erreurs, nous allons définir des concepts généraux, c'est-à-dire des concepts qui se retrouvent dans tout système de gestion d'erreurs.

Ensuite, nous distinguerons les différents types d'utilisateurs, et nous nous demanderons quelles sont leurs attentes par rapport à un système de gestion d'erreurs.

Pour chacun des types d'utilisateurs concernés, nous décrirons le système de gestion d'erreurs optimal selon la structure suivante :

dans un premier temps, nous envisagerons une solution "artisanale", une solution que n'importe quel programmeur peut inclure dans son programme à moindre frais; nous en donnerons le principe général et l'implémentation dans le langage particulier choisi dans le cadre de ce mémoire, le langage Pascal;

ensuite, nous ferons un inventaire rapide des solutions logicielles existantes, qui répondent aux mêmes fonctionnalités que notre solution personnelle;

et, si une telle solution est encore inexistante sur le marché, nous définirons ce que l'on appellera une solution souhaitable, c'est-à-dire ce que nous aimerions voir implémenter dans une solution toute faite.

2.2 CONCEPTS GENERAUX DANS UN SYSTEME DE GESTION D'ERREURS

Tout programme peut être considéré de deux manières différentes :

- un logiciel peut être vu comme une "Black Box", c'est-à-dire que l'on ne connaît du logiciel que les données qu'il reçoit et les résultats qu'il fournit. Le programme est alors vu comme un tout supposé correct et dont le fonctionnement interne n'a pas d'importance.

- un logiciel peut aussi être vu comme une "White Box" : dans ce cas, on ne s'intéresse plus à ses entrées et ses sorties, mais bien à son fonctionnement interne, à la manière dont les résultats sont produits en fonction des données reçues.

Ces deux manières de considérer un programme vont permettre de définir des concepts différents mais complémentaires.

A.Le programme vu comme une "Black Box"

Un programme vu comme une "Black Box" est décrit par ses spécifications; mais, comme on l'a dit plus haut, si le logiciel est capable de traiter les erreurs, il s'en suit une transformation des spécifications par rapport à un programme qui aurait la même fonctionnalité mais qui serait incapable de gérer les erreurs.

Les données à l'entrée d'un programme peuvent être de deux sortes :

- les données correctes, c'est-à-dire celles qui respectent les spécifications du programme, autant sur le plan de leur syntaxe que sur le plan de leur sémantique;

- les données erronées, c'est-à-dire celles qui impliquent un comportement indéterminé pour un logiciel ne contenant pas de gestion d'erreurs, mais qui font l'objet de spécifications explicites dans le logiciel contenant une gestion d'erreurs.

Les résultats sont également séparés en deux types :

- les résultats corrects, produits par des données correctes et spécifiés à la fois dans le logiciel avec gestion d'erreurs et dans le logiciel sans gestion d'erreurs;

- les résultats erronés, produits par des données erronées, qui ne font l'objet de spécifications que dans le cas d'un logiciel avec une gestion d'erreurs.

Si les résultats corrects sont ponctuels, distincts, les résultats erronés peuvent être regroupés en classes d'erreurs : il existe des circonstances reconnues par le programme et définies par des conditions qui doivent être satisfaites.

La classe d'erreur est alors déterminée par la condition non respectée par les données. Il faut alors se donner un moyen de distinguer ces classes d'erreurs entre elles : si elles sont en nombre fini, ce qui sera bien souvent le cas, on peut, par exemple, leur attribuer un numéro servant d'identificateur interne de la classe d'erreur. On peut aussi associer à la classe d'erreur un message d'erreur qui a une fonction d'identificateur externe de la classe d'erreur.

Il faut encore éventuellement pouvoir identifier précisément une erreur par rapport à la classe à laquelle elle appartient : c'est pourquoi on donnera en plus une valeur caractéristique, par exemple, la valeur particulière de la donnée qui a provoqué l'erreur.

B.Le programme vu comme une "White Box"

Le programme est découpé en entités appelées procédures et chaque procédure a les mêmes propriétés que le programme en ce qui concerne ses données et ses résultats (données / résultats corrects et erronés par rapport aux spécifications de la procédure).

Chaque procédure a donc ses propres classes d'erreurs qui lui sont associées, numérotées indépendamment des autres procédures qui constituent le programme.

Dans un programme vu comme une "White Box", il existe deux concepts relatifs aux erreurs. Ces concepts se trouvent cachés si l'on voit le programme de l'extérieur : ce sont la remontée des erreurs et le traçage des erreurs.

1°) la remontée des erreurs

Si le programme appelle une procédure dans une situation erronée, c'est-à-dire, avec des valeurs qui ne correspondent pas à ce qu'attend la procédure, il existe a deux solutions en réponse à cette erreur : on peut soit corriger cette erreur, soit l'interpréter :

- soit la procédure est capable d'agir en conséquence pour faire ce qu'on lui demande, malgré l'erreur ou bien elle est capable de répondre comme il se doit au cas d'exception qui se présente; en d'autres termes, elle est capable de corriger elle-même l'erreur ou l'exception;

- soit la procédure n'est pas apte à traiter elle-même l'erreur, parce que, par exemple, elle ne possède pas tous les éléments nécessaires ou qu'il s'agit d'une faute de l'utilisateur pour laquelle il est nécessaire de le consulter : il faut alors que la procédure force un retour au programme qui l'a appelée, en lui transmettant une valeur qui indique que la procédure a été appelée dans une situation erronée. Si il y a plusieurs procédures avec des appels imbriqués, on procédera de façon récursive pour chacune des procédures. Cependant, à chaque niveau de la remontée, il faut que la classe d'erreur soit significative pour ce niveau, et la classe de l'erreur devra être traduite, interprétée en fonction de l'opération que l'on avait demandé de réaliser à la procédure courante.
  Interpréter une erreur, c'est donc la documenter en amont pour pouvoir la corriger en aval : à chaque niveau d'appel où on remonta, il faut que la classe d'erreur ait une signification pour cette procédure.

2°) le traçage de l'erreur

Connaître la nature de l'erreur qui a eu lieu ne suffit pas toujours, il est aussi parfois intéressant de connaître le chemin d'exécution suivi à travers tout le programme, par exemple, par la succession des appels de procédures qui ont conduit à cette erreur.

Les renseignements qu'il est utile de connaître à propos d'un appel à une procédure sont les suivants :

- quelle est la procédure appelante ?
- quelle est la procédure appelée ?
- l'endroit où se fait l'appel et le nombre d'appels, dans le cas où une procédure appelle plusieurs fois une autre procédure au cours de son exécution

Le traçage permet éventuellement de détailler les causes qui sont à l'origine de la faute et de découvrir une inadéquation du logiciel par rapport aux desiderata de l'utilisateur et ainsi de faciliter l'élimination de cette inadéquation.
Ce mécanisme facilite également la détection de défaillances résiduelles du logiciel (inadéquation par rapport aux spécifications).

2.3 LES UTILISATEURS IMPLIQUES

Etant donné que c'est le type de l'utilisateur qui détermine la façon dont le programme est perçu, il est important de distinguer les différents types d'utilisateurs impliqués.

a) Nous appellerons utilisateur final du logiciel celui qui se sert de l'ensemble logiciel/machine comme d'un outil mis à sa disposition. Il ne connaît ni la réalisation interne du logiciel, ni le fonctionnement interne de la machine. Du programme, il ne voit que les écrans qu'il reçoit et les ordres qu'il introduit au clavier; ceux-ci réalisent l'interface entre l'homme et la machine.

L'utilisateur final n'est intéressé que par les erreurs telles qu'elles sont perçues dans un programme vu comme une "Black Box" : en effet, les seules erreurs qui le concernent sont celles causées par une faute commise par lui-même. Ainsi, par exemple, il lui est inutile de connaître le chemin d'exécution parcouru entre l'introduction de sa donnée et la procédure qui a découvert l'erreur. Par contre, un message indiquant la nature de l'erreur et, le cas échéant, la valeur qui a provoqué l'erreur seront pour lui des renseignements précieux.
Cependant, un mécanisme de remontée des erreurs reste nécessaire à l'intérieur du programme car il faut interpréter, documenter la classe de l'erreur à chaque niveau où l'on remonte, pour que l'erreur soit reportée avec une signification suffisante aux yeux de l'utilisateur final.

b) Le programmeur est celui qui, à partir des spécifications fonctionnelles, a conçu et mis au point le programme. Il intervient avant que le programme ne soit livré à l'utilisateur, ou, lorsque des anomalies de fonctionnement ont été détectées au cours de l'utilisation du logiciel ou, encore, quand des modifications lui sont demandées.

Il est concerné par les erreurs dans le programme vu comme une "White Box", et plus particulièrement l'aspect de traçage de l'erreur est important, pour lui, surtout durant la phase de test. Si la classe de l'erreur est intéressante en ce qui concerne la procédure qui a provoqué l'erreur, l'interprétation de cette erreur à chaque niveau de remontée peut néanmoins renseigner le programmeur sur la logique du mécanisme.

c) L'opérateur est celui qui gère l'ordinateur en tant que machine : il s'occupe de la mise en route de la machine et s'il y a lieu, des périphériques : lecteurs de disques ou de bandes, imprimantes,...
Lorsque la machine utilisée est simple (PC, par exemple), l'opérateur et l'utilisateur sont bien souvent une seule et même personne.

Les erreurs qui le concernent sont les mêmes que pour l'utilisateur final, en se limitant aux erreurs et exceptions relatives à l'état de la machine et des périphériques.

Les objectifs des trois catégories d'utilisateurs face à un système de gestion d'erreurs sont donc, on l'a vu, assez différents :

D'une part, l'utilisateur final et l'opérateur souhaitent être informés des erreurs qu'ils ont commises ou des exceptions qui les concernent, et ces erreurs doivent leur être signalées de façon compréhensible, en adéquation avec l'action qu'ils tentaient d'effectuer et non par rapport à l'instruction qui a provoqué l'erreur. Ce qu'il leur faut, c'est donc un système de gestion d'erreurs capable, à chaque niveau du programme où l'on remonte, de transformer la classe d'erreur en une autre classe d'erreur qui soit significative pour ce niveau; Nous appellerons ce système, un "système de gestion des erreurs de l'utilisateur".

D'autre part, le programmeur désire connaître un maximum d'informations à propos de l'erreur qui a eu lieu, afin d'en trouver la cause, de la localiser avec précision et de pouvoir corriger rapidement et efficacement les instructions du programme qui sont à l'origine de cette erreur.
Ce qu'il lui faut, c'est un système de gestion d'erreurs qui soit une aide lors de la mise au point ou de la correction d'un programme; ce système sera désigné comme un "système de gestion des erreurs du programmeur".

En raison de la différence d'exigences entre ces deux types d'utilisateurs, nous analyserons séparément les deux systèmes de gestion d'erreurs. Nous commencerons d'abord par parler du système de gestion des erreurs du programmeur; nous consacrerons la fin du chapitre au système de gestion des erreurs de l'utilisateur, ce qui permettra de faire la transition avec la partie suivante qui décrit l'application d'un tel système dans le logiciel retenu dans le cadre de ce mémoire, un logiciel de tests sur la mémoire humaine.

2.4 SYSTEME DE GESTION DES ERREURS DU PROGRAMMEUR

A.Solutions envisageables

1°) Principe général

Nous avons dit que les renseignements qui intéressaient le programmeur face à une erreur causée par une faute dans le programme étaient les suivants :

- la nature de l'erreur qui a été détectée;
- le chemin d'exécution à travers les différentes procédures du programme qui a abouti à l'erreur, c'est-à-dire le traçage de l'erreur.

Les informations relatives au traçage de l'erreur correspondent à la pile des appels et retours de procédures associée à tout programme. Cependant, cette pile, bien qu'elle existe, n'est pas facilement accessible ni traitable par un programme.
C'est pourquoi, il est préférable d'utiliser une représentation plus manipulable : une pile dont chaque élément contient le nom ou le numéro de la procédure appelée ainsi qu'une indication de l'endroit où se fait l'appel.

Le principe est alors le suivant :

- Chaque appel à une procédure est suivi, à son retour, d'un test qui vérifie si aucune erreur n'a eu lieu durant l'exécution de la procédure appelée. Si aucune erreur n'a eu lieu, l'exécution poursuit son cours normal; Si une erreur est survenue, on ajoute à la pile l'identifiant de la procédure ainsi que l'indication de l'endroit de l'appel; et, on force un retour à la procédure appelante qui elle-même fait la même chose, et on remonte ainsi jusqu'au programme principal qui traite l'information reçue et communique au programmeur le contenu de la pile, la classe de l'erreur et la valeur caractéristique d'une variable en rapport avec l'erreur.

- Toute instruction susceptible de provoquer une erreur est elle aussi suivie du test sur erreur. Si il n'y a pas d'erreur, la procédure continue son exécution normale. Si une erreur est détectée, on affecte au code d'erreur le numéro de la classe de l'erreur détectée et on positionne la valeur caractéristique; on force ensuite un retour à la procédure appelante jusqu'au programme principal.

L'avantage de cette méthode réside dans le fait qu'elle est simple et systématique à appliquer. Elle est aussi complètement indépendante du langage utilisé puisqu'elle n'emploie que des structures de données qui représentent le code d'erreur et la pile des appels.

2°) Application en langage Pascal

En Pascal, on peut représenter la pile par une liste chaînée de "records" ayant comme champs une chaîne de caractères, contenant le nom de la procédure appelée, et un nombre représentant l'endroit de l'appel :

```
TypProc = record
              nomproc : string;
              nappel  : integer;
              procsuiv: ^Typproc;
          end;

TypPile = ^Typproc;
```

Le code d'erreur, contenant l'identifiant de la classe de l'erreur, telle qu'elle a été traduite sous forme d'un numéro et d'un message explicatif, sera représenté par un "record" contenant un entier et une chaîne de caractères :

```
TypErrorCode = record
                   num : integer;
                   mess : string;
               end;
```

La valeur caractéristique d'une variable concernée sera contenue dans un record ayant une partie variable en fonction du type de cette valeur :

```
Types = (entier, car, booléen);

TypValCar = record
              case t : Types of
                         entier : (intval:integer);
                         car : (charval:string);
                         booléen : (boolval:boolean);
              end;
              nom : string;
            end;
```

* On déclare une procédure que l'on nomme "Empiler", ayant pour but d'ajouter la procédure de nom NOM, appelée à l'endroit NUM au sommet de la pile PILE

```
Procedure Empiler (var pile :  TypPile;
                           nom :  string;
                           num : integer);


  var Proc : TypProc;


  begin
       new (proc);
       proc := nil;
       Proc^.nomproc := nom;
       Proc^.nappel := num;
       Proc^.suiv := pile;
       pile := proc;
       dispose (proc);
  end;
```

* On déclare une procédure que l'on nomme "Vider", ayant pour but de vider la pile

```
Procedure Vider (var Pile : TypPile);


  var ec : TypPile;

  begin
       new (ec);
       ec := nil;
       while pile <> nil do
            begin
                 ec := pile;
                 pile := pile^.suiv;
                 ec^.suiv := nil;
                 dispose (ec);
            end;
       dispose (pile);
  end;
```

Comme chaque procédure du programme a la forme suivante :

```
Procedure Name (.........)

     begin
          ...

          xxxxx {instruction  susceptible de provoquer
                  une erreur}
          if erreur
          then begin
                 ErrorCode.num  := xxx;
                 ErrorCode.mess := xxx;
                    {positionner le code d'erreur}
                 Valcar.nom := xxx;
                    {et la valeur caractéristique}
                 fin procedure;
               end;

          ...

          B ( ... ) {appel à une procédure}
          if ErrorCode <> 0
          then begin
                    Empiler (pile , Name, x);
                    fin procedure;
               end;

          ...


end;
```

en cas d'erreur, on remonte de proche en proche jusqu'au programme principal, il ne reste plus qu'à afficher à l'écran le message correspondant à l'erreur, le nom et la valeur de la variable caractéristique et le contenu de la pile et sans oublier de vider la pile. Le programmeur, connaissant la nature de l'erreur, la valeur d'une variable en rapport direct avec l'erreur et la succession des procédures, pourra rapidement trouver la cause de l'erreur et la corriger.

3°)Conclusion

Cette solution est simple et systématique à implémenter, quoique un peu lourde : en effet, chaque appel à une procédure doit être suivi du test sur erreur; il faut aussi repérer toutes les instructions susceptibles de provoquer une erreur et les faire suivre par un test d'erreur.

De plus, si le programmeur utilise cette méthode au cours de la mise au point du programme, il devra au minimum, une fois le programme devenu opérationnel, enlever les parties de code relatives à l'affichage du contenu de la pile, puisque ces informations ne sont pas intéressantes pour l'utilisateur. Ou bien, on peut imaginer un logiciel ayant deux modes d'exécution : un mode "test", qui exécute les parties de code relatives à la gestion d'erreurs de programmation, et un mode opérationnel, qui n'exécute pas ces parties. On pourrait automatiquement passer d'un mode à l'autre.

Quant aux parties de code relatives à la gestion de la classe d'erreur, de la valeur caractéristique et de la pile des appels de procédures, il peut paraître intéressant de les retirer du code final si l'on vise une exécution la plus performante et la plus rapide possible.
Cependant, leur suppression dans le code final prendra du temps; et, si, par la suite, une correction est nécessaire après la mise en service du logiciel, il faudrait à nouveau réinsérer ces parties de code, ce qui peut être une tâche tout aussi fastidieuse. De plus, ajouter à nouveau ce code modifiera le programme.

Un grand avantage de cette solution est, néanmoins, le fait qu'elle est simple et systématique et qu'elle n'exige aucun outil matériel ou logiciel en plus de ceux déjà utilisés pour l'implémentation d'un simple programme, et ce quel que soit le langage de programmation utilisé.

B.Solutions existantes

1°) offertes par certains langages

Si les langages comme Pascal offrent peu de facilités en matière de gestion des cas d'exception, il existe au contraire d'autres langages qui implémentent de façon standard des techniques d'aide à la détection des cas d'exception; nous parlerons en particulier du langage PL/1 et du langage ADA . Nous allons voir, dans la suite, comment ces deux langages permettent de réaliser un système de gestion des erreurs.
Dans l'absolu, les facilités offertes par ces langages en matière de gestion d'erreurs peuvent être utilisées aussi bien pour concevoir un système de gestion des erreurs du programmeur ou de l'utilisateur. Cependant, nous verrons, après avoir décrit brièvement les techniques employées, qu'elles sont plus adaptées, en pratique, à la réalisation d'un système de gestion des erreurs du programmeur. C'est pour cette raison que nous mentionnons ces langages dans la partie de chapitre consacrée à la gestion des erreurs du programmeur.

a) principe général : objectif poursuivi par ces langages en matière de gestion d'erreurs

Dans un langage de programmation, il est utile de pouvoir remplacer n'importe quelle procédure fournie par le système par une autre définie par le programmeur. Ceci peut être utile, par exemple, lorsqu'il faut réagir à des erreurs ou à des conditions d'exceptions durant l'exécution d'une procédure, ou encore pour répondre de façon adéquate à des fautes de l'opérateur.

En effet, pendant l'exécution du programme, bien souvent, les erreurs provoquent une interruption qui clôture la session interactive et replace l'utilisateur au niveau du système ou, même, ce qui est plus grave, provoque un blocage complet du système.
Si on parvient à remplacer la procédure fournie par le système par une autre plus appropriée, on pourra éviter ce désagrément.

Les mécanismes de traitement d'exceptions rendent possible la gestion de ces événements et permettent de réaliser des actions qui empêchent des situations dangereuses ou irréversibles pour l'utilisateur.

En bref, grâce à ces mécanismes, il devient possible d'écrire des applications extrêmement robustes, capables de faire face à presque toutes les situations d'erreurs humaines ou de programmation.

b) le langage PL/1

Pour le langage PL/1, une "condition exceptionnelle" est définie comme étant une condition qui lorsqu'elle est rencontrée provoque une interruption de l'exécution du logiciel. L'interruption est détectée par le système qui effectue alors le traitement approprié.
Ces conditions exceptionnelles ont reçu différents noms dans le langage PL/1 et le programmeur a la possibilité, d'une part, d'autoriser ou d'interdire la détection d'une de ces conditions, et, d'autre part, de définir un traitement particulier associé à cette condition. Mais ces conditions sont très spécifiques, comme on le verra par la suite.

* Détection d'une condition exceptionnelle

Chaque condition peut être mise en fonction ou hors fonction. Une interruption ne peut être produite par une condition que lorsque celle-ci est en fonction. Si la condition est hors fonction, aucune interruption n'apparaît.
La portée d'une condition est définie de façon statique, à la compilation, sur toutes les instructions entre la mise en fonction et la mise hors fonction.

Il existe deux types de conditions exceptionnelles :

- d'abord, les "conditions de calcul" qui sont Underflow, Zerodivide, Fixedoverflow, Conversion et Size;

- ensuite, les "conditions de mise au point" qui sont Stringrange, Subscriptrange et Check.

* traitement après détection

A chaque condition exceptionnelle est affectée une action par défaut que le programmeur peut redéfinir grâce à une instruction particulière (ON) contenant une suite d'instructions.

L'association entre une condition et un traitement n'est déterminée qu'à l'exécution. La portée est dynamique et l'association peut ainsi être changée temporairement dans les blocs activés, puis restaurée au retour dans le bloc appelant.

Après l'exécution du traitement de la condition exceptionnelle, le programme reprend juste après l'instruction "ON" qui a provoqué l'exception.

Il existe également une instruction (instruction de simulation) qui permet au programmeur de forcer l'apparition d'une interruption. L'action associée à cette condition est exécutée. Ceci permet en particulier de tester le traitement associé à la condition.

* les outils de mise au point

Le programmeur peut également définir un nouveau nom de condition exceptionnelle. Cette définition se fait en spécifiant le traitement associé. Toutefois, ce traitement ne peut être activé que par une instruction de simulation.

Les conditions de mise au point les plus intéressantes sont les suivantes : CHECK et SUBSCRIPTRANGE.

Le traitement associé à la condition CHECK est d'afficher à l'écran la liste des identificateurs avec leur nouvelle valeur, si ce sont des variables.

La condition SUBSCRIPTRANGE vérifie que la valeur des indices utilisés dans une référence à un nom indicé soit comprise entre les bornes déclarées pour cette dimension.

* fonctions incorporées spéciales

Avec ce langage, il est impossible de transmettre directement, à un traitement associé à une condition, une quelconque information sur la cause précise de l'interruption.
Des fonctions spéciales sont fournies au programmeur afin qu'il puisse déterminer avec plus de précision la cause de l'erreur et qu'il puisse agir plus efficacement.

c) le langage ADA

Pour le langage ADA, une "exception" est un événement qui nécessite l'arrêt du traitement normal d'une unité de programme et implique un traitement exceptionnel particulier.
Une exception peut être positionnée, ou levée, par le programme à l'aide d'une instruction spécifique.
Il existe un certain nombre d'exceptions standard prédéfinies dans le langage et d'autres qui peuvent être utilisées à condition d'être déclarées explicitement par le programmeur.

* les déclarations d'exceptions

Les exceptions non standard doivent être déclarées dans le programme avant de pouvoir y faire référence. La déclaration est établie de façon statique.

Les exceptions correspondant à des erreurs prédéfinies sont: Constraint-Error (erreur de contrainte), Numeric-Error (erreur de calcul), Select-Error (erreur dans une instruction de sélection), Storage-Error (erreur d'allocation de place réservée) et Tasking-Error (erreur de communication entre les tâches).

* les traitements d'exceptions

Les traitements d'exception doivent être effectués dès qu'une exception est levée dans une unité de programme. Ils sont spécifiés à la fin du corps d'une unité de programme, sous la forme d'une structure similaire à une sélection multiple où chaque choix correspond à chacune des exceptions.

* les instructions de positionnement des erreurs

Pour pouvoir effectuer le traitement associé à une exception, il faut lever cette exception de façon explicite par une instruction spéciale.
Quand une instruction est levée, le cours normal d'exécution du programme est interrompu et il y a branchement vers le traitement d'exception correspondant.

* la propagation des exceptions

Normalement, le positionnement d'une exception au cours de l'exécution d'une unité de programme provoque l'arrêt du déroulement normal de cette unité et le branchement vers un traitement d'exception associé. Ceci est possible si ce traitement a été prévu dans le corps de l'unité et, dans ce cas, le traitement d'exception termine l'exécution de cette unité. Il existe cependant des cas où le traitement d'exception ne se trouve pas dans l'unité elle-même parce qu'on désire la traiter à un autre niveau.
Dans le cas de sous-programmes, toute exception provoque l'abandon immédiat de l'exécution de l'unité dans laquelle elle apparaît ou est propagée. Ceci est le cas, par exemple, des exceptions prédéfinies qui peuvent remonter jusqu'au niveau où l'on juge bon de les traiter.

d) Conclusion et comparaison

Le langage PL/1 propose des outils de traitement des conditions exceptionnelles qui permettent de détecter l'apparition de ces conditions et d'exécuter une action associée.

Le programmeur peut définir d'autres actions que les actions standard; mais si la définition de nouvelles conditions est possible, elle n'est que peu utile car une condition définie par le programmeur n'est détectée et traitée que lorsqu'on force l'apparition de la condition, par une instruction de simulation.

Les outils permettant de traiter les conditions exceptionnelles proposés par PL/1 sont orientés vers l'aide au test et à la mise au point de programme; ou bien, ce sont des conditions utilisables dans des programmes où les calculs sont nombreux.
L'application de ces outils pour concevoir des systèmes de gestion des erreurs de l'utilisateur est peu intéressante.

Les outils de gestion d'erreurs proposés par ADA correspondent beaucoup mieux à ce qui a été décrit auparavant, en terme de remontée et d'interprétation des codes d'erreurs.

Par comparaison avec ce qui est proposé dans le langage PL/1, ce type de traitement des exceptions s'adapte mieux à la conception d'un système de gestion des erreurs de l'utilisateur.

2°) offertes par certains logiciels

Actuellement, il existe sur le marché de plus en plus d'outils logiciels ayant pour but l'aide à la mise au point de programmes : ces outils sont par exemple, des éditeurs syntaxiques, des conducteurs de tests, des vérificateurs de couverture... qui offrent une aide considérable pour la vérification d'un programme.

Il existe aussi des environnements d'aide au développement logiciel contenant entre autres de puissants outils d'aide au "debugging" dont le rôle est l'aide à la localisation et à la correction d'une erreur de programmation. Ces outils réalisent automatiquement les mécanismes décrits ci-dessus (traçage de l'erreur, surveillance de la valeur d'une variable caractéristique...).
Certains de ces outils sont parfois directement fournis avec les compilateurs, c'est dire l'importance qu'on leur accorde.

Les possibilités généralement offertes par ces logiciels sont :

- l'exécution pas à pas d'un programme;
- la surveillance des valeurs de certaines variables;
- le traçage de l'erreur, donné par la suite des appels de procédures.
- ...

L'avantage de ces outils est le fait qu'ils ne modifient en rien le code du programme sur lequel ils travaillent et qu'ils peuvent s'appliquer à tout moment, que ce soit en phase de mise au point du programme ou plus tard, lors d'une modification.

Un inconvénient non négligeable est le fait qu'il faut encore que de tels outils soient à disposition, ce qui n'est pas toujours le cas.
En outre, ils modifient le déroulement du programme pour passer la main à l'outil à chaque instruction (parfois, à certaines seulement), ce qui l'empêche de s'appliquer à des applications dites en temps réel.

Cependant, face à des outils si puissants et relativement simples à utiliser, notre solution envisageable semble bien piètre.
Elle présente cependant l'avantage de n'exiger aucun outil supplémentaire par rapport à l'implémentation d'un programme normal et de ne pas en modifier la vitesse d'exécution.

Mais comme nous le verrons par la suite, si les progrès sont énormes dans le domaine des outils d'aide à la gestion des erreurs du programmeur, il n'en est pas de même en ce qui concerne les outils d'aide à la gestion des erreurs de l'utilisateur.

2.5 SYSTEME DE GESTION DES ERREURS DE L'UTILISATEUR

A.Solution envisageable et réalisable

1°) Principe général

Lorsque l'utilisateur a commis une erreur, il souhaite connaître la nature de cette erreur de façon significative et compréhensible pour lui. Peu lui importe quelle est la procédure qui a détecté l'erreur et toutes les procédures au travers desquelles l'erreur a dû remonter.

Nous avons également vu que lorsqu'une procédure détecte qu'une erreur a eu lieu, deux solutions se présentent à elle :

soit, la procédure est capable de corriger l'erreur;

soit, la procédure n'est pas capable de corriger l'erreur, par exemple, parce qu'il lui manque des éléments pour le faire : elle doit alors faire remonter et interpréter, documenter, l'erreur au travers de la suite des appels de procédures pour que cette erreur soit significative par rapport aux classes d'erreurs associées à ces procédures.

Une erreur sera représentée par

- un numéro de classe d'erreur, auquel est associé un message, significatif au niveau de la procédure courante;
- une(des) valeur(s) caractéristique(s) en rapport direct avec l'erreur survenue.

On remonte ainsi les indications d'erreurs jusqu'à une procédure capable de supprimer l'erreur ou jusqu'à l'utilisateur qui doit être informé le plus clairement possible de son erreur.

Le principe général est alors le suivant :

> Chaque appel à une procédure ou chaque instruction susceptible de provoquer une erreur est suivi d'un test qui vérifie si aucune erreur n'a eu lieu durant l'exécution de la procédure appelée ou durant l'exécution de l'instruction. Si aucune erreur n'a eu lieu, l'exécution du programme poursuit son cours normal.
> Si une erreur est survenue, on interprète la classe de l'erreur pour qu'elle soit significative à ce niveau (c'est-à-dire qu'on traduit la valeur de la classe d'erreur renvoyée par la procédure appelée en une autre valeur significative pour la procédure appelante). Puis, en fonction de la valeur de la classe d'erreur, soit la procédure exécute un traitement spécifique afin de corriger l'erreur et continuer son exécution normale, ou, soit, la procédure suspend son exécution et force un retour via chacune des procédures appelantes jusqu'à atteindre une procédure capable de corriger l'erreur ou jusqu'au programme principal qui envoie alors à l'utilisateur un message relatif à son erreur.

Cette solution est systématique à appliquer et indépendante du langage utilisé. Il faut cependant apporter beaucoup de soin à la définition des classes d'erreurs associées à chacune des procédures du programme. Il faut aussi veiller à programmer correctement l'interprétation des classes d'erreurs dans chacune des procédures.

2°) Stratégies de test

La question qui se pose au programmeur est la suivante :
à quel endroit d'un programme doit-on faire le test d'erreur?

Il existe deux stratégies, ayant chacune leurs avantages et inconvénients :

* faire le test le plus près possible du traitement

c'est-à-dire on saisit les données, mais on ne vérifie leur validité que lorsqu'on utilise ou que l'on traite ces données.

Cette stratégie est tout à fait correcte du point de vue de la logique de conception du programme : même si, lors de la conception de la procédure qui saisit les données, on connaît les traitements que l'on effectuera par la suite sur ces données (et à fortiori, les contraintes relatives à la validité de ces données imposées par les traitements), ce n'est pas à cette procédure de prendre en charge les vérifications des contraintes imposées par ces traitements.

Cependant, sur le plan de l'efficacité du programme et de sa convivialité, cette solution est moins optimale : en effet, puisqu'on ne vérifie la validité des données que lorsqu'on veut effectuer un traitement sur ces données, on risque de répéter cette vérification plusieurs fois inutilement, avec des conséquences sur la vitesse d'exécution du programme.
De plus, ce n'est qu'au moment du traitement que l'on peut prévenir l'utilisateur qu'il a commis une erreur. Or, comme ce traitement peut avoir lieu relativement longtemps après la saisie, l'utilisateur sera surpris de recevoir un message d'erreur concernant cette donnée invalide après un si long moment.

* faire le test le plus près possible de la saisie

c'est-à-dire on saisit les données et on en vérifie immédiatement la validité en anticipant les contraintes relatives à la validité des données, imposées par des traitements ultérieurs.

Cette stratégie est, elle, adéquate du point de vue de l'efficacité et de la convivialité du programme : en effet, la vérification est faite une seule fois, lors de la saisie, et l'utilisateur est prévenu immédiatement de ses erreurs éventuelles.

Cependant, sur le plan de la logique du programme, cette solution est moins bonne, puisqu'il faut connaître, dans la procédure de saisie, tous les traitements que l'on effectuera par la suite, pour en vérifier les préconditions.

* **stratégie mixte**

Après avoir détaillé ces deux stratégies, il est bon de préciser ce que l'on entend par le mot "près de" : en effet, lorsque l'on dit que de faire les tests **près de** la saisie, il s'agit en fait d'une proximité temporelle : il faut exécuter les tests de validité des données le plus rapidement possible après la saisie de ces données.
Et, lorsqu'on dit que l'on veut tester "le plus **près** possible du traitement", on entend par là une proximité textuelle : le texte de la procédure de vérification doit se trouver juste à proximité du texte de la procédure de traitement.

On peut alors imaginer un système où on déclarerait les procédures de vérification dans un bloc textuellement proche du traitement, mais, on exécuterait ces procédures de vérification juste après avoir réalisé la saisie, en anticipant les traitements à effectuer par la suite.

Cette solution semble, en théorie, la plus optimale car elle rassemble les avantages des deux stratégies précédentes sans en présenter les inconvénients.

Cependant, en pratique, il existe très peu de langages permettant l'implémentation de cette solution.

* **solution réalisable en pratique**

En pratique, il est des cas particuliers où on peut adopter une des deux premières stratégies, sans devoir en supporter les inconvénients :

En effet, certains programmes (en informatique de gestion, entre autres) réalisent surtout des transferts d'informations, en effectuant pas ou peu de traitements sur ces informations. Dans ce cas, il est inutile de vouloir rapprocher le test du traitement, étant donné que le traitement ne s'exécute que si la nature de l'information est respectée. On peut alors sans problème rapprocher le test et la saisie; ce test aura pour but de vérifier que la nature de l'information soit respectée.

D'autres programmes (programmation scientifique) réalisent beaucoup de manipulations d'informations à partir d'une seule procédure de saisie. Dans ce cas, il est impossible de vérifier la validité des données par rapport aux contraintes imposées par les traitements qui peuvent éventuellement s'effectuer par la suite. Ici, la stratégie qui consiste à faire les vérifications près du traitement s'impose sans discussion.

On constate donc que dans certains cas bien précis, une des deux stratégies s'impose sans discussion.
Face à une application particulière, on se posera d'abord la question de savoir s'il il y a réellement un choix entre une des deux stratégies.
Si il y a un choix à faire, on tentera de réduire le plus possible les inconvénients liés à la stratégie retenue.

3°) Stratégies de traitement des erreurs

La question qui se pose ici est :

à quel endroit du programme doit-on faire le traitement de correction de l'erreur ?

Il existe également plusieurs stratégies :

* traitement centralisé

Toute erreur détectée dans une procédure quelconque devra être rapportée au programme principal. Celui-ci réalise la gestion des exceptions soit par lui-même, soit par l'appel à des procédures spécialisées, mais tout en restant sous le contrôle du programme principal.

* traitement décentralisé

Une erreur détectée ne doit pas forcément retourner au programme principal; elle peut être traitée soit localement, soit par une procédure de niveau intermédiaire, sans que le programme principal n'intervienne.
Cette gestion décentralisée n'exclut pas un relevé des erreurs, utilisable par le programme principal, par exemple, en vue de statistiques ou de maintenance.

* traitement mixte

Le traitement se fera d'abord de façon décentralisée avec un temps ou un nombre limité d'essais de correction, et ensuite, de façon centralisée, en cas de dépassement des limites fixées.

* traitement par couches

Le traitement des erreurs peut aussi se faire par couche, si le programme concerné est ainsi structuré : chaque couche a des compétences bien définies, et, pour une couche particulière, on suppose que les couches inférieures de la hiérarchie ont bien effectué leurs tâches.
Dans la description des compétences d'une couche, on inclura également une énumération des cas d'exceptions que la couche est apte à traiter elle-même.

Il faut donc choisir la stratégie de traitement des erreurs en fonction de la structure du programme et de la hiérarchie de ses diverses parties.

4°) Application en Pascal

En Pascal, on peut représenter la classe de l'erreur par un entier contenant le numéro de celle-ci. Il est inutile d'ajouter, à ce numéro, le message correspondant à la classe d'erreur pour chaque procédure où l'on remonte; en effet, il est seulement nécessaire de connaître ce message au niveau de la procédure qui signale à l'utilisateur son erreur et affiche ce message.

```
TypErrorCode = record
                    num : integer;
                    mess : string;
               end;
```

La valeur caractéristique d'une variable en rapport avec l'erreur sera contenue dans un record ayant une partie variable en fonction du type de cette valeur :

```
Types = (entier, car, booléen);

TypValCar = record
              case t : Types of
                        entier : (intval:integer);
                        car : (charval:string);
                        booléen : (boolval:boolean);
              end;
              name : string;
            end;
```

Chaque procédure aura alors la forme suivante :

```
Procedure Name (.........)


begin
     ...

     xxxxx { instruction  susceptible de provoquer
             une erreur }
     if erreur
     then begin
            ErrorCode.num  := xxx;
            ErrorCode.mess := xxx;
               {positionner le code d'erreur}
            Valcar.nom := xxx;
               {et la valeur caractéristique}
            fin procedure;
           end;

     ...

     B ( ... ) { appel à une procédure }
     if ErrorCode <> 0
     then begin
               case errcode of
                         x : begin {interprétation}
                                   errcode := a;
                                   fin procedure;
                             end;
                         y : begin {correction}
                                   errcode := b;
                                   traitererreur;
                             end;
               end;
          end;
     ...


end;
```

La procédure "TraiterErreur" est une partie de code spécifique à la procédure appelante qui traitera l'erreur en fonction du numéro de la classe de l'erreur et d'une valeur caractéristique en rapport avec l'erreur.

5°) Conclusion

Cette solution présente l'avantage de pouvoir se réaliser en Pascal, comme dans tout autre langage, sans demander d'outils supplémentaires.

Elle présente cependant l'inconvénient suivant : il est assez lourd de faire suivre chaque appel de procédure d'une instruction de sélection multiple ("case") portant sur le numéro de la classe d'erreur.

Mais, comme nous allons le voir par la suite, étant donné qu'il n'existe aucune solution toute faite, cette solution est la seule utilisable pour obtenir une application contenant une gestion d'erreurs systématique.
Néanmoins, nous parlerons par après d'une solution qui serait souhaitable, quoique non réalisable dans le cadre d'un mémoire.

B.Solution existante.

Si l'aide à la détection des erreurs du programmeur a beaucoup progressé et a donné naissance à un grand nombre d'outils puissants, il n'en est pas de même en ce qui concerne la détection et la gestion des erreurs de l'utilisateur : il n'existe aucun outil mettant en application l'idée que le programme pourrait gérer des erreurs dues à l'inadéquation des données, fournies par l'utilisateur, par rapport aux contraintes imposées par les procédures du programme.

Pourquoi cette différence, alors qu'il s'agit d'erreurs dans les deux cas?

Le fait que l'on soit très peu avancé en matière de gestion des erreurs de l'utilisateur est dû à un problème déjà évoqué plus haut : le bouleversement de l'écriture des procédures de vérification par rapport à leur exécution. Comme on l'a vu auparavant, il faudrait un système qui permette d'exécuter, au moment de la saisie, des parties de code décrites dans un bloc situé près des traitements que l'on effectue sur ces données.
D'où la difficulté d'implémenter un tel système et par conséquent le vide en ce qui concerne l'existence d'outils analogues aux outils d'aide au "debugging".

C.Solution souhaitable

Dans un environnement Pascal (entre autres), implémenter un système de gestion d'erreurs tel qu'on l'a décrit représente une tâche assez lourde, car il faut faire suivre chaque appel de procédure d'un "case", qui, en fonction de la classe de l'erreur, effectuera soit une correction, soit une interprétation de l'erreur.

Ce que l'on souhaiterait, idéalement, ce serait pouvoir décrire de façon tout à fait séparée, d'une part, l'algorithme qui réalise les traitements normaux, c'est-à-dire, lorsqu'il n'y a pas d'erreur, et d'autre part, les procédures d'exception, c'est-à-dire les algorithmes qui décrivent les opérations à exécuter dans le cas où tel appel de procédure a donné tel code d'erreur.

Ayant ces deux parties séparées, il faudrait alors un mécanisme systématique qui construirait le test adéquat après chaque appel de procédure : on pourrait utiliser un outil, un précompilateur, par exemple, qui aurait pour fonction de transformer un programme qui décrit les cas normaux et une description des traitements d'erreurs en un seul programme contenant une système complet de gestion des erreurs de l'utilisateur tel qu'on l'a décrit auparavant.

Cependant, si un tel système reste faisable, sa réalisation sort du cadre de ce mémoire; nous nous contenterons donc d'en rester à l'ébauche de solution décrite ci-dessus.
Nous resterons ainsi confrontés aux limitations imposées par le langage Pascal, et nous implémenterons dans l'application, la solution, lourde, nous en conviendrons, de faire suivre tout appel de procédure d'un "case".

2.6 CONCLUSION :

Lorsque l'on parle d'un système de gestion d'erreurs, il convient tout d'abord de préciser la manière dont on se situe par rapport au programme : soit, le programme est vu comme une "Black Box", à laquelle on fournit des données et qui renvoit des résultats; soit, le même programme est vu comme une "White Box" qui exécute une suite d'opérations.

Une erreur dans un programme que l'on voit de l'extérieur est décrite par sa classe (c'est-à-dire la circonstance particulière non respectée par les données) et par la valeur d'une ou plusieurs variables en relation directe avec l'erreur.

Une erreur dans un programme vu de l'intérieur est gérée par le biais de deux mécanismes : la remontée de l'erreur, au cours de laquelle l'erreur peut être corrigée ou interprétée, et le traçage de l'erreur qui décrit le chemin d'exécution suivi.

Mais, c'est le type d'utilisateur qui détermine la façon dont on se situe par rapport à un programme : l'utilisateur final et l'opérateur voient le programme de l'extérieur et n'ont que faire du chemin d'exécution suivi. Par contre, puisqu'il faut que l'erreur ait un sens pour eux, il faut qu'elle soit remontée jusqu'à eux en transportant son interprétation successive par rapport aux procédures qu'elle traverse.
Le programmeur, lui, voit le programme de l'intérieur, et le traçage de l'erreur est pour lui de première importance; l'interprétation de l'erreur peut néanmoins lui apporter des informations intéressantes.

C'est en raison de ces divergences de vues que nous avons considéré séparément les deux système de gestion d'erreurs : un système de gestion des erreurs du programmeur, utilisé lors de la mise au point ou de la modification du programme, et un système de gestion des erreurs de l'utilisateur, employé dès que le logiciel est opérationnel.

En ce qui concerne le système de gestion des erreurs du programmeur, l'étude des solutions logicielles existantes et la description d'une solution envisageable nous a permis de conclure que si nous avions à notre disposition un outil logiciel réalisant ce système de gestion d'erreurs, il était plus intéressant de se servir de cet outil.
Néanmoins, comme de tels outils ne sont pas toujours à disposition, nous avons décrit une solution de remplacement applicable dans toutes les situations.

Par contre, du côté du système de gestion des erreurs de l'utilisateur, peu de résultats ont été atteints, pour des raisons que nous avons expliquées en détails : c'est pourquoi nous nous sommes attachés à décrire avec précision une solution réalisable en Pascal. c'est d'ailleurs la solution que nous retiendrons pour l'application à réaliser dans le cadre de ce mémoire.
Nous avons également discuté des diverses stratégies relatives à l'emplacement du test d'erreurs et du traitement d'erreurs. Pour chacune des stratégies, nous avons donné les avantages et les inconvénients. Mais, en pratique, la stratégie à adopter est bien souvent imposée par le style ou la structure du programme.
Nous avons enfin ébauché les fonctionnalités souhaitées d'un outil qui réaliserait automatiquement un tel système de gestion d'erreurs.

Pour décrire ces techniques, nous avons adopté une approche des erreurs par les traitements : ce sont en effet les traitements qui imposent des contraintes de validité et qui constatent l'erreur.
Une autre approche tout aussi valable pourrait se baser sur la structure des données : dans ce cas, on décrit une erreur, non plus par le traitement qui la détecte, mais bien par rapport aux structures de données. Les conditions ne sont plus liées aux traitements mais nous avons alors des conditions du même type que les contraintes d'intégrité que l'on associe, par exemple, à une base de données.
Cette dernière approche, bien qu'elle soit possible, semble plus adéquate lorsque le programme lui-même est orienté par les structures, plutôt que par les traitements.

PARTIE II

APPLICATION DE LA METHODE A UN LOGICIEL DE TESTS DE LA MEMOIRE HUMAINE

Nous avons décrit dans la première partie ce que devrait être un système de gestion des erreurs de l'utilisateur. Nous en avons expliqué les concepts et les mécanismes principaux.
Dans cette seconde partie, nous nous efforcerons d'appliquer cette méthode à un logiciel particulier, le logiciel de tests sur la mémoire humaine cité dans l'introduction.

Avant de s'attacher à décrire en détails l'application des concepts théoriques à notre cas particulier, il faut tout d'abord apporter une réponse à certains problèmes généraux concernant entre autres ce que nous avions appelé les stratégies de test et les stratégies de traitement d'erreurs : étant donné le type de l'application, qui se base surtout sur des saisies plutôt que sur des traitements, nous adopterons la stratégie qui consiste à faire les tests le plus près possible de la saisie, même si cette attitude est un peu moins bonne en ce qui concerne la logique du programme.
D'autre part, le traitement des erreurs se fera de façon décentralisée : en effet, notre application ressemblera à une application typique du MacIntosh, guidée par les événements et la sélection par l'utilisateur des items d'une barre de menus unique. Il est donc préférable de ne pas alourdir la boucle principale de traitement des événements par des instructions supplémentaires de gestion d'erreurs.

Nous avons dit dans la première partie que le programme était vu par l'utilisateur comme une "Black Box". Pour être tout à fait précis, le programme serait perçu non comme une seule "Black Box" unique mais plutôt que comme une succession de plusieurs "Black Box", chacune d'entre elles représentant une unité de traitement particulière aux yeux de l'utilisateur externe.

Nous verrons tout d'abord, dans un premier chapitre, comment l'utilisateur voit le programme et sa gestion d'erreurs, c'est-à-dire que nous adopterons, dans un premier temps, un point de vue externe. Pour cela, nous décomposerons le programme en ses diverses parties externes et nous leur associerons les classes d'erreurs externes qui leur correspondent.

Ensuite, dans un second chapitre, nous nous situerons du point de vue interne, à la place du programmeur qui doit réaliser ce que l'utilisateur aimerait voir lorsqu'il commet une erreur.
Dans ce but, nous diviserons le programme en ses unités internes, c'est-à-dire que nous hiérarchiserons ce programme en ses diverses procédures et fonctions, et nous définirons également les classes d'erreurs internes associées.

CHAPITRE 1

POINT DE VUE DE L'UTILISATEUR

1.1 INTRODUCTION

L'objet de ce premier chapitre est de proposer une décomposition de l'application en ses diverses unités et sous-unités de traitement et de les décrire telles qu'elles sont perçues par l'utilisateur.

Ensuite, pour chacune des unités et sous-unités décrites, il faudra définir toutes les classes d'erreurs qui lui sont associées.

Pour chaque classe d'erreur, on donnera le numéro de l'erreur ainsi qu'une brève description, la cause de l'erreur, c'est-à-dire l'action de l'utilisateur qui a provoqué cette erreur, et enfin l'effet de l'erreur, c'est-à-dire ce qui apparaît à l'écran ou les actions de "correction" réalisées suite à cette erreur et perceptibles par l'utilisateur.

L'effet d'une erreur faisant référence à ce qui apparaît à l'utilisateur sur l'écran, nous renverrons le lecteur à la description complète du contenu de tous les écrans présentés tout au long de l'application. Cette description sera expliquée en détails dans l'annexe 3 de ce présent mémoire.

1.2 DECOUPE DE L'APPLICATION EN UNITES DE TRAITEMENT

Nous nous contenterons dans cette section de décrire, de la façon la plus brève possible, les unités de traitement les plus importantes, celles qui nous intéressent particulièrement en ce qui concerne les classes d'erreurs qui leur sont associées.

Pour une description détaillée, qui ne trouverait pas sa place dans cette partie principale, nous renverrons le lecteur à l'annexe 2, annexe qui a pour but de montrer quel est l'enchaînement de toutes les unités de traitement externes qui composent le logiciel à proprement parler.

Les titres utilisés pour chacun des paragraphes qui suivent sont les noms attribués aux différentes unités reprises dans l'annexe 2.

Intro Nom Pédagogue Existant

Cette unité a pour but de demander à l'utilisateur d'introduire le nom d'un pédagogue existant et de saisir ce nom, sans avoir accès à la liste des pédagogues existants. Ce nom sert en quelque sorte de mot clé pour accéder aux options "Intro" et "Analyse", et ce afin limiter aux seuls pédagogues reconnus le droit de créer de nouveaux tests, de nouveaux pédagogues, d'analyser les réponses des sujets testés et de supprimer ou des tests, ou des pédagogues ou encore des réponses de sujets testés.
Cette unité s'exécute au début des options "Intro" et "Analyse".

Intro Nom Test Inexistant

Cette unité a pour but de demander à l'utilisateur d'introduire un nom de test qui n'existe pas encore, de saisir ce nom ainsi que le type du test, tout en lui permettant de consulter la liste des noms de tests qui existe déjà.
Cette unité s'exécute lors de la création d'un nouveau test, c'est-à-dire au début de l'option "Intro", à la suite de l'introduction du nom de pédagogue existant.

Intro Texte de Travail

Cette unité a pour but de demander au pédagogue d'introduire un texte de travail et de le saisir. Ce texte, qui sera utilisé comme base du test, est composé d'une entête, facultative, suivie d'une suite non vide de paragraphes.
Cette unité s'exécute dans l'option "Intro", suite à l'introduction du nom de pédagogue et du nom du test.

Intro Texte Explicatif

Cette unité a pour but de demander au pédagogue d'introduire un texte explicatif et de le saisir. Ce texte, qui indiquera à l'utilisateur ce que l'on attend de lui, est composé d'une entête, facultative, suivie d'une suite non vide de paragraphes. Ce texte est facultatif et c'est au pédagogue de décider s'il souhaite ou non introduire ce texte.
Cette unité s'exécute lors de la création d'un nouveau test, c'est-à-dire à n'importe quel moment avant la fin de l'option "Intro", suite à l'introduction du nom de pédagogue et du nom du test.

Intro Texte Réponse

Cette unité a pour but de demander à l'utilisateur d'introduire un texte réponse et de le saisir. Ce texte qui peut, si l'utilisateur le souhaite, être associé à n'importe quel élément d'une réponse. Il est composé d'une entête facultative suivie d'un seul paragraphes non vide.
Cette unité s'exécute lors de la création d'une réponse. Cette réponse peut être fournie par un pédagogue (réponse type ou réponse exemple, dans l'option "Intro") ou par un sujet testé (dans l'option "Exécution"), suite à l'ajout d'un élément dans une réponse à un test.

Intro Entête

Cette unité a pour but de demander à l'utilisateur d'introduire une entête d'un texte de travail ou d'un texte explicatif et de saisir cette entête. Une entête est composée d'un titre, facultatif, suivi d'un contenu.
Elle s'exécute au début de l'introduction d'un texte de travail ou d'un texte explicatif.

Intro Paragraphe

Cette unité a pour but de demander à l'utilisateur d'introduire un paragraphe et de saisir ce paragraphe. Un paragraphe est composé d'un titre, facultatif, suivi d'un contenu.
Elle s'exécute lors de l'introduction d'un texte, après de l'entête; pour un texte de travail ou un texte explicatif, cette unité est répétée pour chaque paragraphe, jusqu'à ce que l'utilisateur introduise un paragraphe avec un titre et un contenu vides.

Intro Titre

Cette unité a pour but de demander à l'utilisateur d'introduire un titre d'entête ou de paragraphe d'un texte et de saisir ce titre.
Elle s'exécute au début de l'introduction d'une entête ou d'un paragraphe de texte.

Intro Contenu

Cette unité a pour but de demander et de saisir un contenu d'entête ou de paragraphe d'un texte.
Elle s'exécute lors de l'introduction d'une entête ou d'un paragraphe de texte, à la suite de l'introduction du titre.

Intro Nom Personne

Cette unité a pour but de demander à l'utilisateur d'introduire le nom d'un sujet testé existant ou inexistant, tout en permettant à l'utilisateur de consulter la liste des noms existants et d'y sélectionner un nom.
Cette unité s'exécute lorsqu'un sujet testé doit créer une réponse à un test, c'est-à-dire au début de l'option "Exécution".

Intro Nom Test Existant

Cette unité a pour but de demander à l'utilisateur de sélectionner un nom de test dans la liste des noms de tests existants qui sont affichés à l'écran.
Cette unité s'exécute lors de l'exécution d'un test par un sujet testé, au début de l'option "Exécution", suite à l'introduction du nom de sujet testé, ainsi que dans l'option "Analyse" pour consulter ou pour supprimer la réponse d'un sujet testé à un test ou pour supprimer un test donné.

Intro Nom Personne Existante

Cette unité a pour but de demander à l'utilisateur de sélectionner un nom de sujet testé dans la liste des noms de sujets testés existants qui sont affichés à l'écran. Ce nom est le nom du sujet testé dont le pédagogue veut consulter ou supprimer une réponse.
Cette unité s'exécute dans l'option "Analyse" pour consulter ou pour supprimer la réponse d'un sujet testé à un test.

Intro Nom Pédagogue Nouveau

Cette unité a pour but de demander à l'utilisateur d'introduire le nom d'un nouveau pédagogue et de saisir ce nom, tout en permettant à l'utilisateur de consulter la liste des noms de pédagogues existants qui sont affichés à l'écran. Ce nom est le nom d'un pédagogue que l'on veut ajouter à la liste des pédagogues existants.
Cette unité s'exécute dans l'option "Analyse", lors de la création d'un nouveau pédagogue.

Intro Matériaux Réseau

Cette unité a pour but de demander à l'utilisateur d'introduire un matériau qui n'existe pas encore et de saisir l'identifiant long, l'identifiant court et le type (concept ou liaison) du nouveau matériau, tout en permettant à l'utilisateur de consulter la liste des matériaux déjà créés et qui affichés à l'écran.
Cette unité s'exécute dans l'option "Intro", lors de la création d'un nouveau test de type réseau, après l'introduction du texte de travail.

Intro Matériaux Liaison

Cette unité a pour but de demander à l'utilisateur d'introduire un matériau qui n'existe pas encore et de saisir l'identifiant long et l'identifiant court du nouveau matériau, tout en permettant à l'utilisateur de consulter la liste des matériaux déjà créés et affichés à l'écran.
Cette unité s'exécute dans l'option "Intro", lors de la création d'un nouveau test de type liaison, suite à l'introduction du texte de travail.

Intro Réponse Exemple

Cette unité a pour but de demander à l'utilisateur d'introduire une réponse exemple à un test et de la saisir. Cette réponse est facultative et est proposée au sujet testé à titre d'exemple, avant qu'il ne réalise sa réponse.
Cette unité s'exécute lors de la création d'un nouveau test, à la suite de l'introduction des matériaux du test, à n'importe quel moment avant la fin de la création du test.

Intro Réponse Type

Cette unité a pour but de demander à l'utilisateur d'introduire une réponse type et de la saisir. Cette réponse est proposée au sujet testé à titre de comparaison, après qu'il ait réalisé sa réponse.
Cette unité s'exécute lors de la création d'un nouveau test, à la suite de l'introduction des matériaux du test.

Intro Réponse Personne

Cette unité a pour but de demander à un sujet testé d'introduire une réponse à un test et de la saisir.
Cette unité s'exécute lors de l'exécution d'un test par un sujet testé, à la suite de l'introduction du nom du sujet testé et du nom du test.

Ajout Concept

Cette unité a pour but de demander à l'utilisateur de sélectionner un concept dans la liste des concepts existants et affichés à l'écran.
Cette unité s'exécute lors de l'introduction d'une réponse à un test de type réseau.

Ajout Sommet Paragraphe

Cette unité a pour but de demander à l'utilisateur d'introduire un identifiant qui n'existe pas encore et son association à un paragraphe non encore identifié.
Cette unité s'exécute lors de l'introduction d'une réponse à un test de type liaison ou de type hiérarchisation de paragraphes.

Ajout Liaison

Cette unité a pour but de demander à l'utilisateur de sélectionner un nom de liaison, un concept (ou un sommet de paragraphe) origine et un concept (ou un sommet de paragraphe) cible dans la liste des concepts (ou des sommets de paragraphe) et dans la liste des liaisons existantes affichées à l'écran.
Cette unité s'exécute lors de l'introduction d'une réponse à un test de type réseau ou de type liaison.

Ajout LR

Cette unité a pour but de demander à l'utilisateur de sélectionner dans la liste des concepts existants ceux qui seront regroupés au sein d'une même liaison rectangle.
Cette unité s'exécute lors de l'introduction d'une réponse à un test de type réseau.

1.3 DESCRIPTION DES ECRANS

De façon générale, un écran présentant un message d'erreur est représenté par une "boîte d'alerte" contenant un message et deux boutons : un bouton "OK" et un bouton "Help".
Comme nous l'avons dit dans le premier chapitre de la première partie, ce logiciel doit être conçu pour être employé par des utilisateurs ayant des niveaux d'expérience différents. Il doit respecter les exigences des utilisateurs expérimentés, qui eux cherchent à tirer du logiciel le plus d'efficacité possible, tout comme celles des utilisateurs débutants qui veulent apprendre à se servir du logiciel.

Un logiciel à deux niveaux de messages peut être une solution qui réponde simultanément à ces deux exigences qui semblent, à première vue, contradictoires.

Ces exigences sont aussi valables pour les messages d'erreurs : si un utilisateur expérimenté préfère un message court qui ne le retarde pas tout en lui offrant un bref rappel, un débutant souhaite plutôt des messages qui lui expliquent de la façon la plus précise et la plus détaillée possible la cause de l'erreur.

Pour réaliser cette solution dans notre logiciel, nous avons travaillé de la manière suivante : en cas d'erreur, un premier message est affiché. Ce message est aussi court que possible, mais l'utilisateur peut s'il le souhaite obtenir des informations supplémentaires sur l'erreur qu'il a commise dans la situation précise où il se trouve. C'est grâce aux boutons "OK" et "Help" que l'utilisateur manifeste son choix : un clic sur "Help" signifie que l'utilisateur souhaite de plus amples informations concernant l'erreur commise; un clic sur "OK" signifie que le message suffit à l'utilisateur.

1.4 DEFINITION DES CLASSES D'ERREURS EXTERNES

Une classe d'erreur dite externe est une classe d'erreurs telle que perçue par l'utilisateur.

Dans les descriptions qui suivent, pour chaque unité, repérée par son titre, on donnera les différentes classes d'erreurs identifiées par le numéro de l'erreur suivi d'une brève description. On indiquera ensuite la cause de l'erreur, c'est-à-dire l'action de l'utilisateur qui a provoqué cette erreur, et enfin on expliquera l'effet de l'erreur, c'est-à-dire ce qui apparaît à l'écran ou les actions de "correction" réalisées suite à cette erreur et perceptibles par l'utilisateur externe.

Intro Nom Pédagogue Existant

-> *affiche l'écran n° 1*

* 0 OK

* 1 Annulation saisie d'un nom de pédagogue existant
cause : clic sur le bouton "Cancel" de l'écran n° 1
effet : restauration de la barre des menus dans son état initial

* 2 Nom de pédagogue trop long
cause : la longueur de la chaîne de caractères introduite est supérieure à 6 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-1 qui indique :
Le nom de pédagogue introduit est trop long

* 4 Nom de pédagogue trop court
cause : la longueur de la chaîne de caractères introduite est inférieure à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 5 Demande d'aide supplémentaire concernant l'erreur n° 4
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-1 qui indique :
Le nom de pédagogue introduit est trop court

* 6 Nom de pédagogue inexistant
cause : le nom de pédagogue introduit ne fait pas partie de la liste des pédagogues existants
effet : affichage de l'écran n° 4 qui indique :
Nom inexistant

* 7 Demande d'aide supplémentaire concernant l'erreur n° 6
cause : clic sur le bouton "Help" de l'écran n° 4
effet : affichage de l'écran n° 4-1 qui indique :
pour accéder à l'option "Intro" ou "Analyse', il faut donner un nom de pédagogue existant

Intro Nom Test Inexistant

-> *affiche l'écran n° 5*

* 0 OK

* 1 Annulation saisie d'un nom de test inexistant
cause : clic sur le bouton "Cancel" de l'écran n° 5
effet : restauration de la barre des menus dans son état initial

* 2 Nom de test trop long
cause : la longueur de la chaîne de caractères introduite est supérieure à 6 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-2 qui indique :
Le nom de test introduit est trop long

* 4 Nom de test trop court
cause : la longueur de la chaîne de caractères introduite est inférieure à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 5 Demande d'aide supplémentaire concernant l'erreur n° 4
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-2 qui indique :
Le nom de test introduit est trop court

* 6 Nom de test existant
cause : le nom de test introduit fait déjà partie de la liste des tests existants
effet : affichage de l'écran n° 7 qui indique :
Nom existant

* 7 Demande d'aide supplémentaire concernant l'erreur n° 6
cause : clic sur le bouton "Help" de l'écran n° 7
effet : affichage de l'écran n° 7-1 qui indique :
Pour créer un nouveau test, il faut donner un nom de test inexistant

Intro Texte de Travail

-> affiche la fenêtre texte

* 0 OK

* 1 Texte sans paragraphe
cause : tous les paragraphes introduits ont un contenu vide
effet : affichage de l'écran 15 qui indique :
Texte sans paragraphe

* 2 Demande d'aide supplémentaire concernant l'erreur n° 1
cause : clic sur le bouton "Help" de l'écran n° 15
effet : affichage de l'écran n° 16 qui indique :
Un texte de travail doit contenir au moins un paragraphe non vide

Intro Texte Explicatif

-> affiche la fenêtre texte

* 0 OK

* 1 Texte sans paragraphe
cause : tous les paragraphes introduits ont un contenu vide
effet : affichage de l'écran 17 qui indique :
Texte sans paragraphe

* 2 Demande d'aide supplémentaire concernant l'erreur n° 1
cause : clic sur le bouton "Help" de l'écran n° 17
effet : affichage de l'écran n° 18 qui indique :
Un texte explicatif doit contenir au moins paragraphe non vide

Intro Texte Réponse

-> affiche la fenêtre texte

* 0 OK

* 1 Texte sans paragraphe
cause : tous les paragraphes introduits ont un contenu vide
effet : affichage de l'écran 17 qui indique :
Texte sans paragraphe

* 2 Demande d'aide supplémentaire concernant l'erreur n° 1
cause : clic sur le bouton "Help" de l'écran n° 17
effet : affichage de l'écran n° 19 qui indique :
Un texte réponse doit contenir un paragraphe non vide

Intro Entête

* 0 OK

* 1 Entête avec titre mais sans contenu
cause : le titre introduit est une chaîne de caractères non vide, alors que le contenu est une chaîne de caractères vide
effet : affichage de l'écran 12 qui indique :
Entête avec titre mais sans contenu

* 2 Demande d'aide supplémentaire concernant l'erreur n° 1
cause : clic sur le bouton "Help" de l'écran n° 12
effet : affichage de l'écran n° 13 qui indique :
Une entête ou un paragraphe doit avoir un contenu non vide si son titre est non vide

Intro Paragraphe

* 0 OK

* 1 Paragraphe avec titre mais sans contenu
cause : le titre introduit est une chaîne de caractères non vide, alors que le contenu est une chaîne de caractères vide
effet : affichage de l'écran 14 qui indique :
Paragraphe avec titre mais sans contenu

* 2 Demande d'aide supplémentaire concernant l'erreur n° 1
cause : clic sur le bouton "Help" de l'écran n° 14
effet : affichage de l'écran n° 13 qui indique :
Une entête ou un paragraphe doit avoir un contenu non vide si son titre est non vide

Intro Titre

* 0 OK

* 1 Titre trop long
cause : la longueur de la chaîne de caractères introduit est supérieure à 20 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 2 Demande d'aide supplémentaire concernant l'erreur n° 1
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-6 (ou 3-7) qui indique :
Le titre de l'entête (ou du paragraphe) est trop long

Intro Nom Personne

-> *affiche l'écran n° 20*

* 0 OK

* 1 Annulation saisie d'un nom de personne
cause : clic sur le bouton "Cancel" de l'écran n° 21
effet : restauration de la barre des menus dans son état initial

* 2 Nom de personne trop long
cause : la longueur de la chaîne de caractères introduite est supérieure à 6 caractères
effet : affichage de l'écran n° 3 qui indique : Nom trop long

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-5 qui indique : Le nom de personne introduit est trop long

* 4 Nom de personne trop court
cause : la longueur de la chaîne de caractères introduite est inférieure à 1 caractère
effet : affichage de l'écran n° 2 qui indique : Nom trop court

* 5 Demande d'aide supplémentaire concernant l'erreur n° 4
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-5 qui indique : Le nom de personne introduit est trop court

Intro Nom Test Existant

-> *affiche l'écran n° 21*

* 0 OK

* 1 Annulation saisie d'un nom de test existant
cause : clic sur le bouton "Cancel" de l'écran n° 21
effet : restauration du logiciel dans son état précédent

Intro Nom Personne Existante

-> *affiche l'écran n° 22*

* 0 OK

* 1 Annulation saisie d'un nom de personne existante
cause : clic sur le bouton "Cancel" de l'écran n° 22
effet : restauration du logiciel dans son état précédent

Intro Nom Pédagogue Nouveau

-> *affiche l'écran n° 23*

* 0 OK

* 1 Annulation saisie d'un nom de pédagogue inexistant
cause : clic sur le bouton "Cancel" de l'écran n° 23
effet : restauration du logiciel dans son état précédent

* 2 Nom de pédagogue trop long
cause : la longueur de la chaîne de caractères introduite est supérieure à 6 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-1 qui indique :
Le nom de pédagogue introduit est trop long

* 4 Nom de pédagogue trop court
cause : la longueur de la chaîne de caractères introduite est inférieure à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 5 Demande d'aide supplémentaire concernant l'erreur n° 4
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-1 qui indique :
Le nom de pédagogue introduit est trop court

* 6 Nom de pédagogue existant
cause : le nom de pédagogue introduit fait déjà partie de la liste des pédagogues existants
effet : affichage de l'écran n° 7 qui indique :
Nom existant

* 7 Demande d'aide supplémentaire concernant l'erreur n° 6
cause : clic sur le bouton "Help" de l'écran n° 7
effet : affichage de l'écran n° 7-1 qui indique :
Pour créer un nouveau pédagogue, il faut donner un nom de pédagogue inexistant

-> affiche l'écran n° 9

* 0 OK

* 1 Annulation saisie d'un nom de matériau inexistant
cause : clic sur le bouton "Cancel" de l'écran n° 9
effet : arrêt de la saisie de la suite des matériaux associés au test

* 2 Identifiant court trop long
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant court est supérieure à 6 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-3 qui indique :
L'identifiant court introduit est trop long

* 4 Identifiant long trop long
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant long est supérieure à 20 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 5 Demande d'aide supplémentaire concernant l'erreur n° 4
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-4 qui indique :
L'identifiant long introduit est trop long

* 6 Identifiant court trop court
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant court est inférieur à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 7 Demande d'aide supplémentaire concernant l'erreur n° 6
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-3 qui indique :
L'identifiant court introduit est trop court

* 8 Identifiant long trop court
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant long est inférieur à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 9 Demande d'aide supplémentaire concernant l'erreur n° 8
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-4 qui indique :
L'identifiant long introduit est trop court

* 10 Identifiant court existant
cause : l'identifiant court introduit fait déjà partie de la liste des matériaux existants associés à ce test
effet : affichage de l'écran n° 7 qui indique :
Nom existant

* 11 Demande d'aide supplémentaire concernant l'erreur n° 10
cause : clic sur le bouton "Help" de l'écran n° 7
effet : affichage de l'écran n° 7-2 qui indique :
Pour créer un nouveau matériau, il faut donner un identifiant court inexistant

* 12 Identifiant long existant
cause : l'identifiant long introduit fait déjà partie de la liste des matériaux existants associés à ce test
effet : affichage de l'écran n° 7 qui indique :
Nom existant

* 13 Demande d'aide supplémentaire concernant l'erreur n° 12
cause : clic sur le bouton "Help" de l'écran n° 7
effet : affichage de l'écran n° 7-3 qui indique :
Pour créer un nouveau matériau, il faut donner un identifiant long inexistant

-> affiche l'écran n° 11

* 0 OK

* 1 Annulation saisie d'un nom de liaison inexistante
cause : clic sur le bouton "Cancel" de l'écran n° 11
effet : arrêt de la saisie de la suite des liaisons associées au test

* 2 Identifiant court trop long
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant court est supérieure à 6 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-3 qui indique :
L'identifiant court introduit est trop long

* 4 Identifiant long trop long
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant long est supérieure à 20 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 5 Demande d'aide supplémentaire concernant l'erreur n° 4
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-4 qui indique :
L'identifiant long introduit est trop long

* 6 Identifiant court trop court
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant court est inférieur à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 7 Demande d'aide supplémentaire concernant l'erreur n° 6
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-3 qui indique :
L'identifiant court introduit est trop court

* 8 Identifiant long trop court
cause : la longueur de la chaîne de caractères introduite représentant l'identifiant long est inférieur à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 9 Demande d'aide supplémentaire concernant l'erreur n° 8
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-4 qui indique :
L'identifiant long introduit est trop court

* 10 Identifiant court existant
cause : l'identifiant court introduit fait déjà partie de la liste des liaisons existantes associées à ce test
effet : affichage de l'écran n° 7 qui indique :
Nom existant

* 11 Demande d'aide supplémentaire concernant l'erreur n° 10
cause : clic sur le bouton "Help" de l'écran n° 7
effet : affichage de l'écran n° 7-2 qui indique :
Pour créer une nouvelle liaison, il faut donner un identifiant court inexistant

* 12 Identifiant long existant
cause : l'identifiant long introduit fait déjà partie de la liste des liaisons existantes associées à ce test
effet : affichage de l'écran n° 7 qui indique :
Nom existant

* 13 Demande d'aide supplémentaire concernant l'erreur n° 12
cause : clic sur le bouton "Help" de l'écran n° 7
effet : affichage de l'écran n° 7-3 qui indique :
Pour créer une nouvelle liaison, il faut donner un identifiant long inexistant

Intro Réponse Exemple, *Intro Réponse Type*, *Intro Réponse Personne*

-> *affiche la fenêtre réponse*

* 0 OK

* 1 Clôture saisie réponse
  cause : clic dans la "close box" de la fenêtre réponse
  effet : déclenche la vérification de la validité de la réponse (la réponse doit contenir tous les concepts ou sommets de paragraphe)

* 2 Un concept ou sommet de paragraphe est inutilisé
  cause : clôture de la réponse alors qu'il existe encore au moins un concept ou un sommet de paragraphe qui ne fait pas partie de la réponse
  effet : affichage de l'écran 29 qui indique :
  un concept ou sommet de paragraphe inutilisé

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
  cause : clic sur le bouton "Help" de l'écran n° 29
  effet : affichage de l'écran n°30-1 (ou 30-2) qui indique : Pour être complète, une réponse à un test de type réseau (ou de type liaison ou hiérarchisation) doit contenir tous les concepts (ou tous les sommets de paragraphes)

Ajout Concept

-> *affiche l'écran n° 33*

* 0 OK

* 1 Annulation ajout d'un concept à la réponse
  cause : clic sur le bouton "Cancel" de l'écran n° 33
  effet : restauration du logiciel dans son état précédent

Ajout Sommet Paragraphe

-> *affiche l'écran n° 36*

* 0 OK

* 1 Annulation ajout d'un sommet de paragraphe
cause : clic sur le bouton "Cancel" de l'écran n° 36
effet : restauration du logiciel dans son état précédent

* 2 Identifiant de paragraphe trop long
cause : la longueur de la chaîne de caractères introduite est supérieure à 6 caractères
effet : affichage de l'écran n° 3 qui indique :
Nom trop long

* 3 Demande d'aide supplémentaire concernant l'erreur n° 2
cause : clic sur le bouton "Help" de l'écran n° 3
effet : affichage de l'écran n° 3-8 qui indique :
L'identifiant de paragraphe introduit est trop long

* 4 Identifiant de paragraphe trop court
cause : la longueur de la chaîne de caractères introduite est inférieure à 1 caractère
effet : affichage de l'écran n° 2 qui indique :
Nom trop court

* 5 Demande d'aide supplémentaire concernant l'erreur n° 4
cause : clic sur le bouton "Help" de l'écran n° 2
effet : affichage de l'écran n° 2-6 qui indique :
L'identifiant de paragraphe introduit est trop court

* 6 Identifiant de paragraphe existant
cause : l'identifiant de paragraphe introduit fait déjà partie de la liste des identifiants associés à ce texte de travail
effet : affichage de l'écran n° 7 qui indique :
Nom existant

* 7 Demande d'aide supplémentaire concernant l'erreur n° 6
cause : clic sur le bouton "Help" de l'écran n° 7
effet : affichage de l'écran n° 7-6 qui indique :
Pour créer un nouvel identifiant de paragraphe, il faut donner un nom d'identifiant de paragraphe inexistant

Ajout Liaison

-> *affiche l'écran n° 34*

* 0 OK

* 1 Annulation ajout d'une liaison à la réponse
cause : clic sur le bouton "Cancel" de l'écran n° 34
effet : restauration du logiciel dans son état précédent

Ajout LR

-> *affiche l'écran n° 35*

* 0 OK

* 1 Annulation ajout de concepts à une liaison rectangle
cause : clic sur le bouton "Cancel" de l'écran n° 35
effet : ajout de la liaison rectangle avec tous les concepts qu'elle contient

CHAPITRE 2

POINT DE VUE DU CONCEPTEUR

2.1 INTRODUCTION

L'objet de ce second chapitre est d'analyser le problème de la gestion vu de l'intérieur du programme, c'est-à-dire vu par le concepteur, et de chercher à implémenter ce que l'on a définit au chapitre précédent en termes de souhaits de l'utilisateur en matière de gestion d'erreurs.

Nous avons, dans le premier chapitre, découpé l'application en fonction de critères externes, en adoptant le point de vue de l'utilisateur. Dans ce chapitre, nous allons voir les coulisses du programme, son implémentation interne, et donc, baser notre décomposition sur les différentes procédures et fonctions qui constituent ce programme.

Ensuite, tout comme nous l'avons fait dans le premier chapitre, nous définirons, pour chaque procédure et chaque fonction décrite, les classes d'erreurs qui lui sont associées. C'est sur base de cette définition que l'on pourra interpréter ces classes d'erreurs au cours de la remontée des erreurs, et aboutir en fin de remontée aux classes d'erreurs connues à l'extérieur et spécifiées précédemment.

2.2 DECOUPE DU PROGRAMME EN PROCEDURES

Dans cette section, nous nous préoccuperons uniquement de décrire, brièvement la fonction de chacune des procédures ou fonctions retenues, afin de permettre au lecteur de situer le but de ces procédures sans toutefois alourdir cette partie principale.
Le graphe des appels statiques figure à l'annexe 4.

EvtLoop

Cette procédure a pour fonction de gérer les différents événements qui peuvent survenir dans une application spécifique MacIntosh, c'est-à-dire qu'elle se charge d'appeler des procédures spécialisées capables de gérer ces événements.

ProcessInMenu

Cette procédure a pour fonction de gérer la sélection d'une option du menu.

DoGoAway

Cette procédure a pour fonction de gérer l'action de type "fermeture de fenêtre", réalisée par un clic souris dans la "close box" d'une fenêtre.

DealwthKeyDown

Cette procédure a pour fonction de gérer les événements de type frappe au clavier.

InNomPedaExist

Cette procédure a pour fonction de demander à l'utilisateur un nom de pédagogue existant, utilisé comme mot de passe (sans liste de référence).
Elle en vérifie également la validité syntaxique (longueur du nom) et sémantique (existence du nom dans la liste des pédagogues).

InNomTestInexist

Cette procédure a pour fonction de demander et de saisir un nouveau nom de test ainsi que son type. Elle en vérifie également la validité syntaxique (longueur du nom) et sémantique (inexistence du nom) et elle enregistre ce nouveau nom de test.

InNomTestExec

Cette procédure a pour fonction de demander à l'utilisateur de sélectionner un nom de test existant dans la liste qui lui est présentée, de saisir cette sélection et de vérifier si le sujet testé n'a pas déjà exécuté ce test ou n'a pas atteint le nombre maximum de test autorisés.

InNomST

Cette procédure a pour fonction de demander et de saisir un nom de sujet testé existant (sélection dans la liste proposée) ou de sujet testé inexistant (introduction au clavier).
Dans le cas d'un nouveau nom, elle en vérifie aussi la validité syntaxique (longueur du nom) et enregistre ce nouveau nom de sujet testé.

DoAddCon

Cette procédure a pour fonction de demander à l'utilisateur de sélectionner un concept dans la liste des concepts qui lui est proposée et de saisir cette sélection.
Elle enregistre également ce concept dans la réponse que l'utilisateur est occupé à créer et ajoute ce concept à la représentation graphique de la réponse qui se dessine à l'écran.

DoAddLiai

Cette procédure a pour fonction de demander à l'utilisateur de sélectionner un concept origine, un concept cible et une liaison choisie dans les listes de concepts et de liaisons qui lui sont proposées. La procédure saisit ces trois noms.
Elle enregistre également cette liaison dans la réponse que l'utilisateur est occupé à créer et ajoute cette liaison à la représentation graphique de la réponse qui se dessine à l'écran.

DoAddLR

Cette procédure a pour fonction de demander à l'utilisateur de sélectionner succesivement tous les concepts faisant partie d'une liaison rectangle, de saisir ces différents concepts, d'enregistrer cette liaison rectangle et d'ajouter cette liaison rectangle à la représentation graphique de la réponse qui se dessine à l'écran.

DoAddStPar

Cette procédure a pour fonction de demander et de saisir un identifiant associé à un paragraphe du texte de travail. Elle en vérifie la validité syntaxique (longueur de l'identifiant) et sémantique (inexistence), l'enregistre dans la réponse et ajoute ce sommet de paragraphe à la représentation graphique de la réponse qui se dessine à l'écran.

DoInTexte

Cette procédure a pour fonction de saisir un texte, de vérifier la validité de ce texte dans son ensemble tout comme de chacun de ses éléments pris séparément (titre ou contenu d'entête ou de paragraphe) et d'enregistrer ce texte.

Ecran

Cette procédure a pour fonction de répartir les appels aux écrans des différents types, tout en permettant l'uniformité des paramètres pour les appels. Cette procédure joue en quelque sorte le rôle d'interface entre les procédures appelées et appelantes.

De manière générale ,

EcranX

Cette procédure a pour fonction de répartir les appels entre les différents écrans de type X. Elle joue également le rôle d'interface entre les procédures appelées et les procédures appelantes.

EcranX-Y

Cette procédure a pour but d'afficher l'écran de numéro Y et de type X et d'y saisir les informations introduites par l'utilisateur.

2.3 DEFINITION DES CLASSES D'ERREURS INTERNES

Les classes d'erreurs dites internes sont celles considérées du point de vue du concepteur du logiciel.

Nous commencerons d'abord par définir les classes d'erreurs associées aux procédures de bas niveau, celles qui n'en appellent aucune autre et qui sont capables de détecter des erreurs ou des cas d'exceptions, mais qui sont incapables de les corriger.

Puis, nous poursuivrons par les procédures de niveau intermédiaire, celles qui interprètent les erreurs qui leur sont transmises par les procédures de bas niveau; ces procédures de niveau intermédiaire sont incapables de corriger les erreurs et doivent les faire remonter aux procédures de niveau supérieur.

Enfin, nous terminerons par les procédures de haut niveau, celles qui sont aptes à corriger une erreur reportée jusqu'à elles.

Pour chaque procédure, nous donnerons les classes d'erreurs associées, avec leur numéro et une brève description. Nous indiquerons également l'interprétation des codes d'erreurs remontés ou, le cas échéant, la correction appliquée à l'erreur.

Ecran2-1

0	OK
1	Clic sur le bouton "Cancel"
2	Longueur du nom introduit supérieure à 6 caractères
3	Longueur du nom introduit inférieure à 1 caractère

Ecran3-2-0, *Ecran3-3-0*, *Ecran3-4-0*, *Ecran3-7-0*

0	OK
1	Clic sur le bouton "Help"

Ecran3-12

1	Clic sur le bouton "Introduire Contenu"
2	Clic sur le bouton "Annuler Titre"
3	Clic sur le bouton "Help'

Ecran3-13

1	Clic sur le bouton "Introduire Contenu"
2	Clic sur le bouton "Annuler Titre"

Ecran3-14

1	Clic sur le bouton "Introduire Contenu"
2	Clic sur le bouton "Annuler Titre"
3	Clic sur le bouton "Help'

Ecran3-15

1	Clic sur le bouton "OK"
2	Clic sur le bouton "Help"

Ecran3-17

1	Clic sur le bouton "Introduire Paragraphe"
2	Clic sur le bouton "Annuler Texte"
3	Clic sur le bouton "Help"

Ecran3-18, *Ecran3-19*

1	Clic sur le bouton "Introduire Paragraphe"
2	Clic sur le bouton "Annuler Texte"

Ecran3-29, *Ecran3-31*

0	OK
1	Clic sur le bouton "Help"

Ecran6-21, *Ecran6-22*, *Ecran6-23*,
Ecran6-33, *Ecran6-34*, *Ecran6-35*

0	OK
1	Clic sur le bouton "Cancel"

Ecran7-5

0	OK
1	Clic sur le bouton "Cancel"
3	Longueur du nom introduit supérieure à 6 caractères
4	Longueur du nom introduit inférieure à 1 caractère

Ecran7-9, *Ecran7-11*

0	OK
1	Clic sur le bouton "Cancel"
3	Longueur du nom court introduit supérieure à 6 caractères
4	Longueur du nom long introduit supérieure à 20 caractères
5	Longueur du nom court introduit inférieure à 1 caractère
6	Longueur du nom long introduit inférieure à 1 caractère

Ecran7-20, *Ecran7-28*

0	OK
1	Clic sur le bouton "Cancel"
3	Longueur du nom introduit supérieure à 6 caractères
4	Longueur du nom introduit inférieure à 1 caractère

Ecran2

0	OK
11	Demande d'annulation dans l'écran n° 1
12	Longueur du nom introduit dans l'écran n° 1 supérieure à 6 caractères
13	Longueur du nom introduit dans l'écran n° 1 inférieure à 1 caractère

== > interprétation des codes d'erreur provenant de la procédure Ecran2-1

Ecran3

0	OK
21	Demande d'annulation dans l'écran n° 2
31	Demande d'annulation dans l'écran n° 3
41	Demande d'annulation dans l'écran n° 4
71	Demande d'annulation dans l'écran n° 7
311	Demande d'annulation dans l'écran n° 31

==> interprétation des codes d'erreur en provenance des procédures Ecran3-2, Ecran3-3, Ecran3-4, Ecran3-7 et Ecran3-31

Ecran6

0	OK
211	Demande d'annulation dans l'écran n° 21
221	Demande d'annulation dans l'écran n° 22
231	Demande d'annulation dans l'écran n° 23
331	Demande d'annulation dans l'écran n° 33
341	Demande d'annulation dans l'écran n° 34
351	Demande d'annulation dans l'écran n° 35

==> interprétation des codes d'erreur en provenance des procédures Ecran6-21, Ecran6-22, Ecran6-23, Ecran6-33, Ecran6-34 et Ecran6-35

Ecran7

0	OK
51	Demande d'annulation dans l'écran n° 5
53	Longueur du nom introduit dans l'écran n° 5 supérieure à 6 caractères
54	Longueur du nom introduit dans l'écran n° 5 inférieure à 1 caractère
91	Demande d'annulation dans l'écran n° 9
93	Longueur du nom court introduit dans l'écran n° 9 supérieure à 6 caractères
94	Longueur du nom long introduit dans l'écran n° 9 supérieure à 20 caractères
95	Longueur du nom court introduit dans l'écran n° 9 inférieure à 1 caractère
96	Longueur du nom long introduit dans l'écran n° 9 inférieure à 1 caractère
111	Demande d'annulation dans l'écran n° 11
113	Longueur du nom court introduit dans l'écran n° 11 supérieure à 6 caractères
114	Longueur du nom long introduit dans l'écran n° 11 supérieure à 20 caractères
115	Longueur du nom court introduit dans l'écran n° 11 inférieure à 1 caractère
116	Longueur du nom long introduit dans l'écran n° 11 inférieure à 1 caractère
201	Demande d'annulation dans l'écran n° 20
203	Longueur du nom introduit dans l'écran n° 20 supérieure à 6 caractères
204	Longueur du nom introduit dans l'écran n° 20 inférieure à 1 caractère
281	Demande d'annulation dans l'écran n° 28
283	Longueur du nom introduit dans l'écran n° 28 supérieure à 6 caractères
284	Longueur du nom introduit dans l'écran n° 28 inférieure à 1 caractère

==> interprétation des codes d'erreur en provenance des procédures Ecran7-5, Ecran7-9, Ecran7-11, Ecran7-20 et Ecran7-28

Ecran

0	OK
211	Demande d'annulation dans Ecran2-1
212	Longueur du nom introduit dans Ecran2-1 supérieure à 6 caractères
213	Longueur du nom introduit dans Ecran2-1 inférieure à 1 caractère
321	Demande d'annulation dans Ecran3-2
331	Demande d'annulation dans Ecran3-3
341	Demande d'annulation dans Ecran3-4
371	Demande d'annulation dans Ecran3-7
3311	Demande d'annulation dans Ecran3-31
6221	Demande d'annulation dans Ecran6-22
6231	Demande d'annulation dans Ecran6-23
6331	Demande d'annulation dans Ecran6-33
6341	Demande d'annulation dans Ecran6-34
6351	Demande d'annulation dans Ecran6-35
751	Demande d'annulation dans Ecran7-5
753	Longueur du nom introduit dans Ecran7-5 supérieure à 6 caractères
754	Longueur du nom introduit dans Ecran7-5 inférieure à 1 caractère
791	Demande d'annulation dans Ecran7-9
793	Longueur du nom court introduit dans Ecran7-9 supérieure à 6 caractères
794	Longueur du nom long introduit dans Ecran7-9 supérieure à 20 caractères
795	Longueur du nom court introduit dans Ecran7-9 inférieure à 1 caractère
796	Longueur du nom long introduit dans Ecran7-9 inférieure à 1 caractère

7111	Demande d'annulation dans Ecran7-11
7113	Longueur du nom court introduit dans Ecran7-11 supérieure à 6 caractères
7114	Longueur du nom long introduit dans Ecran7-11 supérieure à 20 caractères
7115	Longueur du nom court introduit dans Ecran7-11 inférieure à 1 caractère
7116	Longueur du nom long introduit dans Ecran7-11 inférieure à 1 caractère
7201	Demande d'annulation dans Ecran7-20
7203	Longueur du nom introduit dans Ecran7-20 supérieure à 6 caractères
7204	Longueur du nom introduit dans Ecran7-20 inférieure à 1 caractère
281	Demande d'annulation dans Ecran7-28
283	Longueur du nom introduit dans Ecran7-28 supérieure à 6 caractères
284	Longueur du nom introduit dans Ecran7-28 inférieure à 1 caractère

==> interprétation des codes d'erreur en provenance des procédures Ecran2, Ecran3, Ecran6 et Ecran7

InNomPedaExist

0	OK
1	Demande d'annulation de la saisie d'un nom de pédagogue existant
3	Longueur du nom de pédagogue introduit supérieure à 6 caractères
4	Longueur du nom de pédagogue introduit inférieure à 1 caractère
5	Demande d'un message plus détaillé à propos de l'erreur n° 3
6	Demande d'un message plus détaillé à propos de l'erreur n° 4
7	Nom de pédagogue introduit inexistant
8	Demande d'un message plus détaillé à propos de l'erreur n° 7

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs, soit en demandant à l'utilisateur de réintroduire le nom, soit en affichant le message supplémentaire demandé

InNomTestInexist

0	Ok
1	Demande d'annulation de la saisie d'un nom de test inexistant
3	Longueur du nom de test introduit supérieure à 6 caractères
4	Longueur du nom de test introduit inférieure à 1 caractère
5	Demande d'un message plus détaillé à propos de l'erreur n° 3
6	Demande d'un message plus détaillé à propos de l'erreur n° 4
7	Nom de test introduit existant
8	Demande d'un message plus détaillé à propos de l'erreur n° 7

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs, soit en demandant à l'utilisateur de réintroduire le nom, soit en affichant le message supplémentaire demandé

InNomTestExec

0	OK
1	Annulation de la saisie d'un nom de test existant
2	Test déjà effectué par le sujet testé
3	Nombre maximum de test autorisés atteint par le sujet testé

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs en demandant un choix au sujet testé ou en slui signalant l'exception

InNomST

0	Ok
1	Demande d'annulation de la saisie d'un nom de sujet testé
3	Longueur du nom de sujet testé introduit supérieure à 6 caractères
4	Longueur du nom de sujet testé introduit inférieure à 1 caractère
5	Demande d'un message plus détaillé à propos de l'erreur n° 3
6	Demande d'un message plus détaillé à propos de l'erreur n° 4
7	Nom de sujet testé introduit au clavier

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs soit en demandant à l'utilisateur d'introduire à nouveau un nom correct, soit en affichant le message demandé

InMatInexist

0	OK
1	Demande d'annulation de la saisie des noms de nouveaux matériaux de test
3	Longueur de l'identifiant court du nouveau matériau supérieure à 6 caractères
4	Demande d'un message d'erreur plus détaillé concernant l'erreur n° 3
5	Longueur de l'identifiant long du nouveau matériau supérieure à 20 caractères
6	Demande d'un message d'erreur plus détaillé concernant l'erreur n° 5
7	Longueur de l'identifiant court du nouveau matériau inférieure à 1 caractère
8	Demande d'un message d'erreur plus détaillé concernant l'erreur n° 7
9	Longueur de l'identifiant long du nouveau matériau inférieure à 1 caractère
10	Demande d'un message d'erreur plus détaillé concernant l'erreur n° 9
11	Identifiant court existant déjà dans les matériaux associé à ce test
12	Demande d'un message d'erreur plus détaillé concernant l'erreur n° 11
13	Identifiant long existant déjà dans les matériaux associé à ce test
14	Demande d'un message d'erreur plus détaillé concernant l'erreur n° 13

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs soit en demandant à l'utilisateur d'introduire à nouveau un identifiant correct, soit en affichant le message demandé

DoAddCon

0	OK
1	Demande d'annulation de l'ajout d'un concept à la réponse
2	Point désigné pour placer le concept déjà occupé
3	Demande d'un message supplémentaire concernant l'erreur n°2

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs soit en demandant à l'utilisateur de désigner à nouveau un autre point correct, soit en affichant le message demandé

DoAddLiai

0	OK
1	Demande d'annulation de l'ajout d'une liaison à la réponse

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs en replacant le logiciel dans l'état précédent

DoAddLR

0	OK
1	Demande d'annulation de l'ajout d'un concept à la liaison rectangle

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs en ajoutant la liaison rectangle

0	OK
1	Demande d'annulation de l'ajout d'un sommet de paragraphe à la réponse
3	Longueur de l'identifiant de paragraphe introduit supérieure à 6 caractères
4	Longueur de l'identifiant de paragraphe introduit inférieure à 1 caractère
5	Demande d'un message plus détaillé à propos de l'erreur n° 3
6	Demande d'un message plus détaillé à propos de l'erreur n° 4
7	Identifiant de paragraphe introduit existant
8	Demande d'un message plus détaillé à propos de l'erreur n° 7

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs, soit en demandant à l'utilisateur de réintroduire l'identifiant, soit en affichant le message supplémentaire demandé

DoInTexte

0	OK
1	Demande d'arrêt de la saisie du texte
2	Longueur du titre de l'entête supérieure à 20 caractères
3	Demande d'un message plus détaillé à propos de l'erreur n° 2
4	Titre d'entête sans contenu
5	Demande d'introduction d'un nouveau contenu d'entête non vide
6	Demande d'un message plus détaillé à propos de l'erreur n° 5
7	Demande de suppression du titre de l'entête
8	Longueur du titre du paragraphe supérieure à 20 caractères
9	Demande d'un message plus détaillé à propos de l'erreur n° 8
10	Titre de paragraphe sans contenu
11	Demande d'introduction d'un nouveau contenu de paragraphe non vide
12	Demande d'un message plus détaillé à propos de l'erreur n° 11
13	Demande de suppression du titre de paragraphe
14	Nombre de paragraphes introduits = 0
15	Demande d'un message plus détaillé à propos de l'erreur n° 14, si le texte est un texte de travail
16	Demande de suppression du texte explicatif
17	Demande d'introduction d'un paragraphe du texte explicatif
18	Demande d'un message plus détaillé à propos de l'erreur n° 14, si le texte est un texte explicatif
19	Demande de suppression du texte réponse
20	Demande d'introduction d'un paragraphe du texte réponse

21 Demande d'un message plus détaillé à propos de l'erreur n° 14, si le texte est un texte réponse

==> interprétation des codes d'erreur en provenance de la procédure Ecran et correction de ces erreurs, soit en demandant à l'utilisateur de réintroduire une partie de texte, soit en affichant le message supplémentaire demandé

DoGoAway

0 OK

1 Demande de fermeture de la fenêtre texte alors que le texte est incomplet

2 Demande de fermeture de la fenêtre réponse alors que la réponse est invalide

3 Demande d'un message d'erreur plus détaillé à propos de l'erreur n° 2

==> établissement des codes d'erreur en fonction du résultat des procédures de vérification et correction de ces erreurs, soit en demandant à l'utilisateur de réintroduire une partie de texte ou de réponse, soit en affichant le message supplémentaire demandé

DealWthMDInContent

0 OK (= rien d'exceptionnel)

1 l'utilisateur a sélectionné l'item de menu "Ajout Concept" (-> un clic à l'intérieur de la fenêtre fait apparaître la boîte de dialogue qui permet une sélection de concepts)

ProcessInMenu

0 OK (-> l'utilisateur a effectué jusqu'au bout l'opération demandée, sans annulation)

sinon, l'utilisateur a interrompu l'opération demandée ce qui implique que si cette opération était une étape obligatoire, il ne faut pas désactiver l'item de menu correspondant.

CONCLUSION

Ce mémoire a tenté de réaliser un double objectif :

Un de ces deux objectifs était, sur un plan pratique, de poursuivre l'implémentation du logiciel entamée par F.Rinclin.

Ce logiciel peut être considéré sous deux points de vue :

D'abord, il a servi de cas pratique d'application de la méthode de gestion des erreurs de l'utilisateur décrite auparavant : en effet, les principes exposés dans la première partie ont été appliqués tels quels pour la réalisation d'une gestion des cas d'exceptions et des erreurs de l'utilisateur. On a ainsi pu constater la faisabilité de la méthode décrite en théorie.

Ensuite, ce logiciel peut être vu comme un logiciel en tant que tel, comme l'outil informatique de tests de mémoire humaine demandé par les psycho-pédagogues. Vu dans cette optique, le logiciel créé est un logiciel de base, c'est-à-dire qu'il réalise de façon opérationnelle les fonctions principales demandées. Bien sûr, des améliorations pourraient être apportées en matière d'efficacité ou sur certains aspects de facilité ou sur certaines restrictions d'utilisation. On pourra aussi par la suite accroître les possibilités du logiciel, ajouter d'autres fonctions de manière aisé, puisque le logiciel s'est voulu le plus extensible possible.

Une remarque concerne le choix du langage de programmation, guidé principalement par la continuité vis-à-vis de la partie de programme déjà réalisé par F.Rinclin.
Il est certain que le langage Pascal offre très peu de facilités en matière de gestion d'erreurs, par comparaison avec d'autres langages évoqués plus haut, tels que ADA ou PL/1, et que la simulation en Pascal de ces instructions est assez lourde.
Cependant, parmi les langages "classiques", Pascal présente de nombreux avantages, notamment en ce qui concerne ses possibilités d'ouverture vers les "ToolBox" et d'autres fonctions non standards, qui n'existent pas encore pour les autres langages.

Un petit mot enfin à propos du choix du matériel : C'est l'Apple MacIntosh qui a servi pour l'ensemble du projet, pour plusieurs raisons :

D'une part, le souhait de réutiliser le travail déjà réalisé par F.Rinclin supposait la continuité dans le choix de la machine. En ce qui la concerne, ce choix avait principalement été guidé par les desiderata des psycho-pédagogues.

D'autre part, en matière d'interfaces homme machine, le MacIntosh présente de nombreuses possibilités par l'intermédiaire des "ToolBox". Ces outils permettent de créer "facilement" des interfaces conviviales; de plus, leur présentation fait l'objet d'un certaine uniformité plus ou moins standardisée, pour toutes les applications tournant sur MacIntosh.

Les outils proposés dans les "ToolBox" sont très puissants. Mais, le revers de la médaille est leur densité et leur complexité, ce qui rend difficile leur maîtrise par le programmeur.

Cependant, de façon générale, l'utilisation de "boîtes à outils" est une expérience profitable, car même si les fonctions offertes peuvent varier selon les cas, la philosophie qui les guide et les principes de base de programmation qu'ils impliquent restent identiques.

Les interfaces créées grâce aux "ToolBox" MacIntosh présentent plusieurs avantages :

D'abord, comme nous l'avons souligné plus haut, elles sont plus ou moins standardisées à travers toutes les applications;

Ensuite, grâce aux ressources, elles sont facilement modifiables, sans exiger une recompilation du programme;

De plus, la programmation MacIntosh est une programmation guidée par les événements, ce qui signifie que c'est l'utilisateur et non le logiciel qui impose son rythme ;

Enfin, l'utilisation de la souris, quoique parfois peu apprécié par certains utilisateurs, présente certains avantages, entre autres en matière de gestion des erreurs, ou plutôt de leur prévention : à la place de demander à l'utilisateur de taper, par exemple, un nom existant et de faire suivre cette introduction d'une vérification, on lui propose une liste des noms répondant à ce critère dans laquelle il doit sélectionner le nom souhaité. On évite ainsi une vérification, l'affichage d'un message d'erreur et l'introduction d'un nouveau nom correct.

On peut procéder de la même façon avec les items des menus représentant les différentes opérations, ils peuvent être activés ou désactivés selon que l'opération est réalisable ou non ce qui rend l'interface trés conviviale pour l'utilisateur.

Pour l'autre objectif au niveau théorique, nous nous sommes efforcés de rechercher une méthode de conception d'un système de gestion des erreurs et plus particulièrement des erreurs de l'utilisateur. Nous avons expliqué l'importance d'une bonne gestion d'erreurs pour l'utilisateur, mais aussi la difficulté que représente pour le programmeur l'implémentation d'une telle gestion.

Nous avons ensuite décrit des solutions réalisables pour concevoir un système de gestion des erreurs de l'utilisateur. Et si notre solution personnelle réalisable d'un système de gestion des erreurs du programmeur semblait bien piêtre comparée à la puissance de certains outils logiciles existants, par contre, dans le domaine de la gestion des erreurs de l'utilisateur, notre méthode présente un intérêt tout particulier, étant donné le vide existant en la matière.

L'avantage de la méthode de conception décrite est qu'elle est systématique à appliquer et indépendante du langage de programmation utilisé et de l'application considérée : en effet, elle met en oeuvre des structures de données simples existant dans tout langage et les instructions qui implémentent cette gestion peuvent s'insérer dans n'importe quel programme même déjà implémenté.

Son inconvénient principal est la lourdeur : comme on l'a expliqué, chaque appel à une procédure susceptible de produire une erreur est suivi d'un test sur la valeur du code d'erreur. Selon la valeur de ce code, on effectue une interprétation du code ou une suppression de l'erreur. Il faut donc repérer les procédures susceptibles de produire une erreur, trouver chacun de leurs appels et insérer à la suite une partie de code supplémentaire ayant pour fonction de gérer les erreurs selon leur code. On traite ainsi à la fois les cas normaux et les cas anormaux simultanément.

Cependant, une recherche future pourrait s'attacher à mettre au point un outil, un précompilateur qui permettrait l'implémentation d'un programme en deux phases distinctes :

Dans un premier temps, un simple compilateur créerait de façon normale, le programme réalisant les cas normaux, c'est-à-dire les traitements à exécuter lorsque tout se passe bien et quand l'utilisateur ne commet aucune erreur.

Dans un second temps, la gestion des erreurs serait décrite séparément, sous un formalisme qui pourrait, par exemple, être un pseudo-langage, afin d'être indépendant du langage cible. on y définirait pour chaque appel de procédure les classes d'erreur qu'il peut renvoyer et le traitement ou l'interprétation à exécuter, en fonction de la valeur du code d'erreur.

Ensuite, un précompilateur serait chargé d'aller, dans le programme, rechercher les appels de procédure mentionnés et d'insérer, automatiquement, à la suite de l'instruction d'appel, la partie de code supplémentaire décrivant les traitements d'exception.

Le travail à réaliser consisterait à implémenter ce précompilateur, de façon générique, c'est-à-dire avec un noyau commun, indépendant du langage cible, et des modules séparés permettant d'adapter ce noyau à un langage cible particulier. Il faudrait aussi définir le formalisme de description des traitements d'exception.

Mais cette recherche, quoique très intéressante, sort du cadre fixé à ce présent mémoire...

BIBLIOGRAPHIE

- D.R.BARSTW, H.E.SHROBE, E.SANDEWALL
"Interactive programming environments"
Mac Graw Hill (1984)

- J.RASMUSSEN
"Information processing and human machine interaction : an approach to cognitive engineering"
Series Vol 12 North Holland Series (1986)

- H. KOPETZ
"Software reliability"
Mac Millan (1979)

- P.LE BEUX
"Introduction à ADA"
SYBEX (1982)

- G.GOOS, J.HARTMANIS
"The programming language ADA, reference manual"
SPRINGER VERLAG (1981)

- F.VEILLON, JM.CAGNAT
"Cours de programmation en langage PL/1"
ARMAND COLIN.

- D.L.SCAPIN
"Guide ergonomique de conception des interfaces homme-machine"
INRIA

-B.SHNEIDERMAN
"Designing the user interface : strategies for effective human computer interaction"
Addison Wesley (1987)

ANNEXE 1 : EXEMPLES DES TROIS TYPES DE TESTS

Lorsqu'il exécute un test, un sujet testé a à sa disposition : un texte explicatif, un texte de travail et différents matériaux selon le type du test. Détaillons ces différents éléments :

- un **texte explicatif** qui peut être, par exemple, pour l'explication d'un test de type réseau :

  Titre du paragraphe 0 : "But"
  Contenu du paragraphe 0 :
  " Le but du test suivant est de représenter le contenu d'un texte sous la forme d'un réseau sémantique."

  Titre du paragraphe 1 : "Définitions"
  Contenu du paragraphe 1 :
  "Les noeuds de ce réseau sont les idées principales du texte; ils sont appelés les concepts. Les arcs qui relient ces concepts sont appelés des liaisons. Il existe également une liaison d'un type particulier appelé liaison rectangle : elle représente un regroupement de concepts."

  Titre du paragraphe 2 : "Données"
  Contenu du paragraphe 2 :
  "La liste des noms des concepts et des noms des liaisons ainsi que le texte sont fournis comme données."

  Titre du paragraphe 3 : "Comment?"
  Contenu du paragraphe 3 :
  "Vous devez sélectionner un à un les concepts et les liaisons proposés pour les ajouter à la réponse que vous créez.
  Cette réponse apparaît à l'écran sous forme graphique : les concepts sont représentés par des ovales, les liaisons par des segments de droite et les liaisons rectangles par des rectangles."

  Titre du paragraphe 4 : "Fin"
  Contenu du paragraphe 4 :
  "La réponse est terminée lorsque tous les concepts proposés ont été utilisés. Vous pouvez alors consulter une réponse proposée par le pédagogue et la comparer avec la vôtre."

- un **texte de travail**, que, pour notre facilité, nous supposerons valable pour les trois types de tests :

Contenu de l'entête :
"L'histoire du petit Chaperon rouge"

Titre du paragraphe 0 : "Qui?"
Contenu du paragraphe 0 :
" Il était une fois une petite fille qui habitait près d'une grande forêt avec sa maman. Comme elle portait toujours une cape rouge, on l'appelait le petit Chaperon rouge."

Titre du paragraphe 1 : "Quoi?"
Contenu du paragraphe 1 :
"Un jour, la maman du Chaperon rouge lui dit : " Ta grand-mère est malade. Va lui apporter ces bonnes choses à manger." La grand-mère habitait de l'autre côté de la forêt."

Titre du paragraphe 2 :"Le loup"
Contenu du paragraphe 2 :
"Elle rencontra un loup affamé qui lui demanda où elle allait. Apprenant qu'elle allait chez sa grand-mère, il prit un raccourci à travers la forêt pendant que le petit Chaperon Rouge continuait à flâner le long du chemin."

Titre du paragraphe 3 : "chez la grand-mère"
Contenu du paragraphe 3 :
"Le loup frappa à la porte de la maison de la grand-mère et entra. Il poussa la grand-mère dans l'armoire et ferma la porte à clé. Alors, le loup mit une chemise et un bonnet de nuit de grand-mère et se glissa vite dans le lit."

Titre du paragraphe 4 : "Arrivée du Chaperon"
Contenu du paragraphe 4 :
"A ce moment-là, le petit Chaperon rouge frappa à la porte. "Entre!", s'écria le loup en imitant la voix d'une vieille femme. La petite fille s'approcha doucement du lit :
"Mais grand-mère, comme tu as de grandes dents!"
"C'est pour mieux te manger", s'écria le loup. En disant ces mots, le loup sauta du lit pour attraper la petite fille."

Titre du paragraphe 5 : "La poursuite"
Contenu du paragraphe 5 :
"Mais la petite fille fut plus rapide que lui et s'enfuit en criant hors de la maison. Le loup la poursuivit."

Titre du paragraphe 6 : "Le chasseur"
Contenu du paragraphe 6 :
"Heureusement, un chasseur qui passait par là l'entendit. Le chasseur prit son fusil et tua le loup avant qu'il ait pu attraper la petite fille.

Titre du paragraphe 7 : "Dénouement"
Contenu du paragraphe 7 :
"Le chasseur ouvrit ensuite la porte de l'armoire pour délivrer la grand-mère. Celle-ci embrassa sa petite fille."

- si le test est de type **réseau** :

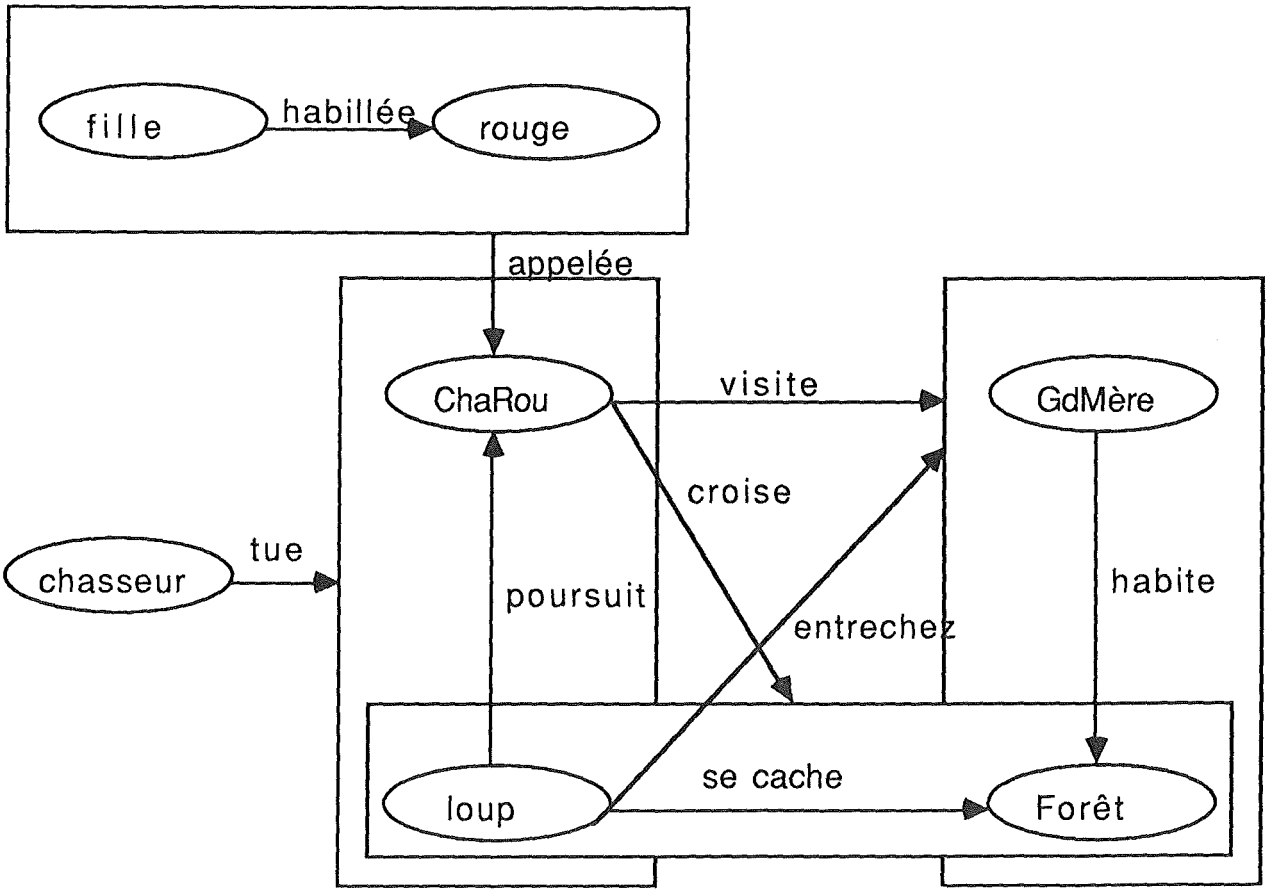
  * la liste des **concepts** :

Identifiants longs	Identifiants courts
Petite fille	Fille
Cape rouge	Rouge
Chaperon rouge	ChaRou
Grand-mère	GdMère
Bout de la forêt	Forêt
Loup	Loup
Chasseur	Chasseur

  * la liste des **liaisons** :

Identifiants longs	Identifiants courts
Habillée	Habillée
Appelée	Appelée
Rendre visite	Visite
Croise	Croise
Habite	Habite
Entrer ds la maison	Entrechez
Se cache	Se cache
Poursuit	Poursuit
Tue	Tue

  * en sélectionnant, un à un, les concepts et les liaisons, le sujet testé doit constituer une **réponse graphique** du style ci-dessous :



- si le test est de type **liaison de paragraphes** :

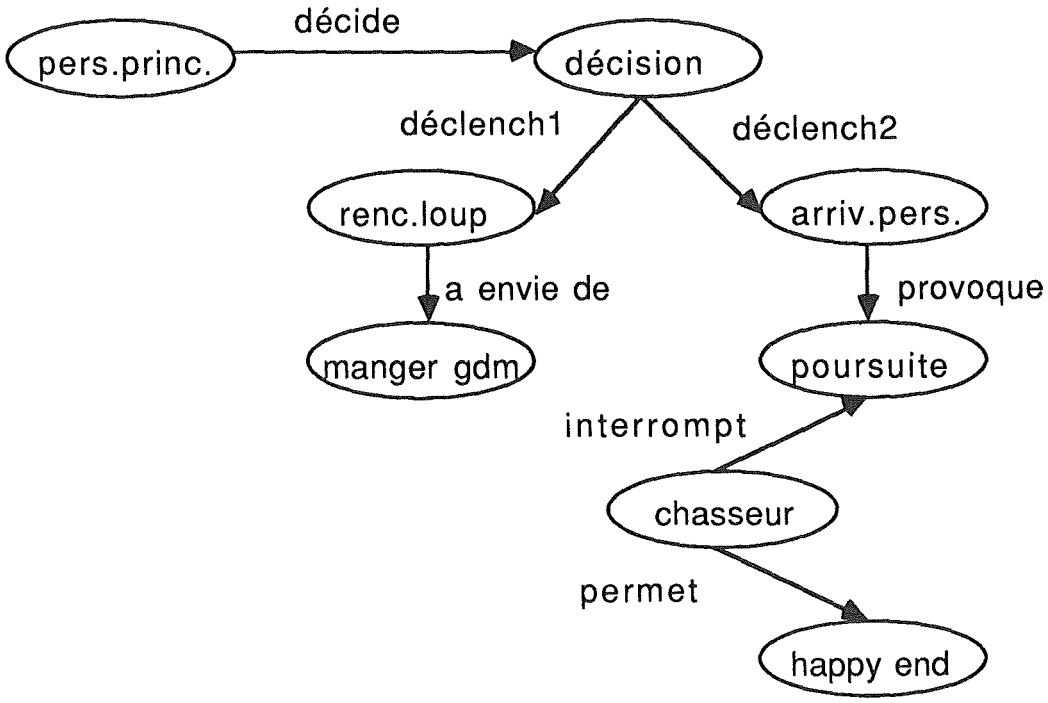
  * la liste des **titres et numéros de paragraphes** :

Numéros	Titres	Identifiants (attribués par la personne qui exécute le test)
0	Qui	Pers.Princ.
1	Quoi	Décision
2	Le loup	Renc.loup
3	Chez la grand-mère	Manger GdM
4	Arrivée du Chaperon	Arriv.Pers.
5	La poursuite	Poursuite
6	Le chasseur	Chasseur
7	Dénouement	Happy End

  * la liste des **liaisons** :

Identifiants longs	Identifiants courts
Décide	Décide
Déclenche d'abord	Déclench1
Déclenche ensuite	Déclench2
Avoir envie de	a envie de
Provoque	Provoque
Interrompt	Interrompt
Poursuit	Poursuit

  * en sélectionnant un à un les liaisons et les paragraphes et en attribuant un nom identifiant à chacun des paragraphes, le sujet testé doit constituer une **réponse graphique** du style ci-dessous :



- si le test est de type **hiérarchisation de paragraphes** :

  * la liste des **titres et numéros de paragraphes** :

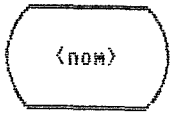
Numéros	Titres	Identifiants (attribués par la personne qui exécute le test)
0	Qui	Pers1
1	Quoi	Décision
2	Le loup	Pers2
3	Chez la grand-mère	P2-GdM
4	Arrivée du Chaperon	P1-GdM
5	La poursuite	Poursuite
6	Le chasseur	Intervention
7	Dénouement	Happy End

  * en sélectionnant un à un les paragraphes et en attribuant un nom identifiant à chacun des paragraphes, le sujet testé doit constituer une **hiérarchie** du style ci-dessous :

1.Pers1
    1.1.Décision
    1.2.P1-GdM
2.Pers2
    2.1.P2-GdM
3.Poursuite
    3.1.Intervention
    3.2.Happy End.

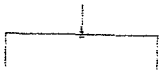
ANNEXE 2 : DECOUPE DE L'APPLICATION EN UNITES EXTERNES

Nous exposerons ici la succession des unités externes qui forme l'application en tant que telle.
Le formalisme de description de cette succession est expliqué dans la légende ci-dessous :



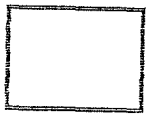
: marque le début d'un traitement

: marque la fin d'un traitement

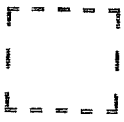


: marque une alternative,
un choix proposé à l'utilisateur

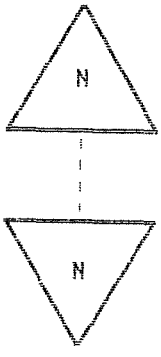
: marque une conditionnelle,
un traitement différencié selon
la valeur d'une condition



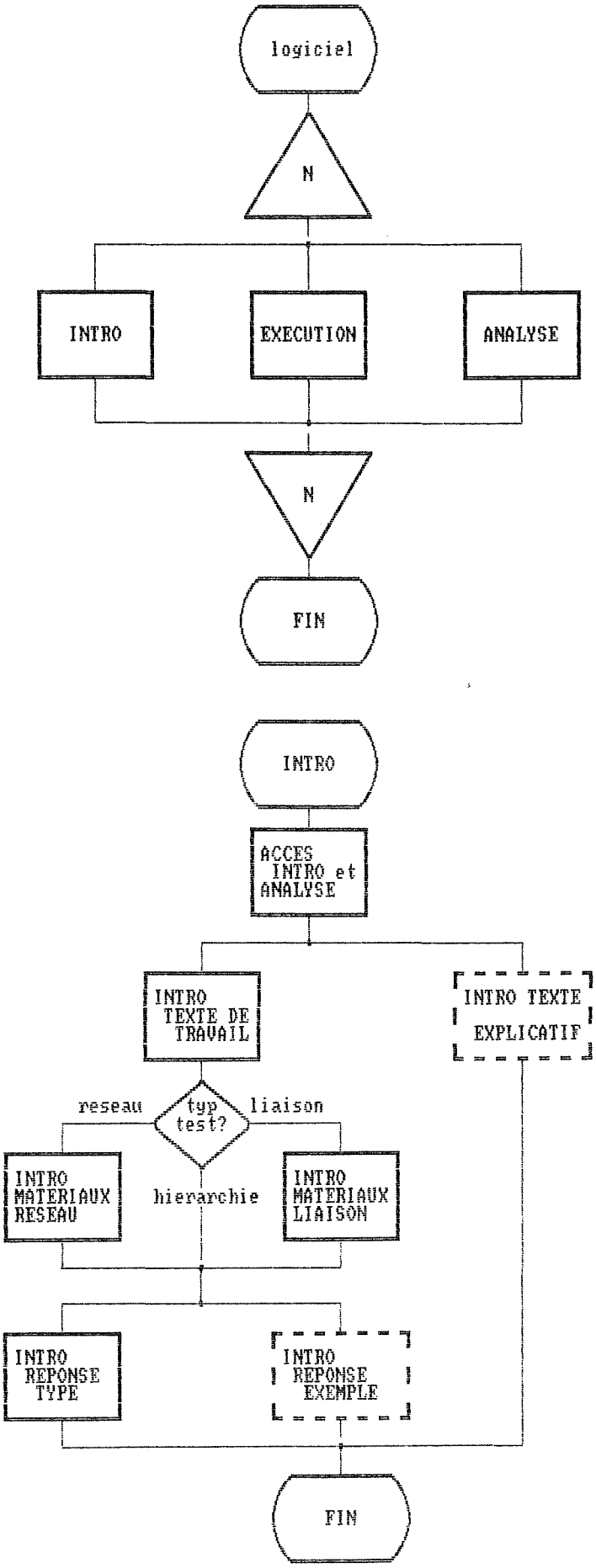
: marque un traitement obligatoire

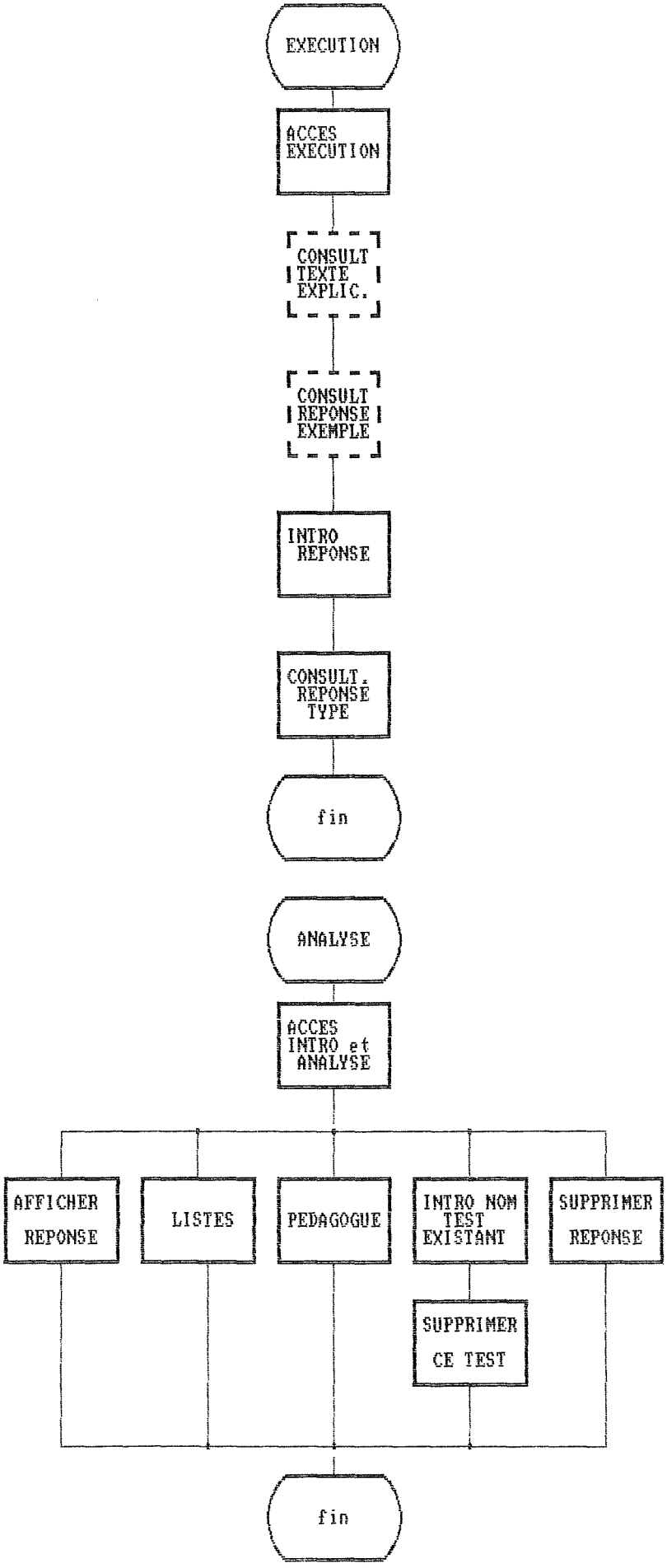


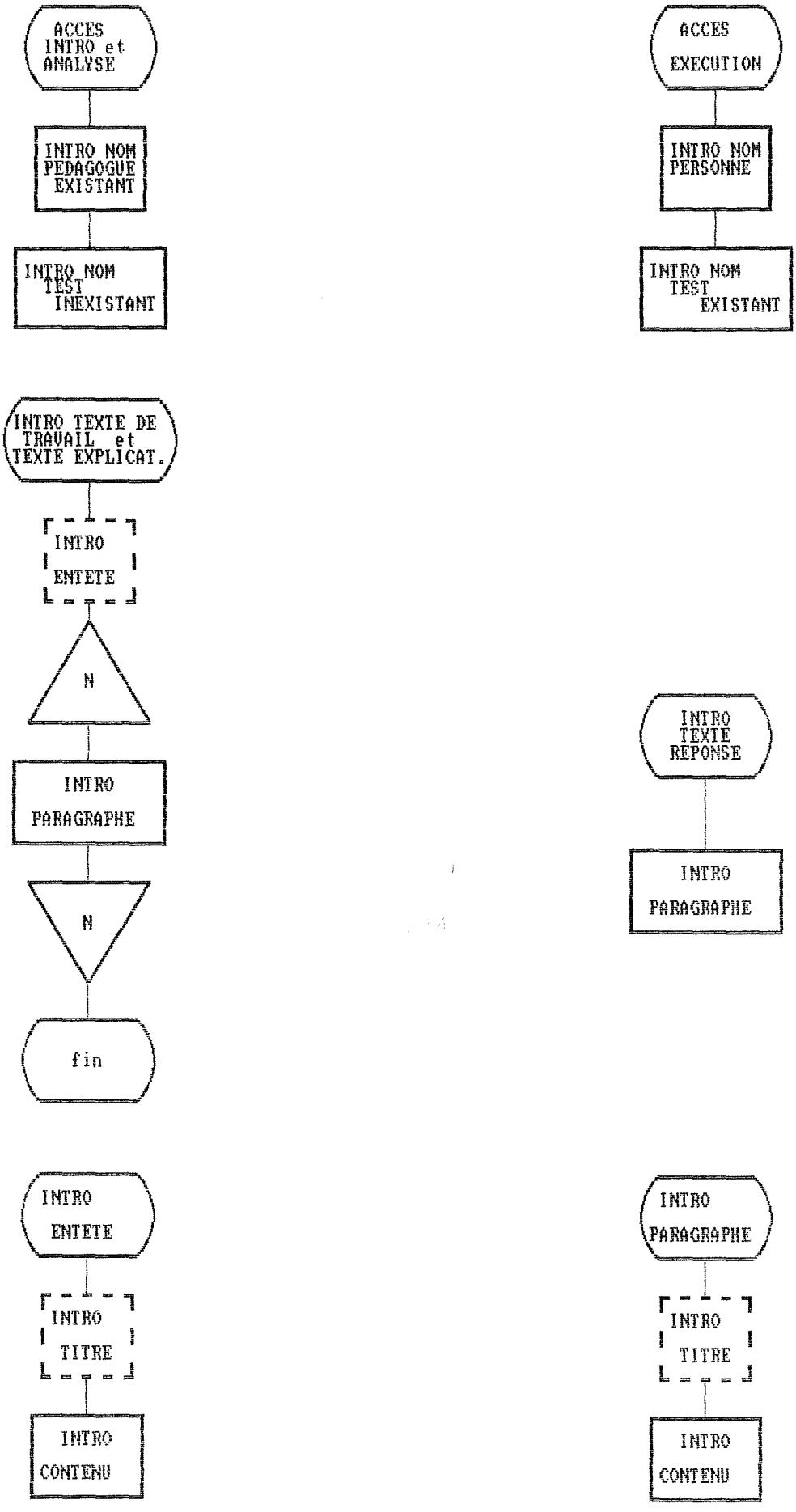
: marque un traitement facultatif

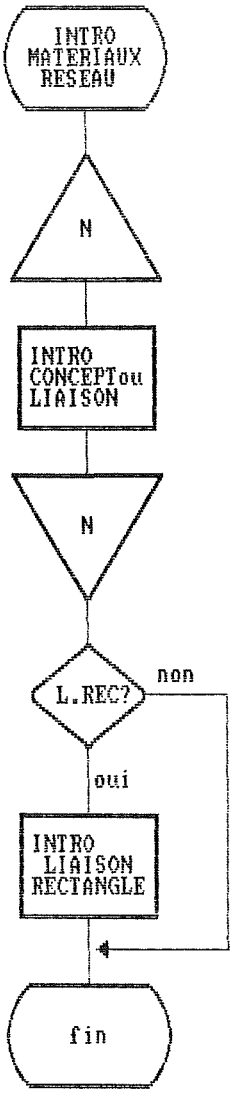


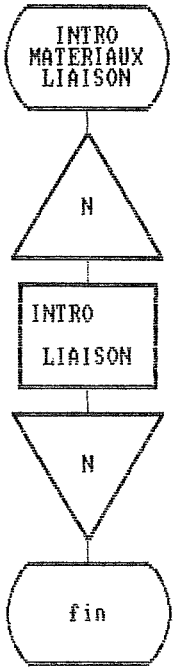
: marque une répétition de N fois le traitement
compris entre les deux triangles.

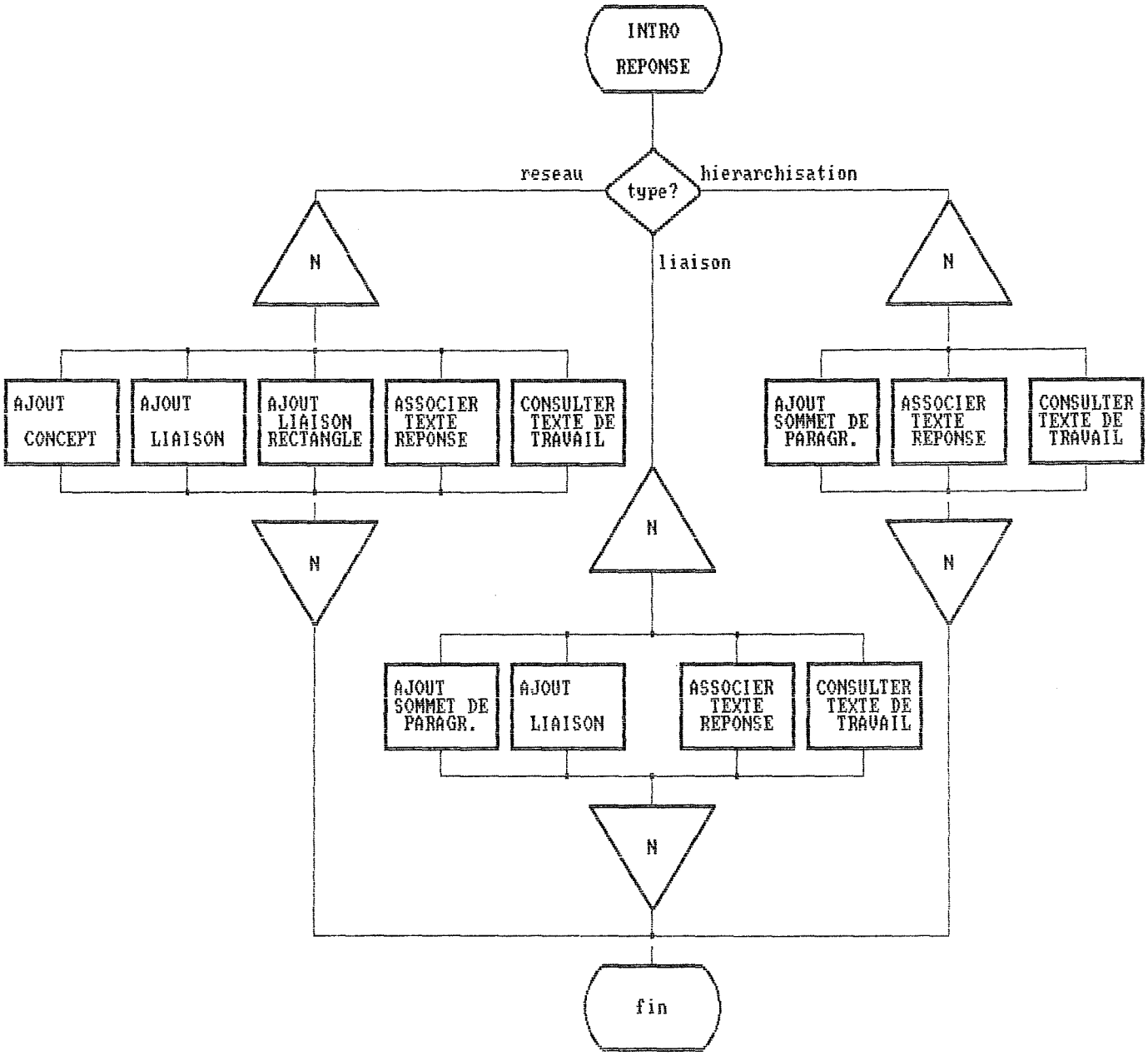

logiciel
N
INTRO
EXECUTION
ANALYSE
N
FIN
INTRO
ACCES
INTRO et
ANALYSE
INTRO
TEXTE DE
TRAVAIL
INTRO TEXTE
EXPLICATIF
reseau
typ
test?
liaison
INTRO
MATERIAUX
RESEAU
hierarchie
INTRO
MATERIAUX
LIAISON
INTRO
REPONSE
TYPE
INTRO
REPONSE
EXEMPLE
FIN

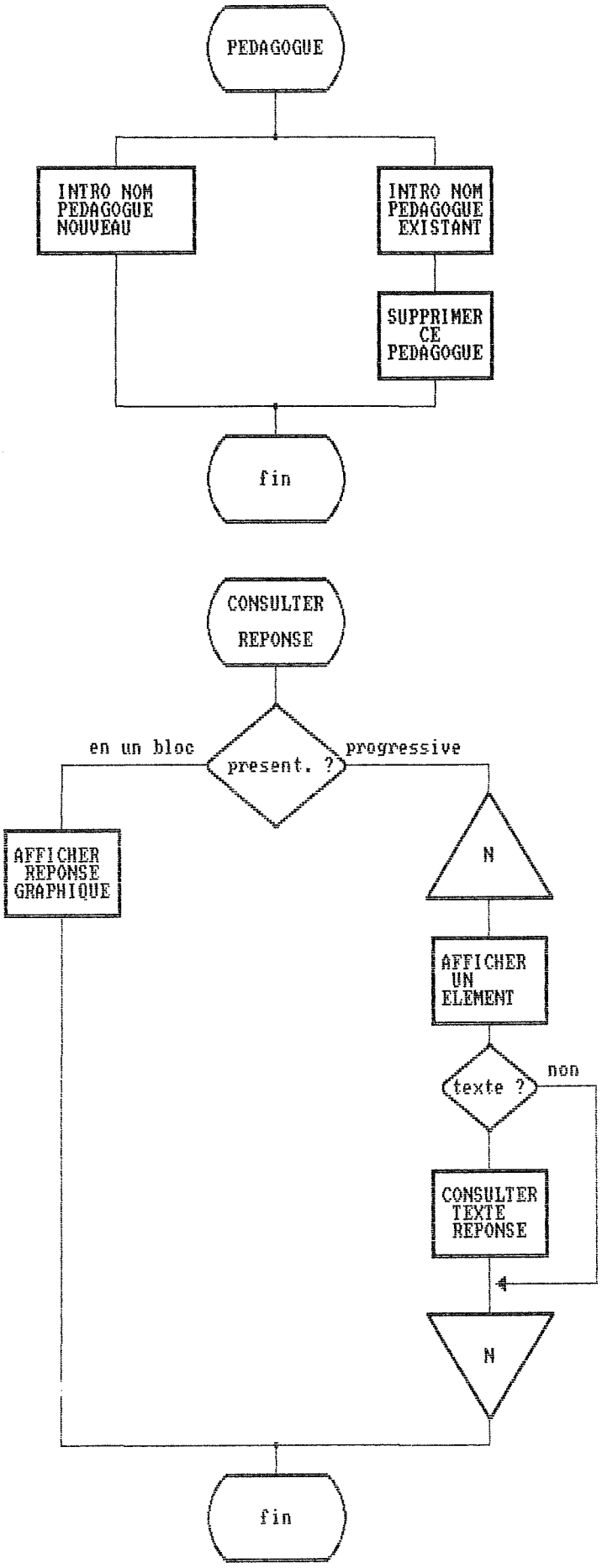

EXECUTION
ACCES EXECUTION
CONSULT TEXTE EXPLIC.
CONSULT REPONSE EXEMPLE
INTRO REPONSE
CONSULT. REPONSE TYPE
fin
ANALYSE
ACCES INTRO et ANALYSE
AFFICHER REPONSE
LISTES
PEDAGOGUE
INTRO NOM TEST EXISTANT
SUPPRIMER REPONSE
SUPPRIMER CE TEST
fin

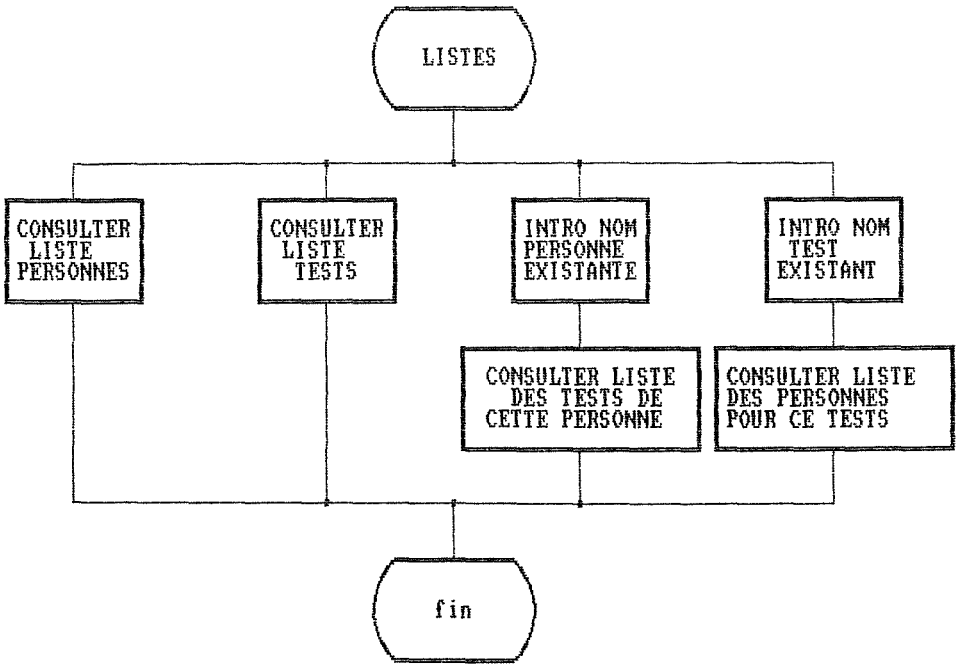

ACCES
INTRO et
ANALYSE
INTRO NOM
PEDAGOGUE
EXISTANT
INTRO NOM
TEST
INEXISTANT
ACCES
EXECUTION
INTRO NOM
PERSONNE
INTRO NOM
TEST
EXISTANT
INTRO TEXTE DE
TRAVAIL et
TEXTE EXPLICAT.
INTRO
ENTETE
N
INTRO
PARAGRAPHE
N
fin
INTRO
TEXTE
REPONSE
INTRO
PARAGRAPHE
INTRO
ENTETE
INTRO
TITRE
INTRO
CONTENU
INTRO
PARAGRAPHE
INTRO
TITRE
INTRO
CONTENU

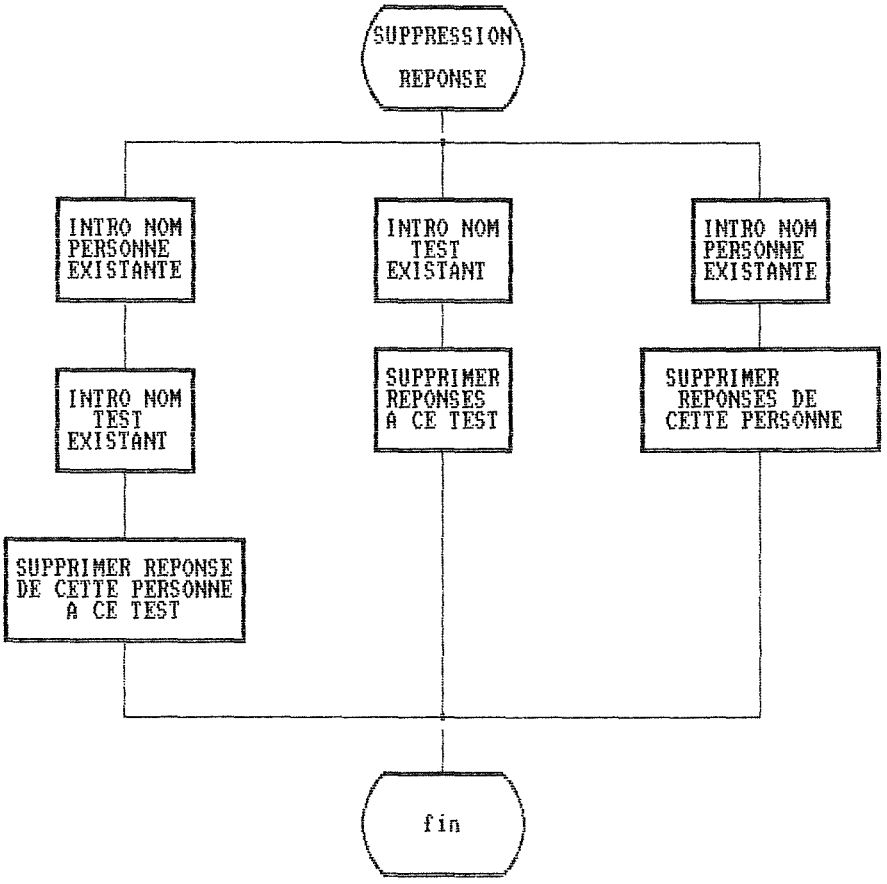

INTRO
MATERIAUX
RESEAU
N
INTRO
CONCEPTou
LIAISON
N
L.REC?
non
oui
INTRO
LIAISON
RECTANGLE
fin

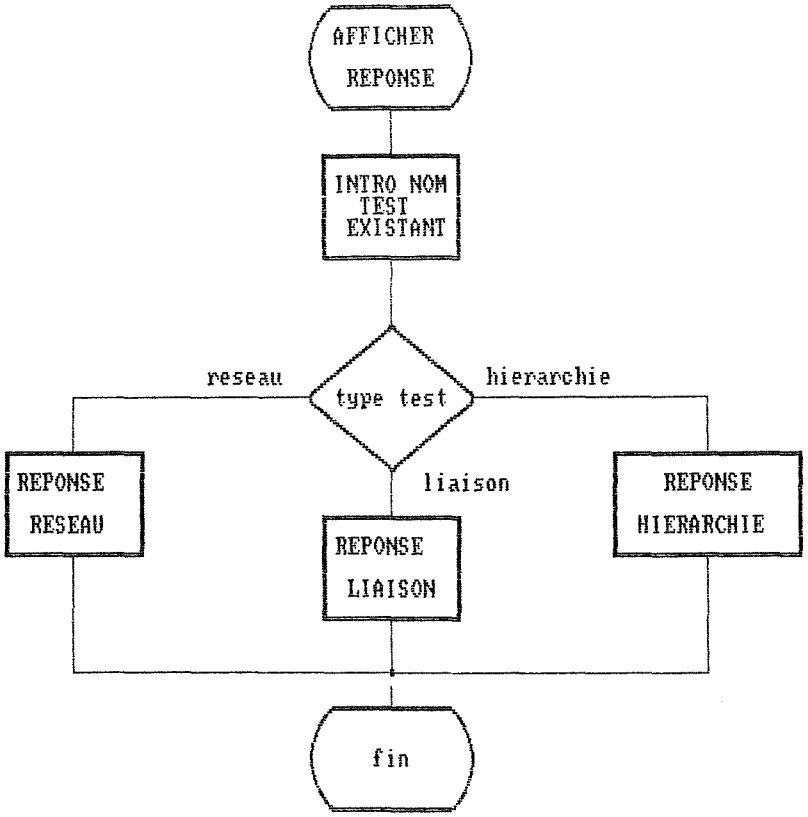

INTRO
MATERIAUX
LIAISON
N
INTRO
LIAISON
N
fin

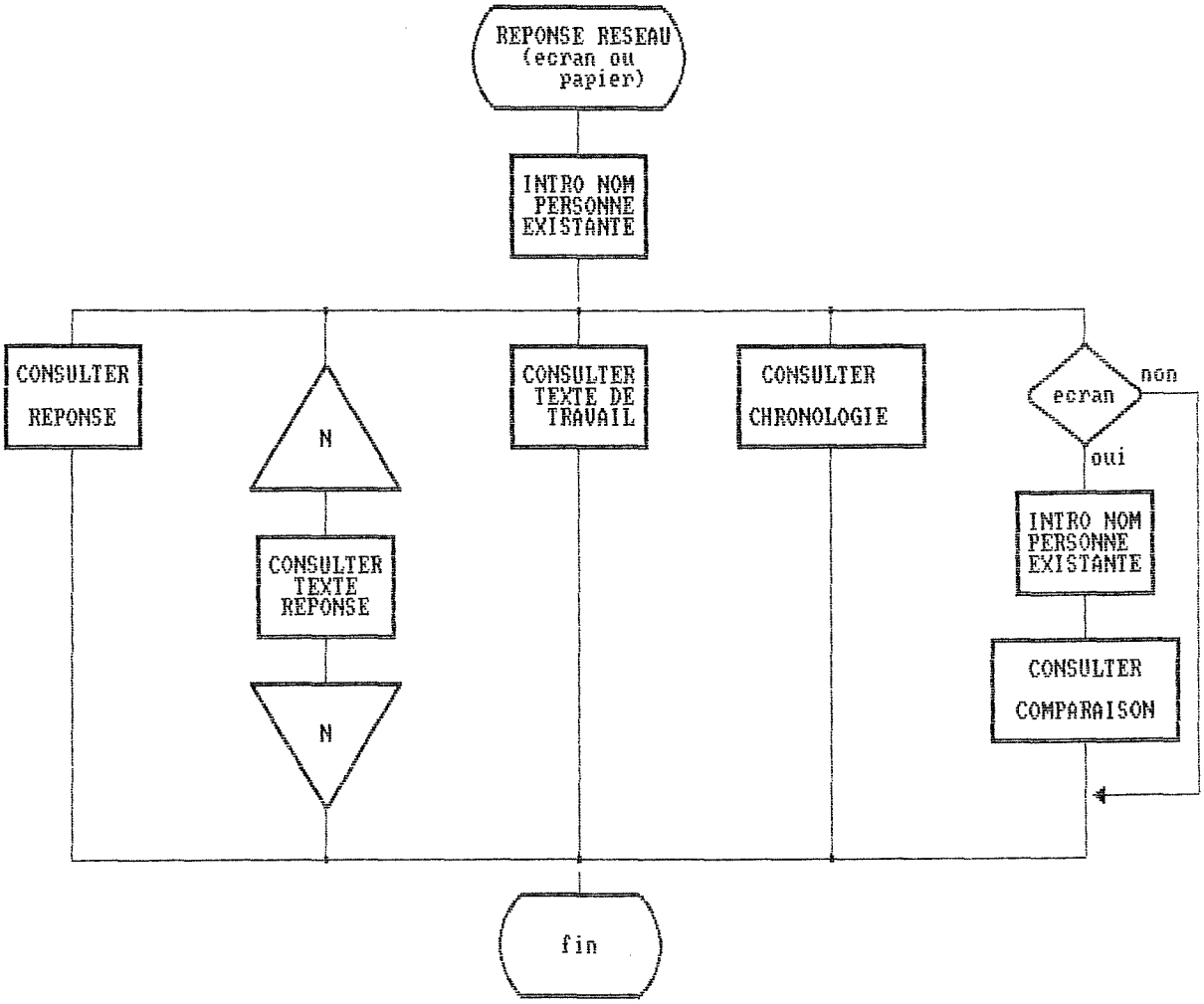

INTRO
REPONSE
type?
reseau
hierarchisation
liaison
N
N
N
N
N
N
AJOUT
CONCEPT
AJOUT
LIAISON
AJOUT
LIAISON
RECTANGLE
ASSOCIER
TEXTE
REPONSE
CONSULTER
TEXTE DE
TRAVAIL
AJOUT
SOMMET DE
PARAGR.
ASSOCIER
TEXTE
REPONSE
CONSULTER
TEXTE DE
TRAVAIL
AJOUT
SOMMET DE
PARAGR.
AJOUT
LIAISON
ASSOCIER
TEXTE
REPONSE
CONSULTER
TEXTE DE
TRAVAIL
fin

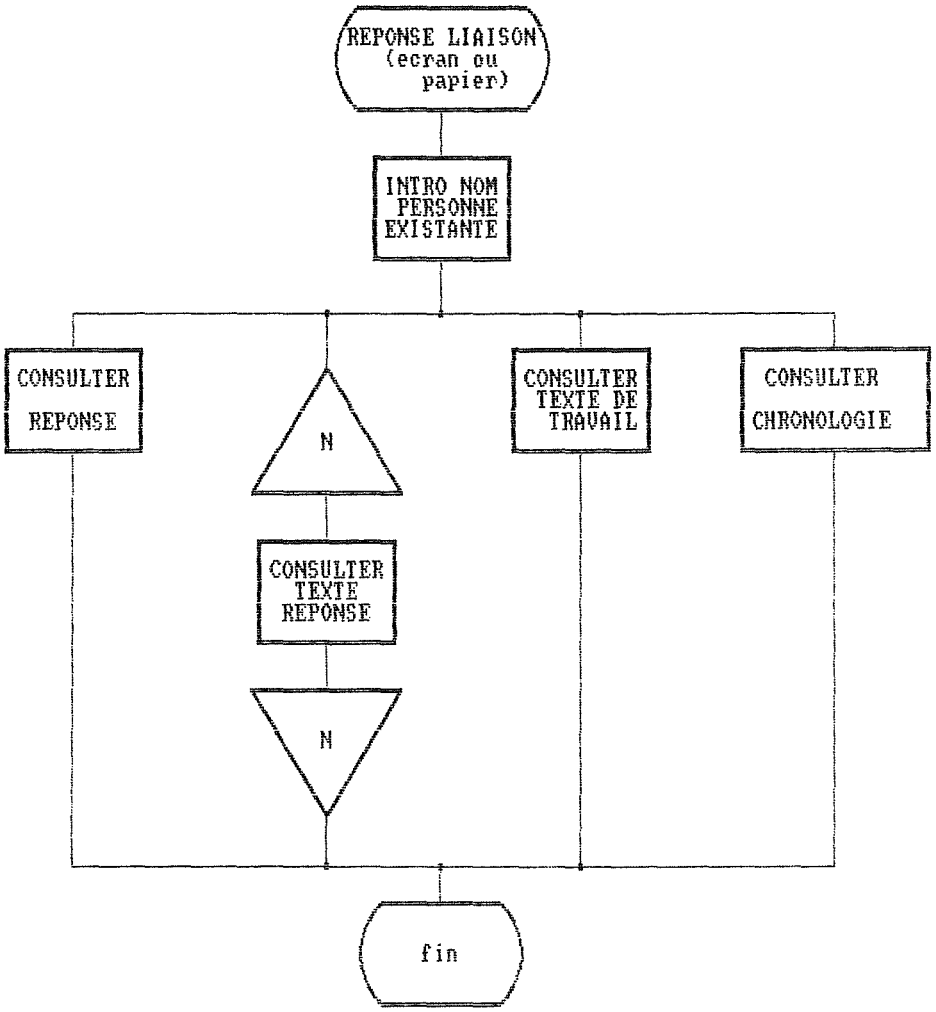

PEDAGOGUE
INTRO NOM PEDAGOGUE NOUVEAU
INTRO NOM PEDAGOGUE EXISTANT
SUPPRIMER CE PEDAGOGUE
fin
CONSULTER REPONSE
en un bloc
present. ?
progressive
AFFICHER REPONSE GRAPHIQUE
N
AFFICHER UN ELEMENT
texte ?
non
CONSULTER TEXTE REPONSE
N
fin

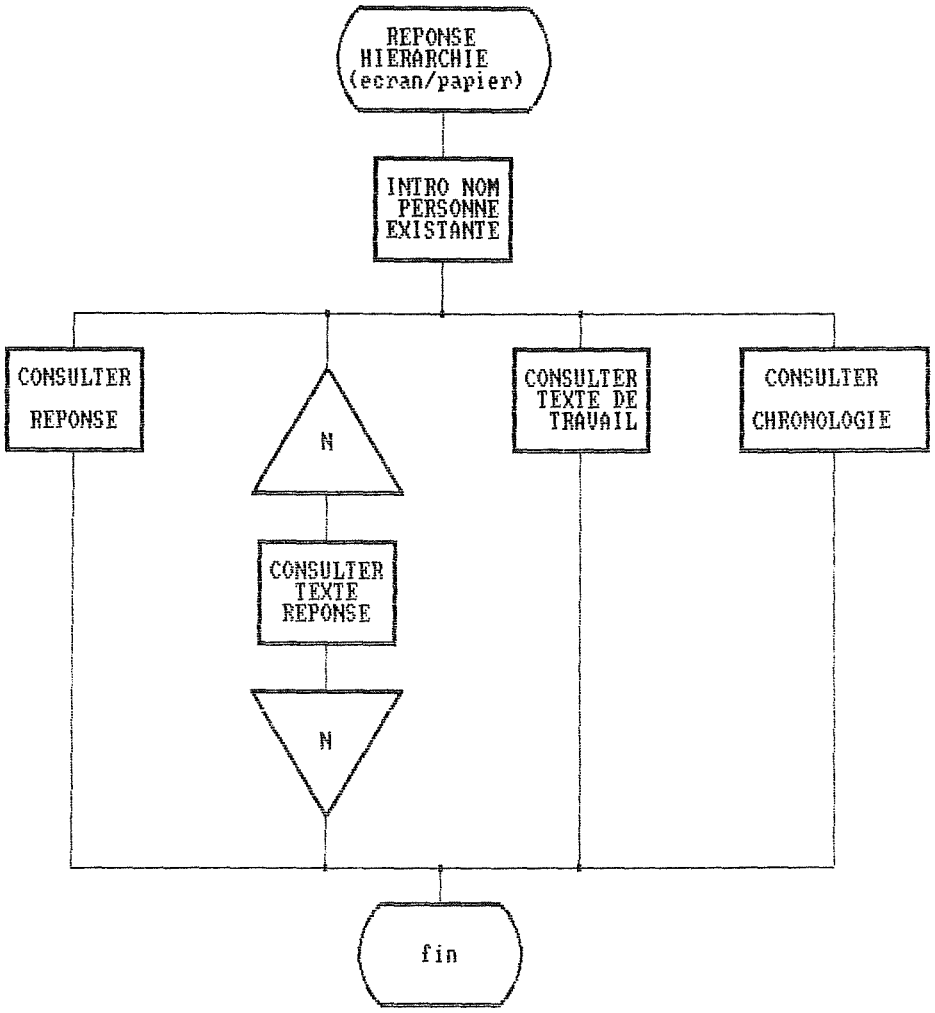

LISTES
CONSULTER LISTE PERSONNES
CONSULTER LISTE TESTS
INTRO NOM PERSONNE EXISTANTE
INTRO NOM TEST EXISTANT
CONSULTER LISTE DES TESTS DE CETTE PERSONNE
CONSULTER LISTE DES PERSONNES POUR CE TESTS
fin


SUPPRESSION REPONSE
INTRO NOM PERSONNE EXISTANTE
INTRO NOM TEST EXISTANT
INTRO NOM PERSONNE EXISTANTE
INTRO NOM TEST EXISTANT
SUPPRIMER REPONSES A CE TEST
SUPPRIMER REPONSES DE CETTE PERSONNE
SUPPRIMER REPONSE DE CETTE PERSONNE A CE TEST
fin


AFFICHER
REPONSE
INTRO NOM
TEST
EXISTANT
reseau
type test
hierarchie
REPONSE
RESEAU
liaison
REPONSE
LIAISON
REPONSE
HIERARCHIE
fin


REPONSE RESEAU
(ecran ou
papier)
INTRO NOM
PERSONNE
EXISTANTE
CONSULTER
REPONSE
N
CONSULTER
TEXTE
REPONSE
N
CONSULTER
TEXTE DE
TRAVAIL
CONSULTER
CHRONOLOGIE
ecran
non
oui
INTRO NOM
PERSONNE
EXISTANTE
CONSULTER
COMPARAISON
fin


REPONSE LIAISON
(ecran ou
papier)
INTRO NOM
PERSONNE
EXISTANTE
CONSULTER
REPONSE
N
CONSULTER
TEXTE
REPONSE
N
CONSULTER
TEXTE DE
TRAVAIL
CONSULTER
CHRONOLOGIE
fin


REPONSE HIERARCHIE (ecran/papier)
INTRO NOM PERSONNE EXISTANTE
CONSULTER REPONSE
N
CONSULTER TEXTE REPONSE
N
CONSULTER TEXTE DE TRAVAIL
CONSULTER CHRONOLOGIE
fin

ANNEXE 3 : PRESENTATION DES ECRANS

F.Rinclin a choisi de classer ses écrans en groupes. Cette décision a été maintenue, quoique aménagée en fonction des modifications apportées à la philosophie même du logiciel : en effet, le logiciel a été implémenté à la façon d'une application typique du MacIntosh. Cela signifie entre autres que c'est l'utilisateur et non plus la machine qui dirige le déroulement des opérations.
De par ce fait, certains groupes d'écrans ont été modifiés, d'autres ont été supprimés, d'autres encore au contraire ont été ajoutés. Pour cette raison, nous allons tout d'abord commencer par redéfinir chacune des classes d'écrans utilisées en se référant à leur représentation en termes des concepts d'une application MacIntosh :

- *groupe 1 :*
  Ce groupe contenait les écrans qui présentent à l'utilisateur un menu, c'est-à-dire qui offrent le choix entre plusieurs actions possibles. Ce groupe a disparu étant donné que le logiciel est guidé par une barre de menus unique qui traverse tout le logiciel; en fonction des étapes déjà réalisées, certains items sont rendus actifs, d'autres sont rendus inactifs.

- *groupe 2 :*
  Ce groupe contient les écrans qui demandent à l'utilisateur d'introduire de façon simple une ou plusieurs données dans le logiciel.
  Ce groupe est implémenté par une boîte de dialogue contenant un texte statique, une boîte d'édition et deux boutons de confirmation ou d'annulation.

- *groupe 3 :*
  Ce groupe contient les écrans qui signalent une erreur à l'utilisateur.
  Il est implémenté par une boîte d'alerte

- *groupe 4 :*
  Ce groupe contient les écrans qui demandent à l'utilisateur un choix binaire de type 'oui' ou 'non', par exemple.
  Il est implémenté par une boîte de dialogue contenant un texte statique et deux boutons présentant les deux choix possibles.

- *groupe 5 :*

  Ce groupe contient les écrans qui indiquent à l'utilisateur à quel niveau il se trouve, ce qu'il peut faire ou les écrans qui annoncent la suite des événements.
  Ce groupe a disparu étant donné que c'est l'utilisateur lui-même qui, par l'intermédiaire des menus, dirige le système.

- *groupe 6 :*

  Ce groupe contient les écrans qui présentent à l'utilisateur une liste de données qu'il peut consulter ou dans laquelle il peut choisir un élément.
  Ce groupe est implémenté par une boîte de dialogue contenant une "list box" et un ou deux boutons.

- *groupe 7 :*

  Ce groupe contient les écrans qui reprennent à la fois l'introduction des données (groupe 2) et la consultation d'une liste (groupe 6).
  Il est implémenté par une boîte de dialogue contenant une ou deux boîtes d'édition, une "list box' et deux boutons de confirmation et d'annulation.

Ecran 1

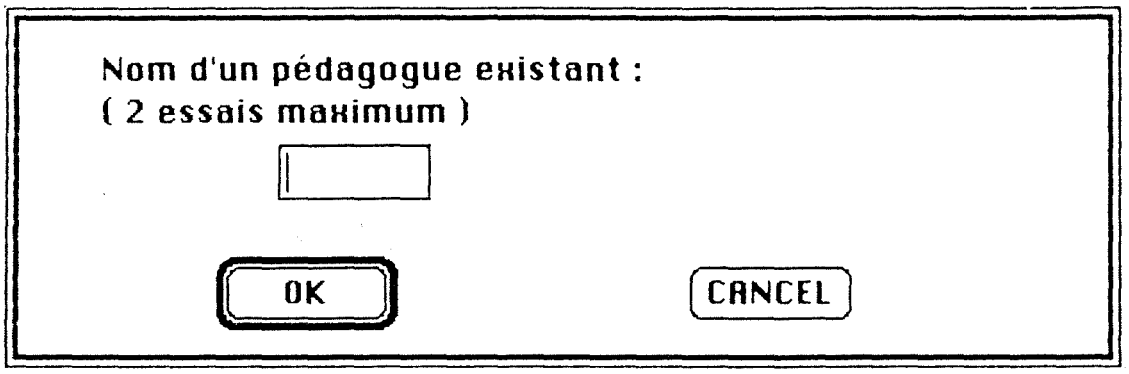
type 2

utilisation : dans la procédure InNomPedaExist

fonction : demande et saisit le nom d'un pédagogue existant

argument/résultat : le nom du pédagogue
un code d'erreur

contenu :



Ecran 2

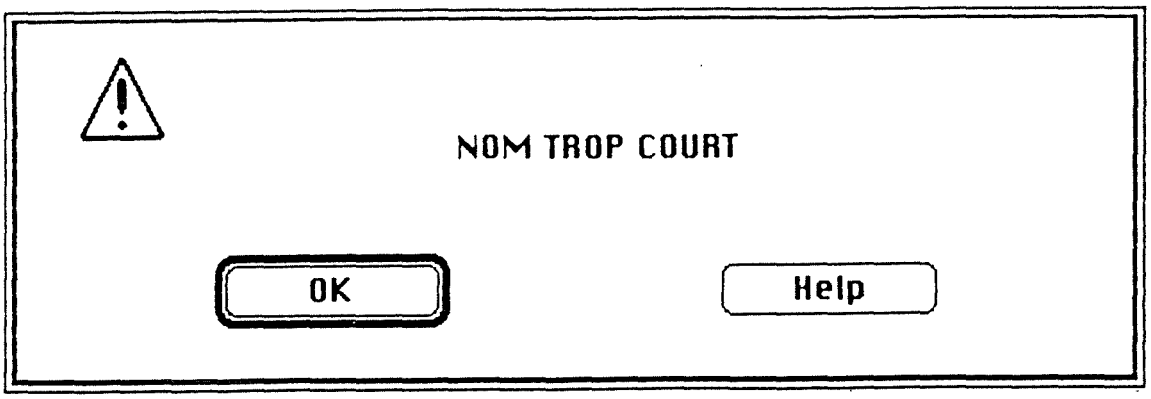
type 3

utilisation : dans les procédures InNomPedaExist, InNomST, InNomPedaInexist, InNomTestInexist et InNomMatInexist

fonction : signale à l'utilisateur que le nom introduit est trop court

argument/résultat : un code d'erreur

contenu :



Ecran 2.1

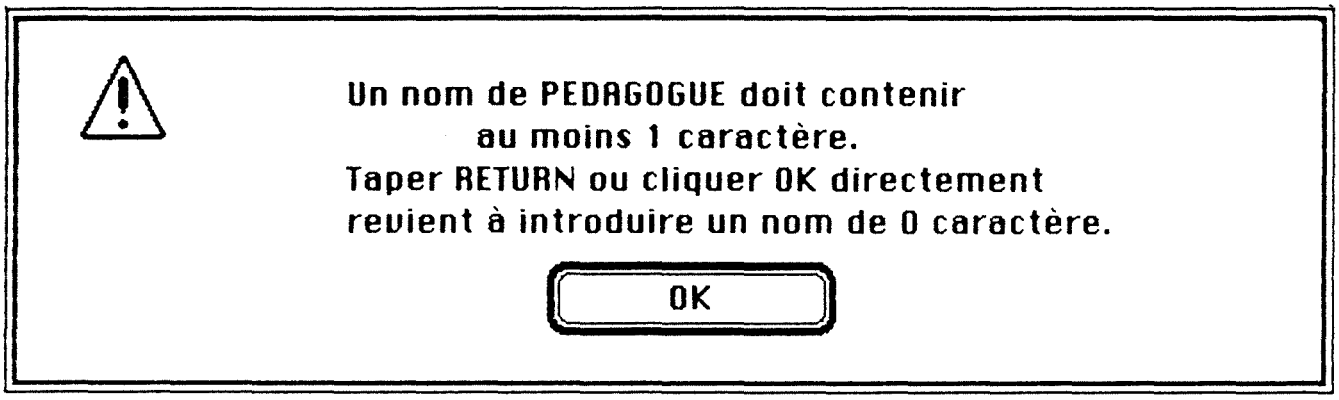
type 3

utilisation : dans les procédures InNomPedaExist et InNomPedaInexist

fonction : détaille l'explication donnée par l'écran 2 dans le cas où le nom est un nom de pédagogue

argument/résultat :

contenu :



Ecran 2.2

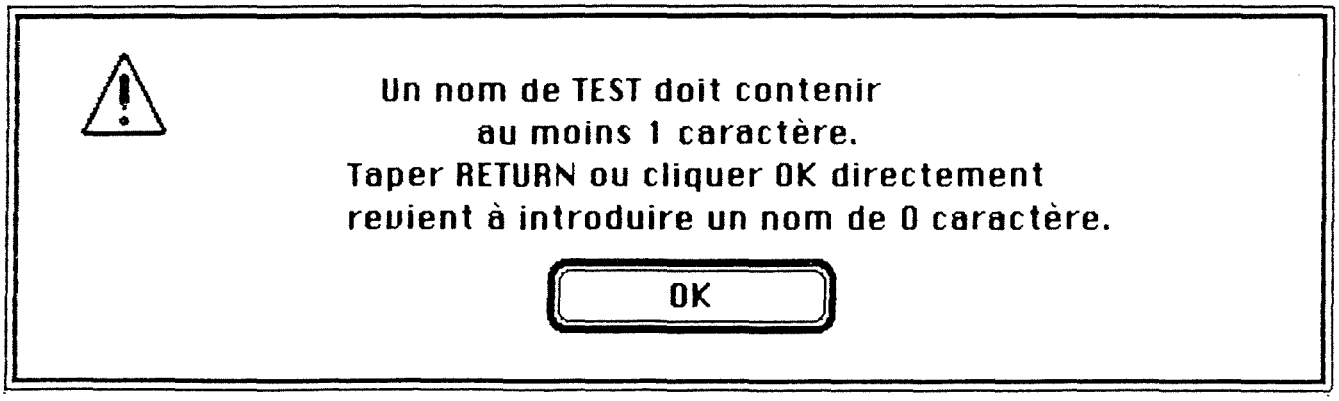
type 3

utilisation : dans la procédure InNomTestInexist

fonction : détaille l'explication donnée par l'écran 2 dans le cas où le nom est un nom de test

argument/résultat :

contenu :



Ecran 2.3

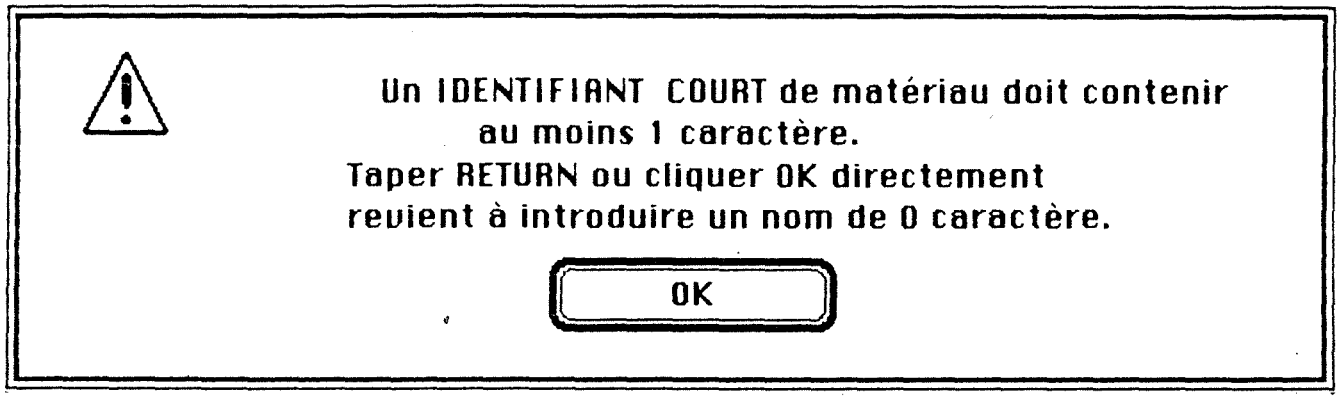
type 3

utilisation : dans la procédure InNomMatInexist

fonction : détaille l'explication donnée par l'écran 2 dans le cas où le nom est un identifiant court de matériau

argument/résultat :

contenu :



Ecran 2.4

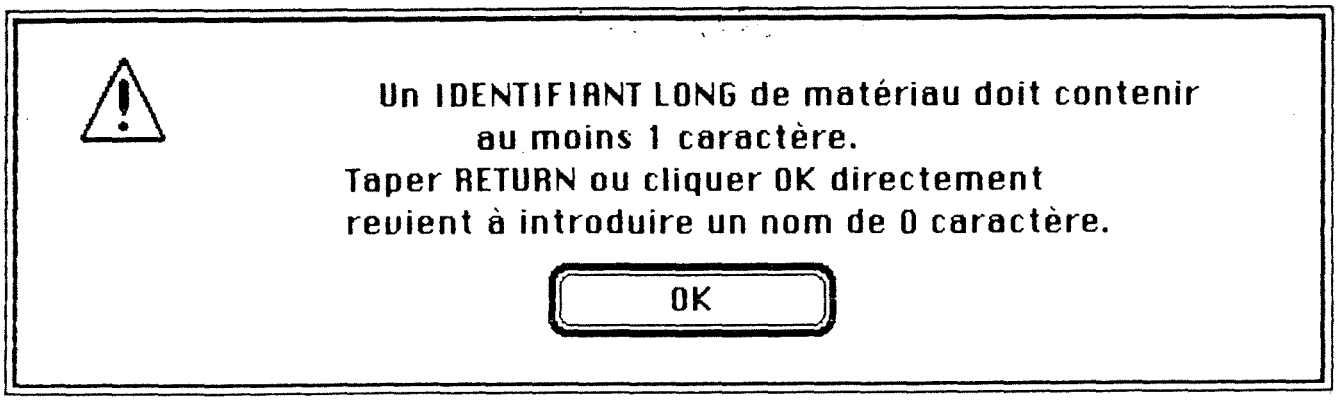
type 3

utilisation : dans la procédure InNomMatInexist

fonction : détaille l'explication donnée par l'écran 2 dans le cas où le nom est un identifiant long de matériau

argument/résultat :

contenu :



Ecran 2.5

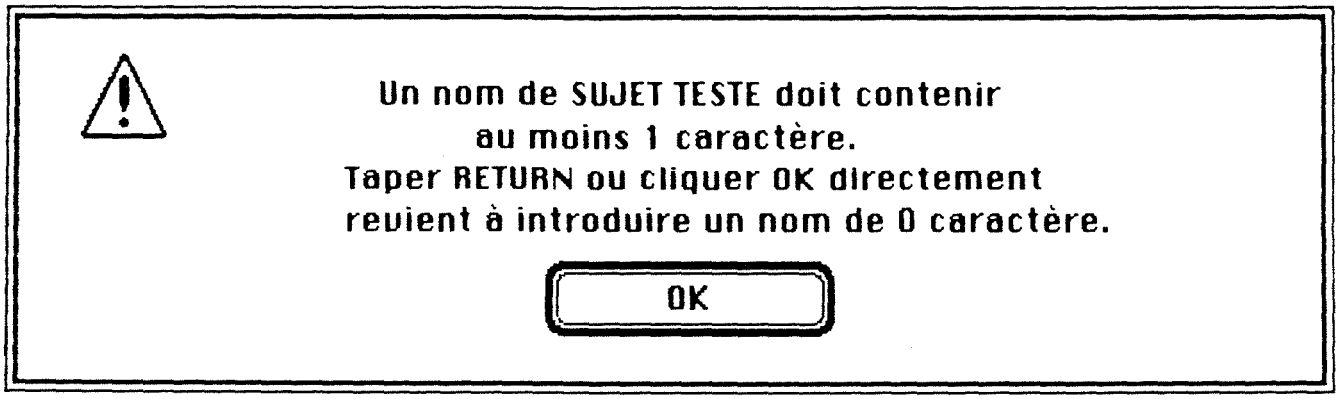
type 3

utilisation : dans la procédure InNomST

fonction : détaille l'explication donnée par l'écran 2 dans le cas où le nom est un nom de sujet testé

argument/résultat :

contenu :



Ecran 2.6

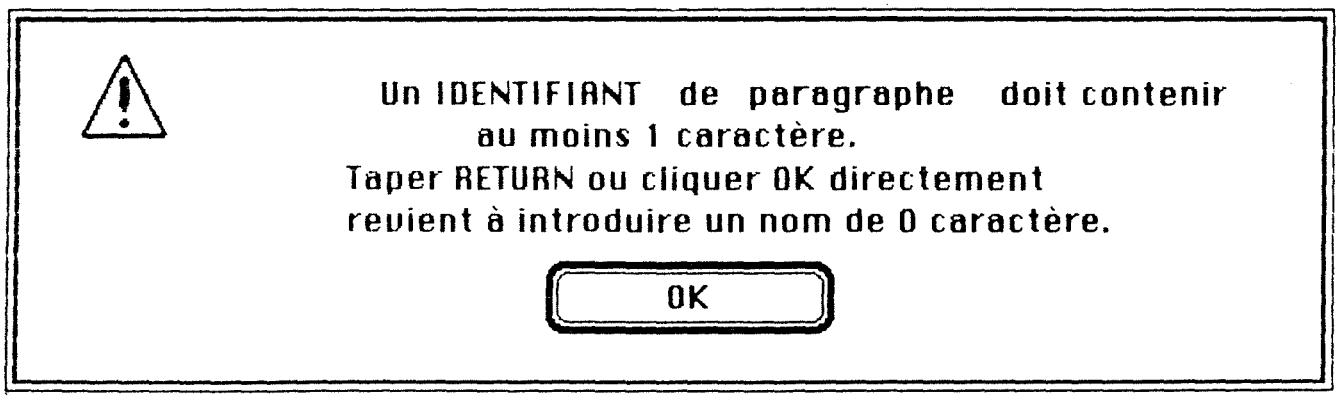
type 3

utilisation : dans la procédure DoAddSTPar

fonction : détaille l'explication donnée par l'écran 2 dans le cas où le nom est un identifiant de paragraphe

argument/résultat :

contenu :



Ecran 3

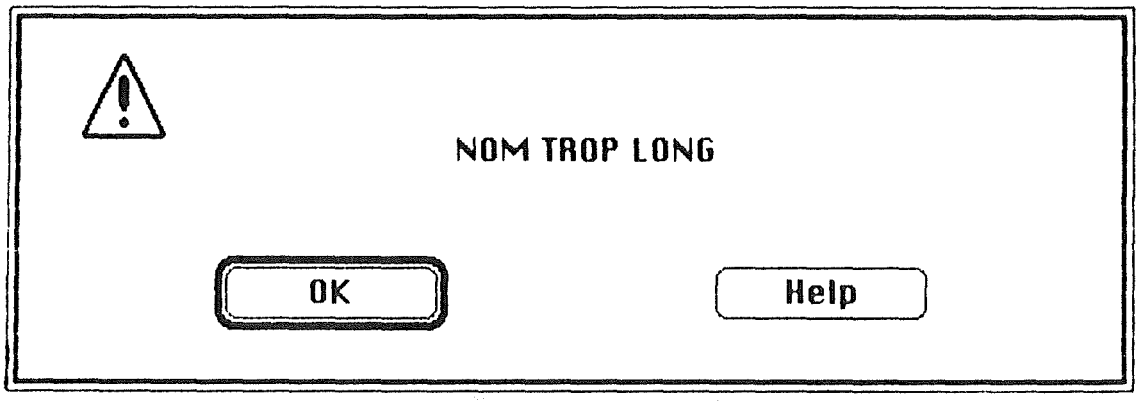
type 3

utilisation : dans les procédures InNomPedaExist, InNomST, InNomPedaInexist, InNomTestInexist et InNomMatInexist

fonction : signale à l'utilisateur que le nom introduit est trop long

argument/résultat : un code d'erreur

contenu :



Ecran 3.1

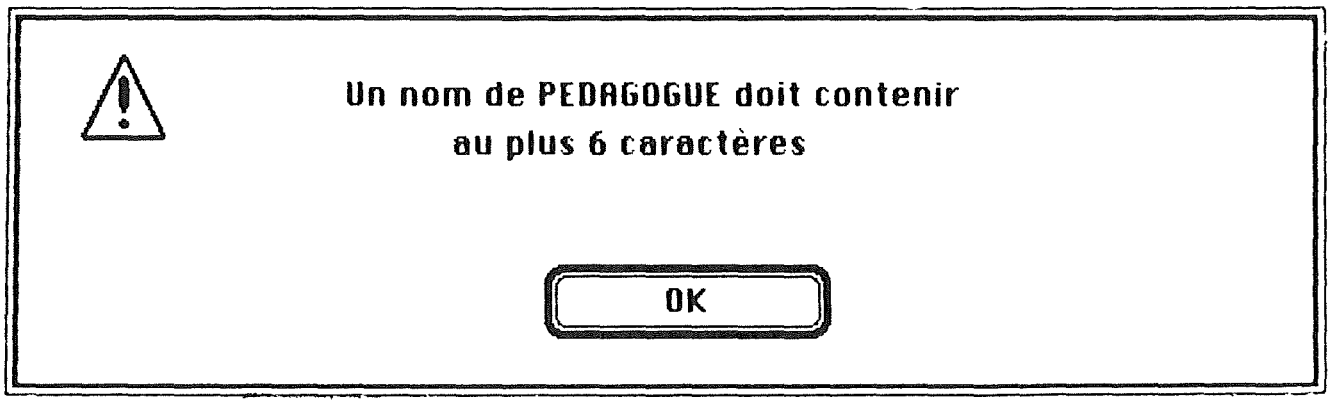
type 3

utilisation : dans les procédures InNomPedaExist et InNomPedaInexist

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un nom de pédagogue

argument/résultat :

contenu :



Ecran 3.2

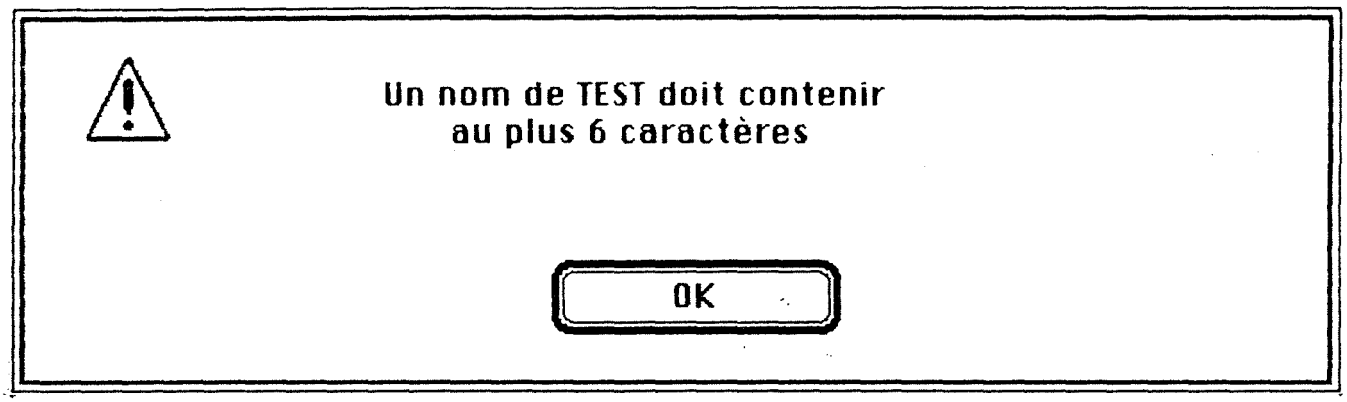
type 3

utilisation : dans la procédure InNomTestInexist

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un nom de test

argument/résultat :

contenu :



Ecran 3.3

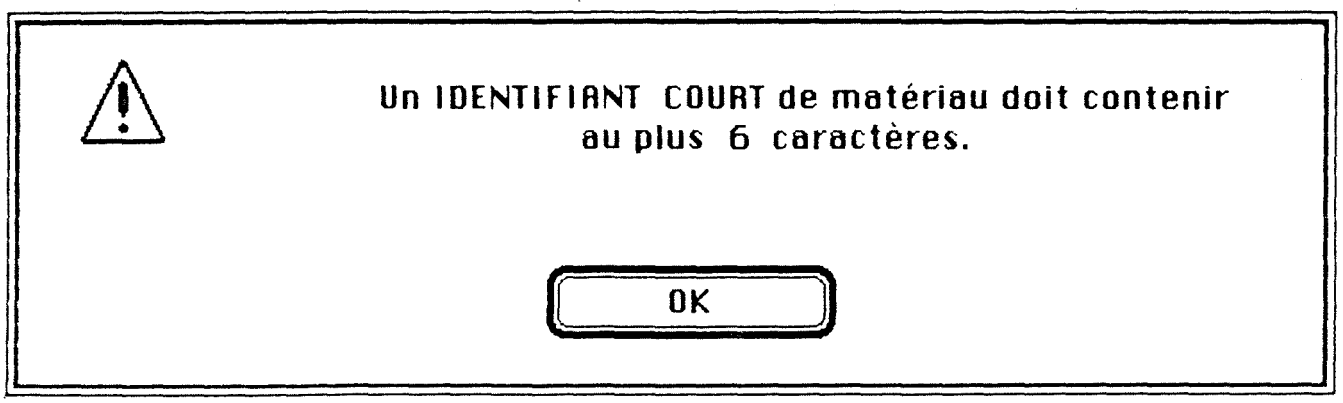
type 3

utilisation : dans la procédure InNomMatInexist

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un identifiant court de matériau

argument/résultat :

contenu :



Ecran 3.4

type 3

utilisation : dans la procédure InNomMatInexist

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un identifiant long de matériau

argument/résultat :

contenu :

Ecran 3.5

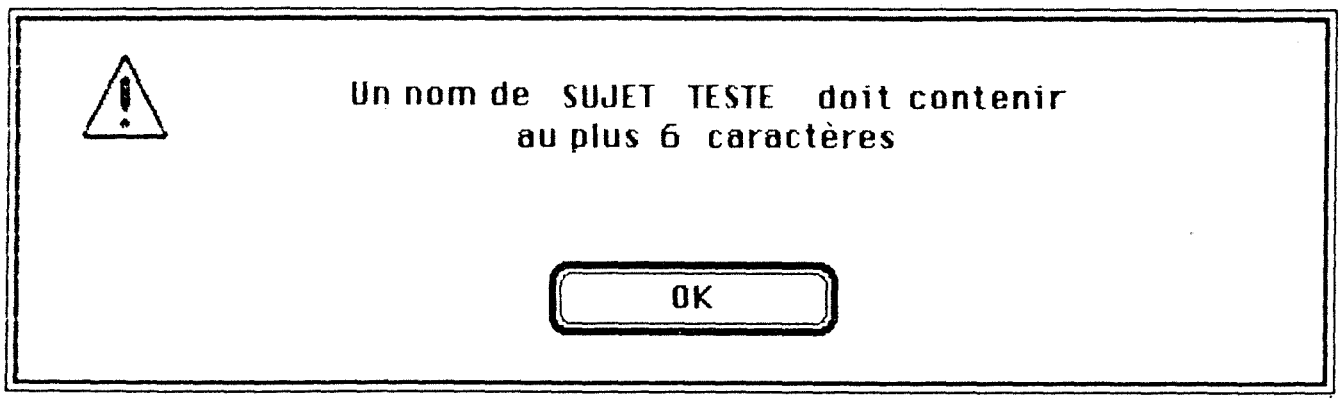
type 3

utilisation : dans la procédure InNomST

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un nom de sujet testé

argument/résultat :

contenu :



Ecran 3.6

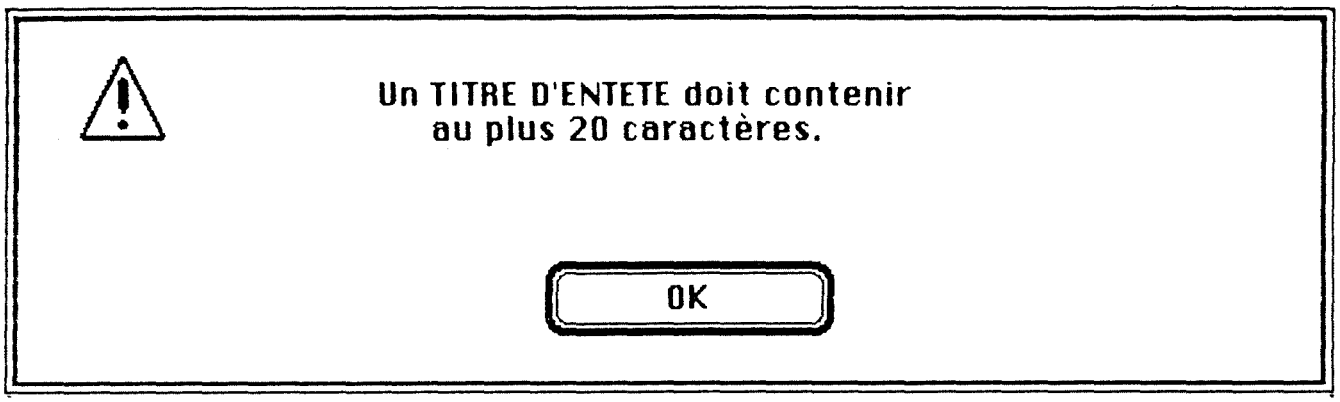
type 3

utilisation : dans la procédure DoIntrotexte

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un titre d'entête

argument/résultat :

contenu :



Ecran 3.7

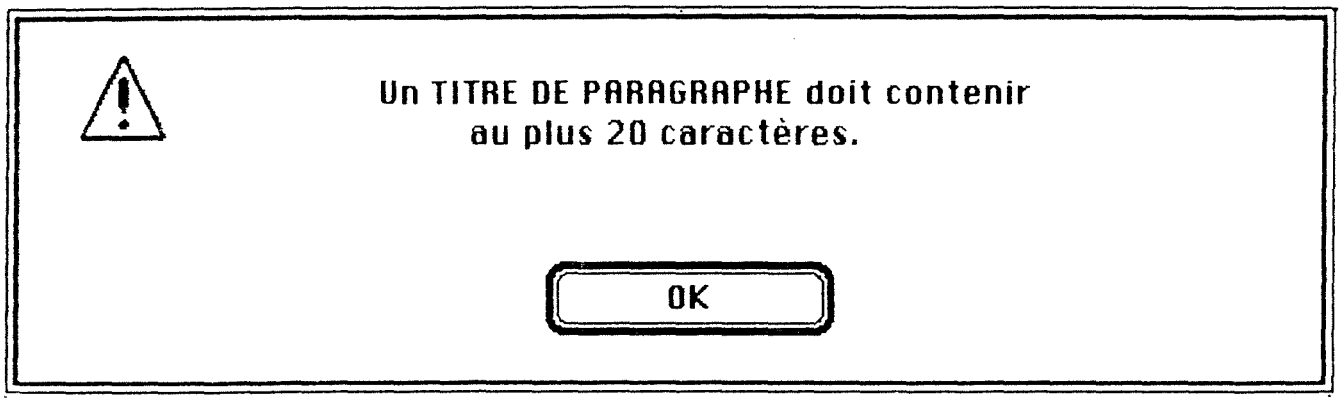
type 3

utilisation : dans la procédure DoIntrotexte

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un titre de paragraphe

argument/résultat :

contenu :



Ecran 3.8

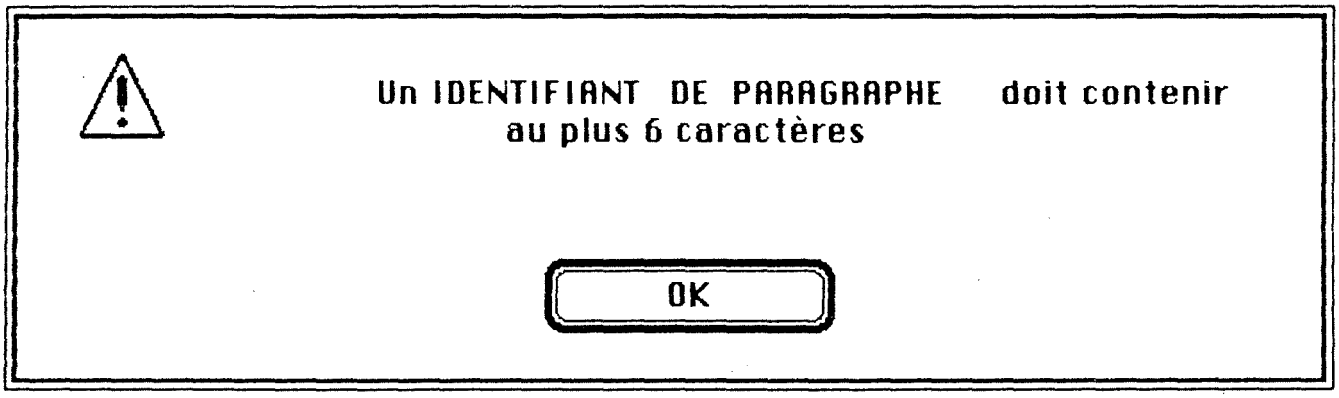
type 3

utilisation : dans la procédure DoIntrotexte

fonction : détaille l'explication donnée par l'écran 3 dans le cas où le nom est un identifiant de paragraphe

argument/résultat :

contenu :



Ecran 4

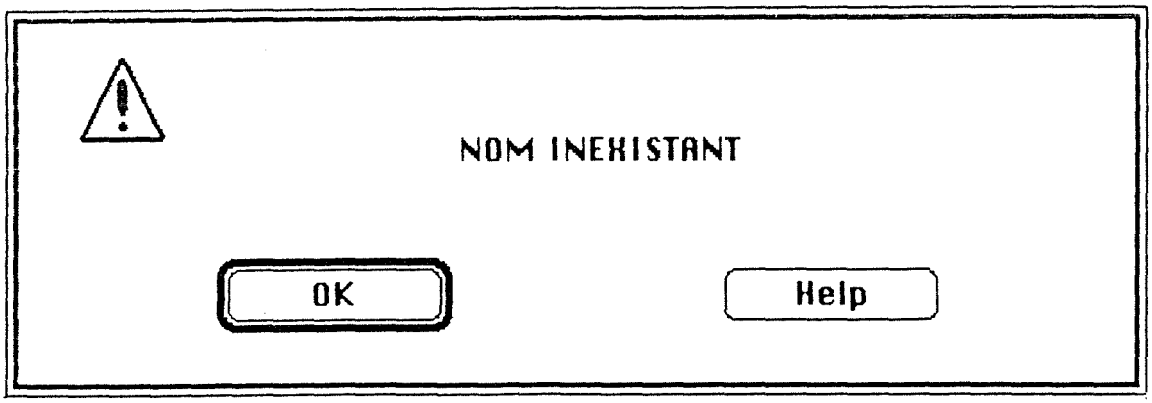
type 3

utilisation : dans la procédure InNomPedaExist

fonction : signale à l'utilisateur que le nom introduit n'existe pas

argument/résultat : un code d'erreur

contenu :



Ecran 4.1

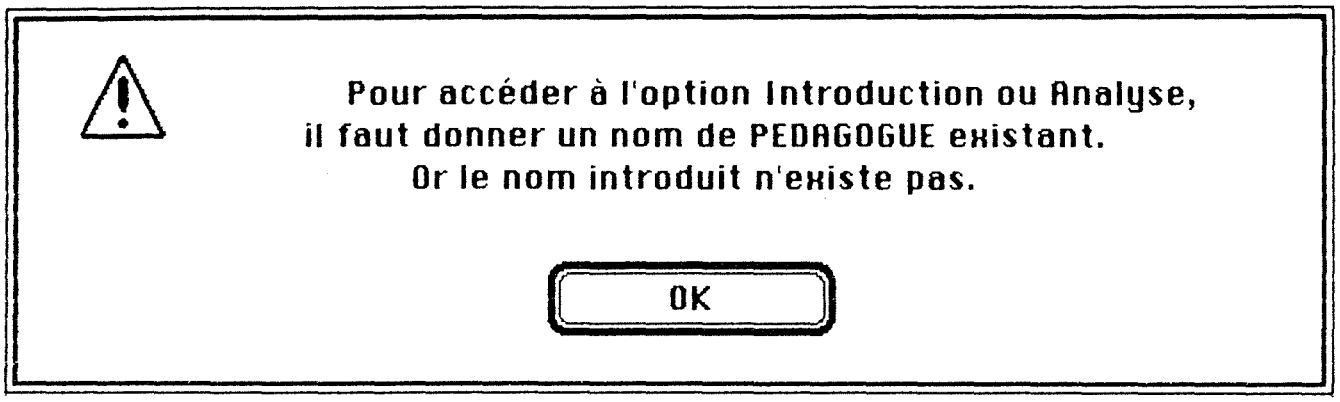
type 3

utilisation : dans la procédure InNomPedaExist

fonction : détaille l'explication donnée par l'écran 4 dans le cas où le nom est un nom de pédagogue

argument/résultat :

contenu :



Ecran 5

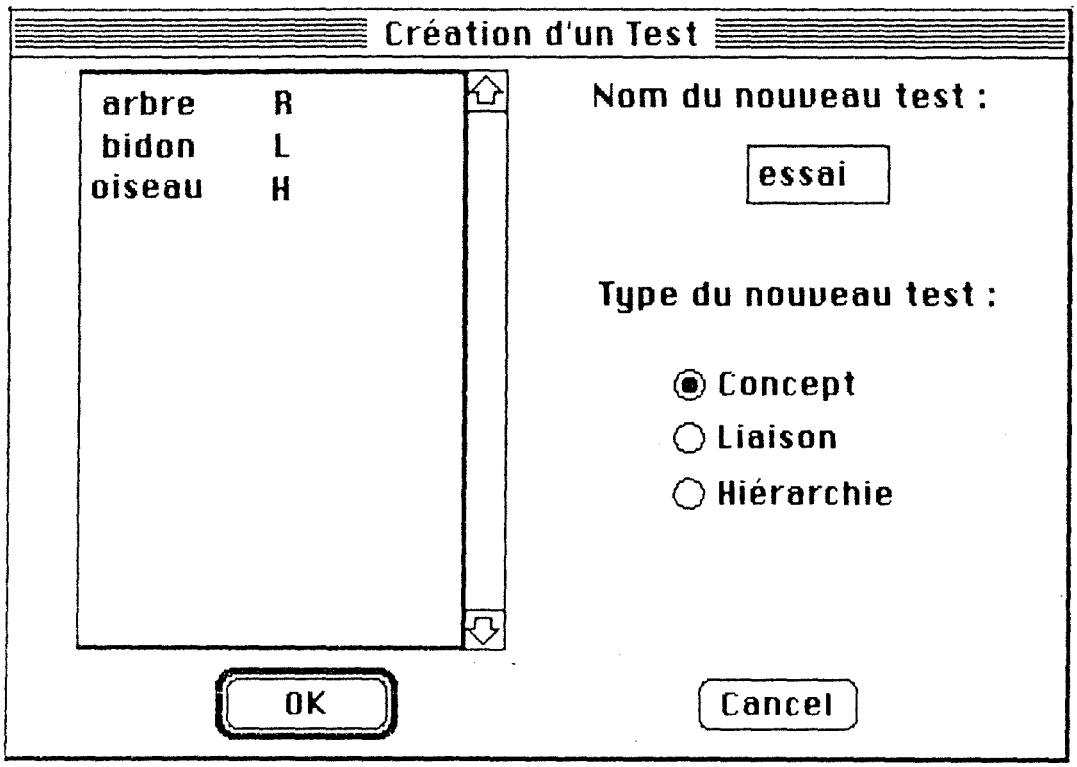
type 7

utilisation : dans la procédure InNomTestExist

fonction : permet à l'utilisateur de consulter la liste des tests existants, d'introduire un nom de test inexistant et un type de test

argument/résultat : la liste des tests existants
le nom du nouveau test
le type du nouveau test
un code d'erreur

contenu :



Ecran 7

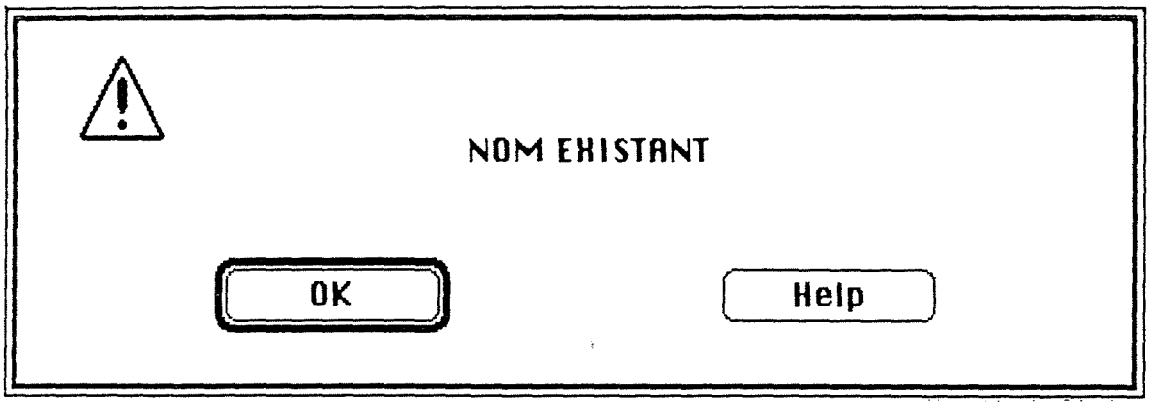
type 3

utilisation : dans les procédures InNomPedaInexist, InNomST, InNomTestInexist et InNomMatInexist

fonction : signale à l'utilisateur que le nom introduit existe déjà

argument/résultat : un code d'erreur

contenu :



Ecran 7.1

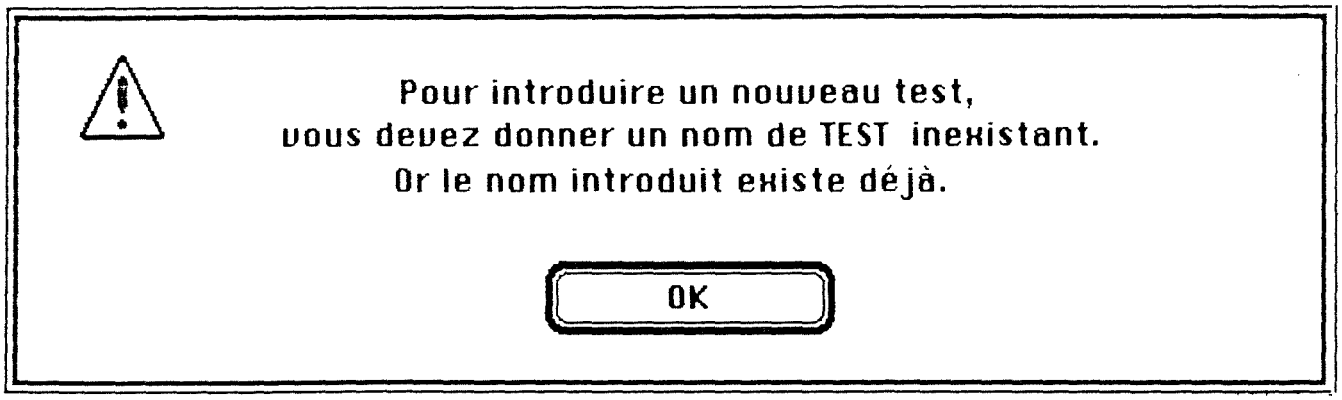
type 3

utilisation : dans la procédure InNomTestExist

fonction : détaille l'explication donnée par l'écran 7 dans le cas où le nom est un nom de test

argument/résultat :

contenu :



Ecran 7.2

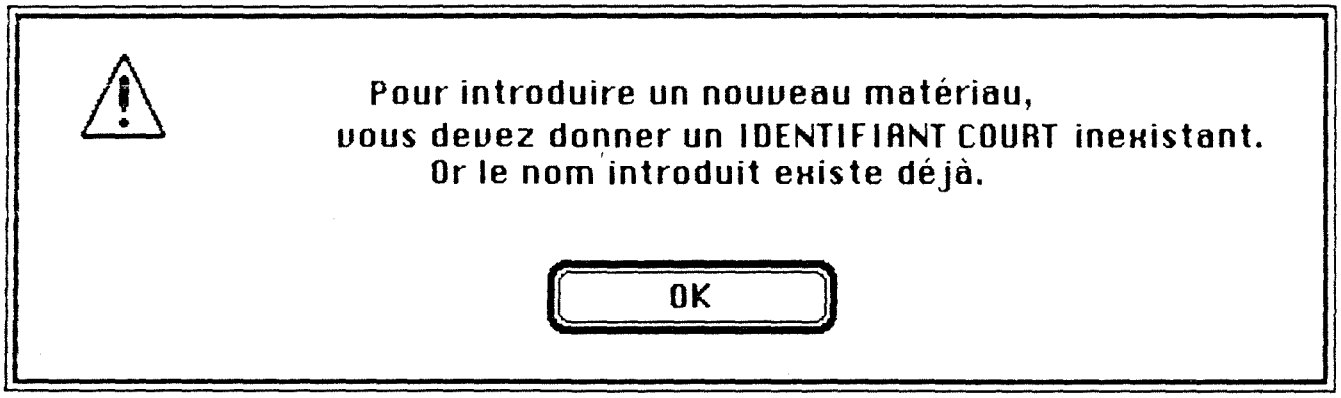
type 3

utilisation : dans la procédure InNomMatInexist

fonction : détaille l'explication donnée par l'écran 7 dans le cas où le nom est un identifiant court de paragraphe

argument/résultat :

contenu :



Ecran 7.3

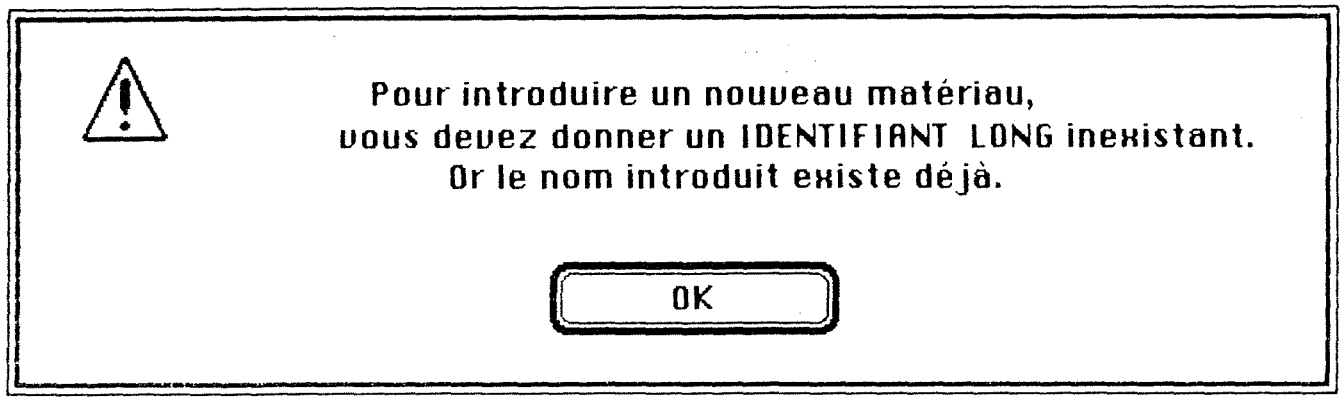
type 3

utilisation : dans la procédure InNomMatInexist

fonction : détaille l'explication donnée par l'écran 7 dans le cas où le nom est un identifiant long de matériau

argument/résultat :

contenu :



Ecran 7.5

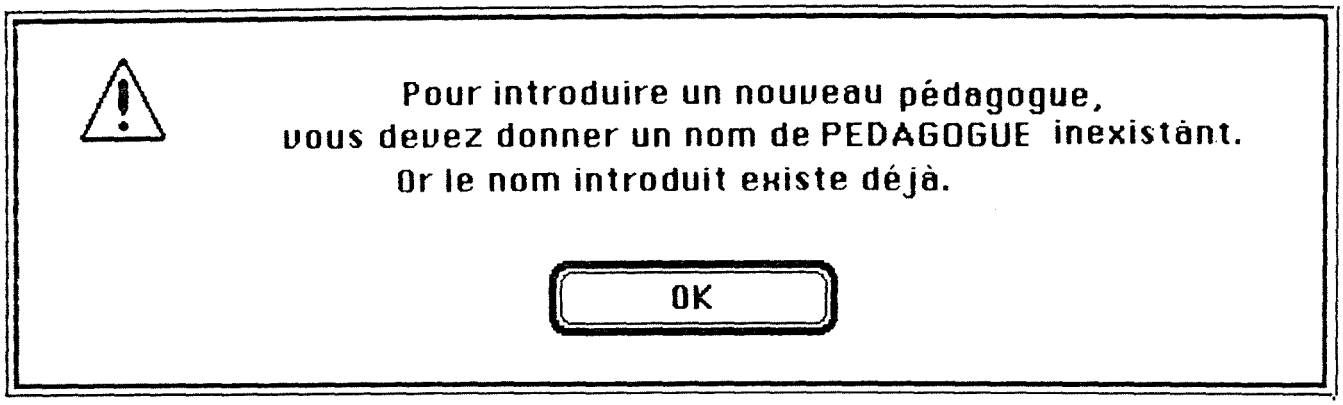
type 3

utilisation : dans la procédure InNomPedaInexist

fonction : détaille l'explication donnée par l'écran 7 dans le cas où le nom est un nom de pédagogue

argument/résultat :

contenu :



Ecran 7.6

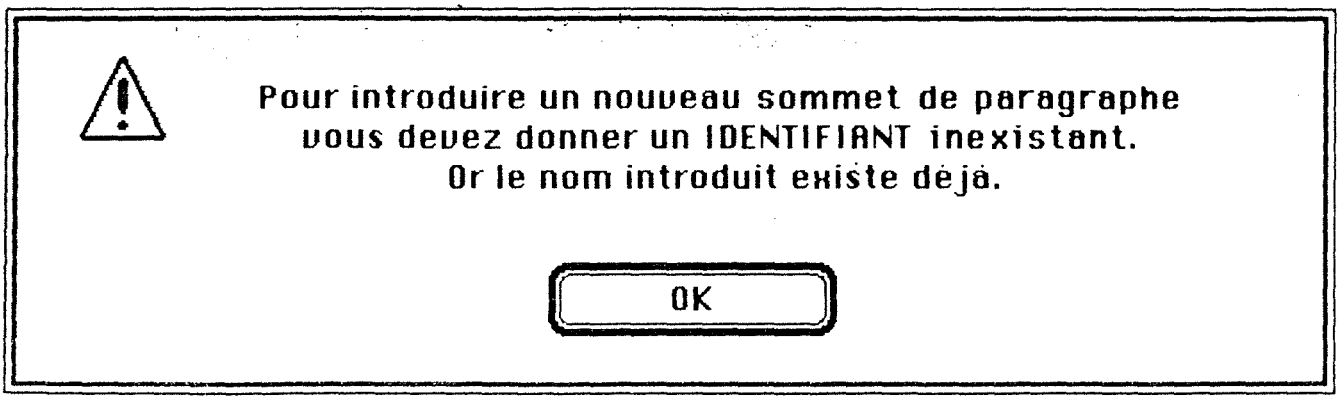
type 3

utilisation : dans la procédure AddSTPar

fonction : détaille l'explication donnée par l'écran 7 dans le cas où le nom est un identifiant de paragraphe

argument/résultat :

contenu :



Ecran 8

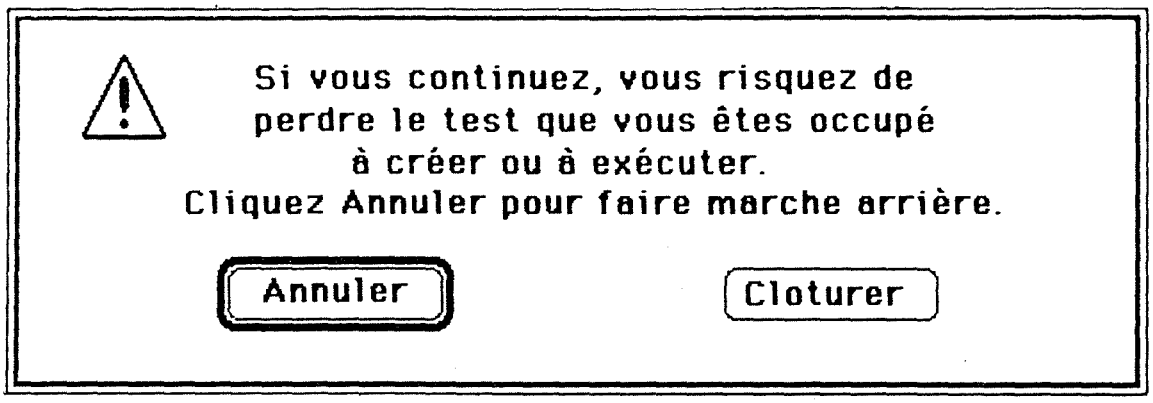
type 4

utilisation : dans la procédure ProcessinMenu

fonction : signale à l'utilisateur qu'il risque de perdre le test qu'il est occupé à créer ou à exécuter et lui demande s'il veut poursuivre l'action ou l'annuler

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 9

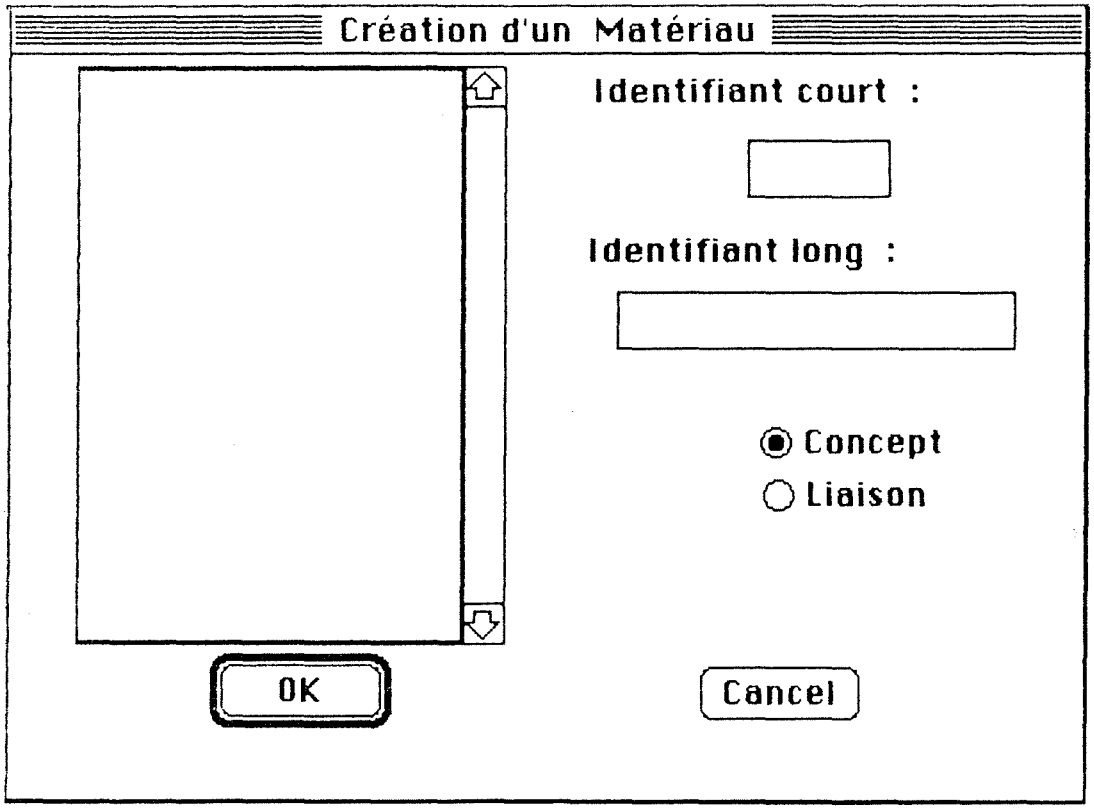
type 7

utilisation : dans la procédure InNomMatInExist, pour un test de type réseau

fonction : permet à l'utilisateur de consulter la liste des matériaux existants, d'introduire un identifiant long, un identifiant court et un type de matériau

argument/résultat : la liste des matériaux existants
l'identifiant long du nouveau matériau
l'identifiant court du nouveau matériau
le type du nouveau matériau
un code d'erreur

contenu :



Ecran 10

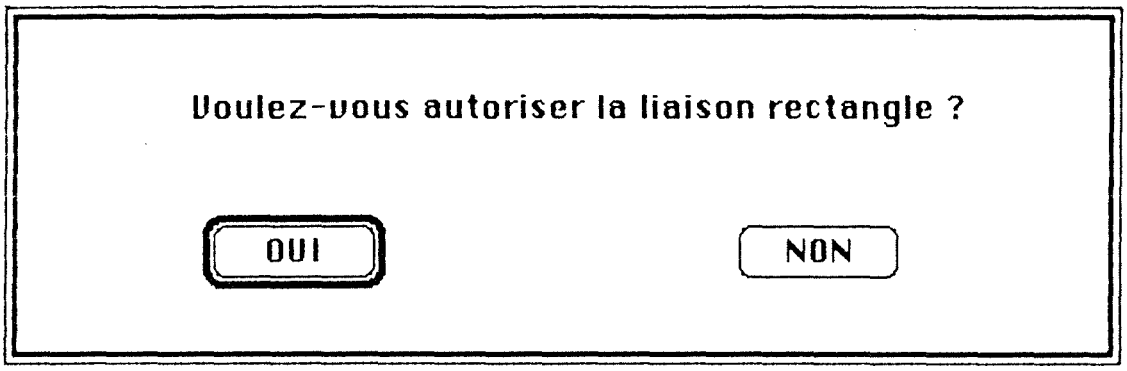
type 4

utilisation : dans la procédure InMatInexist, pour un test de type réseau

fonction : demande à l'utilisateur s'il souhaite autoriser la liaison rectangle dans les matériaux de test

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 11

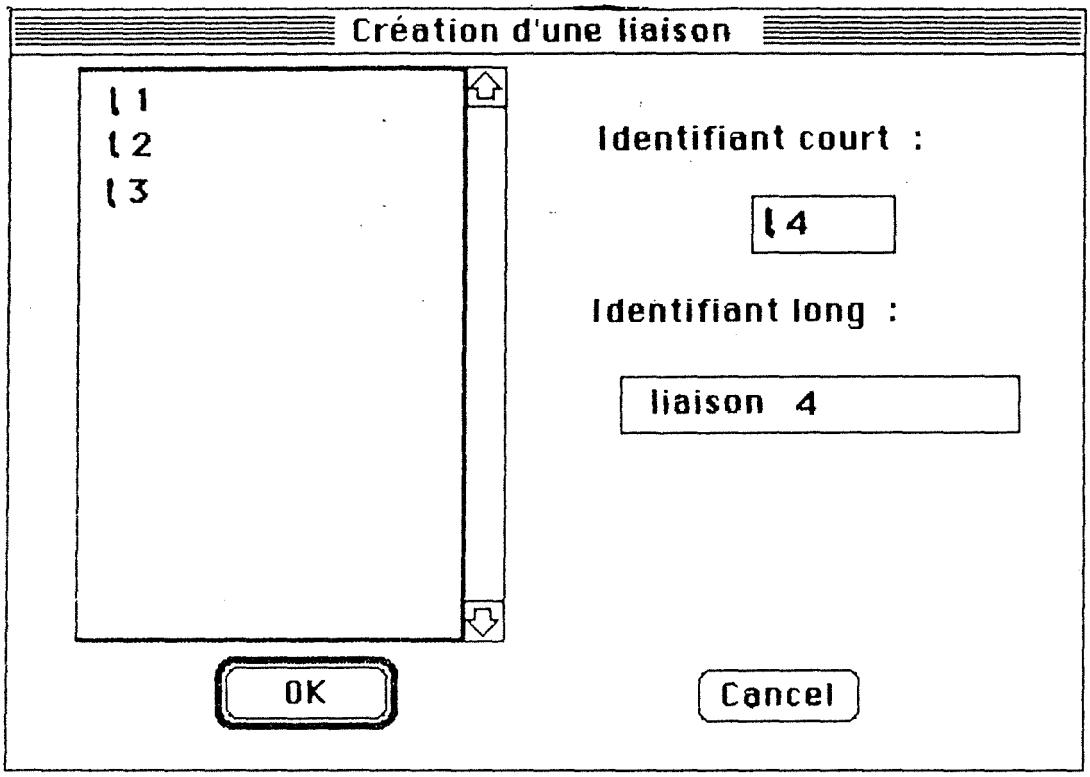
type 7

utilisation : dans la procédure InNomMatInExist, pour un test de type liaison

fonction : permet à l'utilisateur de consulter la liste des matériaux existants, d'introduire un identifiant long et un identifiant court de matériau

argument/résultat : la liste des matériaux existants
l'identifiant long du nouveau matériau
l'identifiant court du nouveau matériau
un code d'erreur

contenu :



Ecran 12

type 3

utilisation : dans la procédure DoIntrotexte

fonction : indique à l'utilisateur qu'une entête doit avoir un contenu si elle a un titre et lui propose diverses solutions

argument/résultat : un code d'erreur

contenu :

Ecran 13

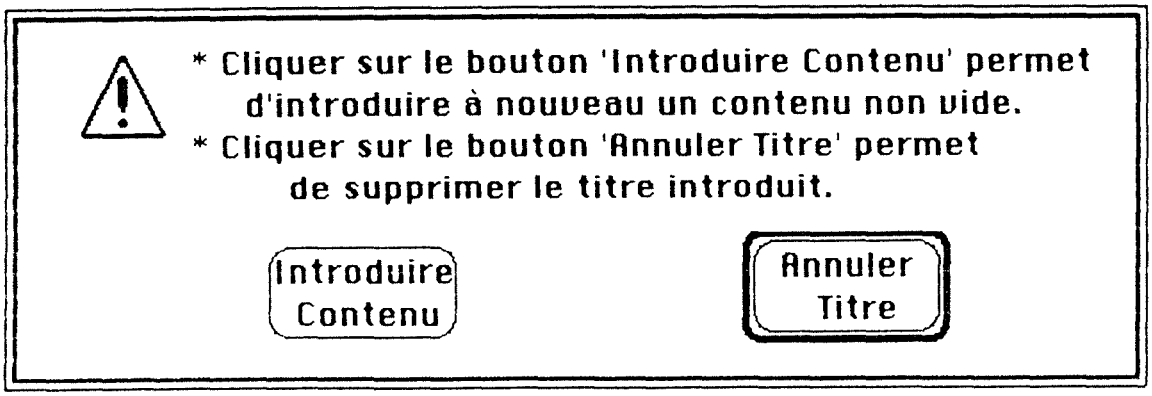
type 3

utilisation : dans la procédure DoIntrotexte

fonction : détaille l'explication donnée par l'écran 12

argument/résultat : un code d'erreur

contenu :



Ecran 14

type 3

utilisation : dans la procédure DoIntrotexte

fonction : indique à l'utilisateur qu'un paragraphe doit avoir un contenu si il a un titre et lui propose diverses solutions

argument/résultat : un code d'erreur

contenu :

Ecran 15

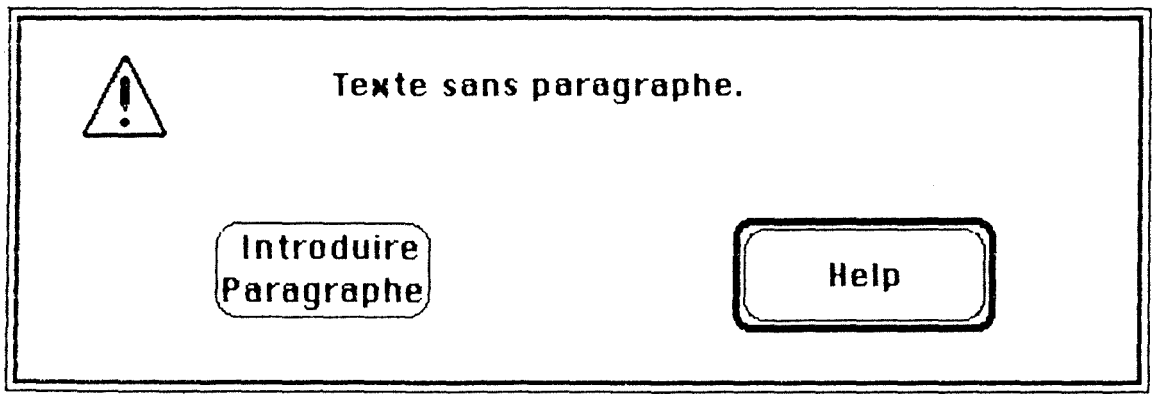
type 3

utilisation : dans la procédure DoIntrotexte

fonction : indique à l'utilisateur que son texte contient un titre mais aucun paragraphe et lui propose diverses solutions relatives à un texte obligatoire

argument/résultat : un code d'erreur

contenu :



Ecran 16

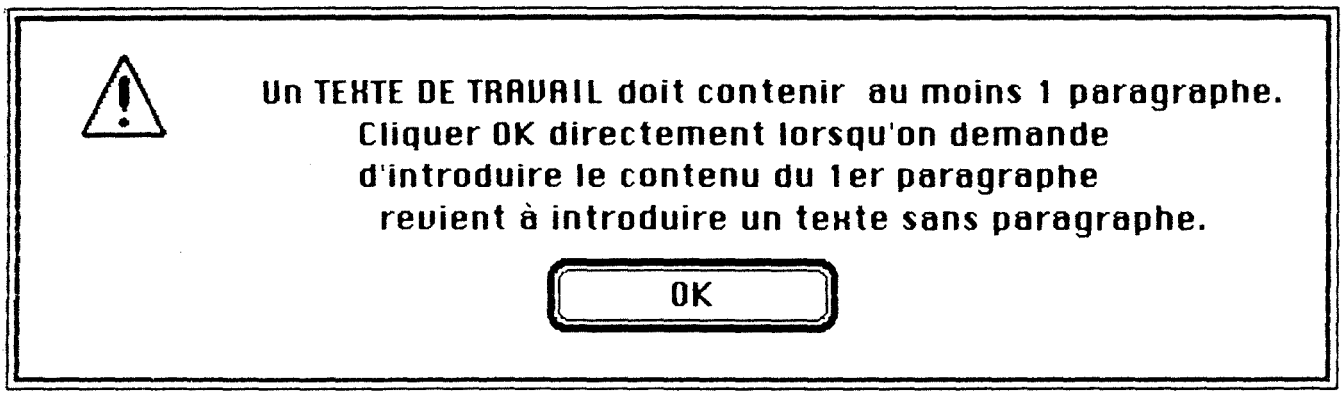
type 3

utilisation : dans la procédure DoIntrotexte

fonction : détaille l'explication donnée par l'écran 15

argument/résultat : un code d'erreur

contenu :



Ecran 17

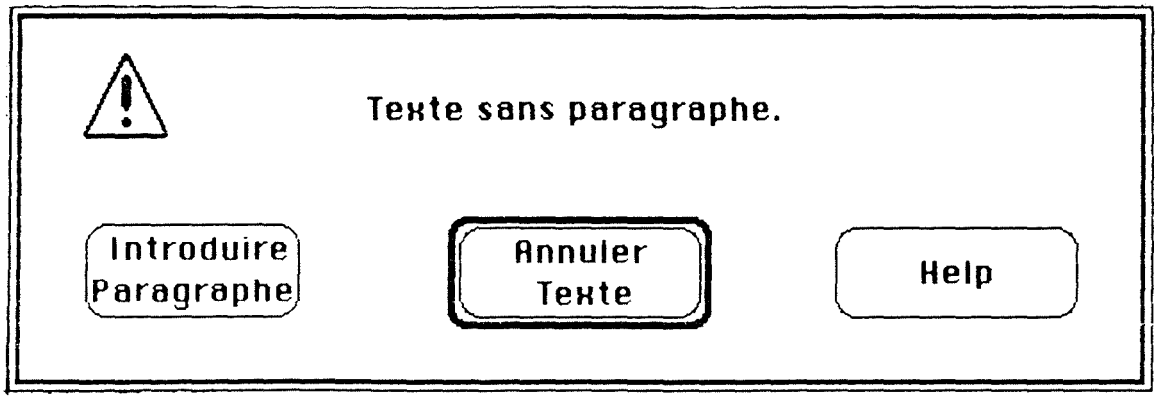
type 3

utilisation : dans la procédure DoIntrotexte

fonction : indique à l'utilisateur que son texte contient un titre mais aucun paragraphe et lui propose diverses solutions relatives à un texte facultatif

argument/résultat : un code d'erreur

contenu :



Ecran 18

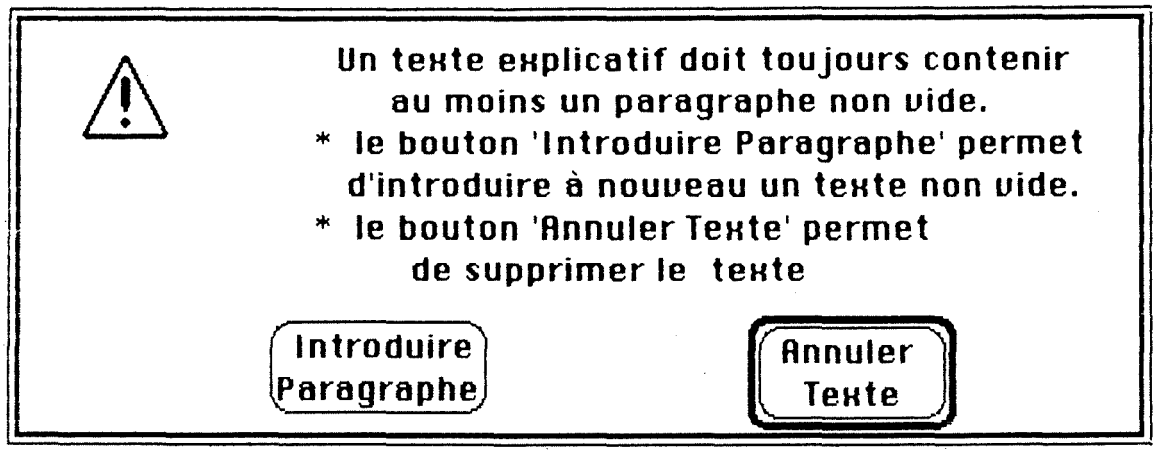
type 3

utilisation : dans la procédure DoIntrotexte

fonction : détaille l'explication donnée par l'écran 17, dans le cas où le texte est un texte explicatif

argument/résultat : un code d'erreur

contenu :



Ecran 19

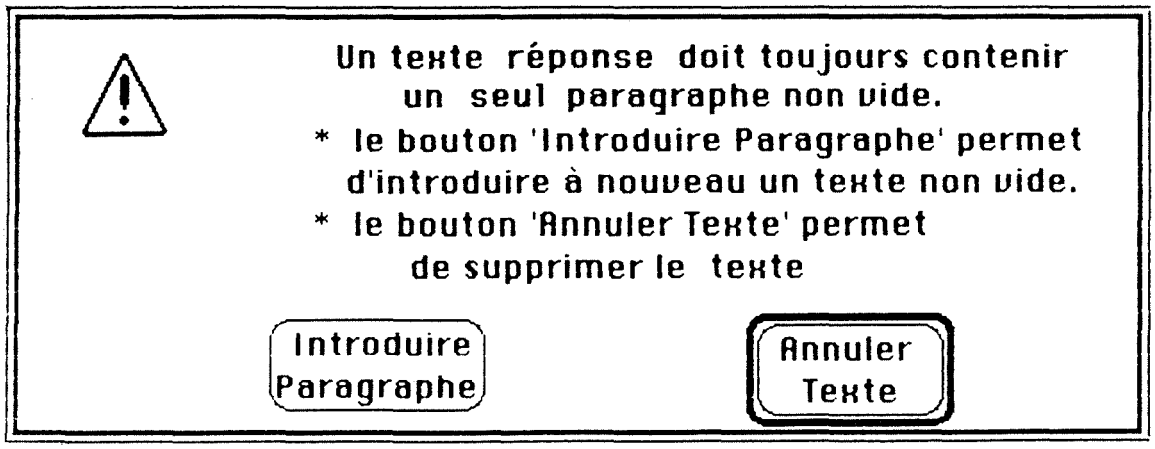
type 3

utilisation : dans la procédure DoIntrotexte

fonction: détaille l'explication donnée par l'écran 17, dans le cas où le texte est un texte réponse

argument/résultat : un code d'erreur

contenu :



Ecran 20

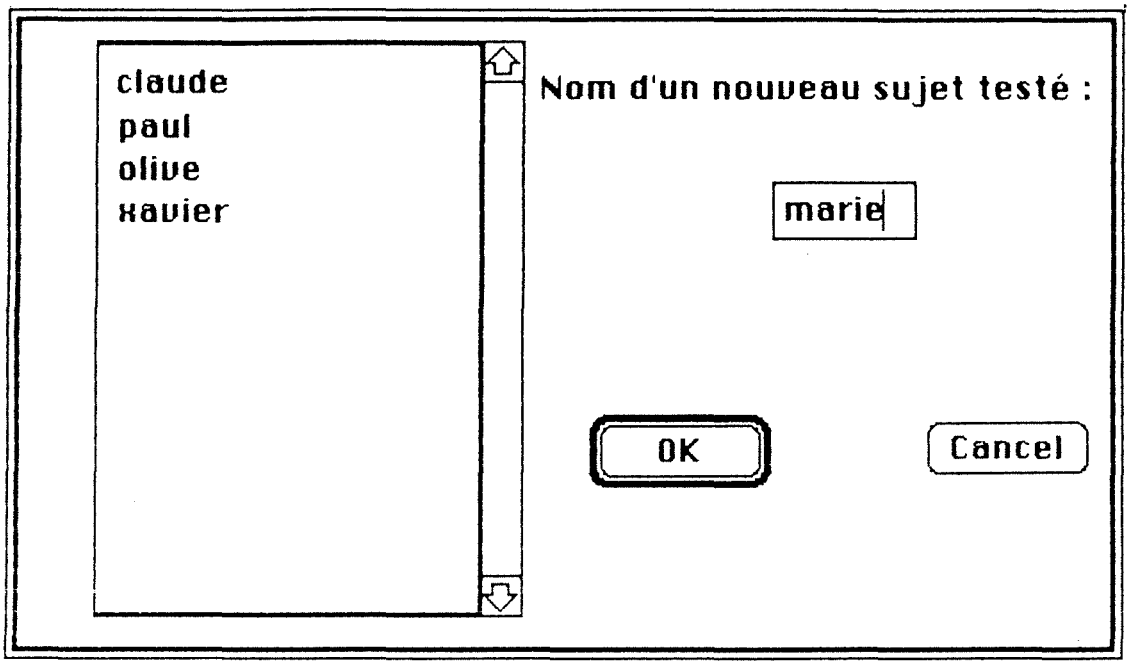
type 7

utilisation : dans la procédure InNomST

fonction : permet à l'utilisateur de consulter la liste des sujets testés existants, d'introduire ou de sélectionner un nom de sujet testé

argument/résultat : la liste des sujets testés existants
le nom du sujet testé
un code d'erreur

contenu :



Ecran 21

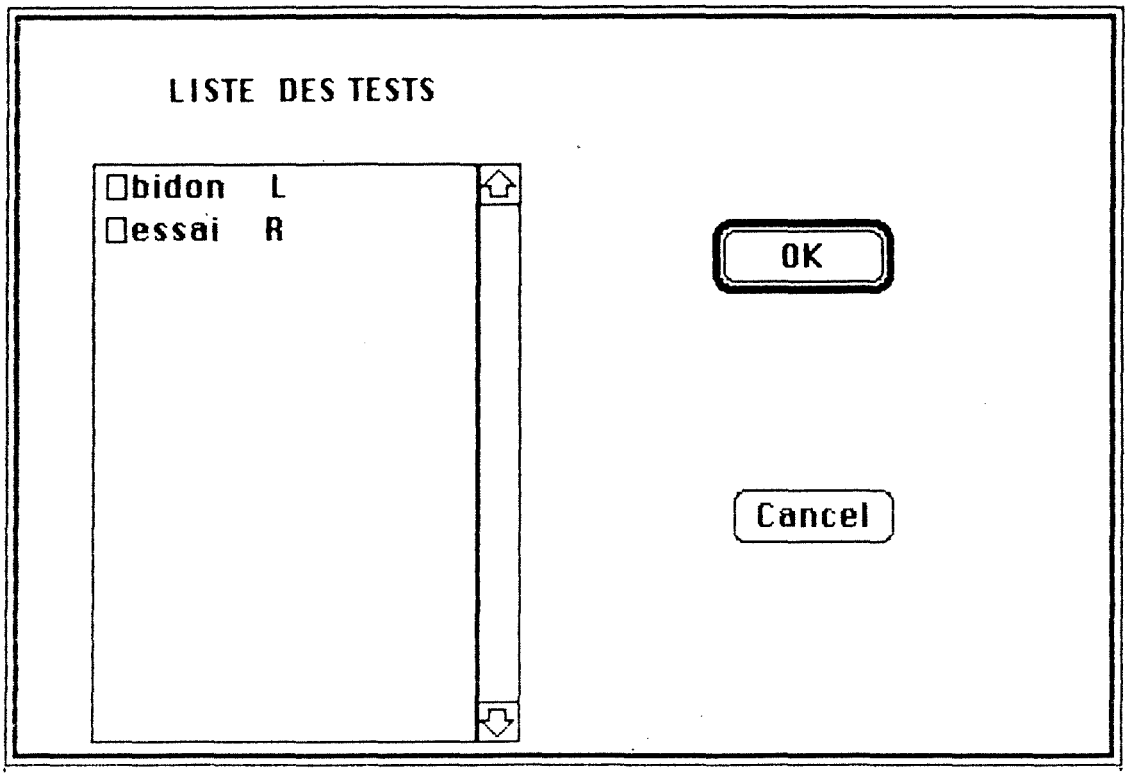
type 6

utilisation : dans les procédures InNomTestExist et ProcessInMenu

fonction : demande à l'utilisateur de sélectionner un nom de test dans la liste des tests existants qui lui est présentée

argument/résultat : la liste des tests existants
le nom du test
le type du test
un code d'erreur

contenu :



Ecran 22

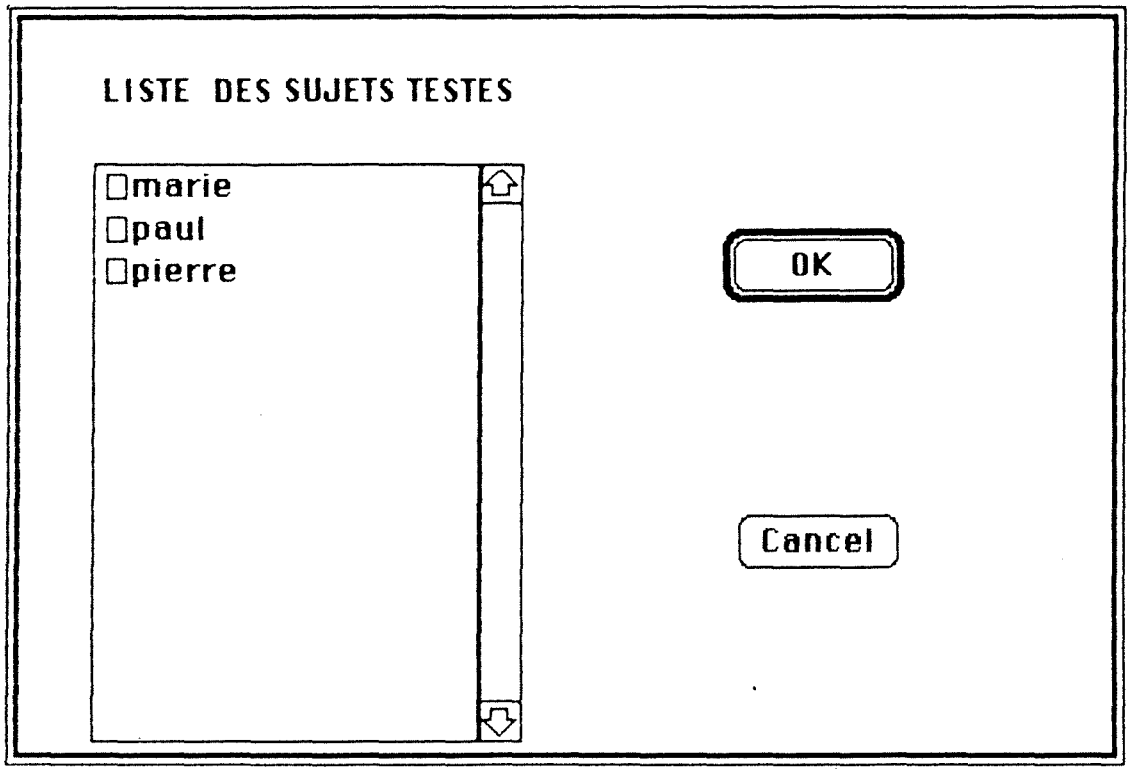
type 6

utilisation : dans les procédures InNomST et ProcessInMenu

fonction : demande à l'utilisateur de sélectionner un nom de sujet testé dans la liste des sujets testés existants qui lui est présentée

argument/résultat : la liste des sujets testés existants
le nom du sujet testé
un code d'erreur

contenu :



Ecran 23

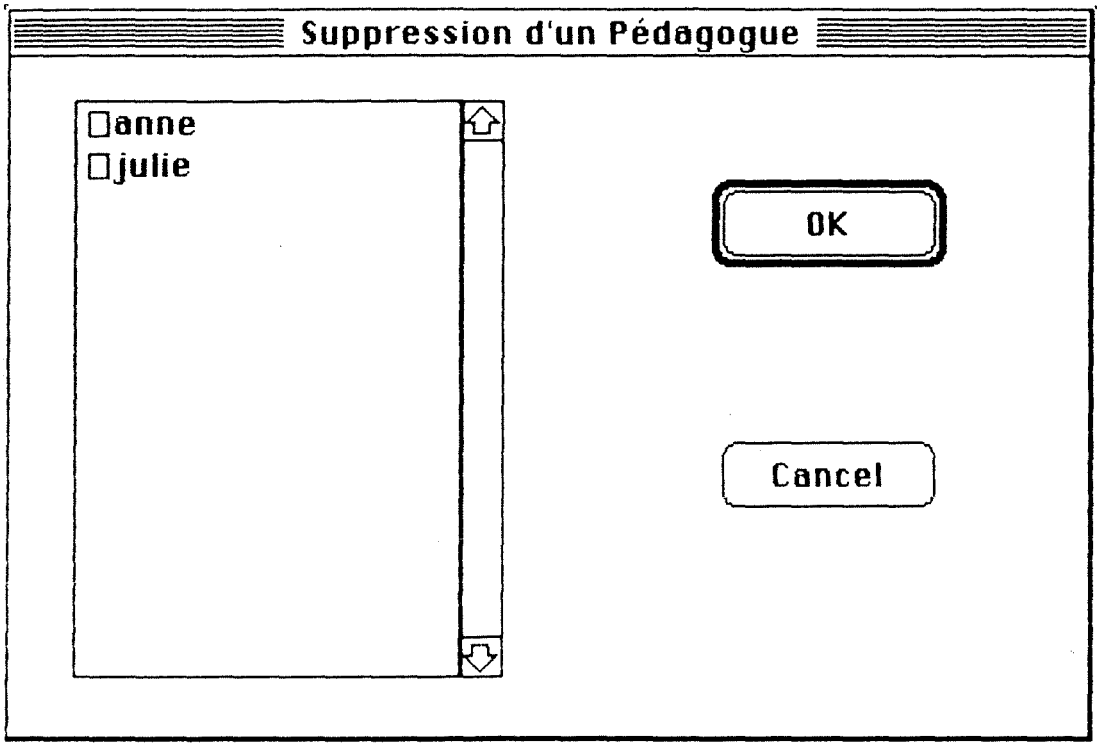
type 6

utilisation : dans les procédures InNomPedaExist et ProcessInMenu

fonction : demande à l'utilisateur de sélectionner un nom de pédagogue dans la liste des pédagogues existants qui lui est présentée

argument/résultat : la liste des pédagogues existants
le nom du pédagogue
un code d'erreur

contenu :



Ecran 24

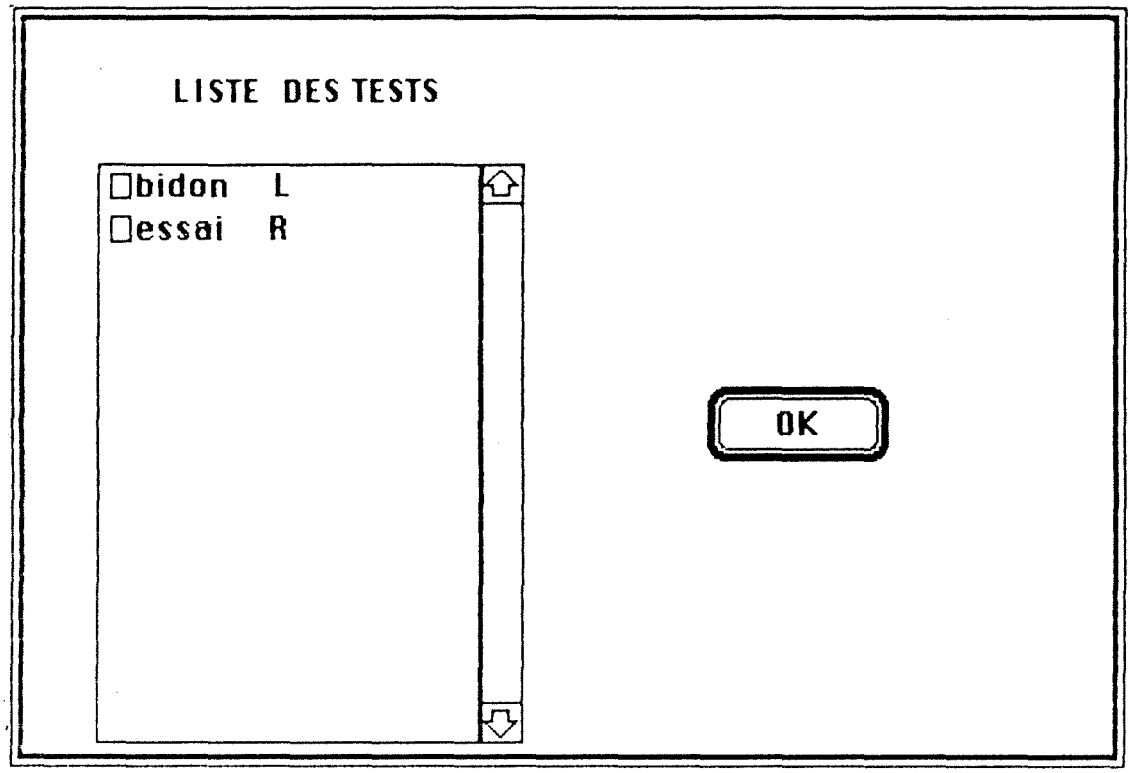
type 6

utilisation : dans la procédure ProcessInMenu

fonction : affiche une liste de tests

argument/résultat : la liste des tests

contenu :



Ecran 25

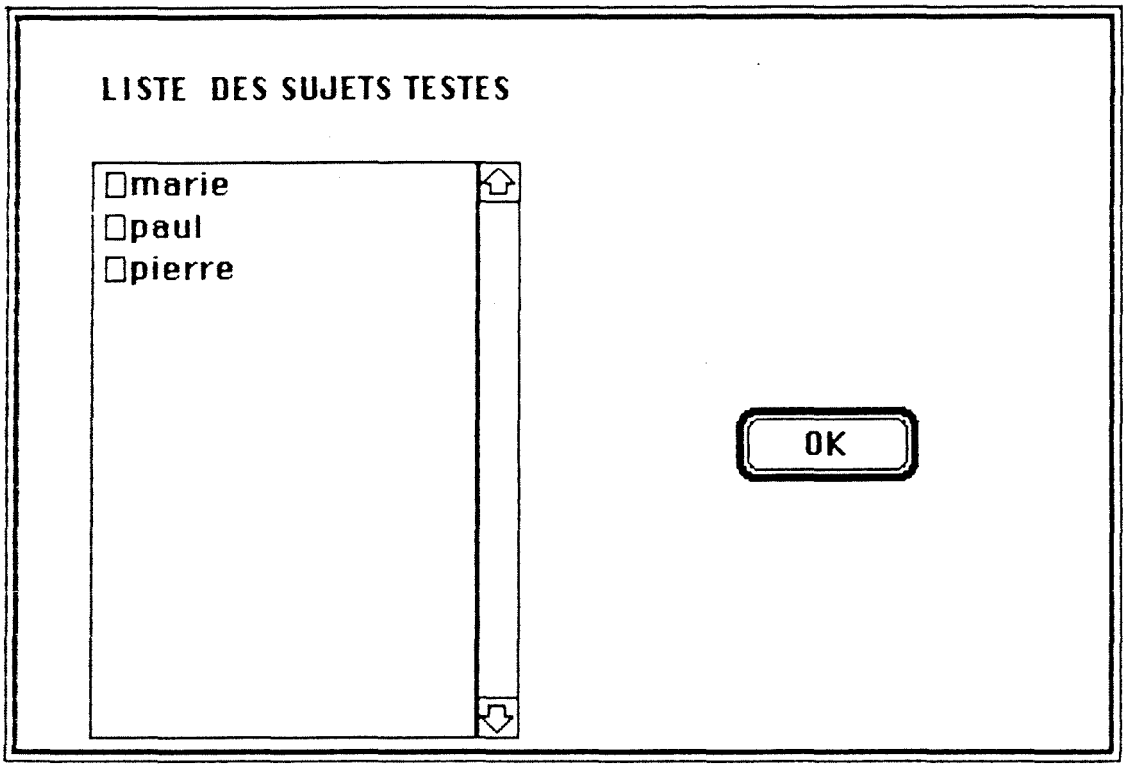
type 6

utilisation : dans la procédure ProcessInMenu

fonction : affiche une liste de sujets testés

argument/résultat : la liste des sujets testés

contenu :



Ecran 28

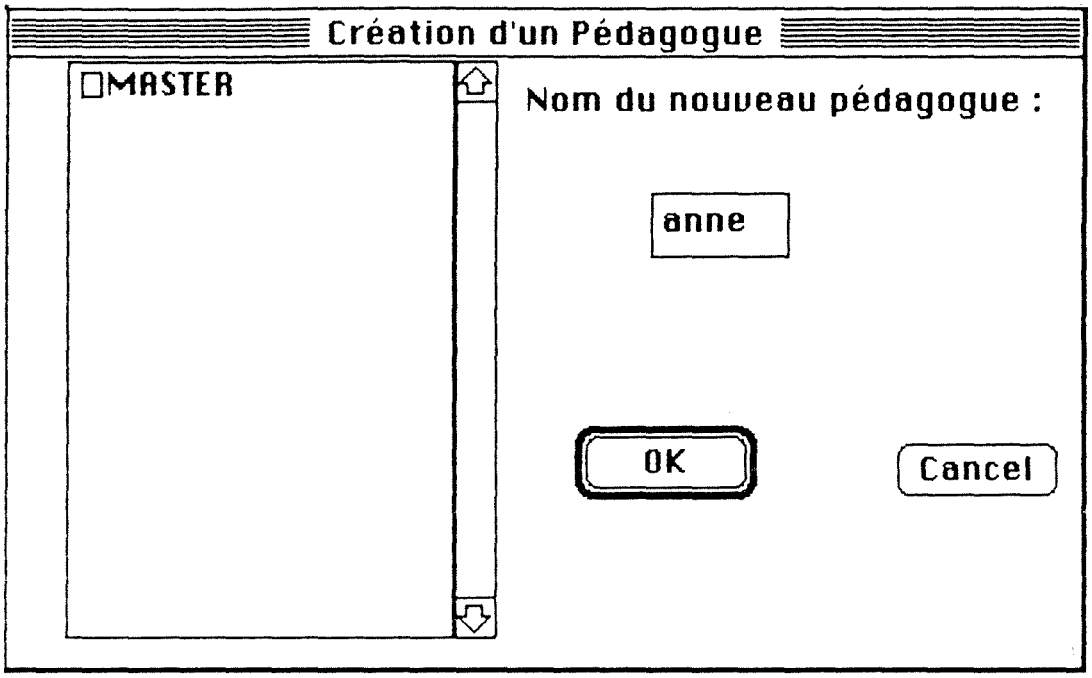
type 7

utilisation : dans la procédure ProcessInMenu

fonction : permet à l'utilisateur de consulter la liste des pédagogues existants, d'introduire un nom de pédagogue inexistant

argument/résultat : la liste des pédagogues existants
le nom du nouveau pédagogue
un code d'erreur

contenu :



Ecran 29

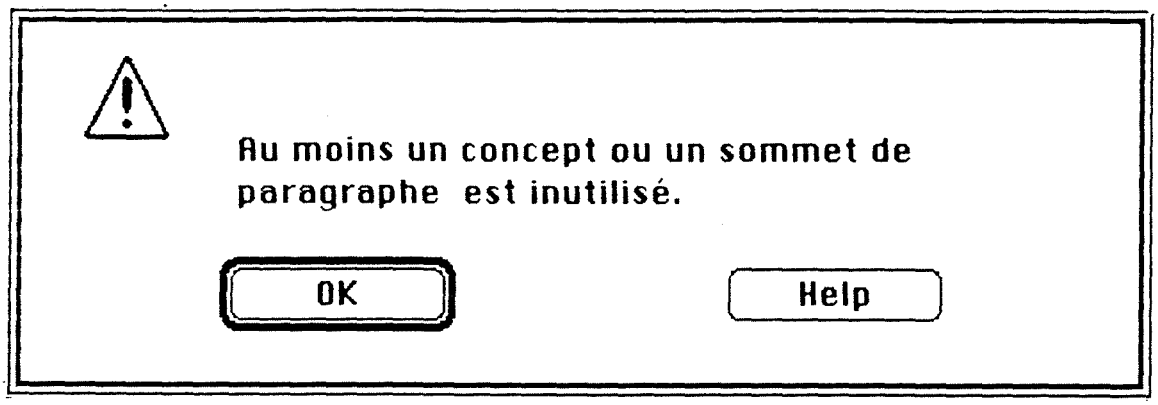
type 3

utilisation : dans la procédure DoGoAway

fonction : signale à l'utilisateur que tous les concepts ou sommets de paragraphe proposés n'ont pas tous été utilisés pour composer la réponse

argument/résultat : un code d'erreur

contenu :



Ecran 30-1

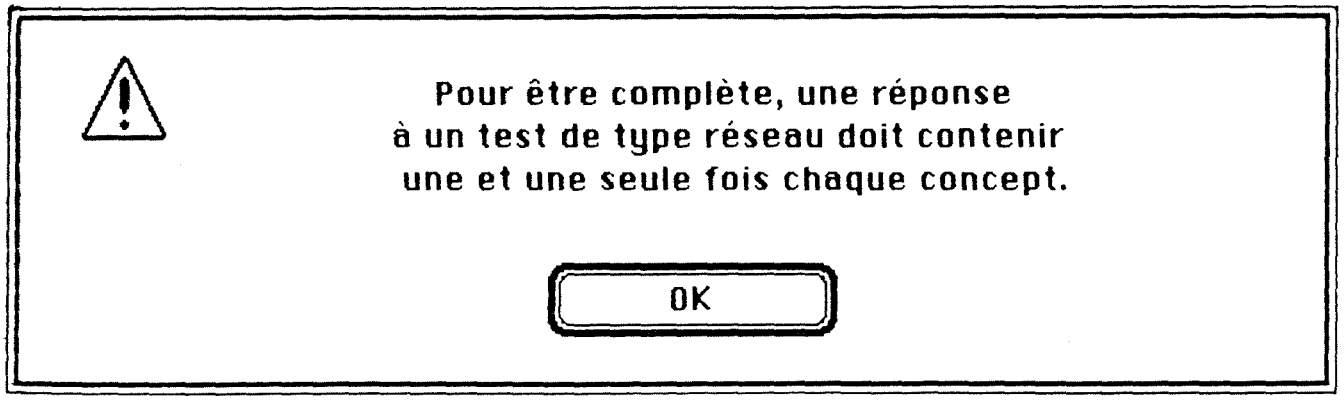
type 3

utilisation : dans la procédure DoGoAway

fonction : détaille l'explication donnée par l'écran 29 dans le cas d'un test de type réseau

argument/résultat : un code d'erreur

contenu :



Ecran 30-2

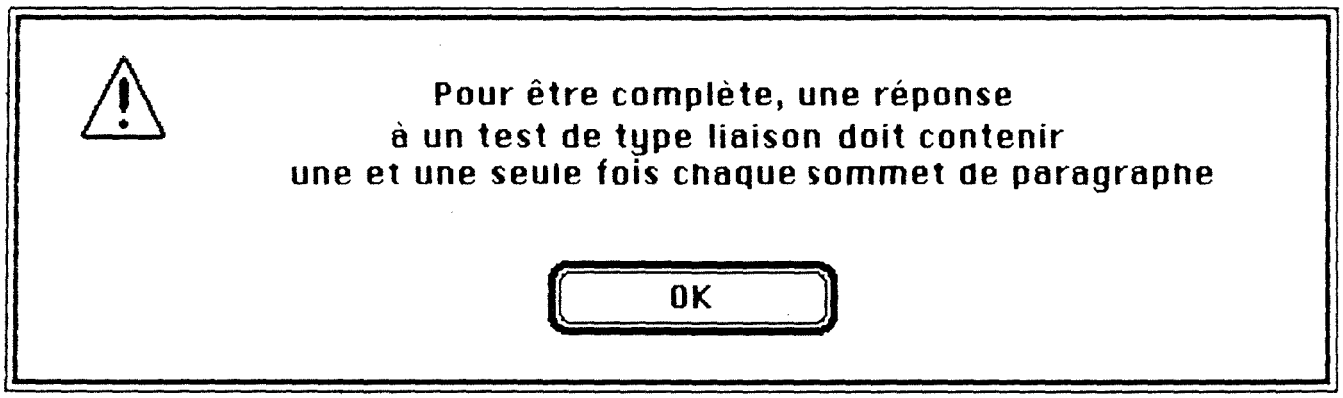
type 3

utilisation : dans la procédure DoGoAway

fonction : détaille l'explication donnée par l'écran 29 dans le cas d'un test de type liaison ou hiérarchisation de paragraphes

argument/résultat : un code d'erreur

contenu :



Ecran 33

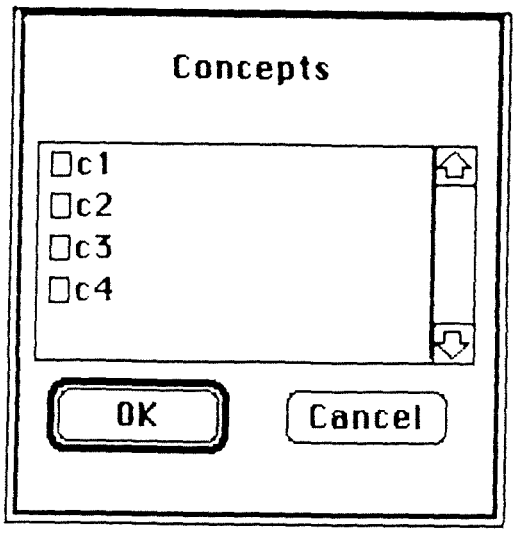
type 6

utilisation : dans la procédure DoAddCon

fonction : demande à l'utilisateur de sélectionner un nom de concept dans la liste des concepts encore disponibles qui lui est présentée

argument/résultat : la liste des concepts disponibles
le nom du concept
un code d'erreur

contenu :



Ecran 34

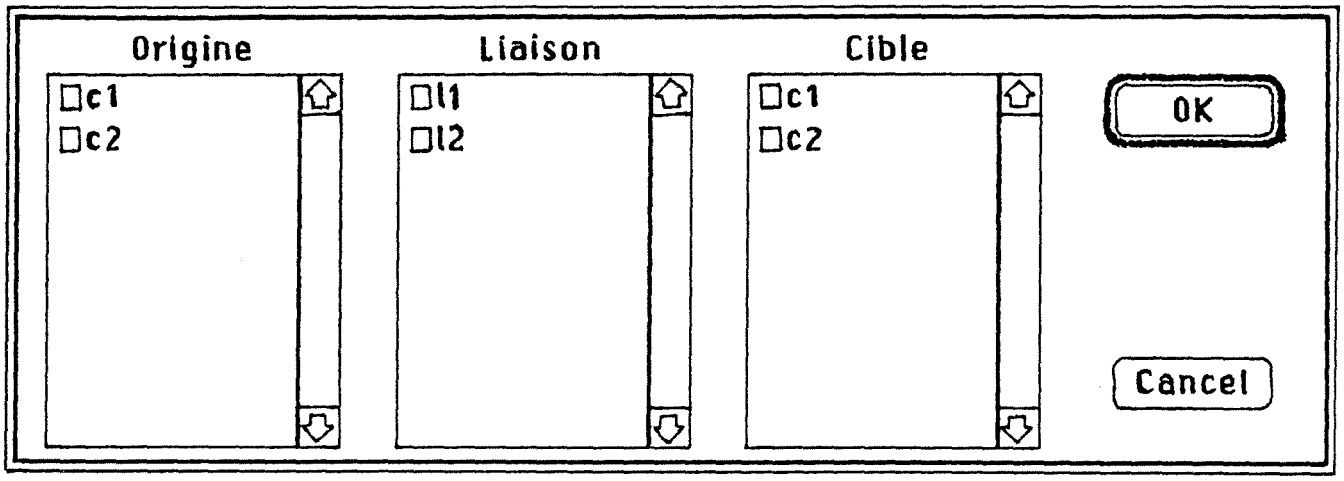
type 6

utilisation : dans la procédure DoAddLiai

fonction : propose à l'utilisateur la liste des concepts origines, la liste des concepts cibles et la liste des liaisons et lui demande de sélectionner un nom dans chacune de ces trois listes

argument/résultat : la liste des concepts origines
la liste des concepts cibles
la liste des liaisons
le nom du concept origine
le nom du concept cible
le nom de la liaison
un code d'erreur

contenu :



Ecran 35

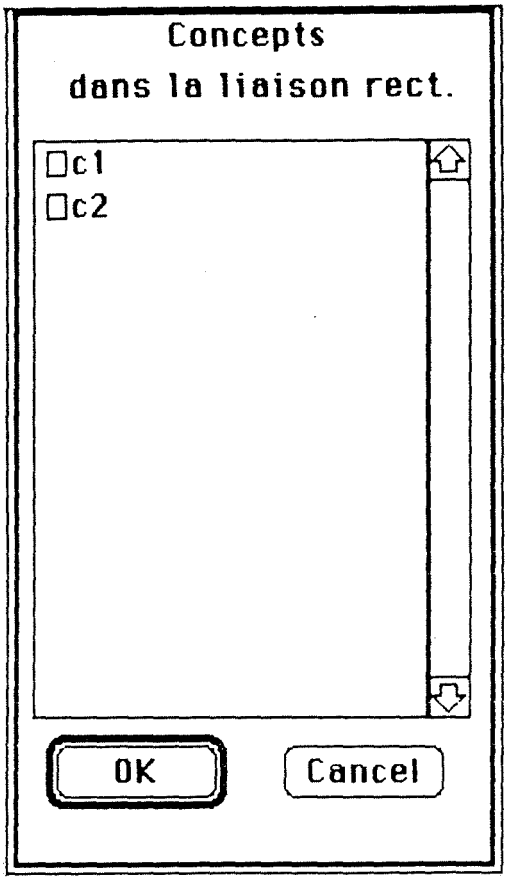
type 6

utilisation : dans la procédure DoAddLR

fonction : propose à l'utilisateur la liste des concepts et lui demande de sélectionner successivement tous les concepts faisant partie de la liaison rectangle

argument/résultat : la liste des concepts
le nom du concept
un code d'erreur

contenu :



Ecran 36

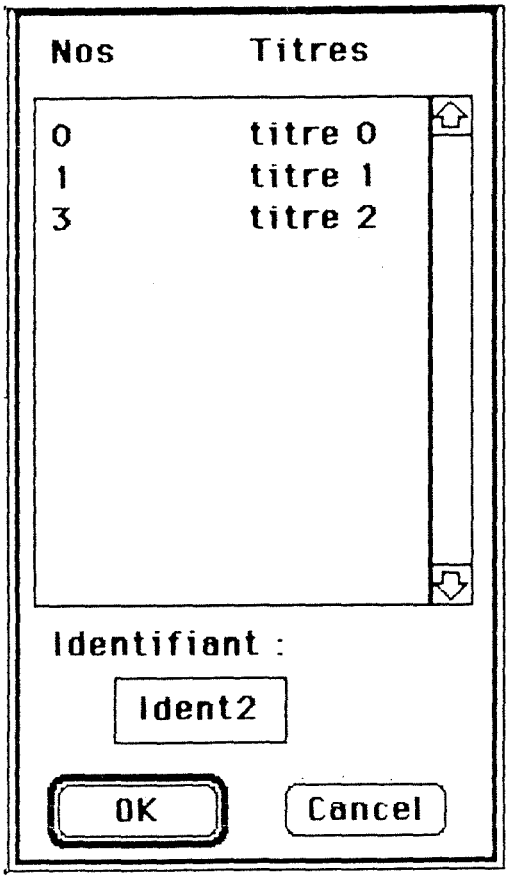
type 7

utilisation : dans la procédure DoAddSTPar

fonction : demande et saisit un identifiant associé à un paragraphe du texte de travail

argument/résultat : l'identifiant introduit
le numéro du paragraphe associé
un code d'erreur

contenu :



Ecran 37

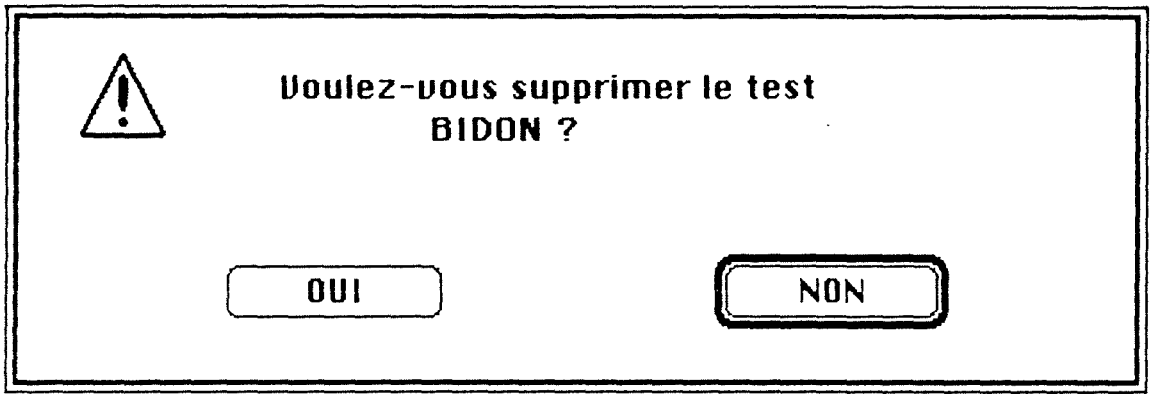
type 4

utilisation : dans la procédure ProcessInMenu

fonction : demande à l'utilisateur de confirmer la suppression du test choisi

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 38

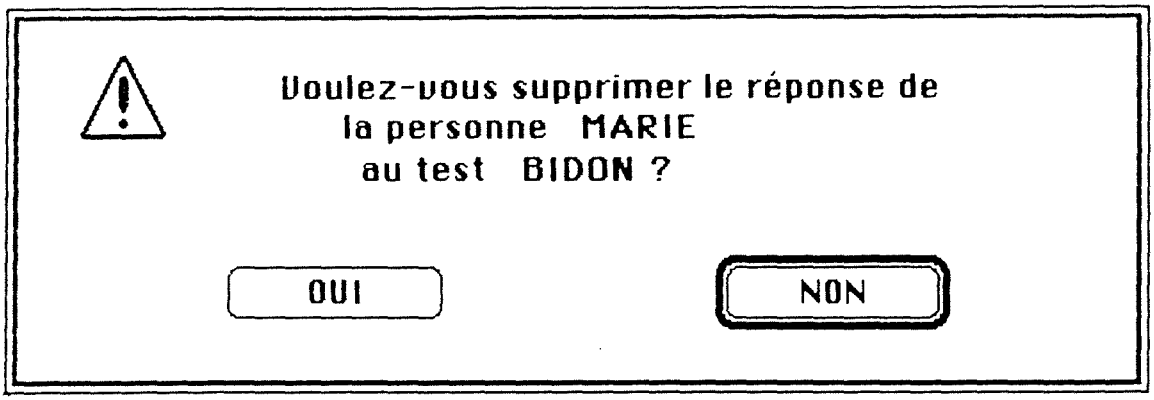
type 4

utilisation : dans la procédure ProcessInMenu

fonction : demande à l'utilisateur de confirmer la suppression de la réponse d'une personne choisie à un test choisi

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 39

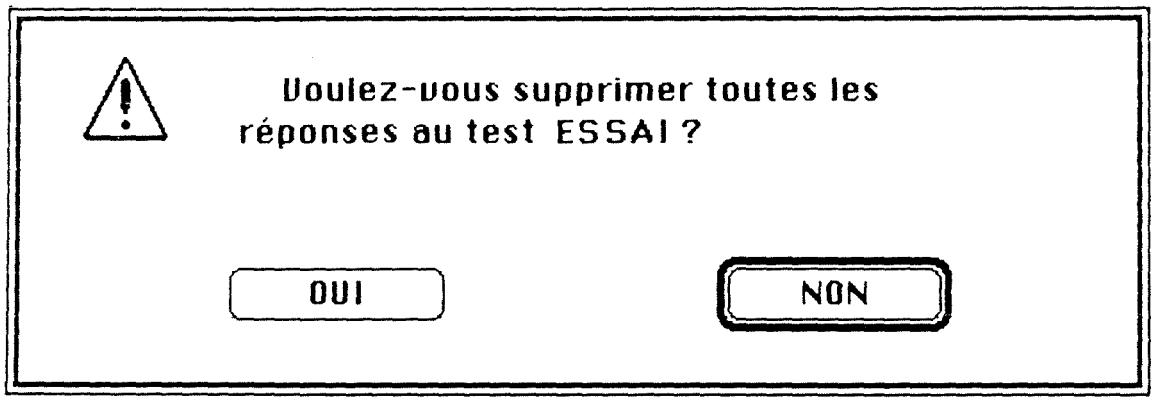
type 4

utilisation : dans la procédure ProcessInMenu

fonction : demande à l'utilisateur de confirmer la suppression de toutes les réponses à un test choisi

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 40

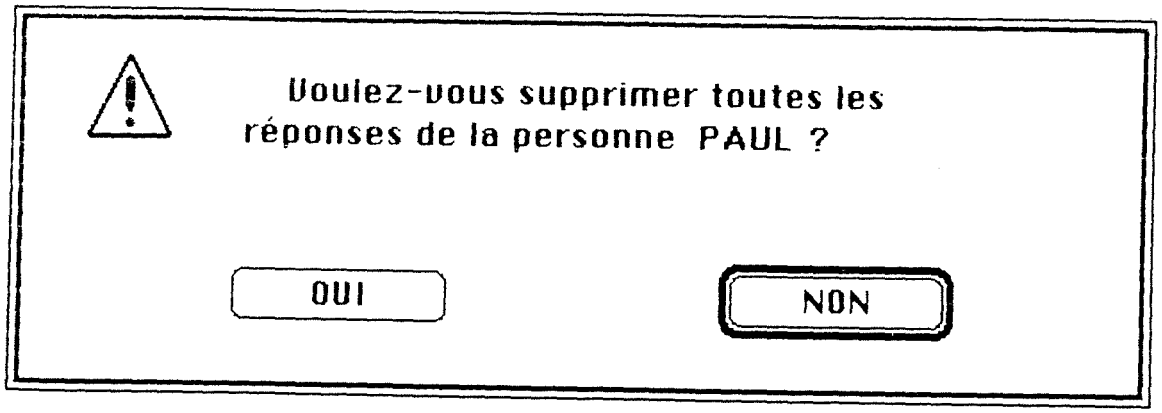
type 4

utilisation : dans la procédure ProcessInMenu

fonction : demande à l'utilisateur de confirmer la suppression de toutes les réponses d'une personne choisie

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 41

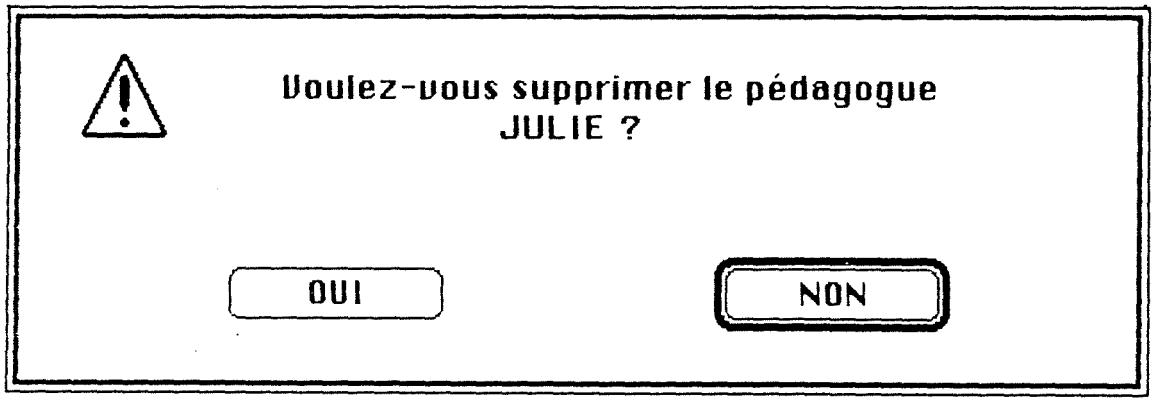
type 4

utilisation : dans la procédure ProcessInMenu

fonction : demande à l'utilisateur de confirmer la suppression du pédagogue choisi

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 42

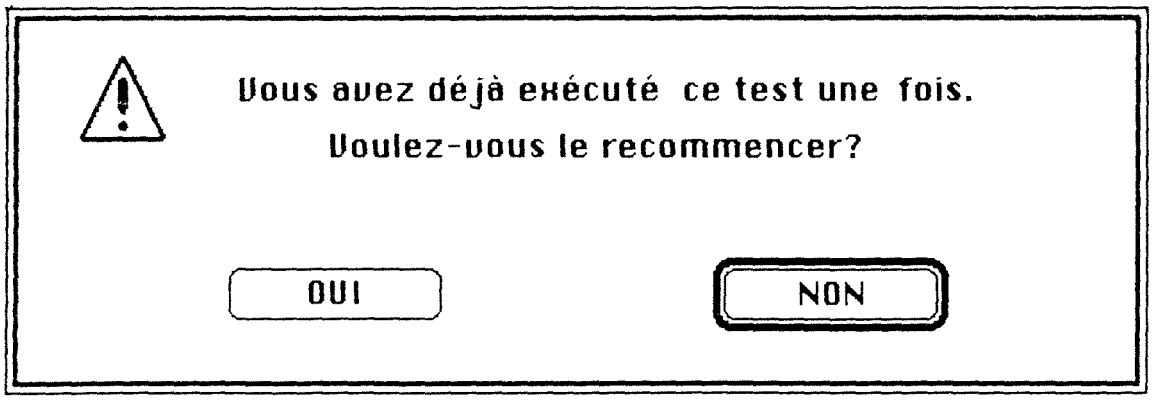
type 4

utilisation : dans la procédure ProcessInMenu

fonction : signale à l'utilisateur qu'il a déjà exécuté le test choisi et lui demande s'il souhaite recommencer ce test

argument/résultat : la réponse de l'utilisateur, c'est-à-dire une variable booléenne

contenu :



Ecran 43

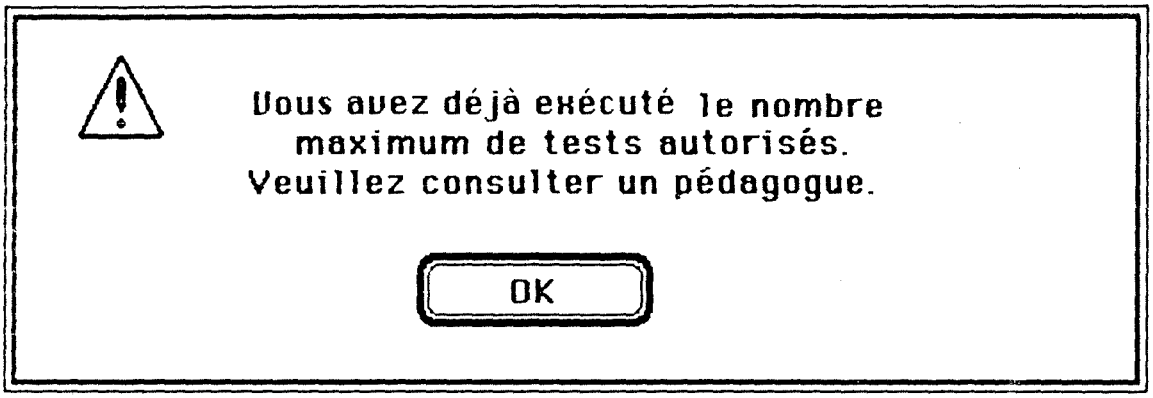
type 3

utilisation : dans la procédure InNomTestexec

fonction : signale à l'utilisateur qu'il a déjà exécuté le nombre maximum de tests autorisés

argument/résultat :

contenu :



Ecran 45

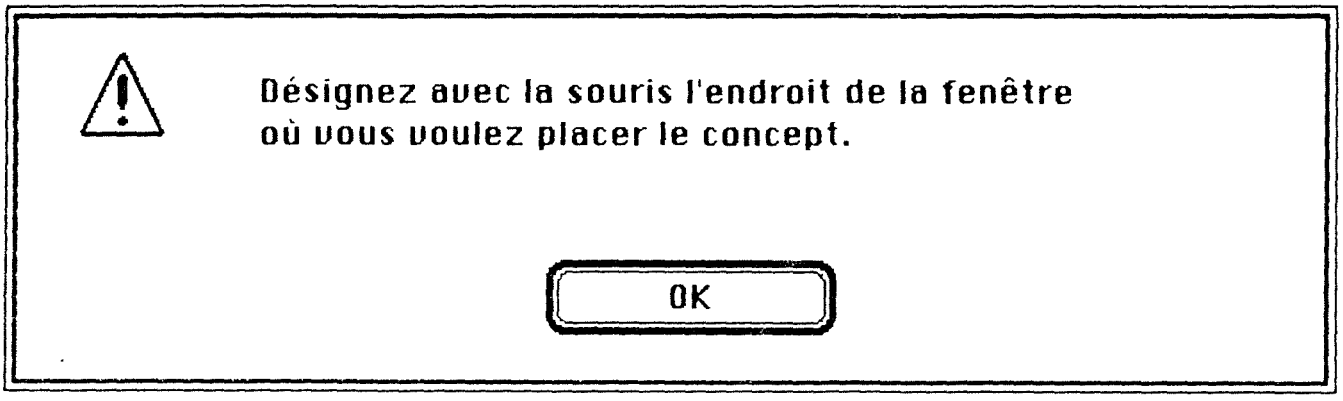
type 3

utilisation : dans la procédure ProcessInMenu

fonction : indique à l'utilisateur que pour ajouter un concept ou un sommet de paragraphe à une réponse, il doit d'abord désigner avec la souris l'endroit de l'écran où ce concept ou sommet de paragraphe sera placé

argument/résultat :

contenu :

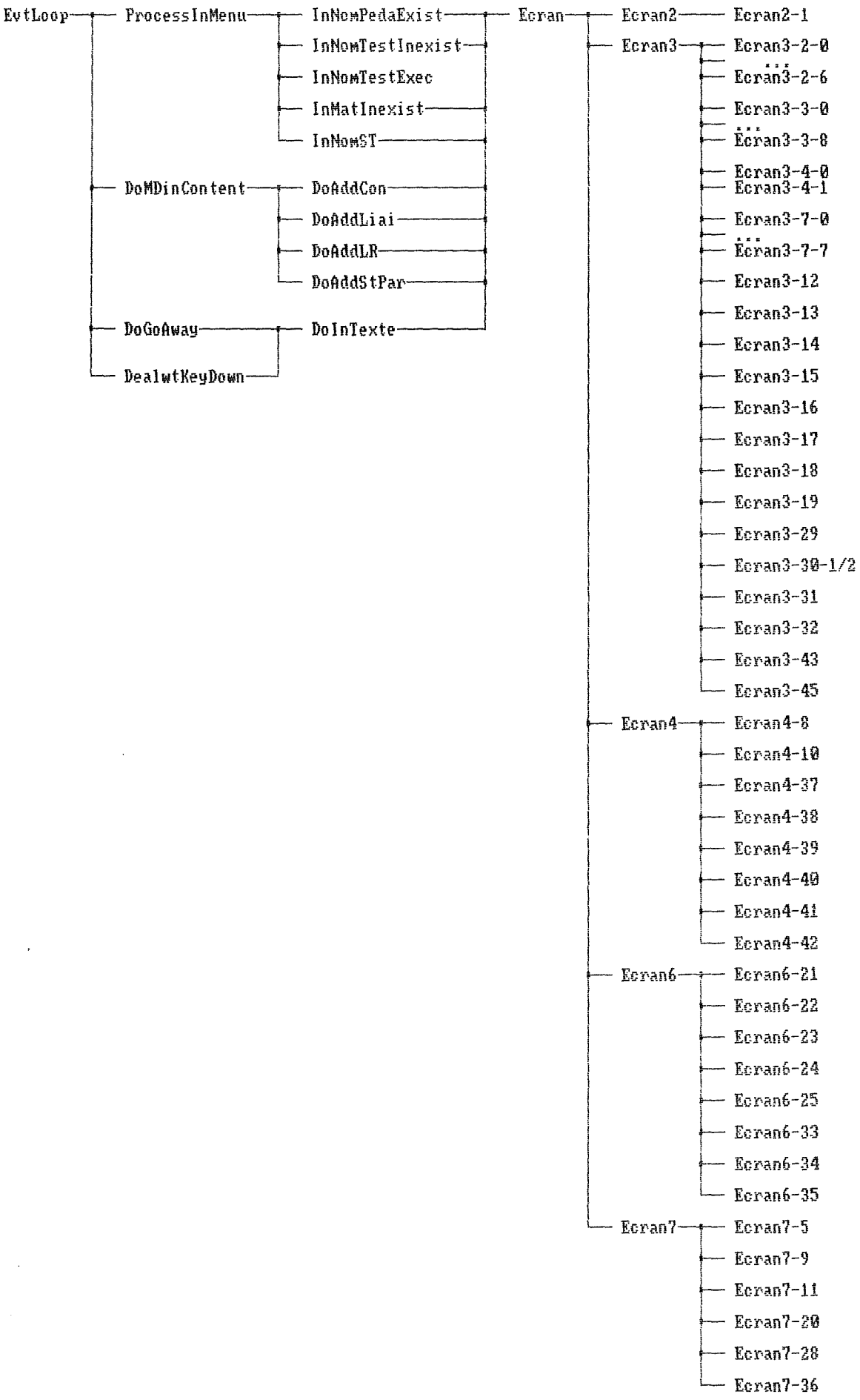


ANNEXE 4 : DECOUPE DU PROGRAMME EN PROCEDURES ET EN FONCTIONS

Nous ne reprendrons ici que les procédures qui présentent un intérêt en ce qui concerne le problème de la gestion des erreurs.

Le programme est présenté sous forme du graphe statique de ses appels de procédures tel qu'on peut le créer à la simple lecture du code qui compose le programme.

Un arc allant d'une procédure de nom A à une procédure de nom B signifie donc que la procédure A, dans son texte, contient un appel à la procédure B.


EvtLoop
ProcessInMenu
InNomPedaExist
InNomTestInexist
InNomTestExec
InMatInexist
InNomST
DoMDinContent
DoAddCon
DoAddLiai
DoAddLR
DoAddStPar
DoGoAway
DoInTexte
DealwtKeyDown
Ecran
Ecran2
Ecran2-1
Ecran3
Ecran3-2-0
...
Ecran3-2-6
Ecran3-3-0
...
Ecran3-3-8
Ecran3-4-0
Ecran3-4-1
Ecran3-7-0
...
Ecran3-7-7
Ecran3-12
Ecran3-13
Ecran3-14
Ecran3-15
Ecran3-16
Ecran3-17
Ecran3-18
Ecran3-19
Ecran3-29
Ecran3-30-1/2
Ecran3-31
Ecran3-32
Ecran3-43
Ecran3-45
Ecran4
Ecran4-8
Ecran4-10
Ecran4-37
Ecran4-38
Ecran4-39
Ecran4-40
Ecran4-41
Ecran4-42
Ecran6
Ecran6-21
Ecran6-22
Ecran6-23
Ecran6-24
Ecran6-25
Ecran6-33
Ecran6-34
Ecran6-35
Ecran7
Ecran7-5
Ecran7-9
Ecran7-11
Ecran7-20
Ecran7-28
Ecran7-36

ANNEXE 5 : ILLUSTRATION DE LA METHODE DECRITE

Cette annexe a pour but d'illustrer les principes relatifs à la gestion des erreurs exposés dans la partie principale de ce présent mémoire.

Nous proposerons, à titre d'illustration, une partie du programme réalisé, qui contient des instructions implémentant la gestion d'erreurs telle que décrite précédemment; nous pourrons ainsi donner des exemples de ce que nous avons appelé plus haut "l'interprétation d'une erreur" et la "correction d'une erreur".

Les procédures choisies pour cet exemple sont les suivantes :

- InNomPedaExist, considérée dans notre exemple comme la procédure appelante;

- Ecran,
- Ecran2,
- Ecran2-1,
- Ecran3,
- Ecran3-2-0 et Ecran3-2-1,
- Ecran3-3-0 et Ecran3-3-1,
- Ecran3-4-0 et Ecran3-4-1, qui sont les procédures appelées.

Le but du morceau de programme choisi pour cet exemple est de demander à l'utilisateur d'introduire un nom de pédagogue existant et de saisir ce nom (appel à Ecran2-1, via Ecran et Ecran2).

Il faut ensuite vérifier la longueur du nom, et le cas échéant, envoyer un message à l'utilisateur (via Ecran et Ecran3, appel à Ecran3-2-0, si le nom est trop court et appel à Ecran3-3-0, si le nom est trop long). Pour chacun de ces messages d'erreurs, l'utilisateur a la possibilité de demander une explication plus détaillée (appel à Ecran3-2-1 ou à Ecran3-3-1, via Ecran et Ecran3).

Enfin, il faut encore contrôler si le nom introduit fait partie de la liste des noms de pédagogues existants et, si ce n'est pas le cas, le signaler à l'utilisateur par un message d'erreur (appel à Ecran3-4-0, via Ecran et Ecran3). Ici aussi, l'utilisateur peut obtenir à la demande un message plus détaillé (appel à Ecran3-4-1, via Ecran et Ecran3).

```
Procedure InNomPedaExist (var nompeda : stlid; var numpeda : integer);

   var nessai : integer;
       ele1 : stlid;
       ele2 : stltitr;
       ele3 : char;
       bool : boolean;
       num : integer;


begin
   nompeda := '';
   nessai := 0;
   repeat
      errcode := 0; { initialisation du code d'erreur }
      ecran  (1, nompeda, ele2, ele3, bool, nessai);
                        { appel à  l'écran qui saisit le nom }

      case errcode of
         0 : { pas d'erreur dans la longueur du nom introduit }
            begin
               if V_Exist_Peda (nompeda, numpeda) <> 0 then
                  begin { nom existant }
                    errcode:=7; { interprétation du code d'erreur }
                    nessai := nessai+1;
                    ecran (4, ele1, ele2, ele3, bool, 0);
                     {appel à l'écran 3-4-0 => correction de l'erreur}
                    if errcode  = 341 then
                       begin  {demande d'un message plus détaillé pour
                                l'erreur précédente }
                         errcode:=8; {interprétation du code d'erreur}
                         ecran (4, ele1, ele2, ele3, bool, 0);
                            {appel à l'écran 3-4-1 =>
                                        correction  de l'erreur }
                        end;
                  end;
            end;

         211 : { annulation saisie nom }
            errcode := 1 ; { interprétation du code d'erreur }

         213 : { longueur nom supérieure à 6 caractères }
            begin
               errcode := 3; { interprétation du code d'erreur }
               nessai := nessai + 1;
               ecran (3,  ele1,  ele2,  ele3,  bool, 0);
                   { appel à l'écran 3-3-0 => correction de l'erreur }
               if errcode = 331 then
                  begin { demande  d'un  message  plus détaillé
                                pour  l'erreur précédente }
                     errcode:=5 ; { interprétation du code d'erreur }
                     ecran (3, ele1, ele2, ele3, bool, 1);
                       { appel à l'écran 3-3-1 =>
                                   correction de l'erreur }
                  end;
            end;

         214: { longueur nom inférieure à 1 caractère }
            begin
```

```
                errcode := 4; { interprétation du code d'erreur }
                nessai := nessai + 1;
                ecran (2,  ele1, ele2, ele3, bool, 0);
                    { appel à l'écran 3-2-0 => correction de l'erreur }
                if errcode = 331 then
                   begin { demande  d'un  message  plus détaillé
                                pour  l'erreur précédente }
                      errcode:=5 ; { interprétation du code d'erreur }
                      ecran (2, ele1, ele2, ele3, bool, 1);
                        { appel à l'écran 3-2-1 =>
                                    correction de l'erreur }
                   end;
             end;
       end;

       if nessai then
          errcode := 1;

   until (errcode=1) or (errcode=0) ;

end;
```

```
Procedure  Ecran ( numec : integer; var ele1 :  stlid;
                   var ele2 :  stltitr; var ele3 :  char;
                   var bool : boolean; var num : integer);


begin
   case numec of :
     ...
      1 :
         begin
            ecran2(numec, ele1, ele2, ele3, bool, nb);
            case errcode of   { interprétation du code d'erreur }
               0  : errcode  :=  0;
               11 : errcode := 211;
               13 : errcode := 213;
               14 : errcode := 214;
            end;
         end;


      2,3,4... :
         begin
            ecran3(numec, nb);
            case errcode of  { interprétation du code d'erreur }
               0 : errcode := 0;
               20 : errcode := 320;
               21 : errcode := 321;
               30 : errcode := 330;
               31 : errcode := 331;
               40 : errcode := 340;
               41 : errcode := 341;
            ...
            end;
         end;
        ...

   end;
end;
```

```
procedure Ecran2 ( numec : integer; var ele1 : stlid;
                        var ele2 : stltitr; var ele3 : char;
                        var ok : boolean; var nb : integer ) ;


begin
   case numec of
      1 : begin
             ecran2-1;

              case errcode of  { interprétation du code d'erreur }
                 0 : errcode := 10;
                 1 : errcode := 11;
                 3 : errcode := 13;
                 4 : errcode := 14;
          end;
      ...
   end;
end;



procedure ecran2-1 ;


begin
   ...

   repeat
      modalDlg( nil, item)
   until (item= OkIt) or (item=CancelIt);

   case item of  { positionnement du code d'erreur }
      CancelIt : errcode := 1 ;
      okIt : begin
                errcode := 0 ;
                if length(nom) < minlgpeda then
                   errcode := 4 ;
                if length(nom) > maxlgpeda then
                   errcode := 3 ;
             end;
   end;

   ...
end;
```

```
procedure Ecran3 ( numec : integer; var nb : integer ) ;


begin
   case numec of
      2 : begin
            case nb of
               0 : ecran3-2-0;
               1 : ecran3-2-1;
               ...
             end ;

             case errcode of  { interprétation du code d'erreur }
                0 : errcode := 20;
                1 : errcode := 21;
         end;

   end;

      3 : begin
            case nb of
               0 : ecran3-3-0;
               1 : ecran3-3-1;
               ...
             end ;

             case errcode of  { interprétation du code d'erreur }
                0 : errcode := 30;
                1 : errcode := 31;
         end;

   end;

      4 : begin
            case nb of
               0 : ecran3-4-0;
               1 : ecran3-4-1;
               ...
             end ;

             case errcode of  { interprétation du code d'erreur }
                0 : errcode := 40;
                1 : errcode := 41;
         end;

   end;

end;
```

```
procedure ecran3-2-0 ;


begin
   ...

   item := cautionalert(alertid, nil);
   case item of  { positionnement du code d'erreur }
      helpIt : errcode := 1 ;
      okIt : errcode := 0 ;
   end;

   ...
end;
```

```
procedure ecran3-3-0 ;


begin
   ...

   item := cautionalert(alertid, nil);
   case item of  { positionnement du code d'erreur }
      helpIt : errcode := 1 ;
      okIt : errcode := 0 ;
   end;

   ...
end;
```

```
procedure ecran3-4-0 ;


begin
   ...

   item := cautionalert(alertid, nil);
   case item of  { positionnement du code d'erreur }
      helpIt : errcode := 1 ;
      okIt : errcode := 0 ;
   end;

   ...
end;
```

```
procedure ecran3-2-1 ;


begin
   ...

   item := cautionalert(alertid, nil);
   errcode := 0 ;  { initialisation du code d'erreur }

   ...
end;



procedure ecran3-3-1 ;


begin
   ...

   item := cautionalert(alertid, nil);
   errcode := 0; { initialisation du code d'erreur }

   ...
end;



procedure ecran3-4-1 ;


begin
   ...

   item := cautionalert(alertid, nil);
   errcode := 0; { initialisation du code d'erreur }

   ...
end;
```

ANNEXE 6 : MANUEL D'UTILISATION

Ce manuel suppose une connaissance préalable de l'utilisation de l'Apple MacIntosh et de ses applications typiques ainsi que du vocabulaire employé dans ces applications.

Ce logiciel a pour objectif de réaliser des tests sur la mémoire humaine; pour cela, il s'adresse à deux types d'utilisateurs, des pédagogues et des sujets testés.

Il permet aux pédagogues de créer de nouveaux tests et d'analyser les réponses fournies par les sujets testés.
Les sujets testés peuvent fournir des réponses aux tests créés par les pédagogues.

Tous les tests réalisables dans le cadre de ce logiciel utilisent un même matériel de base : un texte de travail. A partir de ce texte de travail, trois exercices différents peuvent être proposés :

1°) la représentation du contenu du texte sous forme d'un réseau sémantique, c'est-à-dire sous la forme d'un graphe orienté où les noeuds (appelés concepts) sont les idées principales reprises dans le texte et les arcs représentent les relations existant entre ces idées (appelées liaisons). Il existe également une liaison d'un type particulier, la liaison rectangle qui représente un regroupement de concepts.
Dans le cadre de cette version, les noms identifiants associés aux concepts et aux liaisons sont attribués par le pédagogue et proposés au sujet testé lors de l'exécution du test. On appellera "matériaux du test" l'ensemble des identifiants de concepts et de laisons.

2°) la représentation des liaisons existant entre les paragraphes du texte; cette représentation se fera sous forme d'un graphe orienté dont les noeuds sont les paragraphes et dont les relations sont les liaisons entre ces paragraphes. Ici, seuls les noms identifiants des liaisons sont attribués par le pédagogue et c'est le sujet testé qui attribue des noms identifiants aux différents paragraphes du texte de travail.

3°) la représentation de la structure qui guide le texte, sous la forme d'une hiérarchie de ses paragraphes. Dans ce cas, le sujet testé doit lui-même attribuer des noms identifiants aux diférents paragraphes du texte de travail.

D'une manière générale, le logiciel est divisé en trois phases distinctes :

- l'introduction de nouveaux tests par un pédagogue, y compris le texte de travail et les matériaux (pour un test de type réseau ou de type liaison);

- l'exécution de tests, c'est-à-dire une phase réservée aux sujets testés et au cours de laquelle ils peuvent apporter une réponse à l'un des tests et la comparer par la suite avec la réponse proposée par le pédagogue.

- l'analyse des résultats de tests, c'est-à-dire
  l'affichage des listes de tests et de sujets testés;
  l'affichage des réponses fournies par les sujets testés;
  la suppression de réponses;
  la suppression de tests;
  la suppression de pédagogues dans la liste;
  l'ajout de nouveaux pédagogues.

Chacune de ces trois phases correspond à un titre de la barre des menus qui apparaît lors du lancement de l'application.

Au fur et à mesure des opérations déjà effectuées, les items de menus sont activés ou désactivés, selon les opérations autorisées ou interdites.

Nous allons tout d'abord décrire séparément chacune de ces phases en expliquant l'effet de la sélection de chaque item de menu.

Ensuite, nous donnerons la liste des messages d'erreurs qui peuvent apparaître au cours de l'exécution du logiciel.

1.INTRODUCTION

* *Accès*

Sélectionner cet item permet à un pédagogue d'accéder à l'option Introduction, et donc d'introduire un nouveau test avec le texte et les matériaux associés.
La sélection de cet item fait apparaître successivement **deux boîtes de dialogue :**

la **première boîte** demande d'introduire un nom de pédagogue existant; ce nom sert en quelque sorte de mot de passe pour réserver aux pédagogues l'accès à cette option. Le nom doit contenir au moins 1 caractère et au plus 6 caractères; si ce n'est pas le cas, un message d'erreur apparaît. Si le nom de pédagogue introduit n'existe pas, un message le signale également.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

la **seconde boîte** de dialogue demande d'introduire un nouveau nom de test et affiche la liste des noms de test qui existent déjà. Des "boutons radios" permettent de sélectionner le type du test. Le nom d'un test, tout comme le nom d'un pédagogue, doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si le nom introduit existe déjà, un message vous le signale.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

* *Intro Texte Explicatif*

Le texte explicatif est un texte écrit par le pédagogue et qui sert à expliquer au sujet testé ce qu'il doit faire lors de l'exécution du test. Ce texte est un élément facultatif.
Un texte explicatif est composé d'une entête, facultative, et d'une suite non vide de paragraphes.
Une entête ou un paragraphe contient un titre, facultatif, et un contenu non vide.

Sélectionner l'item "Intro Texte explicatif" fait apparaître une **fenêtre texte** où l'on demande d'abord d'introduire le titre et le contenu de l'entête. Attention, chaque introduction doit être terminée par Command-Return.
Si le texte n'a pas d'entête, tapez directement Command-Return pour le titre et pour le contenu. Si l'entête n'a pas de titre, tapez directement Command-Return pour le titre. Attention, si l'entête a un titre, elle doit aussi avoir un contenu.
Après avoir introduit l'entête, introduisez de la même façon chacun des paragraphes et lorsque tous les paragraphes ont été introduits, signalez la fin du texte en introduisant un paragraphe vide (Command-Return pour le titre et pour le contenu du paragraphe).

* *Intro Texte Travail*

Le texte de travail est la base de tout test, puisque c'est son contenu qui sera représenté sous forme de graphe ou de hiérarchie.
Il est composé d'une entête, facultative, et d'une suite non vide de paragraphes.
Une entête ou un paragraphe contient un titre, facultatif, et un contenu non vide.

Sélectionner l'item "Intro Texte Travail" fait apparaître une **fenêtre texte** où l'on demande d'abord le titre et le contenu de l'entête. Attention, chaque introduction doit être terminée par Command-Return.
Si le texte n'a pas d'entête, tapez directement Command-Return pour le titre et pour le contenu.
Si l'entête n'a pas de titre, tapez directement Command-Return pour le titre.
Attention, si l'entête a un titre, elle doit aussi avoir un contenu.
Après avoir introduit l'entête, introduisez de la même façon chacun des paragraphes et lorsque tous les paragraphes ont été introduits, signalez la fin du texte en introduisant un paragraphe vide (Command-Return pour le titre et pour le contenu du paragraphe).

* *Intro Matériaux*

- pour un test de type réseau :

Il s'agit d'introduire la liste des concepts et des liaisons qui interviendront dans la réponse. A chaque matériau, il correspond un identifiant long, un identifiant court et un type du matériau (Céseau ou Liaison).
La sélection de l'item "Intro Matériau" fait apparaître une **boîte de dialogue** qui demande d'introduire un à un les différents matériaux. La boîte de dialogue contient la liste des matériaux déjà créés pour ce test, deux "boutons radios" pour sélectionner le type du matériau et deux boîtes d'édition qui permettent d'introduire l'identifiant long et l'identifiant court du matériau.
L'identifiant long doit contenir au moins 1 caractère et au plus 20 caractères; l'identifiant court doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si l'un des deux identifiants existe déjà, un message vous le signale.
Lorsque tous les matériaux ont été introduits, signalez-le en cliquant sur le bouton "Cancel".

Une nouvelle **boîte de dialogue** apparaît alors : elle demande si vous souhaitez autoriser la liaison rectangle dans les matériaux du test. Cliquez "Oui" ou "Non" selon votre réponse.

- pour un test de type liaison :

Il s'agit d'introduire la liste des liaisons qui interviendront dans la réponse. A chaque liaison, il correspond un identifiant long et un identifiant court.
La sélection de l'item "Intro Matériau" fait apparaître une **boîte de dialogue** qui demande d'introduire une à une les différentes liaisons. La boîte de dialogue contient la liste des liaisons déjà créées pour ce test et deux boîtes d'édition qui permettent d'introduire l'identifiant long et l'identifiant court du matériau.
L'identifiant long doit contenir au moins 1 caractère et au plus 20 caractères; l'identifiant court doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si l'un des deux identifiants existe déjà, un message vous le signale.
Lorsque toutes les liaisons ont été introduites, signalez-le en cliquant sur le bouton "Cancel".

* *Intro Réponse Type*

La réponse type est la réponse qui est proposée au sujet testé après la création de sa propre réponse. Elle lui permet de comparer sa réponse à celle du pédagogue.
La sélection de l'item "Intro Réponse Type" fait apparaître une **fenêtre réponse** vide qui se manipule de façon standard (barres de défilement, flèches...).
Pour créer la réponse, sélectionnez un à un les items de menus correspondants aux actions souhaitées (ajout d'un concept, d'un sommet de paragraphe, d'une liaison, d'une liaison rectangle ou association d'un texte à un élément de la réponse).
Pour signaler la fin de la création de la réponse, fermez la fenêtre. Attention, une réponse n'est complète que si tous les concepts ou tous les paragraphes en font partie. Si ce n'est pas le cas, un message vous le signalera.

* *Intro Réponse Exemple*

La réponse exemple est la réponse qui est proposée au sujet testé avant la création de sa propre réponse. Elle montre par un exemple ce que l'on attend de lui au cours de ce test.
La sélection de l'item "Intro Réponse Exemple" fait apparaître une **fenêtre réponse** vide qui se manipule de façon standard (barres de défilement, flèches...).
Pour créer la réponse, sélectionnez un à un les items de menus correspondants aux actions souhaitées (ajout d'un concept, d'un sommet de paragraphe, d'une liaison, d'une liaison rectangle ou association d'un texte à un élément de la réponse).
Pour signaler la fin de la création de la réponse, fermez la fenêtre. Attention, une réponse n'est complète que si tous les concepts ou tous les paragraphes en font partie. Si ce n'est pas le cas, un message vous le signalera.

* *Consulter Texte Travail*

Le texte de travail est la base de tout test, puisque c'est son contenu qui sera représenté sous forme de graphe ou de hiérarchie.
Il est composé d'une entête, facultative, et d'une suite non vide de paragraphes.
Une entête ou un paragraphe contient un titre, facultatif, un contenu non vide et, pour les tests de type liaison ou de type hiérarchisation, une zone identifiante .
La sélection de l'item "Consulter Texte Travail" permet de consulter le texte de base durant la création de la réponse.
Elle fait apparaître une **fenêtre texte** qui se manipule de façon standard. Lorsque la consultation est terminée, cliquez dans la "close box" de la fenêtre.

* *Ajouter Concept (test de type réseau uniquement)*

Cet item permet d'ajouter un concept à la réponse en cours de création.
Vous devez tout d'abord cliquer dans la fenêtre réponse à l'endroit où se placera le concept.
Ensuite, apparaît une **boîte de dialogue** qui permet de sélectionner le nom du concept à ajouter dans la réponse.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une forme ovale représentant le concept et contenant son identifiant court se dessine dans la fenêtre à l'endroit que vous aviez choisi.

* *Ajouter Sommet*

Cet item permet d'ajouter un sommet de paragraphe à la réponse en cours de création.

- test de type liaison :

Vous devez tout d'abord cliquer dans la fenêtre réponse à l'endroit où se placera le sommet.
Ensuite apparaît une **boîte de dialogue** où vous devez sélectionner le numéro et le titre du paragraphe à ajouter à la réponse et introduire le nom identifiant à associer à ce paragraphe. Ce nom doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si le nom existe déjà, un message vous le signale.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une forme ovale représentant le sommet et contenant son identifiant se dessine dans la fenêtre à l'endroit que vous aviez choisi.

- test de type hiérarchie :

Une **boîte de dialogue** apparaît où vous devez sélectionner le numéro et le titre du paragraphe à ajouter à la réponse et introduire le nom identifiant à associer à ce paragraphe. Ce nom doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si le nom existe déjà, un message vous le signale.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Ensuite, une **seconde boîte de dialogue** apparaît qui demande de sélectionner le "père" du paragraphe sélectionné auparavant. Le père d'un paragraphe est le paragraphe qui se trouve au niveau juste supérieur dans la hiérarchie.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

* *Ajouter Liaison (test de type réseau ou de type liaison)*

Cet item permet d'ajouter une liaison à la réponse en cours de création.
Une **boîte de dialogue** apparaît contenant 3 listes : la liste des liaisons, la liste des origines de liaisons et la liste des cibles de liaisons. Vous devez sélectionner un élément dans chacune des listes.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une droite reliant l'origine à la cible apparaît dans la fenêtre.

* *Ajouter LR (test de type réseau uniquement)*

Cet item permet d'ajouter une liaison rectangle à la réponse en cours de création, c'est-à-dire une liaison particulière ayant la signification "regroupement de concepts".
Une **boîte de dialogue** apparaît contenant la liste des concepts déjà placés dans la réponse. Vous devez sélectionner successivement tous les concepts figurant dans cette liaison rectangle.
Cliquez "Cancel" lorsque vous avez sélectionné tous les concepts faisant partie de la même liaison rectangle.
Un rectangle entourant tous les concepts sélectionnés se dessine dans la fenêtre réponse.

* *Associer Texte*

A chacun des éléments de la réponse, on peut associer un texte justificatif, appelé texte réponse.
Ce texte réponse est composé d'un seul paragraphe non vide.
Un paragraphe contient un titre, facultatif, et un contenu non vide.

Sélectionner l'item "Associer Texte" fait apparaître une **fenêtre texte** où l'on demande le titre et le contenu du paragraphe.
Attention, chaque introduction doit être terminée par Command-Return.
Si le paragraphe n'a pas de titre, tapez directement Command-Return pour le titre.
Attention, si le paragraphe a un titre, il doit aussi avoir un contenu.

* *Clôture*

Cet item permet de quitter l'option Introduction et de replacer la barre de menus dans son état initial, telle qu'elle était au lancement de l'application.
Attention, l'option Introduction ne peut être clôturée que si la réponse type complète a été introduite. Sinon, un message vous le signale.

2.EXECUTION

* *Accès*

Sélectionner cet item permet à un sujet testé d'accéder à l'option Exécution, et donc de fournir une réponse à un test. La sélection de cet item fait apparaître **deux boîtes de dialogue :**

la **première boîte** demande d'introduire un nom de sujet testé; cette boîte contient la liste des noms de sujets testé existants et une boîte d'édition. Vous pouvez soit sélectionner un nom dans la liste, soit introduire un nouveau nom au clavier. Ce nom doit contenir au moins 1 caractère et au plus 6 caractères; si ce n'est pas le cas, un message d'erreur apparaît.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

la **seconde boîte de dialogue** présente la liste des noms de tests existants et demande de sélectionner un nom dans cette liste. Sélectionner le nom et cliquez "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

* *Consulter Texte Explicatif*

Le texte explicatif est un texte écrit par le pédagogue, qui sert à expliquer au sujet testé ce qu'il doit faire lors de l'exécution du test.
Un texte explicatif est composé d'une entête, facultative, et d'une suite non vide de paragraphes.
Une entête ou un paragraphe contient un titre, facultatif, et un contenu non vide.
La sélection de l'item "Consulter Texte Explicatif" permet de consulter le texte explicatif. Elle fait apparaître une **fenêtre texte** qui se manipule de façon standard. Lorsque la consultation du texte est terminée, cliquez dans la boîte de fermeture de la fenêtre.

* *Consulter Réponse Exemple*

La réponse exemple est la réponse qui est proposée au sujet testé avant la création de sa propre réponse. Elle montre par un exemple ce que l'on attend de lui dans ce test.

La réponse peut être présentée sous deux formes :

soit une présentation progressive, c'est-à-dire que l'affichage de la réponse est entrecoupé de l'affichage des textes réponse associés aux divers éléments de la réponse;

soit une présentation en un bloc, c'est-à-dire sans les textes réponse.

La sélection de l'item "Consulter Réponse Exemple" fait apparaître une **boîte de dialogue** qui propose une présentation en un bloc ou une présentation progressive. Cliquez le bouton qui correspond à votre choix.

- si vous avez choisi une présentation progressive :

Une **fenêtre réponse** apparaît et les divers éléments de la réponse se dessinent un à un. Lorsque s'affiche un élément auquel un texte réponse est associé, une **boîte de dialogue** apparaît;
elle demande si vous souhaitez consulter le texte réponse associé. Cliquez le bouton qui correspond à votre choix.
Si vous avez demandé la consultation du texte réponse, une **fenêtre texte** apparaît contenant le texte réponse. cette fenêtre se manipule de façon standard. Lorsque vous avez terminé la consultation, cliquez dans la "close box" de la fenêtre et l'affichage des éléments de la réponse reprend son cours.
Lorsque tous les éléments ont été affichés et que vous avez terminé la consultation de la réponse, cliquez dans la "close box" de la fenêtre.

- si vous avez choisi une présentation en un bloc :

Une **fenêtre réponse** apparaît et les éléments de la réponse s'affichent. Cette fenêtre se manipule de façon standard. Lorsque vous avez terminé la consultation de la réponse, cliquez dans la "close box" de la fenêtre.

* *Intro Réponse*

La sélection de cet item permet au sujet testé de réaliser sa réponse personnelle au test choisi.
Elle fait apparaître une **fenêtre réponse** vide qui se manipule de façon standard (barres de défilement, flèches...).
Pour créer la réponse, sélectionnez un à un les items correspondants aux actions souhaitées (ajout d'un concept, d'un sommet de paragraphe, d'une liaison, d'une liaison rectangle ou association d'un texte à un élément de la réponse).
Pour signaler la fin de la création de la réponse, fermez la fenêtre. Attention, une réponse n'est complète que si tous les concepts ou tous les paragraphes en font partie. Si ce n'est pas le cas, un message vous le signalera.
..pa

* *Consulter Texte Travail*

Le texte de travail est la base de tout test, puisque c'est son contenu qui sera représenté sous forme de graphe ou de hiérarchie.
Il est composé d'une entête, facultative, et d'une suite non vide de paragraphes.
Une entête ou un paragraphe contient un titre, facultatif, un contenu non vide et, pour les tests de type liaison ou de type hiérarchisation, une zone identifiante .
La sélection de l'item "Consulter Texte Travail" permet de consulter le texte de base durant la création de la réponse. Elle fait apparaître une **fenêtre texte** qui se manipule de façon standard. Lorsque la consultation est terminée, cliquez dans la "close box" de la fenêtre.

* *Ajouter Concept (test de type réseau uniquement)*

Cet item permet d'ajouter un concept à la réponse en cours de création.
Vous devez tout d'abord cliquer dans la fenêtre réponse à l'endroit où se placera le concept.
Ensuite, apparaît une **boîte de dialogue** qui vous permet de sélectionner le nom du concept à ajouter dans la réponse.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une forme ovale représentant le concept et contenant son identifiant court se dessine dans la fenêtre à l'endroit que vous aviez choisi.

* *Ajouter Sommet*

Cet item permet d'ajouter un sommet de paragraphe à la réponse en cours de création.

- test de type liaison :

Vous devez tout d'abord cliquer dans la fenêtre réponse à l'endroit où se placera le sommet de paragraphe.
Ensuite, apparaît une **boîte de dialogue** où vous devez sélectionner le numéro et le titre du paragraphe à ajouter à la réponse et introduire le nom identifiant à associer à ce paragraphe. Ce nom doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si le nom existe déjà, un message vous le signale.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une forme ovale représentant le sommet et contenant son identifiant se dessine dans la fenêtre à l'endroit que vous aviez choisi.

- test de type hiérarchie :

Une **boîte de dialogue** apparaît où vous devez sélectionner le numéro et le titre du paragraphe à ajouter à la réponse et introduire le nom identifiant à associer à ce paragraphe. Ce nom doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si le nom existe déjà, un message vous le signale.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

Ensuite, une **seconde boîte de dialogue** apparaît qui demande de sélectionner le "père" du paragraphe sélectionné auparavant. Le père d'un paragraphe est le paragraphe qui se trouve au niveau supérieur dans la hiérarchie.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

* *Ajouter Liaison (test de type réseau ou de type liaison)*

Cet item permet d'ajouter une liaison à la réponse en cours de création.
Une **boîte de dialogue** apparaît contenant 3 listes : la liste des liaisons, la liste des origines de liaisons et la liste des cibles de liaisons. Vous devez sélectionner un élément dans chacune des listes.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une droite reliant l'origine à la cible apparaît dans la fenêtre.

* *Ajouter LR (test de type réseau uniquement)*

Cet item permet d'ajouter une liaison rectangle à la réponse en cours de création, c'est-à-dire une liaison particulière ayant la signification "regroupement de concepts".
Une **boîte de dialogue** apparaît contenant la liste des concepts déjà placés dans la réponse. Vous devez sélectionner successivement tous les concepts figurant dans cette liaison rectangle.
Cliquez "Cancel" lorsque vous avez sélectionné tous les concepts faisant partie de la même liaison rectangle.
Un rectangle entourant tous les concepts sélectionnés se dessine dans la fenêtre réponse.

* *Associer Texte*

A chacun des éléments de la réponse, on peut associer un texte justificatif, appelé texte réponse.
Ce texte réponse est composé d'un seul paragraphe non vide.
Un paragraphe contient un titre, facultatif, et un contenu non vide.

Sélectionner l'item "Associer Texte" fait apparaître une **fenêtre texte** où l'on demande le titre et le contenu du paragraphe.
Attention, chaque introduction doit être terminée par Command-Return.
Si le paragraphe n'a pas de titre, tapez directement Command-Return pour le titre.
Attention, si le paragraphe a un titre, il doit aussi avoir un contenu.

* *Consulter Réponse Type*

La réponse type est la réponse qui est proposée au sujet testé après la création de sa propre réponse. Elle lui permet de comparer sa réponse à celle du pédagogue.

La réponse peut être présentée sous deux formes :

soit une présentation progressive, c'est-à-dire que l'affichage de la réponse est entrecoupé de l'affichage des textes réponse associés aux divers éléments de la réponse;

soit une présentation en un bloc, c'est-à-dire sans les textes réponse.
La sélection de l'item "Consulter Réponse Exemple" fait apparaître une **boîte de dialogue** qui propose une présentation en un bloc ou une présentation progressive. Cliquez le bouton qui correspond à votre choix.

- si vous avez choisi une présentation progressive :

Une **fenêtre réponse** apparaît et les divers éléments de la réponse se dessinent un à un. Lorsque s'affiche un élément auquel un texte réponse est associé, une **boîte de dialogue** apparaît; elle demande si vous souhaitez consulter le texte réponse associé. Cliquez le bouton qui correspond à votre choix.
Si vous avez demandé la consultation du texte réponse, une **fenêtre texte** apparaît contenant le texte réponse. Cette fenêtre se manipule de façon standard. lorsque vous avez terminé la consultation, cliquez dans la "close box" de la fenêtre et l'affichage des éléments de la réponse reprend son cours.
Lorsque tous les éléments ont été affichés et que vous avez terminé la consultation de la réponse, cliquez dans la "close box" de la fenêtre.

- si vous avez choisi une présentation en un bloc :

Une **fenêtre réponse** apparaît et les éléments de la réponse s'affichent. Cette fenêtre se manipule de façon standard. Lorsque vous avez terminé la consultation de la réponse, cliquez dans la "close box" de la fenêtre.

* *Clôture*

Cet item permet de quitter l'option Introduction et de replacer la barre de menus dans son état initial, telle qu'elle était au lancement de l'application.
Attention, l'option Introduction ne peut être clôturée que si la réponse type complète a été introduite. Sinon, un message vous le signale.

3.ANALYSE

* *Accès*

Sélectionner cet item permet à un pédagogue d'accéder à l'option Analyse et donc de consulter des réponses de sujets testés, de créer ou de supprimer des pédagogues et de supprimer des tests et des réponses.
La sélection de cet item fait apparaître une **boîte de dialogue** qui demande d'introduire un nom de pédagogue existant; ce nom sert en quelque sorte de mot de passe pour réserver aux pédagogues l'accès à cette option. Le nom doit contenir au moins 1 caractère et au plus 6 caractères; si ce n'est pas le cas, un message d'erreur apparaît. Si le nom de pédagogue introduit n'existe pas, un message le signale également.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

* *Liste Tests*

Sélectionner cet item permet de consulter la liste des tests existants : une **boîte de dialogue** apparaît avec la liste des noms de tests existants accompagnés de leur type (Réseau, Liaison ou Hiérarchie). Vous pouvez vous déplacer dans la liste grâce aux barres de défilement.
Lorsque vous avez terminé de consulter la liste, cliquez sur le bouton "OK".

* *Liste Personnes*

Sélectionner cet item permet de consulter la liste des sujets testés existants : une **boîte de dialogue** apparaît avec la liste des noms de sujets testés existants.
Vous pouvez vous déplacer dans la liste grâce aux barres de défilement.
Lorsque vous avez terminé de consulter la liste, cliquez sur le bouton "OK".

* *Liste des tests d'une personne*

Sélectionner cet item permet de consulter la liste des tests réalisés par un sujet testé choisi : une **première boîte de dialogue** apparaît avec la liste des noms de sujets testés existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du sujet testé dont vous voulez consulter la liste de tests et cliquez sur le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans consulter la liste de tests, cliquez "Cancel".
Ensuite, une **seconde boîte de dialogue** apparaît avec la liste des tests exécutés par le sujet testé choisi. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement.
Lorsque vous avez terminé de consulter la liste, cliquez sur le bouton "OK".

* *Liste des personnes pour un test*

Sélectionner cet item permet de consulter la liste des sujets testés ayant fourni une réponse à un test choisi : une **première boîte de dialogue** apparaît avec la liste des noms de tests existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du test dont vous voulez consulter la liste de sujets testé et cliquez sur le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans consulter la liste de sujets testés, cliquez "Cancel".
Ensuite, une **seconde boîte de dialogue** apparaît avec la liste des sujets testés ayant exécuté le test choisi. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement.
Lorsque vous avez terminé de consulter la liste, cliquez sur le bouton "OK".

* *Sortie*

Sélectionner cet item permet de choisir la sortie sur laquelle vous voulez voir afficher la réponse, la chronologie ou les textes : vous pouvez souhaiter un affichage à l'écran ou une impression sur papier.
La sélection de cet item fait passer d'un mode de sortie à l'autre.
Si vous sélectionnez le mode "imprimante", s'affichent les **boîtes de dialogue** standard relatives à une impression; répondez selon votre choix.

* *Réponse*

Sélectionner cet item permet de consulter la réponse donnée par un sujet testé à un test;

La réponse peut être présentée sous deux formes :
soit une présentation progressive, c'est-à-dire que l'affichage de la réponse est entrecoupé de l'affichage des textes réponse associés aux divers éléments de la réponse;
soit une présentation en un bloc, c'est-à-dire sans les textes réponse.

Une **première boîte de dialogue** apparaît avec la liste des noms de sujets testés existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du sujet testé dont vous voulez consulter une réponse à un test et cliquez sur le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans consulter de réponse, cliquez "Cancel".

Ensuite, une **deuxième boîte de dialogue** apparaît avec la liste des tests exécutés par le sujet testé choisi. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement.
Sélectionnez le nom du test dont vous voulez consulter une réponse et cliquez sur le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans consulter de réponse, cliquez "Cancel".
Si vous avez choisi une sortie "imprimante", la réponse s'imprime normalement.

Si vous avez choisi une sortie "écran", une **troisième boîte de dialogue** apparaît qui propose une présentation en un bloc ou une présentation progressive. Cliquez le bouton qui correspond à votre choix.

- si vous avez choisi une présentation progressive :

  Une **fenêtre réponse** apparaît et les divers éléments de la réponse se dessinent un à un. Lorsque s'affiche un élément auquel un texte réponse est associé, une boîte de dialogue apparaît; elle demande si vous souhaitez consulter le texte réponse associé. cliquez le bouton qui correspond à votre choix.
  Si vous avez demandé la consultation du texte réponse, une **fenêtre texte** apparaît contenant le texte réponse. Cette fenêtre se manipule de façon standard. Lorsque vous avez terminé la consultation de la réponse, cliquez dans la "close box" de la fenêtre et l'affichage des éléments de la réponse reprend son cours.
  Lorsque tous les éléments ont été affichés et que vous avez terminé la consultation de la réponse, cliquez dans la "close box" de la fenêtre.

- si vous avez choisi une présentation en un bloc :

  Une **fenêtre réponse** apparaît et les léments de la réponse s'affichent. Cette fenêtre se manipule de façon standard. Lorsque vous avez terminé la consultation de la réponse, cliquez dans la "close box" de la fenêtre.

* *Textes*

Sélectionner cet item permet de consulter les textes associés à une réponse de sujet testé à un test.
Un **texte réponse** se compose d'un seul paragraphe, contenant un titre facultatif et un contenu.

Si vous avez choisi une sortie "imprimante", les textes s'impriment, normalement.

Si vous avez choisi une sortie "écran",

Une **fenêtre texte** apparaît contenant la suite des textes réponses. Cette fenêtre se manipule de façon standard. Lorsque vous avez terminé la consultation des textes, cliquez dans la "close box" de la fenêtre.

* *Chronologie*

Sélectionner cet item permet de consulter la liste des éléments choisis pour constituer la réponse, classés par ordre chronologique de leur insertion dans la réponse.

Si vous avez choisi une sortie "imprimante", la liste des éléments s'imprime, normalement.

Si vous avez choisi une sortie "écran", une fenêtre réponse apparaît contenant la suite des éléments constituant la réponse. Cette fenêtre se manipule de façon standard. Lorsque vous avez terminé la consultation, cliquez dans la "close box" de la fenêtre.

* *Introduction Pédagogue*

Sélectionner cet item permet d'ajouter un nouveau pédagogue à la liste des pédagogues existants : une **boîte de dialogue** demande d'introduire un nouveau nom de pédagogue et affiche la liste des noms de pédagogues qui existent déjà. Le nom d'un pédagogue doit contenir entre 1 et 6 caractères. Si ce n'est pas le cas ou si le nom introduit existe déjà, un message vous le signale.
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".

* *Suppression Pédagogue*

Sélectionner cet item permet de supprimer un pédagogue de la liste des pédagogues existants : une **boîte de dialogue** apparaît avec la liste des noms de pédagogues existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du pédagogue que vous voulez supprimer de la liste des pédagogues existants et cliquez sur le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une **seconde boîte de dialogue** vous demande de confirmer la suppression du nom de pédagogue choisi. Cliquez le bouton "Oui" ou le bouton "Non" selon votre choix.

* *Suppression Test*

Sélectionner cet item permet de supprimer un test, c'est-à-dire de supprimer son nom de la liste des tests existants, de supprimer les matériaux associés ainsi que toutes les réponses fournies à ce test. Une **boîte de dialogue** apparaît avec la liste des noms de tests existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du test que vous voulez supprimer et cliquez le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans rien changer, cliquez "Cancel".
Une **seconde boîte de dialogue** vous demande de confirmer la suppression du nom de test choisi. Cliquez le bouton "Oui" ou le bouton "Non" selon votre choix.

* *Suppression Réponse*

Sélectionner cet item permet de supprimer la réponse d'un sujet testé à un test : Une **première boîte de dialogue** apparaît avec la liste des noms de sujets testés existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du sujet testé dont vous voulez supprimer une réponse et cliquez le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans rien supprimer, cliquez "Cancel".

Une **seconde boîte de dialogue** apparaît avec la liste des noms de tests exécutés par le sujet testé choisi. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du test dont vous voulez supprimer la réponse fournie par le sujet testé choisi et cliquez le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans rien supprimer, cliquez "Cancel".

Une **troisième boîte de dialogue** vous demande de confirmer la suppression de la réponse du sujet testé choisi au tests choisi. Cliquez "Oui" ou "Non" selon votre choix.

* *Suppression Réponse Personne*

Sélectionner cet item permet de supprimer toutes les réponses fournies par un sujet testé aux tests qu'il a réalisés : Une **première boîte de dialogue** apparaît avec la liste des noms de sujets testés existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du sujet testé dont vous voulez supprimer toutes les réponses et cliquez le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans rien supprimer, cliquez "Cancel".
Une autre **boîte de dialogue** vous demande de confirmer la suppression de toutes les réponses du sujet testé choisi aux différents tests qu'il a exécutés. Cliquez "Oui" ou "Non" selon votre choix.

* *Suppression Réponse Test*

Sélectionner cet item permet de supprimer toutes les réponses fournies à un test par tous les sujets testés qui l'ont exécuté : Une **boîte de dialogue** apparaît avec la liste des noms de tests existants. Vous pouvez vous déplacer dans la liste grâce aux barres de défilement. Sélectionnez le nom du test dont vous voulez supprimer toutes les réponses de sujets testés et cliquez le bouton "OK".
Si vous voulez remettre le logiciel dans l'état précédent sans rien supprimer, cliquez "Cancel".
Une autre **boîte de dialogue** vous demande de confirmer la suppression de toutes les réponses de sujets testés au test choisi. Cliquez "Oui" ou "Non" selon votre choix.

* *Clôture*

Cet item permet de quitter l'option Analyse et de replacer la barre de menus dans son état initial, telle qu'elle était au lancement de l'application.

4.LES MESSAGES D'ERREURS

De façon générale, lorqu'une erreur s'est produite au cours d'une opération, une **première boîte d'alerte** apparaît avec un message expliquant l'erreur (message d'erreur) et 2 boutons : un bouton "OK" et un bouton "Help". Si le message d'erreur ne vous paraît pas clair, cliquez le bouton "Help" et vous verrez apparaître une **seconde boîte d'alerte** contenant un message plus détaillé et un bouton "OK". Le bouton "OK" sert à indiquer au logiciel que vous avez pris connaissance du message et que vous souhaitez continuer.

Voici la liste des messages d'erreurs et leur explication :

* *Nom Existant*

  Vous avez introduit un nom qui ne fait pas partie de la liste des noms existants alors que l'on demandait un nom existant.
  Ce message apparaît lorsque le nom de pédagogue que vous introduisez pour accéder à l'option Introduction ou à l'option Analyse n'existe pas.

* *Nom Inexistant*

  Vous avez introduit un nom qui fait déjà partie de la liste des noms existants alors que l'on vous demandait un nom encore inexistant.
  En général, la boîte de dialogue qui saisit le nouveau nom contient aussi la liste des noms existants que vous pouvez consulter en vous déplaçant grâce aux barres de défilement.

* *Nom Trop Court*

  Vous avez tapé "Return" ou cliqué "OK" sans avoir introduit aucun caractère. Or tout nom doit contenir au moins un caractère.
  Si vous souhaitez quitter la boîte de dialogue sans introduire un nom, cliquez "Cancel".

* *Nom Trop Long*

  Vous avez introduit un nom ayant un nombre de caractères supérieur au nombre autorisé. Lorsque l'on revient à la saisie du nom (en cliquant "OK"), ce nom apparaît avec les premiers caractères en nombre autorisé.
  De façon générale, les noms de tests, de pédagogues, de sujets testés, les identifiants courts de matériaux et les identifiants de paragraphe peuvent contenir au plus 6 caractères. Les titres de paragraphes et les identifiants longs de concepts peuvent contenir au plus 20 caractères.

* *Titre sans contenu*

Vous avez introduit une entête ou un paragraphe avec un titre non vide et un contenu vide. Or, si une entête ou un paragraphe a un titre, elle (il) doit aussi avoir un contenu non vide.
Vous avez alors deux solutions :

soit vous annulez le titre et votre texte n'a pas d'entête (s'il s'agit de l'entête)

soit vous introduisez à nouveau un contenu d'entête ou de paragraphe non vide.

* *Texte sans paragraphe*

Vous avez introduit un paragraphe vide comme premier paragraphe du texte c'est-à-dire que vous avez tapé directement Command-Return pour le titre et pour le contenu du premier paragraphe.
Si il s'agit du texte de travail, vous devez introduire à nouveau un paragraphe non vide.
Si il s'agit du texte explicatif ou d'un texte réponse, vous pouvez choisir : soit vous introduisez à nouveau un paragraphe non vide; soit vous annulez la saisie du texte, puisque ces deux textes sont facultatifs.

* *Réponse incomplète*

Vous avez cliqué dans la boîte de fermeture (close box) de la fenêtre réponse; cliquez dans la boîte de fermeture signifie que vous avez terminé d'introduire la réponse.
Or il reste au moins un concept ou un sommet de paragraphe qui ne fait pas partie de la réponse. Et, pour être complète, une réponse doit contenir tous les concepts ou tous les sommets de paragraphe proposés.
Sélectionnez l'item "ajouter concept" (pour un test de type réseau) ou "ajouter sommet" (pour un test de type liaison ou hiérarchisation de paragraphes) pour ajouter à la réponse les concepts ou les sommets de paragraphe restants.

* *Si vous clôturez maintenant, vous risquez de perdre le test...*

Vous avez sélectionné l'item "clôture" alors que tous les éléments obligatoires n'ont pas été introduits. Vous pouvez clôturez si vous le souhaitez, mais alors les éléments déjà introduits ne seront pas sauvegardés.
Pour l'option Intro, vous ne pouvez clôturer que lorsque la réponse type complète a été introduite. Pour l'option Execution, vous ne pouvez clôturer que lorsque la réponse complète a été introduite.

* *Vous avez déjà exécuté ce test. Voulez-vous le recommencer?*

Vous avez sélectionné un nom de test pour lequel vous déjà fourni une réponse. Attention, si vous choisissez de recommencer ce test, la réponse précédente est alors perdue.

TABLE DES MATIERES

PARTIE II : APPLICATION DE LA METHODE A UN LOGICIEL : CAS D'UN LOGICIEL DE TESTS DE LA MEMOIRE HUMAINE