

**DESIGN AND IMPLEMENTATION OF A TOOL  
FOR TEACHING PROGRAMMING**

**A THESIS**

**Submitted To The Department Of Computer  
Engineering And  
Information Sciences  
And The Institute Of Engineering And Sciences  
Of Bilkent University  
In Partial Fulfillment Of The Requirements  
For The Degree Of  
Master Of Science**

**By**

**Mesut Göktepe**

**September 1988**

QR  
26.6  
.118  
G5G1  
1988

# DESIGN AND IMPLEMENTATION OF A TOOL FOR TEACHING PROGRAMMING

A THESIS  
SUBMITTED TO THE DEPARTMENT OF COMPUTER  
ENGINEERING AND  
INFORMATION SCIENCES  
AND THE INSTITUTE OF ENGINEERING AND SCIENCES  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
MASTER OF SCIENCE

By

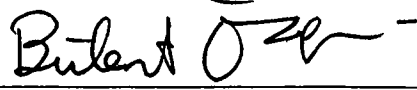
Mesut Göktepe

September 1988

*Mesut Göktepe*  
tarafından bağlanmıştır.

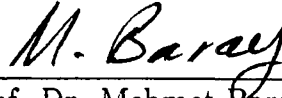
QA  
76.6  
.I 18  
G 561  
1988  
84857

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



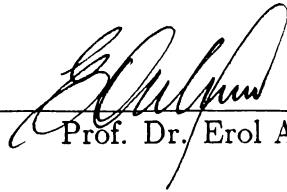
Assoc. Prof. Dr. Bülent Özgüç (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Prof. Dr. Mehmet Baray

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Prof. Dr. Erol Arkun

Approved for the Institute of Engineering and Sciences:



Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Sciences

# ABSTRACT

## DESIGN AND IMPLEMENTATION OF A TOOL FOR TEACHING PROGRAMMING

Mesut Göktepe

M.S. in Computer Engineering and  
Information Sciences

Supervisor: Assoc. Prof. Dr. Bülent Özgüç  
September 1988

In this thesis, a survey on computer applications for teaching programming and some graphical programming tools together with their underlying environments has been carried out and a graphical Pascal teaching tool is designed and implemented.

Graphical programming tools provide the user the ability to solve the problems through the use of icons and symbols allowing very little text. Here, a Pascal teaching tool is presented in a very user friendly environment to teach programming through the use of flowcharts in a visual manner.

Keywords: graphical programming, user interface, window manager

# ÖZET

## BİLGİSAYAR PROGRAMLAMA EĞİTİMİ İÇİN BİR YAZILIM

Mesut Göktepe

Bilgisayar Mühendisliği ve Enformatik Bilimleri Yüksek Lisans

Tez Yöneticisi: Assoc. Prof. Dr. Bülent Özgüç

Eylül 1988

Bu tezde bilgisayar programlama öğretimi için uygulamalar, bazı grafiksel programlama sistemleri ve uygulandığı ortamlar araştırılmış, ve grafiksel bir Pascal programlama öğretim sistemi tasarlanıp uygulaması yapılmıştır.

Grafiksel programlama sistemleri kullanıcıya imge ve semboller yoluyla çok az yazı kullanarak problem çözme olasılığı sağlar. Burada, bir Pascal programlama sistemi dost bir etkileşim ortamı içinde akış diagramları kullanarak görsel olarak programlamayı öğretmek amacıyla sunulmuştur.

Anahtar kelimeler: grafiksel programlama, etkileşim sistemleri, pencere yönetim sistemi

## ACKNOWLEDGEMENT

I would like to acknowledge first the help and cooperation of my supervisors Assoc. Prof. Dr. Bülent Özgüç and Prof. Dr. Mehmet Baray without whom this work could not have been completed. I would like to thank Ahmet Coşar, Aydın Kaya, and Murat Karaorman for their patient suggestions and comments. I also wish to thank Emine Oskargil for her moral support during the preparation of this thesis. Attila Gürsoy and Özgür Ulusoy have also been very helpful by their remarks and suggestions.

# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Computer Based Teaching For Programming . . . . .	2
1.2	Graphical Programming vs. Textual Programming . . . . .	3
1.3	Survey of Graphical Programming . . . . .	5
1.3.1	Systems that are Based on Conventional Programming Languages . . . . .	5
1.3.2	Systems Implementing Programming Languages which are Completely New and Based Completely on Graphics (with very little text) . . . . .	8
1.4	Some Graphical Programming Systems . . . . .	9
1.4.1	PECAN . . . . .	9
1.4.2	IconLisp . . . . .	10
1.4.3	PICT . . . . .	11
1.4.4	HiVisual . . . . .	13
1.4.5	'A Tool for Teaching Programming' . . . . .	14
<b>2</b>	<b>UNDERLYING ELEMENTS AND ENVIRONMENT FOR THE DESIGN AND IMPLEMENTATION OF A TOOL FOR TEACHING PROGRAMMING</b>	<b>17</b>
2.1	User Interface . . . . .	17



2.2	Object Oriented Approach . . . . .	19
2.3	Environment . . . . .	21
2.4	Tools For User Interface . . . . .	22
2.5	A Notification Based System . . . . .	24
2.5.1	The Notifier . . . . .	24
2.5.2	Relationships Between the Notifier, Objects, and the Application . . . . .	27
<b>3</b>	<b>DESIGN OF A TOOL FOR TEACHING PROGRAMMING</b>	<b>28</b>
3.1	Naming Conventions . . . . .	28
3.2	Screen Design of the Tool . . . . .	28
3.3	Other Screen Layout Elements Displayed by Flowchart Han- dling Operations . . . . .	31
3.4	Items of the Flowchart Menu . . . . .	34
3.4.1	Flowchart symbols . . . . .	34
3.4.2	Flowchart Handling Operations . . . . .	35
3.5	Flowchart Design Methodologies . . . . .	37
<b>4</b>	<b>IMPLEMENTATION OF A TOOL FOR TEACHING PRO- GRAMMING</b>	<b>42</b>
4.1	Data Structures . . . . .	42
4.2	Basic Routines Used for Implementation . . . . .	46
4.3	Low Level Operation Of The Tool . . . . .	52
4.3.1	Code Generation in Low Level . . . . .	53
4.3.2	Code Execution in Low Level . . . . .	53
4.4	Unimplemented Features . . . . .	55

<b>5</b>	<b>CONCLUSION</b>	<b>57</b>
<b>A</b>	<b>USER'S MANUAL</b>	<b>59</b>
A.1	Generation of a Flowchart . . . . .	59
A.1.1	Creating a New Flowchart . . . . .	59
A.1.2	Editing a Flowchart . . . . .	63
A.1.3	Redisplaying a Flowchart . . . . .	66
A.1.4	Saving a Flowchart to a File on the Disk . . . . .	66
A.1.5	Loading a Flowchart from a File on the Disk . . . . .	66
A.1.6	Erasing a Flowchart from the Screen . . . . .	67
A.2	Code Generation . . . . .	67
A.2.1	Creating the Flowchart for Main Body of a Program .	67
A.2.2	Creating Flowcharts for Subprograms of a Program .	67
A.2.3	Generating the Pascal Code Corresponding to a Flowchart	71
A.3	Execution of the Pascal Code for a Flowchart . . . . .	77
A.3.1	Stepwise Execution of the Code . . . . .	79
A.3.2	At Each Step Highlighting the Corresponding Flowchart Box . . . . .	81
A.3.3	At Each Step Displaying the Values of Simple Vari- ables of the Currently Executing Routine . . . . .	85
A.4	Leaving the Tool . . . . .	85
A.5	Miscellaneous . . . . .	85

## LIST OF FIGURES

1.1	Hierarchical classification of graphical programming tools. . .	6
1.2	A snapshot of the screen. . . . .	14
2.1	Class hierarchy of a window manager system. . . . .	23
2.2	Control structure of a notifier based system. . . . .	26
3.1	Screen layout of the tool. . . . .	29
3.2	Flowchart drawing area. . . . .	30
3.3	Flowchart menu. . . . .	30
3.4	Panels. . . . .	31
3.5	Text edit window. . . . .	32
3.6	File name window. . . . .	32
3.7	Scale window. . . . .	32
3.8	Declaration window. . . . .	33
3.9	Code display window. . . . .	33
3.10	Step window. . . . .	34
3.11	Connection of arcs. . . . .	38
3.12	if-then-else. . . . .	38
3.13	while-do. . . . .	38

3.14 repeat-until. . . . .	39
3.15 for. . . . .	39
3.16 start/stop/procedure-start/return. . . . .	40
3.17 subroutine call. . . . .	40
3.18 output/display. . . . .	40
3.19 process. . . . .	40
3.20 disk i/o. . . . .	40
3.21 Only one entry and one exit point. . . . .	41
4.1 Structure for box coordinates. . . . .	43
4.2 Structure for box pins. . . . .	43
4.3 Box structure. . . . .	44
4.4 Text structure. . . . .	45
4.5 Box pointers. . . . .	46
4.6 Array of sub-flowcharts . . . . .	47
4.7 Flowchart stack. . . . .	47
4.8 Low Level Operation of the Tool. . . . .	54
A.1 Picking and placing boxes. . . . .	60
A.2 Filling in the boxes. . . . .	61
A.3 Connection of boxes. . . . .	62
A.4 Moving a box. . . . .	64
A.5 Scaling a box. . . . .	65
A.6 Main flowchart. . . . .	68
A.7 Sub-flowchart. . . . .	69

A.8	Generating the Pascal code. . . . .	72
A.9	A file name for the Pascal source code. . . . .	73
A.10	Declaring data types in the main body. . . . .	74
A.11	Declaring data types in the main body. . . . .	75
A.12	Declaring data types of the sub-flowchart of the previous figure. . . . .	76
A.13	Displaying the Pascal source code. . . . .	78
A.14	Execution of the code. . . . .	79
A.15	Erroneous code and error messages. . . . .	80
A.16	'STEP'ping. . . . .	81
A.17	End of 'STEP'ping while printing values of simple variables. . . . .	82
A.18	'STEP'ping in a main flowchart. . . . .	83
A.19	'STEP'ping in a sub-flowchart. . . . .	84

# 1. INTRODUCTION

Since the beginning of Computer Science, the main trend on the development of systems has been in the direction of how computer systems can be designed to accommodate the user needs in an easy, fast, and the most natural fashion. The most common approach concerning the human-computer interaction has been in the field of programming languages. Many programming languages are developed to build better user interfaces to simplify the computer use, teach computer programming and increase the programmer productivity.

As the problem domain to be solved on the computer increases, inadequacies of common sequential, procedural, text based programming languages arise. A radical departure from current programming styles becomes a necessity. In order to make the programming more accessible, new, improved programming languages have been created with the capabilities of handling specific types of problems. For example, object oriented languages deal with the problem with objects and various actions that can be performed on that object. Generally common programming languages and environments are quite difficult to use because they are not friendly and very adaptable to users, especially for ones new in computer programming. This lack of adaptation exists both at the representation level -in the graphical layout seen by the user- and at the conceptual level -in the computational model-. Graphical programming systems based on interactive computer graphics have been working on the former problem. Research in the functional programming (Lisp) and logic programming (Prolog) environments have been attempting to find remedies for the latter problem. Artificial Intelligence research has been doing extreme work on techniques for communicating with computer in a natural language. Currently, a great deal of work on expert systems has been attempting to simplify the human-computer interaction by making deductions within the given application domains. Graphical programming systems are the most user friendly tools for programming environments, especially ones used for teaching programming.

In the following sections, topics related to computer based teaching for 'programming' and major systems which support programming with graphics are surveyed to exemplify the various approaches of graphical programming. Then follows the discussion of our 'Pascal teaching tool' in comparison with the related systems to explain the matter in detail. Finally, a particular attention is paid to 'Design and Implementation of a tool for teaching programming' describing its usage, components, user interface, higher programming constructs applied, implementation environment, and low level implementation details. A list of its unimplemented features are given at the end.

## 1.1 Computer Based Teaching For Programming

The way one is to follow for programming with a classical textual language goes through a number of steps. Once the problem to be solved by a computer is analyzed in detail, the most important step, selection of an algorithm and derivation of the flowchart for the algorithm comes next. These steps may lead to an efficient and correct program for solving the problem. An algorithm gives the logical steps of a program in textual form so that a program can be generated from it. The flowchart presents the same information graphically. The structure of a program is derived by dividing the problem into logically complete blocks - subproblems- and analyzing each separately. After defining the structure, the code for the program can be written easily, ideally through a structured language. Now comes the time for running the program and waiting for the results. If the results seem incorrect, it is required to figure out where the 'bugs' are , then modify the program and rerun again.

Pascal facilitates writing structured programs, programs that are relatively easy to read, understand, manipulate, and maintain. It is currently one of the most widely used language for teaching programming. Its popularity is due to the fact that the syntax is relatively easy to learn.

The goal of structured programming is to organize and discipline the program design and coding process in order to prevent most logic errors and to detect ones that remain. Structured programming concentrates on one of the most error-prone factors of programming, the logic. It has three major characteristics: top-down design, modular programming, structured coding.

In top-down design, first the program is specified in the broadest outline and then, the structure is gradually refined to fill in the details. In each step

a given task is broken down into a number of subtasks until the subtasks are simple enough to be coded with a highly reliable process of programming. Modular programming is the process of dividing a program into logical parts called modules and then successively programming each part. Here, size of the modules must not be too large and each module must be independent from others. Structured coding is a method of writing programs with a high degree of structure providing for understandable programs for testing, maintenance and manipulation. Any proper program -a program with one entry and one exit point and no infinite loops or unreachable code- can be accomplished only by the logic structures of sequence of two or more operations, selection of one or two operations, and iteration of an operation while a condition is true.

Pascal is a structured language facilitating the mentioned characteristics of structured programming. It has BEGIN-END blocks, PROCEDURES, and FUNCTIONS providing constructs for modular programming. Pascal executes the code from top to bottom in sequence, enabling the sequential processing. REPEAT-UNTIL, WHILE-DO, and FOR statements allow iteration over a number of operations. IF-THEN-ELSE, and CASE statements facilitate a selection mechanism for structured programming [9].

This is the way how one commonly programs the computer through a textual language. The use of pictures in programming is also another methodology for programming which differs from the classical textual programming.

A sketch of the graphical approach for programming can be outlined as follows: First, selection of images that visually represent the data structures and variables needed. Next, drawing the desired algorithm as a logically structured, multidimensional picture. Last, watching the program run and the results being generated. If the program is not doing what is expected, identifying where and when the error(s) occur, visually.

## **1.2 Graphical Programming vs. Textual Programming**

In general, programming environments can be classified as textual -Pascal, Cobol, Lisp etc.- and graphical, graphical environments can further being divided as visual or iconic, according to the degree to which they employ text as opposed to graphics. These categories are not disjoint, however.

Some of the features of images that make them potentially powerful and



useful tools in the computational environment are :

- First, images provide a natural supplement to other means of communication. People gather information at a higher rate by discovering graphical relationships in pictures than by reading ordinary text.
- Images are often easily learned, retained, and recalled as single units of information, since the human mind is strongly visually oriented.
- Images may possess a universality that natural languages and their associated programming languages do not. For example, standardized signs in many applications can be understood by people who speak many languages, such as traffic signs.
- The rapidly falling cost of graphics-related hardware with their increasing performance show that images are becoming more and more suitable for human-machine communication.
- When one programs in a textual language, one is forced to learn its syntax. On the other hand in programming with graphics, pictures keep associated syntax and semantic rules allowing them to represent language elements.
- Text can be thought of as a one dimensional stream of characters [14]. Indentation and bold-facing of keywords are often applied in text to help making text multi-dimensional. Pictures, on the other hand, inherently provide two or three dimensions. In addition, other properties of pictures such as shape, size, color, texture, direction, etc. provide further information.
- Pictures are used to illustrate abstract ideas and make them simple to think about. All well-drawn pictures provide good metaphors, using text alone forces the reader to come up with his own mental image.

While all of the above are certainly advantages of graphical programming, there are some disadvantages. One can argue, for example, that as a programming language becomes more friendly and easy to use, it also becomes more specialized and its domain decreases [15]. As the level of programming increases, the domain for which programs can be written decreases. For example, in a menu based system, the user has very few options (limited domain) but it is certainly very easy to use.

## 1.3 Survey of Graphical Programming

Graphical programming can be defined as any kind of visual interaction between a computer and a human in order to complete some programming task. This interaction, however, does not only involve manipulation of text or programs written in a linear string of characters, but rather, of pictures or symbols of two-dimensional objects such as graphs, maps, flowcharts, etc. Thus, two-dimensional picture or pictures are used to represent the program.

Recently, many graphical tools have been developed to provide the programmer and the user the ability to solve the problems through the use of icons and symbols allowing a little text.

In reality, the divisions among the various tools are not as discrete as Figure 1.1 indicates. In general, the various systems that implement a graphical programming environment can be divided into two major groups:

### 1.3.1 Systems that are Based on Conventional Programming Languages

These tools implement a graphical programming 'layer' on top of conventional programming languages like Pascal or Lisp. So, underlying graphics environment is a textual programming language. The visibility of this textual language to the user depends on the particular tool. Some tools merely offer graphics as another way to view a fundamentally textual program. Other tools give more flexibility to the user by allowing him/her to completely create and debug a program graphically. The tools which are based on conventional programming languages can be further subdivided into the following categories:

#### Tools which Show Control Flow/Program Structure Graphically

- 'Flowcharts'

The most common method of showing control flow of a program is by a flowchart. Flowcharts were originally developed as a tool for assembly language programming, which allows completely unstructured control transfers. While flowcharts do allow a graphical representation of the control mechanism, the disciplined control constructs, scoping, and data

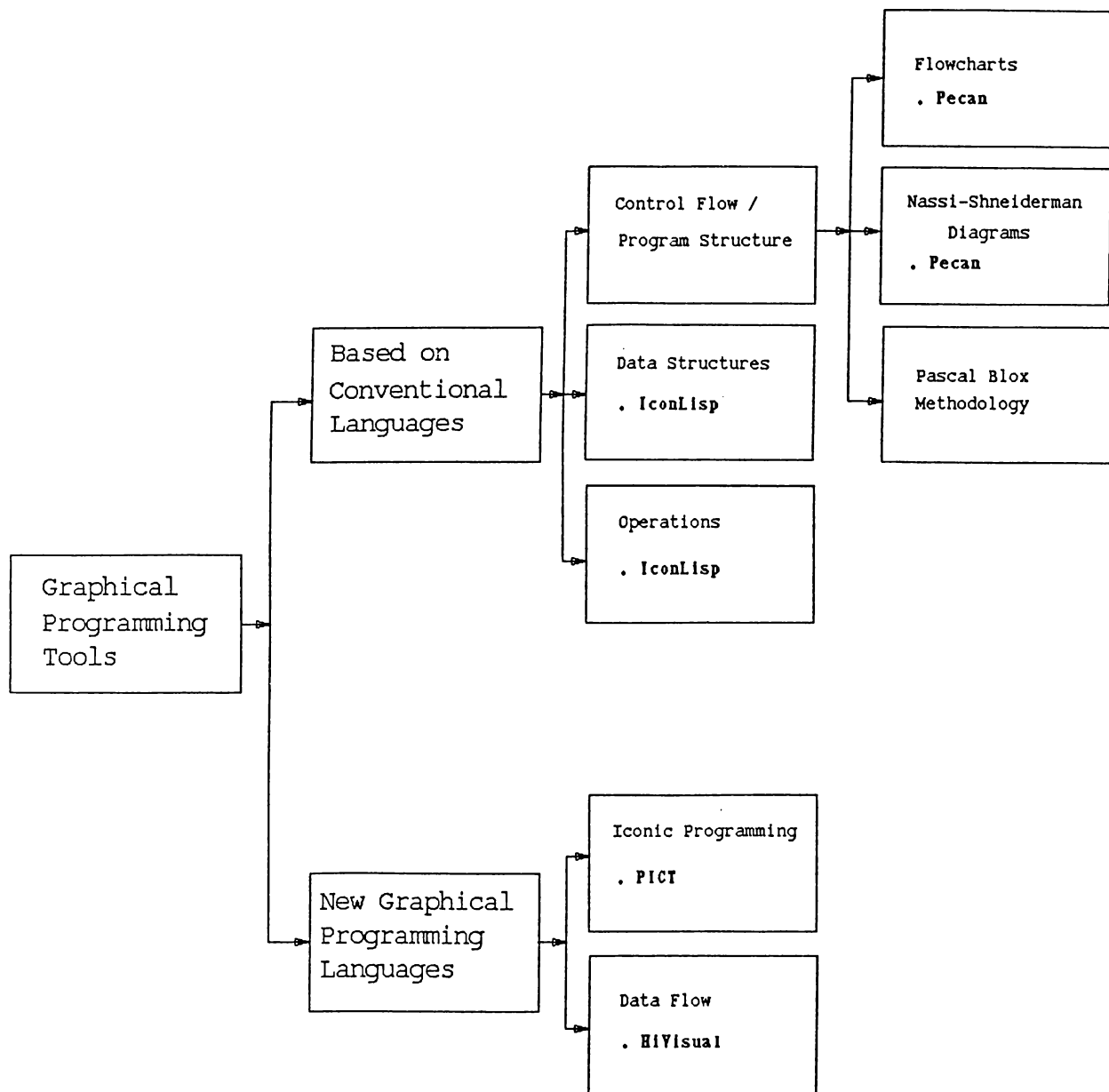


Figure 1.1: Hierarchical classification of graphical programming tools.

structures of present-day programming languages render them obsolete as a graphical programming tool. Even with such features though, flowcharts cannot be used to display data structures of programs. They also need to be modified to allow common control constructs such as for loops, while loops, repeat/until loops, etc. Also the diagrams still are mostly text.

- 'Nassi-Shneiderman diagrams'

Nassi-Shneiderman diagrams allow the structure of a program to be displayed graphically. These types of diagrams when combined with syntax-directed editors provide a systematic (top-down) method for building programs. While these types of programs are useful for showing program structure, they do not show expressions, procedure calls, types, and data graphically. The diagrams still are mostly text.

- 'Pascal Blox Methodology'

The Pascal Blox Methodology [16] developed by Glinert represents program structure by using 'jigsaw'-like blocks in an interconnected fashion. Each jigsaw piece represents a particular construct in the language. The pieces are designed in such a way that the only way to combine the various types of structures is according to the syntax rules of the language. Thus, the user creates entire program in a manner analogous to constructing jigsaw puzzles.

## **Systems which Show the Data Structures in the Language Graphically**

Surprisingly, there has been very little work on ways to display data structures graphically. One of the reasons for this is that data is usually very complex compared to control flow. Another reason for this is that data is often represented at many different levels of abstraction. Thus, it is difficult for the computer to determine what exactly a given data structure represents. For example, a graph may be represented by an adjacency matrix. But, unless told explicitly, the computer has no idea that the adjacency matrix represents a graph. For this reason, it can not represent this information graphically.

## **Tools which Show the Actual Operations in the Language Graphically**

IconLisp, in addition to representing data structures graphically, allows the actual operations to be performed graphically. So, the user uses a mouse attached to the computer to 'drag' an icon of an input list on top of an icon of a function. The computer computes the output list and represents the list by creating a new icon for the output list. Thus, the actual operation of applying a function to some input is done graphically. Again, this is simple in the case of Lisp because of the functional nature of the language. Such a technique would be difficult with Pascal since the language is procedural.

All of the tools described above are based on conventional programming languages like Pascal and Lisp.

### **1.3.2 Systems Implementing Programming Languages which are Completely New and Based Completely on Graphics (with very little text)**

#### **'Iconic Programming'**

Some tools allow complete programs to be written by symbols called Icons. By putting these icons together, a runnable program is created. Once the program is created, it can be run directly without being converted to another textual language. The method of creating programs is so free of text that according to the creators the keyboard is never used! However, even though this may seem nice at first, one can imagine how cumbersome the process of creating programs may get if the mouse is used for everything.

#### **'Data Flow Languages'**

In a dataflow graph, the flow of data and control coincide. These languages allow the user to create dataflow graphs on the screen and then specify inputs which are fed to this graph to produce a set of outputs. An advantage of these types of languages is that they are well suited for program animation. The user can watch the 'dataflow' from one module to another. The simple structure of dataflow graphs encourages modular design and can be used at

all levels of system description. Such diagrams are also useful in identifying potential units which can be executed in parallel. However, one problem this scheme shares with flowcharts is that the lack of high level control constructs usually lead to messy diagrams.

## 1.4 Some Graphical Programming Systems

### 1.4.1 PECAN

The PECAN [17] family of programs are program development systems developed at Brown University. PECAN provides users with a rich programming and run-time environment that allows graphical views of the user program, its semantics, and its execution. In PECAN, for example, users for the most part indicate the sequence of actions they wish to perform by pointing with a mouse to entries in fixed or pop-up menus. The built-in editor recognizes Pascal syntax and thus can immediately highlight erroneous statements. At run-time, users can follow the computer's progress through the program with each statement being highlighted as it is executed. The system was developed to run on APOLLO workstations. The main objective of the system is to provide a graphical programming environment for the novice and experienced programmers. PECAN contains many of the best features of other similar systems. These include:

- immediate feedback of syntax and semantic errors while the user is editing the source program (syntax-directed editor),
- an undo facility whereby the user can undo and redo any action back to the beginning of the session,
- structured templates for building the program,
- the flexibility to type text at any time instead of using templates,
- the use of on-screen and pop-up menus as an alternative to typing most commands,
- a multiple window display to make effective use of the screen,
- incremental semantics that allows the program to be compiled as it is edited,

- a framework that handles a variety of (algebraic) programming languages with the same commands.

The PECAN system differs from other program development systems in its use of multiple views of user programs. One such view is a syntax-directed editor. This view displays the user program with a format for automatic indentation and boldfacing of Pascal keywords. Another view displays the program as a Nassi-Shneiderman diagram. A third view shows a module inter-connection diagram showing how the program is organized. Others are control flow diagram, expression tree etc. Those several views may be displayed simultaneously, each in its own window.

In addition to the different program views, PECAN allows the user program to be compiled incrementally. The user can also view the symbol tables, data types, expression trees, and the control-flow graph of the program in separate windows.

According to the classification discussed earlier, PECAN fits into the category of programming environments which are based on conventional programming languages. This is due to the fact that the entire system is based on Pascal. In general, PECAN is a complete development system for Pascal programs. Actually, the system is modular enough to allow other languages to be supported. One of the drawbacks of the system, though, is that the actual data structures cannot be rendered graphically. Also, the user cannot create the program graphically; he/she must create the textual Pascal program first, and then may choose to view it in different ways.

### 1.4.2 IconLisp

IconLisp [18] is a system designed to offer an iconic extension to the standard programming in Lisp. The system consists of a row of pull-down menus, a palette of tools on the lefthand side of the screen, and a working space which occupies the rest of the screen. The user with an input device such as a mouse can then use the menus and palette of tools to create lists and thus create programs (lists and programs are the same thing in Lisp). The IconLisp system was still in the design phase and thus had not been actually implemented on a computer. Nevertheless, the designed version of the system will be used to talk about its capabilities.

The IconLisp system implements iconic programming in the following way:

The user clicks on and drags objects on the screen to create and modify lists. For example, once an initial list is created, the user selects the list and clicks on the CDR icon in the palette area to produce the CDR (all except the first part) of the list. This action produces a new icon on the screen which contains the CDR of the list and which can be used in a further computation.

In order to implement iconic programming, the IconLisp system, for each list (function), keeps track of three components: the physical component, the logical component, and the name. The physical component is the actual picture/icon to represent the list. The logical component is the actual text of the list. The name component is used to label the icon and is normally the same as the function name. By using these three components, users can create arbitrary functions and assign icons to them. With these icons, the user can execute programs by creating the data lists and then dragging them on top of the appropriate function icons in the user workspace. Also different buttons on the mouse represent different actions. The first mouse button selects a function/list and opens a window which contains its textual definition. The second mouse button evaluates (calls the EVAL function in Lisp) the selected list.

The IconLisp system can be classified among those systems which are based on conventional programming languages, in this case Lisp. The system not only allows programs to be displayed graphically, but the data structures are also displayed graphically. Even the operations are done by graphical manipulation. Of course, this is very simple to do because of the functional nature of Lisp. Also, since programs and data are in the same format (lists) in Lisp, graphic representation is very easy. The IconLisp system is nice in that programs, data structures, and operations are all represented graphically. However, underlying this graphic layer, the user still needs to deal with the text of the Lisp functions and data. Also the system is nice in design but has not been actually implemented to demonstrate the practicality of such an approach.

### 1.4.3 PICT

PICT [19] is a system developed by Glinert and Tanimoto at the University of Washington. The system was primarily designed to aid in program implementation rather than algorithm design and selection. The system, however, does not allow general programming, it serves as an experimental prototype



capable of supporting small, but nontrivial programs.

Each of the systems previously described provide many of the features which are important to graphical programming. However, all of them suffer from the following two problems:

- users have to use one of the standard programming languages to code their programs, and
- the display is monochrome; color graphics is not used. PICT solves these two problems.

The PICT system provides a new type of algorithmic programming environment in which computer graphics plays a central role. With PICT, users do not write computer programs using letters, digits, and punctuation marks; in fact, users never use the keyboard. The user uses the mouse and a color display terminal to create his/her program. The program creation is not free form, but rather like constructing jigsaw puzzles. Graphics, color, sound, and animation are used to make the program appear as multidimensional and concrete as possible. With the PICT system, programs, subprograms, parameters, data structures, variables and program operations are all represented by icons of various sorts. The control structure is represented by paths that are followed by the system to show the current execution stage. Also, the syntax of the user's program is checked continuously during program construction, thus eliminating the need for syntax checking before executing the program.

PICT provides all tools users need to compose, edit, and run their programs integrated within a simple, consistent command structure. Users communicate with the PICT system by pointing to icons in a menu tree; PICT responds by altering its display in an appropriate manner or, if the user has made an error, by displaying a help message.

Unlike PECAN, PICT is not based on a textual language. The language is completely iconic and needs no text information at all. Thus, in the classification scheme, PICT falls in the category of tools which implement completely new and graphical programming languages. Specifically, PICT is an example of an iconic programming language.

PICT has the advantage over other systems in that it is completely graphical. In fact, one can even claim it is too graphical. According to a survey

conducted by the developers, many experienced programmers thought that such a system would be too cumbersome to work with. On the other hand, most beginners really appreciated using the system and actually enjoyed using the system to create programs.

#### 1.4.4 HiVisual

HiVisual (Hiroshima Visual) [20] is a graphical programming system developed at the Hiroshima University in Japan. In HiVisual, programming is carried out by simply arranging icons on the screen. The system features icon-based programming, visualization of data flow, interactive programming capability, and animation of program execution.

HiVisual, like PICT, supports iconic programming. In PICT, however, the control of program flow is visually specified on the basis of its flowchart-like representation. In contrast, the representation of programs in HiVisual is based essentially on the flow of data. Icons are regarded as functional modules which have inputs and outputs, and the program is constructed by specifying the connection between the inputs and outputs of the various function icons. In order to display different levels of abstraction, the actual definition of a module may be hidden from the user.

In addition, HiVisual provides an interactive programming environment in the following way: When the user selects an icon and puts it at a suitable place on the display, the system activates the icon to execute the associated function and returns an iconic representation of the resultant data. This icon is displayed next to the activated icon. After this, programming proceeds by referring to the output and connecting another icon to it. Once a program is completed, the program itself can be defined as a new icon and be used later to make a higher level program.

HiVisual also allows simple data types such as records to be displayed graphically. Although text is used to display the individual fields, the fact that the structure of the data is displayed graphically can be helpful. HiVisual, like PICT, is not based on any conventional textual language. So, in the classification scheme, it falls in the category of tools that implement completely new graphical programming languages. Unlike PICT, it uses the data flow model to display programs graphically. Because it uses the data flow diagrams, HiVisual shares the advantages and disadvantages of this diagramming method: although data can be represented graphically, control flow is

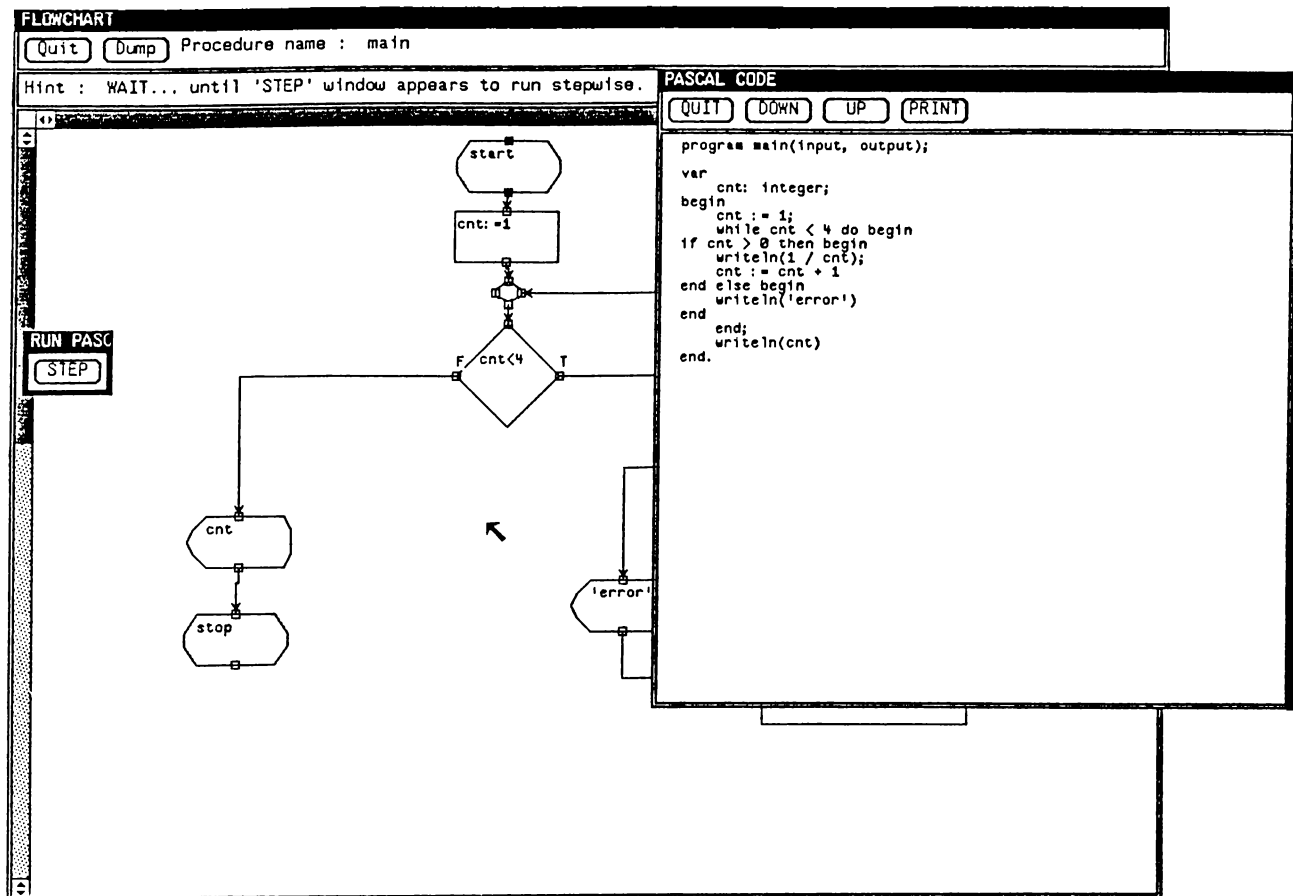


Figure 1.2: A snapshot of the screen.

not obvious from the diagram.

#### 1.4.5 'A Tool for Teaching Programming'

Our programming teaching system implements a graphical programming layer on top of the Pascal programming language. The tool is primarily meant to be used as a learning aid to students new to the Pascal language, but can also be used as an aid to write PASCAL programs in a top-down, graphical fashion using flowchart symbols.

The tool provides the user with a means to draw a program's flowchart in a window through a menu of icons and commands, then displays the corresponding Pascal code of the flowchart through the directives from the user with the help of a pointing device, a mouse. A snapshot of the screen while the tool is running is shown in Figure 1.2.

Flowchart elements are picked from a menu containing flowchart symbols as icons and are drawn on the graphic screen. The text information of each symbol is filled in by selection of a text entry command from the menu.

A text window can display the Pascal code corresponding to a flowchart in a graphics window. A single menu contains both the flowchart symbols and commands to operate on a flowchart as well as to accomplish the task of Pascal code generation and execution. Although explicit editing of the Pascal code that is generated from a flowchart is not permitted, editing of the flowchart itself is possible by means of appropriate commands. Therefore, each flowchart symbol on the screen can be identified for further operations. Once a flowchart is edited, the corresponding Pascal code also changes accordingly.

Both the graphic portion of the screen (which is the window containing the flowchart) and the text portion (which is the window containing the Pascal code) are scrollable. Therefore, the whole Pascal code corresponding to the flowchart can be displayed part by part by scrolling. This is true for the flowchart itself as well. Exit from the tool is provided by means of a button on the panel of the window in which the flowchart resides.

Once the code is generated, it can be executed step by step, as if using a debugger. During stepwise execution, values of the simple variables concerned are displayed and related flowchart elements of the code are highlighted.

Our teaching tool for programming fits into the category of programming environments which are based on conventional programming languages. This is because, the tool is based on a strategy for teaching Pascal programming through flowcharts [21]. A general comparison of the tool with the existing systems can be listed as:

- The tool is a flowchart based system for teaching Pascal programming.
- Both graphics and text involve in the tool. That is, in spite of graphical representation of programs through flowcharts, the user still needs to deal with the text of the flowchart boxes.
- Graphical part of the tool represents only the control flow of programs, since it applies flowcharts.
- The actual data structures can not be rendered graphically. They are entered to the program by the user through a declaration window by

typing in the identifiers of listed data types of Pascal, during the code generation.

- The tool applies a restricted domain of Pascal language, e.g., it does not support 'go to' statement, and Pascal 'Functions'.
- During the execution of the tool multiple windows and on-screen/pop-up selection mechanisms are applied for effective use of screen and simplicity for users.

In order to satisfy user needs in a simple way, we have to present the application in such a friendly manner that the user need not spend too much time in learning the system itself, but instead play with it while learning programming in Pascal. This is accomplished best by means of user-friendly tools.

## **2. UNDERLYING ELEMENTS AND ENVIRONMENT FOR THE DESIGN AND IMPLEMENTATION OF A TOOL FOR TEACHING PROGRAMMING**

### **2.1 User Interface**

As sophistication of computer systems increase with technology, the requirement for computers to remain friendly, comprehensible, and effective to continue to appeal to users also increases. A crucial factor in all three of these attributes is the quality of communication between the user and the machine.

User interface is the part of a program that determines how the user and the computer communicate. The design of interactive user interfaces is very important for the performance of the application. Not only are bad user interfaces difficult to learn, but they make programs difficult to use even in the hands of experienced users. For the naive user, graphical interfaces have been introduced that represent a significant improvement over traditional command languages as a means of realizing the full potential of a computer. As an example, iconic interfaces present command and system information in a nonverbal manner i.e., in the form of icons by which the learning curve can be reduced in both time and effort. Thus, it improves user performance while reducing errors. The reason for upgrading the user interface and thus performance in this manner follows the argument that communicating through images is the most natural way of communication. Furthermore, images can be easily recognized and learned, and thus may be better than their lexical equivalents [1].

Generally, any graphical user interface is composed of two parts, the presentation part and the interaction part. The presentation or layout part

defines what pictures are on the screen i.e., placing graphics and images on various parts of the screen. The interactions or behaviour part determines how these pictures change with user actions i.e., transforming an object on the screen. Models of user interface can be considered to fall into two broad categories : linguistic models, and spatial models. Linguistic models view the interface as a dialogue between user and computer, and focus on issues that occur within the syntax, semantic, and lexical levels of the dialogue. Spatial models include interactive graphics or direct manipulation models. A well designed user interface normally incorporates both linguistic and spatial components [2].

User interface components [3] naturally can be grouped into four :

- User's Model

This is a conceptual model acting as a framework for the development of strategies for operating the program. It enables the user to have a broad understanding of what the program is doing. The use of familiar concepts makes the user's model more intuitive and easier to learn. It is difficult to gain acceptance for a program that presents the user with unfamiliar objects that behave in highly unexpected ways.

- Command Language

Once the model is understood by the user, user needs commands to manipulate it, a system of which forms a command language. Principal issues of command languages are command modes, selection sequence, a command abort mechanism and error handling. Command modes are the states in which a given operation by the user is interpreted differently by the program. Selection sequence is the order to manipulate a command. Command abort mechanism is necessary to disable a currently working command. An error handling mechanism is how a program must respond to incorrect input.

- Feedback

Feedback is a component of the user interface through which the computer assists the user in operating the program. Some forms of feedback are provided only for naive users, and ignored by experienced users. Acknowledgement of reception of commands, explanatory messages, indication of selected objects, echoing of typed characters are some forms of feedback that help the user to be informed about the correct receipt of his commands.

- **Information Display**

This component shows the user the state of the information he is manipulating. By means of well chosen symbols and graphics, the user can be given confirmation that his model is correct. For a more effective utilization of the screen for information display, the way menus, windows, informative icons or graphics are handled, together with selection mechanisms, must be taken care of properly. A device independent user interface that functions with a wide variety of devices is the most feasible design issue for a user interface, since it provides for applications to be portable from one hardware configuration to another. On the other hand, such a design is quite difficult and the results might not be equally satisfactory for all classes of devices.

## **2.2 Object Oriented Approach**

During the implementation phase of our Pascal teaching tool the object oriented programming paradigm is made use of.

The object paradigm provides a natural mechanism for representing a system at several levels of abstractions from a very low level concept, such as a number, to a very high level concept, such as an office information system. Graphical applications, data base applications, and user interface designs are the main areas where object oriented programming techniques have been widely used recently. The innovative work of Kay and Goldberg at Xerox corporation in developing objects for the Smalltalk language obviously contributed much towards the advancement of these techniques [11,12,13]. A few other languages have such capabilities. Objects in C are a recent addition that this project makes use of mostly as an extension to the toolkit on which the code is built [5].

Object oriented programming differs from the traditional control oriented programming in that all conceptual entities are modelled with a single concept, an object. An object is an entity with attributes and having relationships with other objects.

Temperature range, size, danger level, color can be the attributes of an object representing a temperature gauge. The identity of an alarm object could be the value of a relationship. Methods are a kind of procedure in which the behaviour of an object is embodied. The method for a temperature



gauge could be the display of a temperature dial by graphics routines.

A message is the specification of an operation to be performed on an object, similar to a procedure call. Messages provide for objects to communicate with other objects to request information or some form of behaviour from them, for example a temperature gauge can be requested to display itself by messages from other objects. A method for an object is a function that implements the response when a message is sent to that object. An object reacts to a message by executing its corresponding method, which is a piece of code manipulating or returning the state of the object.

Objects can be represented in a class hierarchy inheriting properties down the hierarchy. Therefore different classes of objects can be defined, each object being an instance of a class. A class describes the form of its instances and operations applicable to its instances. Objects of the similar type are grouped into the same class. An object differs from other instances of the same class by the values of its attributes and relationships, but all the instances do have the same methods of the class. Lower down the class hierarchy, a class can also have subclasses as well as objects (instances of the class). All subclasses of a class inherit the properties of its superclass and additionally have its local properties [6,7].

An object oriented system for the construction of a window based user interface consists of a set of objects and actions for the interface. Objects present a functional interface and are manipulated by passing their unique identifier to their associated functions.

In an object oriented window manager system each element of the window manager belongs to a basic class *object*. *Window* is a subclass of object class having *frame* as an object and *subwindow* as a subclass. *TTY*, *canvas*, *panel* and *text* are the objects of subwindow. Additionally, *text*, *button*, *message*, *choice* and *slider* are the objects of *panel items* subclass belonging to object class. Also, *menu*, *scrollbar*, and *icon* are objects of object class [5].

While developing our object oriented education tool, we have used the window manager's own objects in addition to the objects designed specifically for our application.

Each entity in the application is designed as an object. At the top every object belongs to a basic class, OBJECT, that has subclasses GRAPHIC-OBJECT-CLASS, and TEXT-OBJECT-CLASS. GRAPHIC-OBJECT-CLASS

has subclasses, say, DECISION-BOX, PROCESS-BOX, START-STOP-RETURN-BOX, DUMMY-BOX, etc.

Instances of TEXT-OBJECT-CLASS that are the Pascal code segments corresponding to each graphic object, are created during the code generation for each graphic object. For example, each DECISION-BOX created at different times and places, while running the tool generates one instance for that class, that is an object for DECISION-BOX, and so do all the other graphic symbols. The following sections discuss the environment on which these objects act and how they act.

## 2.3 Environment

The Pascal Teaching Tool is designed on SUN 3.50 WorkStations running under 4.2 BSD UNIX<sup>1</sup> Operating System and using SunView<sup>2</sup> (SUN Visual/Integrated Environment for Windows) window manager system in C language. Sun WorkStations are fast devices with a high resolution screen for graphical applications. A mouse is used as a pointing device and a keyboard for text entry. A matrix printer is used as the output device [8].

Graphics architecture for the application is based on raster graphics. In raster graphics every visible picture element on the screen is called a *pixel* and has a corresponding storage location in memory. In workstations, display rasters are stored in dedicated regions of memory called *frame buffers* or video memory. A high speed frame buffer offers flexibility for the display of arbitrary objects, including variable width characters, foreign alphabets, mathematical symbols, lines, curves, shaded regions, and photographic images.

A frame buffer is generally organized by plane used for monochrome applications, mapping one bit of memory to one pixel. Also a frame buffer may feature additional raster operation hardware to speed up raster operations [8].

A window manager system such as SunView is used in implementation because of its interactive user interface constructs. SunView defines the relationships among the many possibilities of a workstation with multiple windows, running multiple processes. SunView manages and controls the interaction

---

<sup>1</sup>UNIX is the trademark of AT&T Bell Laboratories.

<sup>2</sup>SunView is the trademark of SUN Microsystems Inc.

of the varied processes within the appropriate window [5].

From the system's point of view, the tool is an interactive application running according to user directives from many input sources. It waits for input from mouse, keyboard, or disk. The output goes to any one of the output devices such as graphics screen, disk, or matrix printer according to the selection of user. The environment in which the editor runs is a notification based window manager system. In a notification based system the notifier acts as the controlling entity within a user process, reading UNIX input from the kernel and formatting it into higher level events, that it distributes to the different SunView objects. The tool uses the objects of SunView which provide a means for easy development of advanced user interfaces.

## 2.4 Tools For User Interface

In current user friendly systems, the hardware environment for applications generally consists of a high resolution bit addressable screen, a small pointing device called a mouse, and multitasking capabilities. On a single screen, multiple overlapped windows can be concurrently active and each window can contain multiple special purpose subwindows, each of which can have an active process. A wide variety of subwindow types provides for easy development of user interfaces. In such a system, management of the common interface functions is provided by a notifier/selection service, a window manager, and a broadcast service. The notifier and selection service are the runtime systems distributing input to the appropriate windows. The window manager manages the overlapping windows. The broadcast service makes it possible to activate task and exchange data between applications running in separate windows in the same or in different processes [4].

Such a notifier based system running in a multiple window environment contains various objects -tools- and actions for the user interface. An object is a software entity providing a functional interface and is manipulated by passing its unique identifier, called a handle, to its associated functions. Classes of these objects are defined in a hierarchy and can contain subclasses that may inherit common behaviour from their superclass. For a window manager system [5] this class hierarchy can be represented as in Figure 2.1.

- WINDOWS

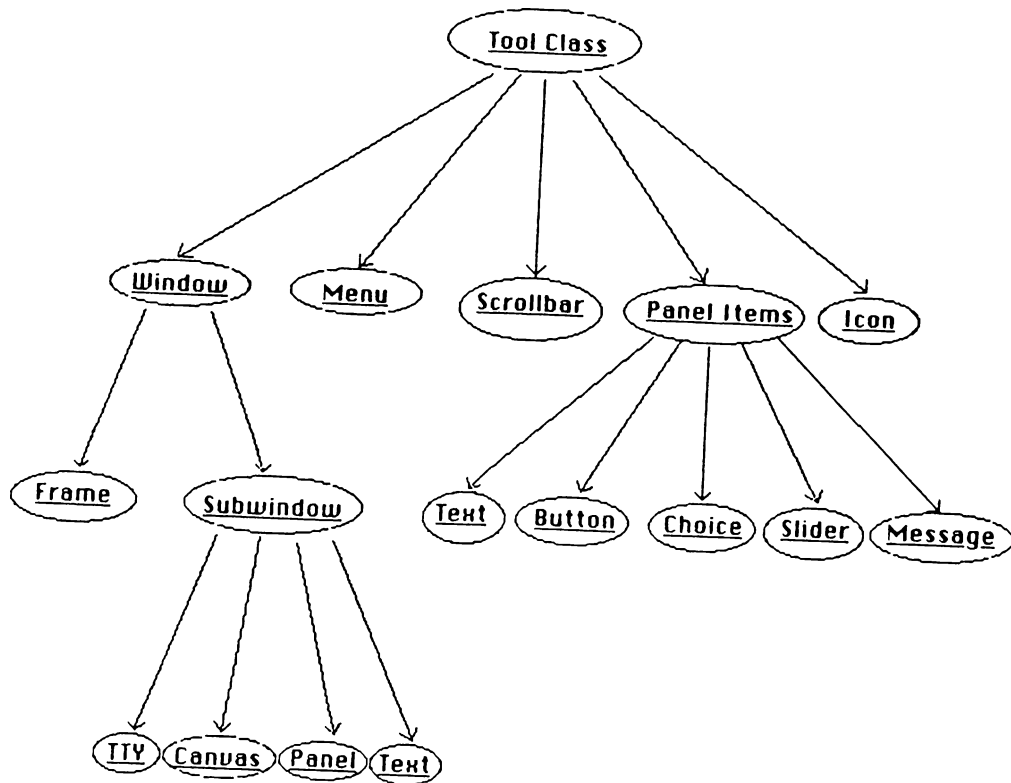


Figure 2.1: Class hierarchy of a window manager system.

Windows refer to a visible component of the display that the user perceives as a single entity -tool- and manipulates as a unit.

– **Frames**

Frames are windows containing nonoverlapping subwindows within their borders with the purpose of bringing different types of subwindows together into a common framework, providing them to be operated on as a single unit.

– **Subwindows**

Subwindows are windows never existing independently and always owned by a frame.

\* **Panel**

A panel is a subwindow containing panel items of buttons, message items, choice items, text items, or sliders. Buttons are simple activation confirmation items, message items provide an output area, choice items are to select among a number of items, text items present a type-in area, and sliders are to provide a continuous range of values.

\* **Text**

Text is a subwindow with capabilities for editing text.

- \* **Canvas**

A canvas is a subwindow into which programs can draw.

- \* **TTY**

A TTY is a system subwindow in which programs can be run and commands can be given.

- **PANEL ITEMS**

Panel items are panel components facilitating a particular interaction between the user and the application by button, message, slider, and choice items.

- **SCROLLBARs**

Scrollbars are lower class objects and dependent elements that may be attached to a subwindow to control the display of the specified portion of the window by sending appropriate locational messages. Another name used for scrollbars is the *elevator*.

- **MENUs**

Menus are special purpose subwindows allowing the user to choose from a list of actions.

- **ICONS**

Icons are small images symbolizing an application or its state.

These tools support the design of the style of user interfaces where the user typically uses a mouse to select and manipulate objects on the screen [5,10].

A window is activated when a user points to it say for input, and the window generates an event. Therefore events and their notification becomes a significant part of such a system.

## **2.5 A Notification Based System**

### **2.5.1 The Notifier**

Having a multiplicity of independent input sources, control of flow has to be taken away from the application, i.e., without performing input and output

explicitly, the application must register interest, describing the routines it wishes to be called when certain input/output events occur. A *notifier* is an example of this approach. Thus, notification based systems have some characteristics rather opposite to those of conventional sequential programming systems. In a notification based system, the main control loop of the application does not reside in the application itself but it is managed by the concurrent notifier that is the controlling entity for user processes.

In the conventional style of interactive programming, the main control loop resides in the application. An editor, for example, will read a character, take some action based on the character, then read the next character, and so on. When a character is received that represents the user's request to quit, the program exits. Notification based systems invert this 'straight line' control structure. The notifier reads events and 'notifies' or 'calls out' to a procedure that the application has previously registered with the notifier for those events.

Thus, the duty of the notifier is to sense the events and activate the procedures of the applications -described as *callback procedures*- already specified for the events. An event is almost anything that is prescribed. The functions to be activated by the occurrence of an event for an application are registered with the notifier. This registration process is provided through specification of callback procedures for events [5,10].

The control structure of a notifier based program is given in Figure 2.2.

Without a centralized notifier, the application itself would have to carry the duty of detection and dispatching of events to its components as in the case of conventional interactive programming. For programmers who are not used to a notification based system, this callback style of programming takes some getting used to. Its big advantage is that it takes over the burden of managing a complex, *event-driven* environment. In SunView, an application typically has many objects. In the absence of a centralized notifier, each application must be responsible for detecting and dispatching events to all the objects in the process. With a centralized notifier, each component of an application receives only the events the user has directed towards it and each function receives only the events related to itself.

Therefore, in a notification based system, the programmer creates the required functions and object definitions for the application, then registers such definitions with the notifier, based on the occurrence of a class of events.

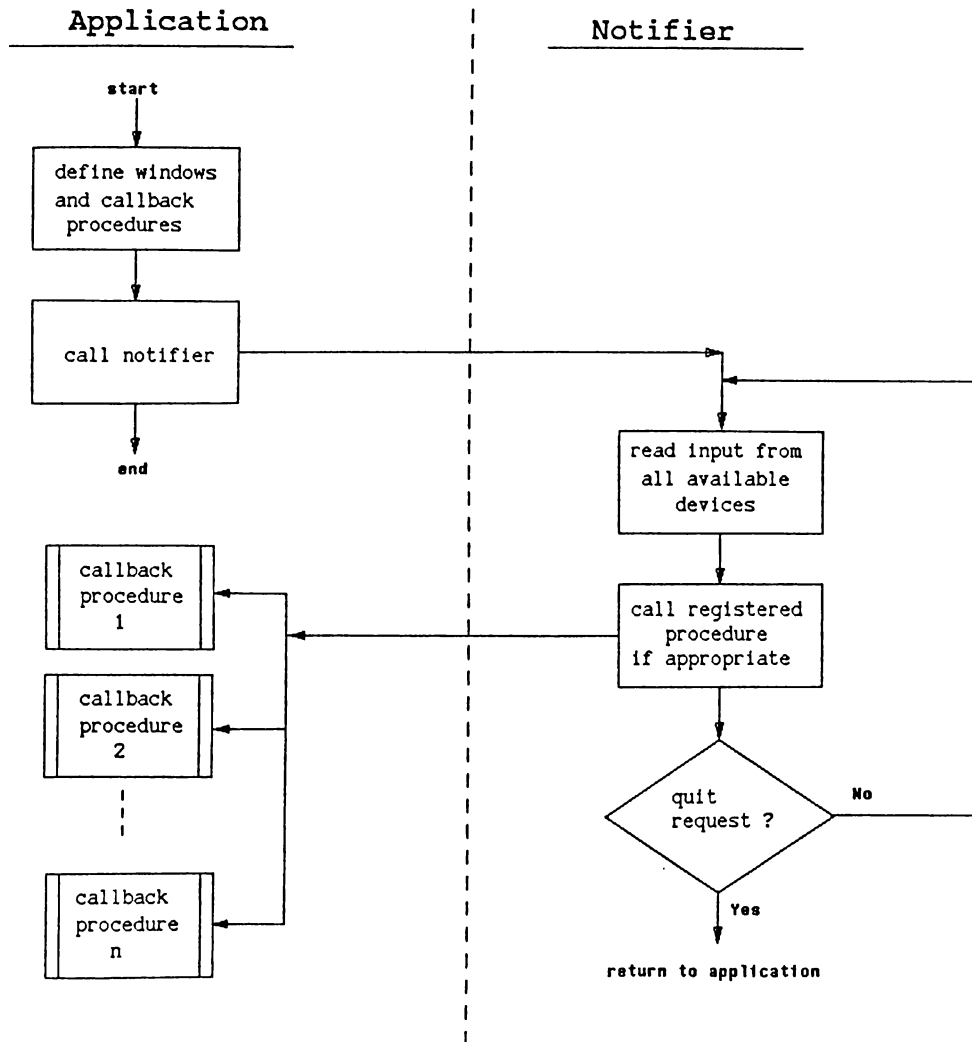


Figure 2.2: Control structure of a notifier based system.

The control is then passed to the notifier/selection mechanism and if an event from the prescribed class should occur, the selection service activates the necessary tools [4,5].

### **2.5.2 Relationships Between the Notifier, Objects, and the Application**

It is not necessary for the user (programmer here) to interact with the notifier directly in the application. SunView has a two-tiered scheme in which the packages that support the various objects -panels, canvases, scrollbars, etc.- interact with the notifier directly, registering their own callback procedures. The application, in turn, registers its own callback procedures with the object. Typically, when writing a SunView application the programmer first creates the various windows and other objects needed for his/her interface, and registers his/her callback procedures with the objects. Then he/she pass control to the notifier. The work is done in the various callback procedures. There is one distinction between the 'event procedures' for the canvases and the 'notify procedures' for the panel items. They are all callback procedures, but they have different purposes. The canvas's event procedure does not do much work -basically it calls out to the application's event procedure each time an event is received. The application sees every event and is free to interpret the events however it likes.

The notify procedure for panels, on the other hand, does quite a bit of processing. It determines which item should receive the event, and places its own interpretation on events such as, the middle mouse button is ignored, or left mouse button down over an item is interpreted as a 'tentative' selection of the item, etc. It does not call back to the notify procedure for the item until it receives a left mouse button up over the item. So panel item notify procedures are not so much concerned with the event which caused them to be called, but with the fact that the button was selected, or a new choice made, etc.

Within the context of the issues related to user interface and its tools, the design and implementation of 'a tool for teaching programming' is discussed in the following sections.



## 3. DESIGN OF A TOOL FOR TEACHING PROGRAMMING

### 3.1 Naming Conventions

In the following sections, when describing our Pascal teaching tool, it will be mentioned as 'the tool' in short.

Flowchart symbols consist of two main types of elements; box and arc. A box is a flowchart symbol representing an action or process in the chart and may contain textual information inside. Some examples of boxes are start/stop box, decision box, input box, output box, etc. An arc is the flowchart element providing a path for connecting the boxes of a flowchart with an arrow tip indicating the flow direction.

Pins are the connection points of boxes to connect them to other boxes via arcs. The pin at which an arc path begins is called the *beginning* pin of the arc and the pin where the arc path ends indicated by an arrow tip is named *ending pin* of the arc. Arcs can be connected to other arcs through the use of dummy boxes.

If a flowchart has some sub-routines to reference by a 'call subroutine box', flowcharts of those sub-routines are named as sub-flowcharts and generated by means of 'proc body On' menu item. The owner flowchart -calling one- is named as main flowchart, on the other hand.

### 3.2 Screen Design of the Tool

As Figure 3.1 shows, there are three main areas on the screen layout of the tool : A drawing area, a flowchart menu, and information panels. Design

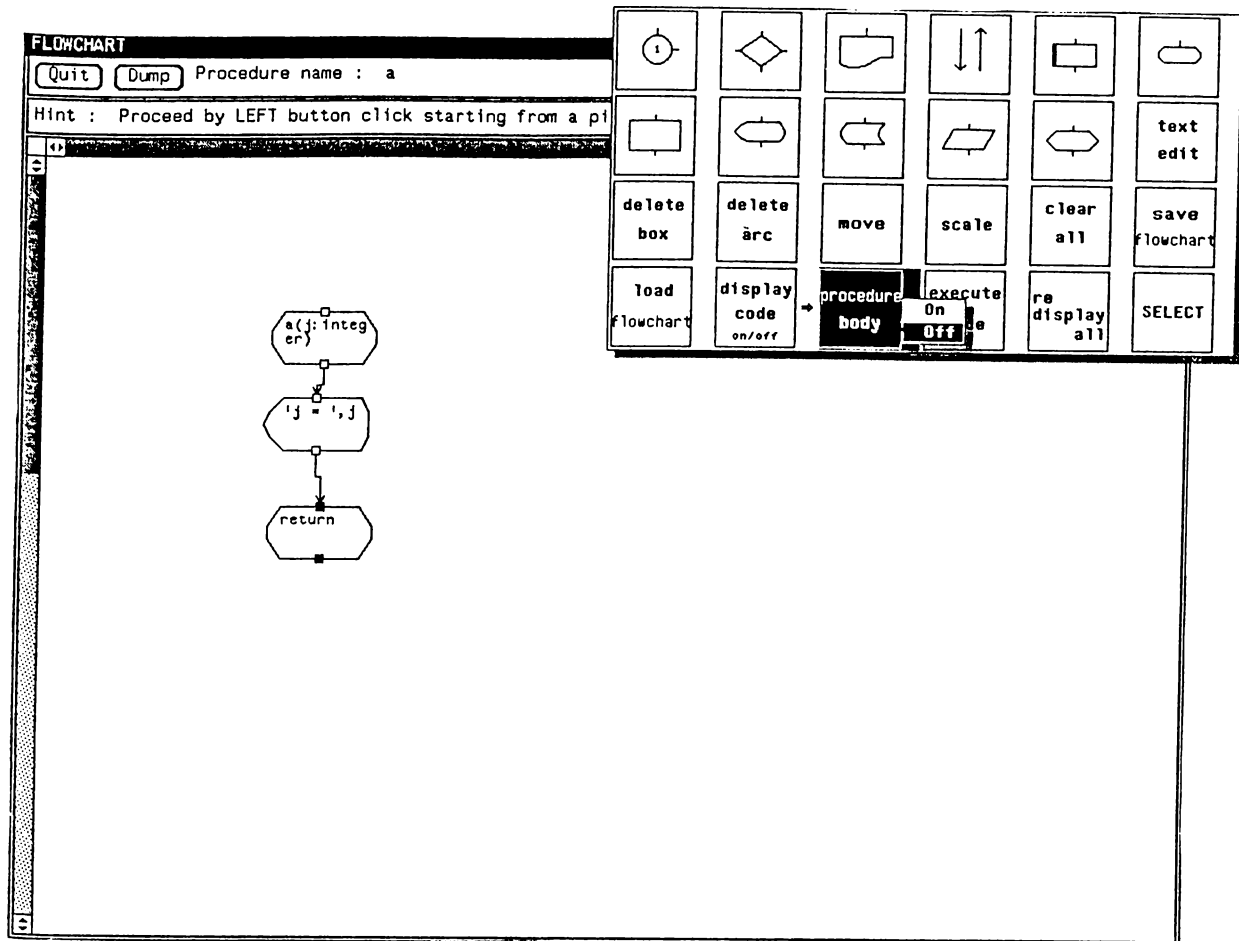


Figure 3.1: Screen layout of the tool.

of all these areas are supported by the user interface tools of the window manager on which the tool is written. In the following section main parts of the tool's screen layout with window manager's constructs used for their implementation is presented.

- Flowchart drawing area

This area is a 'canvas' used for drawing flowchart elements on and writing text in during flowchart generation. Its size is larger than the frame surrounding it and it can be scrolled by means of 'scrollbars' attached to left and upper sides of the frame around (Figure 3.2).

Drawing symbols on the screen as well as text entry from keyboard are controlled by the selections from flowchart menu through mouse buttons. Activation of scrollbars are also provided by mouse buttons for scrolling the drawing area.

- Flowchart Menu

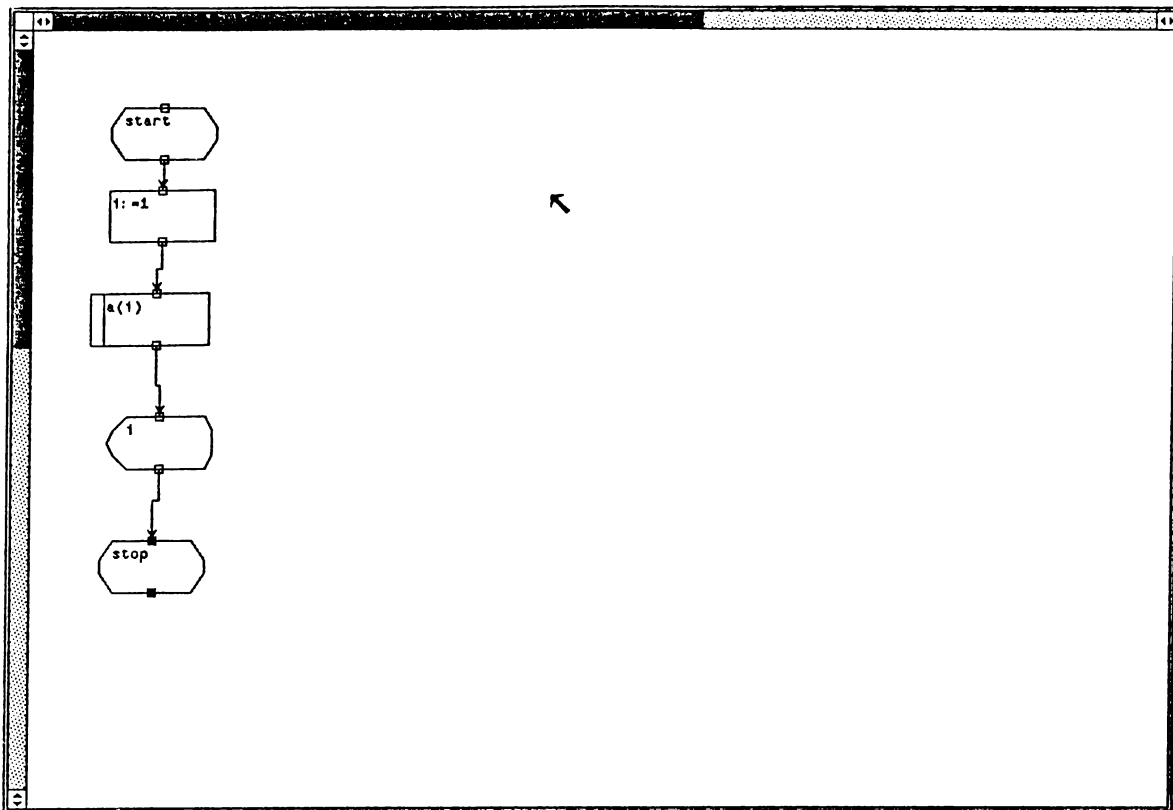


Figure 3.2: Flowchart drawing area.

Flowchart menu is implemented using 'menu' element of the window manager to present a selection mechanism for flowchart symbols and operations to handle the tool. Menu items to select among are composed of 'icons' depicting the flowchart symbols and operations. The menu is displayed by pressing right mouse button inside the drawing area or left mouse button on the frame covering the drawing area and panels (Figure 3.3).

Releasing the button on one of the items -which becomes dark- chooses that item to operate. Some menu items are 'pull right menu items'

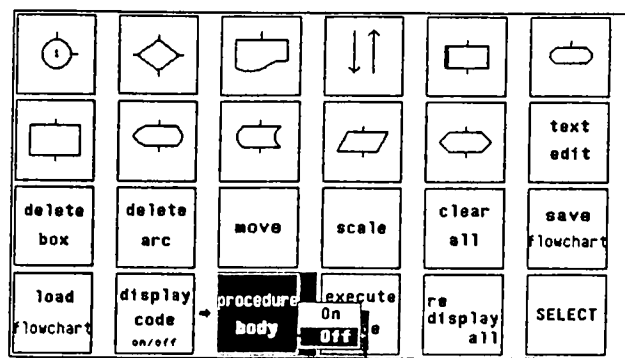


Figure 3.3: Flowchart menu.

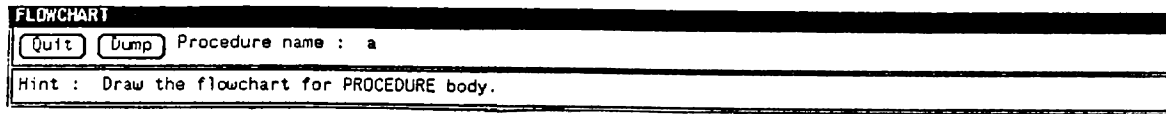


Figure 3.4: Panels.

displaying another menu, when mouse is pulled right over them. For example 'display code On/Off' and 'procedure body On/Off' items are pull right menu items indicated by a right arrow nearby, and present a deeper selection mechanism.

- Information panels

- Panel 1

First information panel is used for displaying the name of the currently working flowchart/sub-flowchart appearing on the drawing area. Since a flowchart symbolically represents a routine and can contain sub-flowcharts for sub-routines of the routine, a message giving the name of the working routine/flowchart is displayed to user near the 'Procedure name :' message. Name of the initial flowchart is 'main' by default, and its sub-flowcharts which are named by the user can be displayed by choosing 'procedure body On' menu item. Returning back to the main flowchart is possible by activation of 'procedure body Off' item.

The panel also contains two buttons, one for exiting from the tool, and other for dumping the screen image to disk for hard copy from the laser printer (Figure 3.4).

- Panel 2

This panel guides the user about what to do next after selection of a menu item. The action to be taken is announced for the user near the word 'Hint :' (Figure 3.4).

### 3.3 Other Screen Layout Elements Displayed by Flowchart Handling Operations

- Text edit window

This is a 'text subwindow' appearing each time a new symbol is drawn or by selection of the 'text edit' menu item. It is used for editing textual informations of flowchart boxes (Figure 3.5).

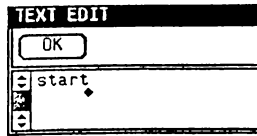


Figure 3.5: Text edit window.



Figure 3.6: File name window.

- File name window

This window contains a 'panel text item' to read a file name required for some operations like 'flowchart save/load', and 'code generate' (Figure 3.6).

- Scale window

This window consists of two 'panel slider items' to enlarge or shrink a flowchart box on the drawing area. It appears after selection of 'scale' menu item and its sliders are operated on by left mouse button and size of the flowchart box changes with the amount of percentage value of the activated sliders (Figure 3.7).

- Declaration window

This window contains a number of 'panel text items' near the messages listing the Pascal data types to get the corresponding identifiers of data types from the user, during Pascal code generation. In addition to declaration of variables/types in the main flowchart, declaration of local variables/types for each sub-flowchart is asked from the user too. Declaration window and its related flowchart appears for main and each sub-flowchart, after 'code generate On' item of menu is chosen (Figure 3.8).

- Code display window

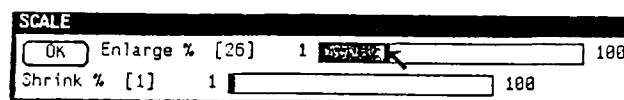


Figure 3.7: Scale window.

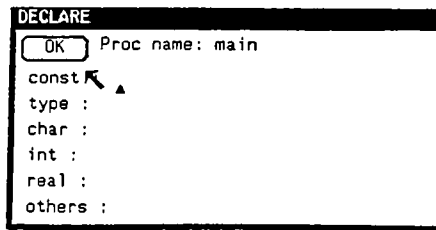


Figure 3.8: Declaration window.

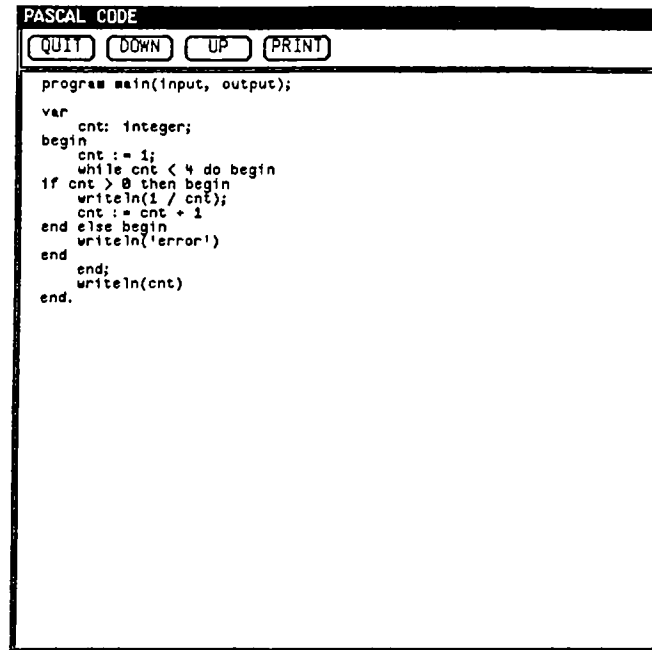


Figure 3.9: Code display window.

This window contains a 'panel' and a 'canvas', canvas displaying the Pascal code corresponding to the flowchart residing in the drawing area, when the 'code generate On' operation is completed. The panel consists of four buttons to remove the window from the screen, to scroll the Pascal code up and down, and to print the Pascal code from the printer (Figure 3.9).

- Step window

Step window is used during the stepwise execution of the generated Pascal code. It contains a 'panel button' to continue execution until the next box and display step window again by suspending execution (Figure 3.10).



Figure 3.10: Step window.

## 3.4 Items of the Flowchart Menu

Items of the flowchart menu can be analyzed in two categories: flowchart symbols and flowchart handling operations as shown in Figure 3.3.

### 3.4.1 Flowchart symbols

- Dummy box

Dummy box has no effect as a flowchart box, but only serves for the connection of two or three arcs with each other.

- Decision box

Decision box is used to carry out a conditional branch operation according to the textual expression residing inside the box. If/then/else, while/do, and repeat/until constructs of Pascal are represented through the use of decision box on the flowchart, since these three constructs do not have distinct flowchart symbols for each. They are differentiated by their connection to other boxes and to the dummy box from the decision box. For making a connection from the decision box to another, the first arc beginning at the decision box shows the 'False' exit, next one the 'True', by default. False and True exit points can be changed by deleting those arcs and redrawing in reverse creation order again.

- Output box

Output box is used for printing values for a list of variables/constants delimited by commas from the printer.

- Arc symbol

Arc symbol connects the boxes of a flowchart through a path at the connection points of boxes named pins.

- Call subroutine box

This box appears on the flowchart where a reference to a predefined sub-flowchart representing a sub-routine is required. As a textual information it contains the name of the sub-routine to be called with its parameters in parenthesis.

- **Start/Stop/Return box**

This box symbolizes the starting/stopping element of a flowchart according to 'start'/'stop' string contained in the box respectively. If this symbol is a starting element of a sub-flowchart it must contain the name of the sub-routine with its parameters and their data types separated by commas enclosed in parenthesis as textual information. Return from a sub-flowchart is symbolized again by this symbol with a 'return' string in it.

- **Process box**

This box represents an assignment operation with an arithmetic, logic, or string expression.

- **Display box**

This box is the same as the output box except that this prints the values of data listed in its text string onto the screen.

- **Disk i/o box**

This box is used for input/output operations from a disk file. The file type is always 'text file' of Pascal. Operation type (i/o), a file name, and list of variables are written as its textual information.

- **Input box**

In order to get some values of variables from the user, the input box is used. Its textual part contains a list of variables to read the values from the standard input device -screen-.

- **For box**

This box is used for incremental/decremental iterations over a range of boxes. Its textual part consists of the name of the index for the iteration with its starting and ending values.

### **3.4.2 Flowchart Handling Operations**

- **text edit**



This operation is used for filling in the textual part of a box or editing previously filled one.

- delete box

In order to delete a specific box from the flowchart together with its textual part, this operation is applied. To have the operation in effect, all the arcs beginning or ending at that box must be deleted before.

- delete arc

This operation deletes an arc whose beginning pin is pointed by the mouse and clicked over, after 'delete arc' operation is chosen.

- move operation

Selection of this operation moves the box pointed by the mouse accordingly with the movement of the mouse, from one location to another.

- scale operation

Scale operation changes the size of a selected box. It enlarges or shrinks the box according to user directives controlled by the scale window.

- clear all operation

It clears the whole drawing area together with all of the data structures allocated for the flowchart and its sub-flowcharts. After this operation, generation of a new flowchart is possible.

- save flowchart operation

User can save the flowchart residing on the drawing area together with its sub-flowcharts to a disk file named by the user.

- load flowchart operation

A previously saved flowchart can be loaded from a disk file named by the user and displayed on the drawing area.

- Display code On/Off operation

'Display code On' operation creates the Pascal code of the flowchart residing on the drawing area and lists in the code display window.

'Display code Off' operation removes the display window with the Pascal code from the disk, to be regenerated later, when required.

- Procedure body On/Off operation

'Procedure body On' is used to create/display a sub-flowchart of a flowchart. If the name of the sub-flowchart requested from the user is a new one, drawing area is cleared to generate it. If it already exists it is displayed to the user for editing.

'Procedure body Off' operation is used to return to the main flowchart of a sub-flowchart while editing the sub-flowchart.

The name of the edited sub-flowchart is displayed on top of the drawing area near the message 'Procedure name :'.

- execute code operation

This operation has to be applied to start stepwise execution of Pascal code for the flowchart, after the code generation operation.

- Redisplay all operation

In order to redraw the whole flowchart on the drawing area, i.e., refreshing the drawing area, this operation has to be activated.

- SELECT operation

Select operation selects a flowchart box from the drawing area to be used by other operations such as scale, delete box, etc.

### 3.5 Flowchart Design Methodologies

Since flowcharts are first developed for unstructured languages like assembly or Fortran language, a flowchart can not support all the constructs such as repeat-until, while-do, etc. of a structured language such as Pascal. In order to implement almost all constructs of Pascal in our Pascal teaching tool, a number of flowchart design methodologies are developed to represent some of the Pascal constructs which do not have corresponding single box symbols. For example, since there is no distinct symbols for if-then-else, while-do, and repeat-until constructs of Pascal, they are represented by the use of decision box. Differentiation of these constructs are provided by connection to dummy box and other boxes from the decision box.

- connection of arcs through dummy boxes (Figure 3.11).
- if-then-else (Figure 3.12).
- while-do (Figure 3.13).



Figure 3.11: Connection of arcs.

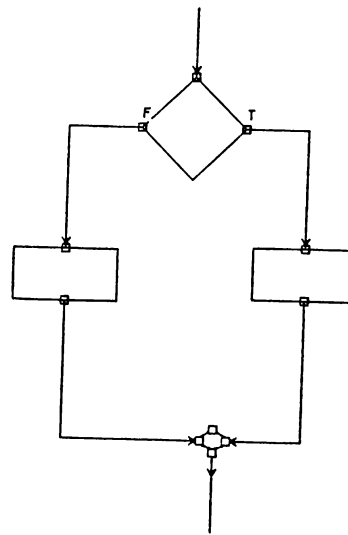


Figure 3.12: if-then-else.

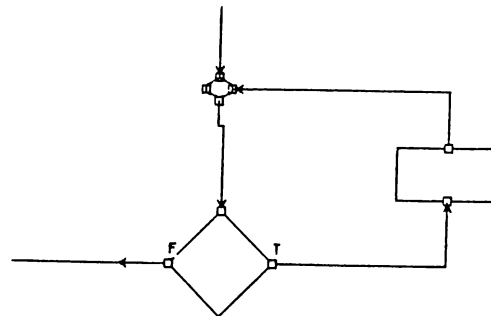


Figure 3.13: while-do.

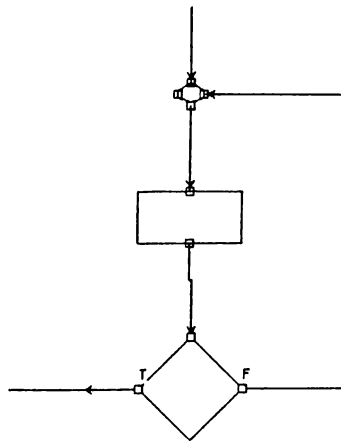


Figure 3.14: repeat-until.

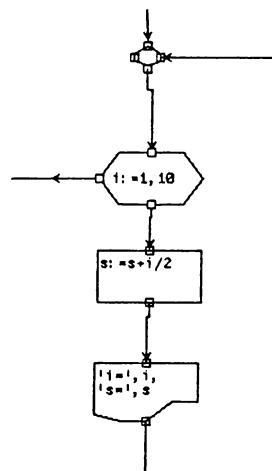


Figure 3.15: for.

- repeat-until (Figure 3.14).
- for (Figure 3.15).
- start/stop/procedure-start/return (Figure 3.16).
- subroutine -procedure- call (Figure 3.17).
- output/display (Figure 3.18).
- process (Figure 3.19).
- disk i/o (Figure 3.20).
- only one entry and one exit point for each flowchart/sub-flowchart (Figure 3.21).



Figure 3.16: start/stop/procedure-start/return.

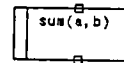


Figure 3.17: subroutine call.

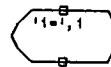
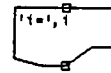


Figure 3.18: output/display.

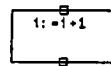


Figure 3.19: process.

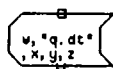
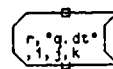


Figure 3.20: disk i/o.

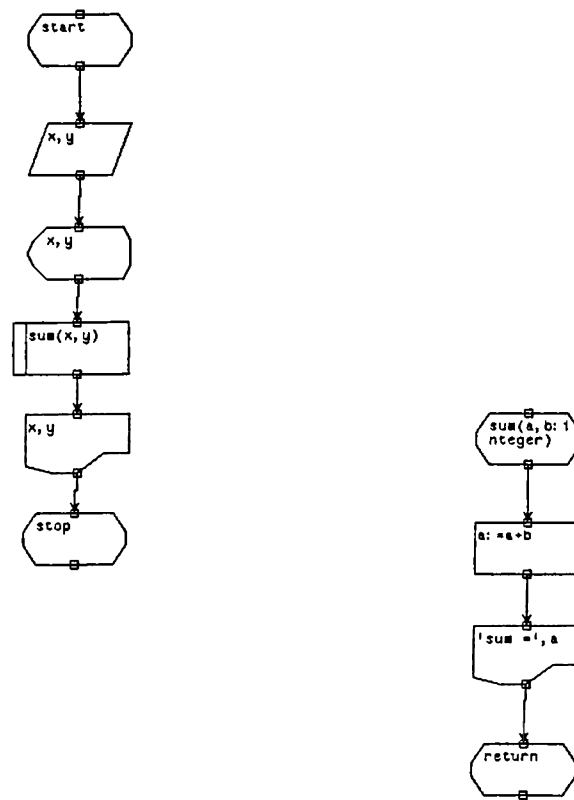


Figure 3.21: Only one entry and one exit point.

## 4. IMPLEMENTATION OF A TOOL FOR TEACHING PROGRAMMING

### 4.1 Data Structures

There are a number of data structures employed during the implementation of the tool. The flowchart on the drawing area is kept in memory as a list of structures containing information about the location of the box, its connection points, its text string, connections to other boxes, and connection paths. Also some additional structures keep the pointers to sub-flowcharts in the application. The data structures keeping those kind of information are described below.

- A structure for list of coordinates of a box.  
Keeps the locations of the corner points of a box drawn on the screen (Figure 4.1).

```
struct coords
{
    int x;
    int y;
    struct coords *next_coord;
} ;
```

- A structure for list of connection points -pins- of a box.  
Keeps the locations of the pins of a box (Figure 4.2).

```
struct pins
{
    int x;
    int y;
```

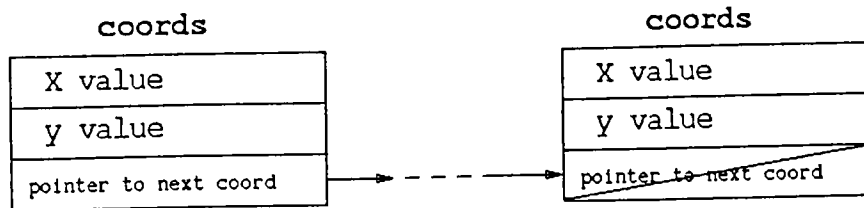


Figure 4.1: Structure for box coordinates.

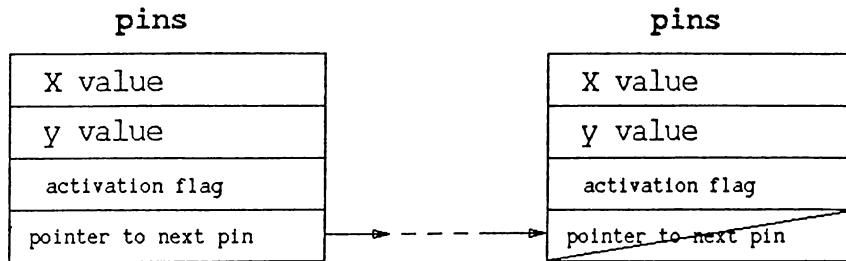


Figure 4.2: Structure for box pins.

```

    short active;
    struct pins *next_pin;
} ;

```

- A structure for list of boxes of a flowchart.  
Keeps the id, type, coordinates, pins, textual information, and box pointers of a flowchart's boxes in a list (Figure 4.3).

```

struct box
{
    int box_id;
    char box_type;
    char *code_ptr;
    struct pins *box_pins;
    struct coords *box_coord;
    struct btext *box_text;
    struct box *connect_to[2];
    struct coords *arc[2];
    struct box *connect_back[3];
    struct box *next;
    struct box *previous;
} ;

```

- A structure for textual information of a box.  
Keeps the starting coordinate for text in a box, length of box to fit the text in it, and text string itself for the box (Figure 4.4).



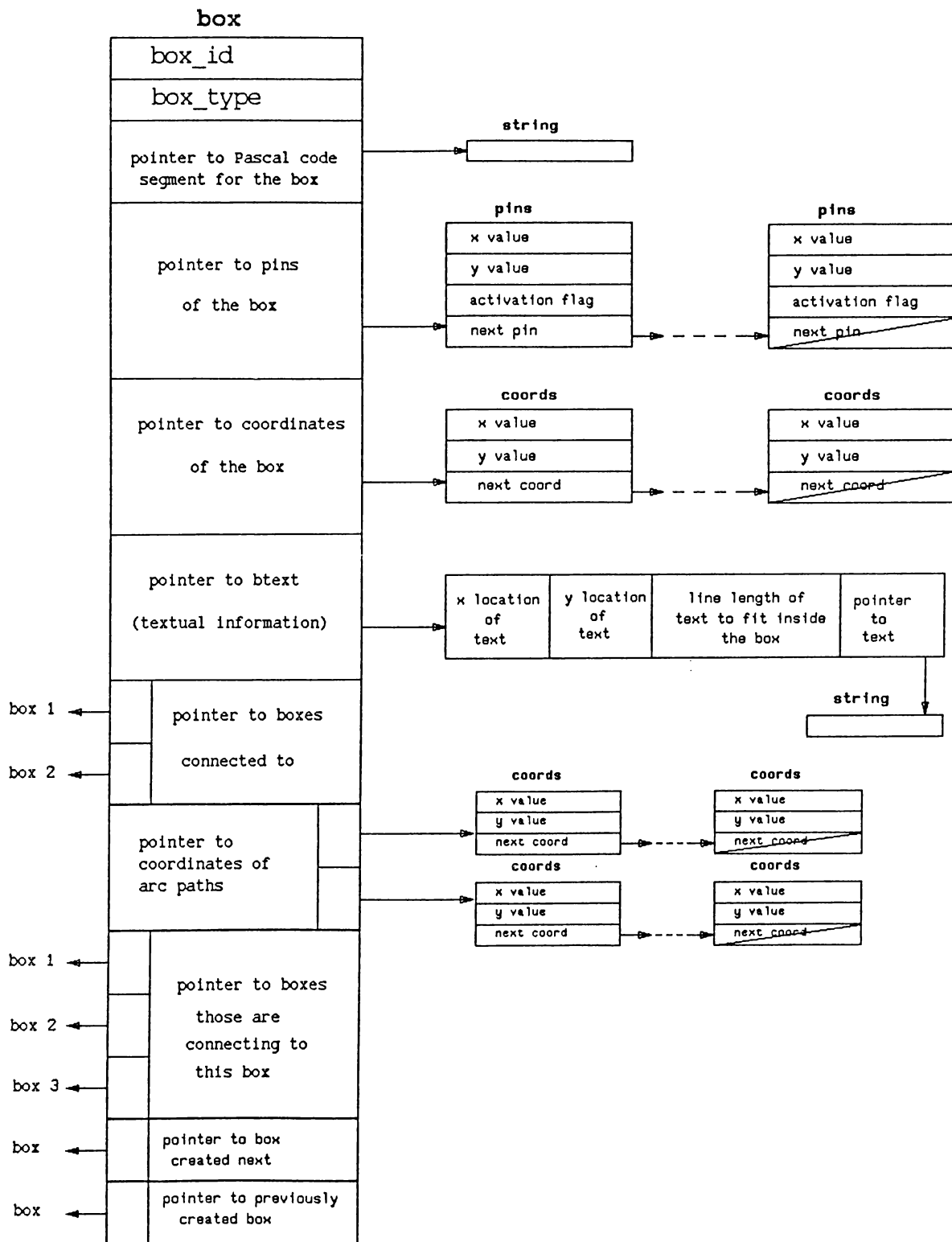


Figure 4.3: Box structure.

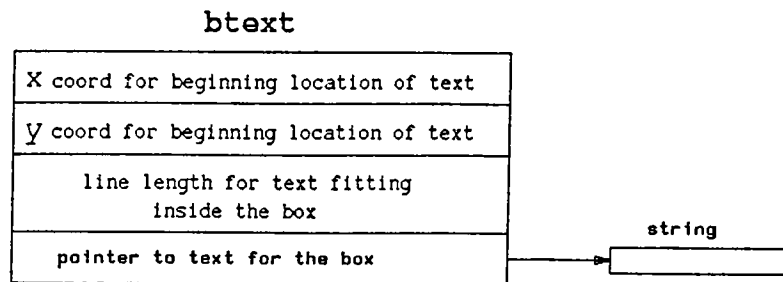


Figure 4.4: Text structure.

```
struct btext
{
    int x;
    int y;
    int line_len;
    char *text_entered;
} ;
```

- first\_box.  
Keeps the address of the first created box for the flowchart (Figure 4.5-a).

```
struct box *first_box;
```

- last\_box.  
Keeps the address of the last created box (Figure 4.5-b).

```
struct box *last_box;
```

- current\_box.  
Keeps the address of the currently selected box to operate on (Figure 4.5-c).

```
struct box *current_box;
```

- A structure for list of sub-flowcharts of a flowchart.  
Keeps the information about the name, its caller's name, and box pointers of sub-flowcharts in an array (Figure 4.6).

```
struct proc_list_type
{
```

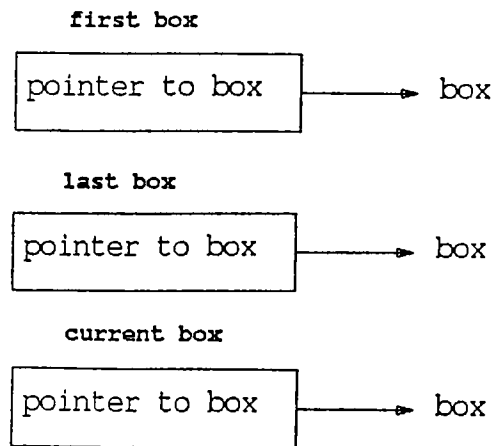


Figure 4.5: Box pointers.

```

char *proc_name;
char *belonging;
struct box *current_box;
struct box *first_box;
struct box *last_box;
} proc_list[10];

```

- A structure for stacking owner flowcharts during creation/editing of sub-flowcharts.

Keeps the name of sub-flowchart and its owner -main/calling one-, and pointer to the boxes of the main flowchart (Figure 4.7).

```

struct proc_stack_type
{
    char *proc_name;
    struct box *current_box;
    struct box *first_box;
    struct box *last_box;
    char *return_to;
} proc_stack[10];

```

## 4.2 Basic Routines Used for Implementation

- Default-draw-box routine

For each kind of flowchart box, one specific 'default-draw-box' routine exists in the program. These routines are activated when placing a box

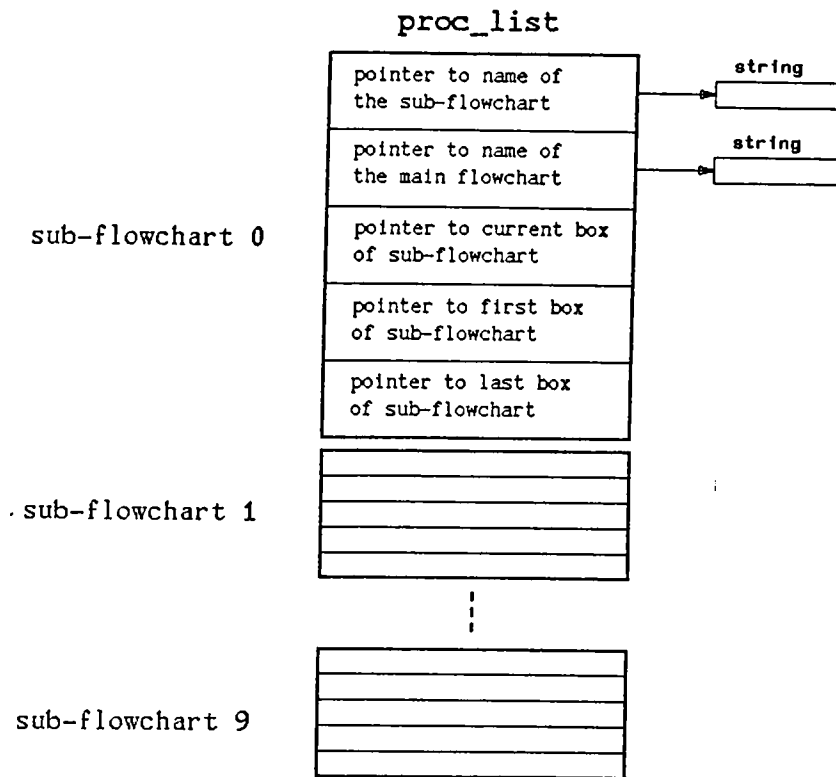


Figure 4.6: Array of sub-flowcharts

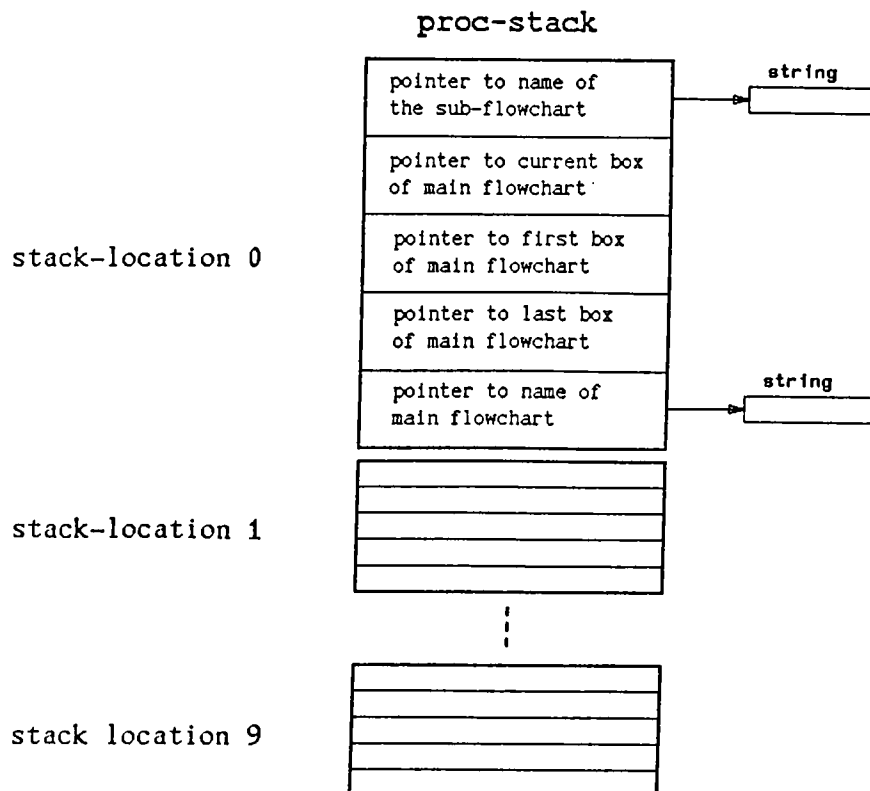


Figure 4.7: Flowchart stack.

onto drawing area. The routine for the chosen box type then allocates a 'box' structure, fills in the appropriate fields of the box such as id, type, pointer to previously created box, some textual information as location, length of one line of text, then nullifies the empty pointers. The figure of the box is drawn by this routine with its predefined size and pin location information at the cursor position. At the same time, locational information for the box structure is filled in by allocating the necessary structures. Then follows the execution of text edit routine to fill in the textual information field of the box structure.

- Text-edit/Write-to-box routines

'Text-edit' routine displays a text window for the user to type in the textual information for a box or to edit existing text. After text editing is completed, 'write-to-box' routine is called to display the text inside the box on the drawing area by fitting the text string to the box size. The text is divided into lines each line fitting to the borders of the figure of the box. The text string is also inserted to a structure pointed by the textual information field of the box structure.

- Select-box routine

Purpose of this routine is the identification of a particular box on the flowchart, pointed by the mouse. When select operation is chosen from the menu and mouse button is clicked near the pins of a box, 'select-box' routine is activated. It searches through the list of boxes beginning from the first box to the last box to catch the box, where one of the pin coordinates of it is the closest to the cursor position. A half cm threshold length is applied when comparing the distance between the box pins and the mouse position. If one is found, that box is selected as the current box and its nearest pin is identified. The box is emphasized by highlighting the pins of the box. If no such box is found close as the threshold distance, then previous current box does not change.

- Move-box routine

This routine becomes active in a coordinated way with 'canvas- event-procedure' after 'move' item from the menu is chosen. By pressing left mouse button near a box, 'select-box' routine is called to identify the box to be moved. While left button is down, according to the movement of the mouse, through canvas- event-procedure, the coordinate values of the box are changed clearing the old figure and redrawing it at the new location at the same time . Therefore the figure is perceived by the user as if it is moving.

- Enlarge/Shrink routines

Before scaling the size of a box, first a box must be selected by 'select' menu item. Choosing 'scale' menu item creates a window containing two sliders, one for enlarging, other for shrinking of boxes. When the enlarge slider is activated, it calls 'enlarge-routine' to get the value of the slider. 'Shrink-routine' is activated alike by shrink slider. Slider values are used for the percentage enlarging/shrinking of the box. Then x and y offset values from old coordinates to new coordinates are calculated. Coordinate values of the box and its pins are updated according to x and y offsets, then old figure is erased, and redrawn with the new size.

- Canvas-event-procedure

'Canvas-event-procedure' controls whole operations occurring inside the drawing area and follows the behaviour of the mouse. Events generated by mouse buttons and mouse movement are all processed in this routine. All of the operations requiring coordinates of the cursor are processed here, such as default-draw-box, move, select, arc-draw/arc-delete operations.

- Arc-draw/arrow routines

Activation of these routines comes into action when an arc will be drawn or deleted on the flowchart. Creation and deletion of arcs are controlled by 'canvas-event-proc' . When creating an arc, first pressing the left button near a box, the closest non active -unused- pin of the nearest box is marked as the beginning pin of the arc. An arc is drawn from the beginning pin to the point where the mouse is pointing at, as the mouse moves. Releasing the button puts the arc permanently, the coordinates of the arc are recorded into a coordinate list pointed by the arc[] field of the box structure for that nearest box. Pressing the left button continues the arc path from the last position where it left to the current position of the mouse as the mouse moves. Releasing the button adds the coordinate of the arc to the list pointed by arc[]. After repeating these steps , pressing the middle button selects the ending pin for attaching the path to the connection box. If ending pin is already active, then the path is cleared, otherwise arc path is connected to ending pin and its activation flag is set together with that of starting pin. An arrow tip is drawn at the ending pin in the direction of the arc using 'arrow' routine.

Deletion of an arc is provided again first selecting the nearest pin as a beginning pin for an arc by 'select-box' routine. Then the arc is

removed from the screen and from the box structure where it is kept in a list pointed by `arc[]` field. Beginning and ending pins are made inactive and arrow tip at the ending pin is removed.

- Procedure-body-On/Procedure-body-Off routines

'Procedure-body-On' routine is used for generation of sub-flowcharts of a flowchart. It is activated from the menu and the name of the sub-flowchart is asked from the user. To generate a sub-flowchart for a flowchart, first-box, last-box, and current-box pointers -which provide access to the list of boxes- of the main flowchart are pushed onto the 'proc-stack' with the names of the sub-flowchart and main flowchart. After that, drawing area is cleared and initialized to generate the sub-flowchart -its name is displayed near the 'Procedure name : ' message. After completion of the sub-flowchart, activation of 'procedure-body-Off' routine returns back to the main flowchart. If the sub-flowchart is not empty, that is, contains some boxes, it is added to the 'proc-list' array recording its name, pointers to its first/last/current boxes, and name of the calling flowchart. If the sub-flowchart is empty, it is not recorded to be used for sub-flowchart editing and code generation in the 'proc-list' array, so, simply deleted. The drawing area is cleared and the main flowchart that it belongs is loaded onto the drawing area from the 'proc-stack' by popping the stack, and displayed being ready for editing at the state where it is left.

- Code-generate-On/Code-display routines

'Code-generate-On' routine generates the Pascal code for the flowchart by selection from the menu. Code-generate-On routine, first, requests a file name from the user to write the generated Pascal statements on. For each box in the flowchart one or more Pascal statements are generated. For example, output box generates a 'writeln' statement, start box generates statements for 'program heading', 'declaration' sentences, and a 'begin'. Also some of the boxes are grouped into one Pascal statement such as decision box and a dummy box with different connections to other boxes generate if-then-else/while- do/repeat-until statements.

The corresponding Pascal statements of each box is appended to their box structures at the time of first analysis of the flowchart during code generation described above. This first pass over the flowchart produces and prepares the Pascal statements to write to a disk file. In order to complete the code generation phase, the second pass over the flowchart picks the statement(s) for each box from box structures and combine

them in the file named by the user at the beginning.

At the time of code generation for a flowchart, sub-flowcharts correspond to Pascal 'Procedures'. All sub-flowcharts are processed after start box of the main flowchart generating 'Procedures', since procedures are written before the main body in Pascal.

After completion of the code generation for all sub-flowcharts and main flowchart, the Pascal code is displayed to the user from a window by using 'Code-display' routine. This routine reads the Pascal statements line by line from the file and prints them on a canvas inside the display window, just created. It displays only a portion of the code fitting to the canvas size. 'Down routine' activated by the 'down' button of the window displays next canvas-size portion of the code and 'up' button displays previous canvas-size portion by 'up routine', if so many lines of the code exist.

Before displaying the Pascal code, it is processed through a Pascal beautifier of the Unix named 'pxb', to make the code more clear and understandable.

- Execute-code routine

As its execution will be described in more detail in the next section, the Pascal code will be executed stepwise, each step corresponding to one box of the flowchart. At each step the values of the simple variables will be printed and the executing flowchart box will be highlighted.

For the execution phase, Pascal source code is converted to object code by calling the Pascal compiler 'pc' of the system inside 'execute-code' routine. Execution of the Pascal object code simultaneously with the executing tool itself is performed by a system call to the Pascal object code to execute it in background. During the stepwise execution of the code communication between the tool and the Pascal code is provided by message passing over a temporary file.

During the compilation of Pascal source code, if Pascal compiler generates some errors resulting from the user's incorrect flowchart design or syntax of the text strings inside the boxes, execution of the tool continues by displaying the error messages at the display window by calling 'code-display' routine for the error file containing error messages of the compiler.

- Flowchart-save/flowchart-load routines



These two routines used for putting/getting the flowcharts prepared to/from the disk for using them later.

'Flowchart-save' routine saves the values of the fields of all box structures in the box structure list of the flowchart, orderly, and those of the sub-flowcharts recorded in the 'proc-list' array of sub-flowcharts.

While writing pointer values of boxes, box-ids of the boxes that are pointed to are recorded instead of addresses. For some fields of the boxes containing a list of elements, such as list of coordinates, each element is written in order and terminated by a delimiter to identify the end of the list.

'Flowchart load' routine reads the data on the disk for a flowchart in the order it is written. While reading the data, it allocates the box structures and structures for their fields and fills in them with these data values, just read. Since pointer fields for box structures are read as box\_ids, they are replaced by box addresses by searching through the box\_ids in the box list and writing the addresses found for the box\_ids instead. All sub-flowcharts are read in the same way and recorded to 'proc-list', array.

### **4.3 Low Level Operation Of The Tool**

How the generation and the execution of Pascal code happens is slightly different than a user perceives it. When the Pascal code is generated for a flowchart, the user sees a pure Pascal program displayed in the display window corresponding to the flowchart on the drawing area. When this code is executed, the user follows the execution steps from the flowchart. But, in fact, the executing code is not the pure Pascal code that the user sees at the display window, but another one having the additional facilities to communicate with the executing tool. While code generation phase is working, two Pascal source files are generated, one for the pure Pascal code to display, another containing some additional transparent code for each Pascal statement. This transparent code before each statement -when executed- provides the communication between the tool and the executing Pascal code to highlight the flowchart boxes corresponding to each executing Pascal statement.

### 4.3.1 Code Generation in Low Level

In code generation phase two Pascal files are created first, one for 'pure code', the other for 'transparent code'. Through the 'code generate On' routine, the flowchart is analyzed and during the analysis for each box of the flowchart a set of Pascal statements are placed to the box structures of boxes. When the analysis is completed, code for each box is written to two files, but, for the transparent code file some additional Pascal code is also appended.

For the 'start box', declaration of an External C routine is added to the transparent code, that is, procedure 'talk\_to\_parent (int\_adr :integer);'. This is the C routine linked to the Pascal code during compilation time. This routine provides the communication of the executing Pascal program with the tool. As will be described below, this routine is to be called with an integer parameter which is the box address in a flowchart to be highlighted.

For each executable Pascal statement written to the pure code file and transparent code file, an additional statement 'talk\_to\_parent (<box address>);' is written to transparent code file. This is the call to that C routine with a parameter being the box address of the box that generates that Pascal statement.

Also another statement for transparent code file is appended to print the values of simple variables in the flowchart, that is, 'writeln (\* simple vars \*, <list of simple variables>);'. For main flowchart only simple variables in the main flowchart, and for sub-flowcharts only local variables are processed. Therefore, execution of this transparent code will display the values for the variables in the executing flowchart.

### 4.3.2 Code Execution in Low Level

Code execution is provided by the 'execute-code' routine. In this routine Pascal object code for the transparent Pascal source file is generated by Unix system call to the Pascal compiler 'pc', and the object code generated is executed by a system call to the name of the object.

During the code execution phase, two processes execute simultaneously; the tool itself being the parent process in fore-ground, and the 'transparent Pascal object' being the child process executing in back-ground. At this moment the tool displays the flowchart/sub-flowchart highlighting its boxes,

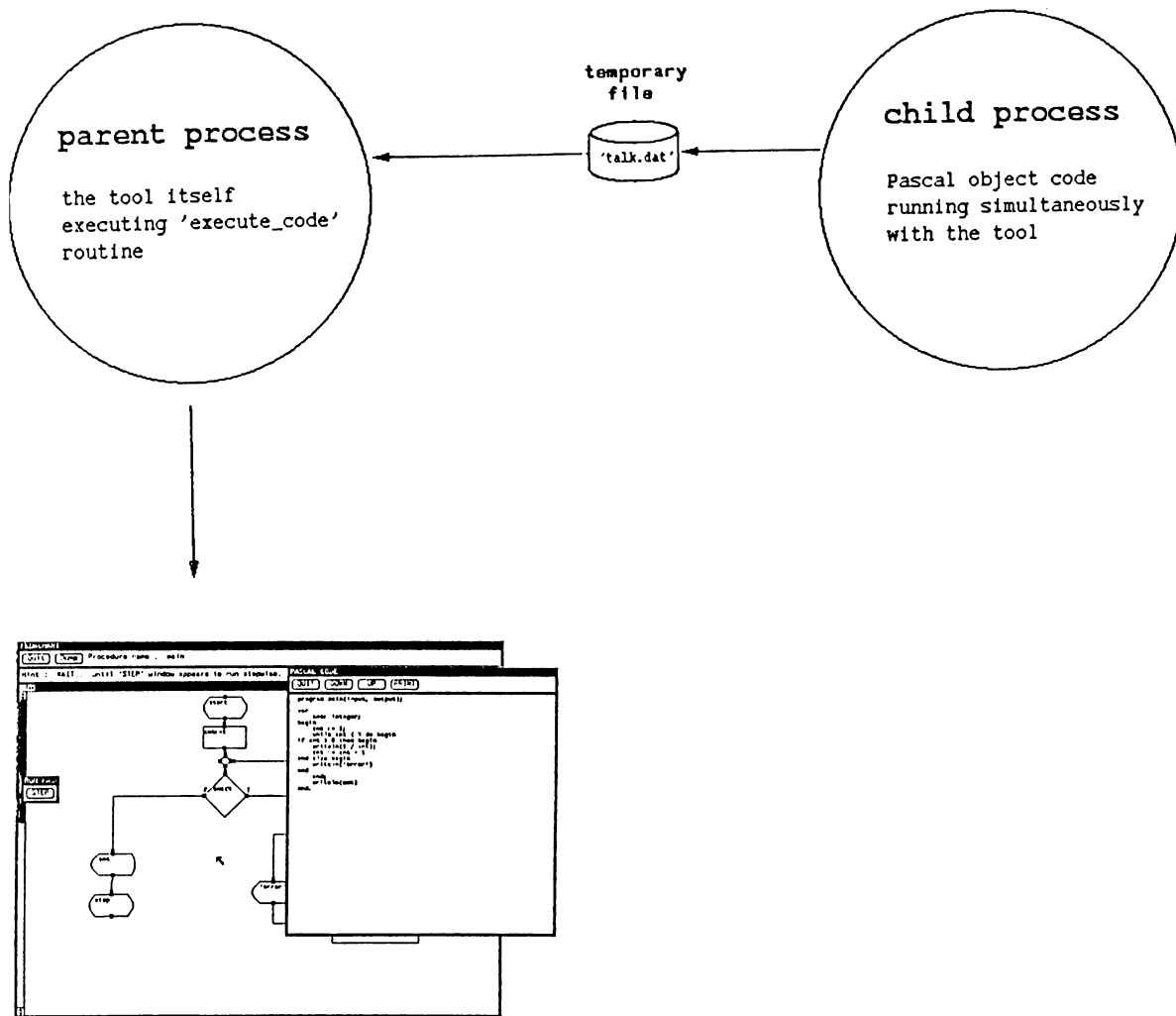


Figure 4.8: Low Level Operation of the Tool.

identified by the simultaneously executing Pascal code. Communication between the two processes for passing the address of the box from child to parent is provided over a temporary file 'talk.dat'. Child process writes the box id to 'talk.dat' file by executing a call to 'talk\_to\_parent' routine for highlighting. Parent process reads the box.id from 'talk.dat' and highlights that box on the flowchart (Figure 4.8). These statements as mentioned, are appended to transparent code file during code generation and box.id is the parameter of 'talk\_to\_parent' routine specifying the boxes. The routine simply does the following:

```

back :
If 'talk.dat' file does not exist, then

```

```

        { create 'talk.dat' file
          write box_id coming as the parameter to 'talk.dat'
          /* box_id is the address of the box corresponding
             to the Pascal statement to be executed */
        }
else { wait a little
      go back.
    }

```

The parent process communicates with the Pascal code by reading that box id to be highlighted as follows :

```

back :
If 'talk.dat' exist AND box id inside exists then
    { read box id for highlighting
      highlight the box on the flowchart
      remove the 'talk.dat' from the disk
      go back
    }
else { wait a little
      go back
    }

```

Note that at the beginning of code execution phase 'talk.dat' file does not exist in the system.

## 4.4 Unimplemented Features

- An 'UNDO' facility for the tool does not exist during its execution. It is not implemented, because the user can 'undo' any operation himself through the operations presented in the tool. For example, draw box/delete box, draw arc/delete arc, edit text operations etc.
- No syntax-directed check for flowchart boxes are made during the construction of flowcharts. Incorrect text strings for boxes generates syntactically incorrect Pascal statements and these errors are reported by the Pascal compiler, during the code execution phase to inform the user.

- No semantic check for the complete flowcharts are made before the code generation phase. Pascal beautifier reports the errors for incorrect/incomplete flowcharts during the code generation phase to inform the user.

## 5. CONCLUSION

Graphical programming has many advantages over conventional programming with text. Programming through pictures is much more natural and thus very easy to learn. Graphics systems provide an environment which is much more comfortable and user friendly for program development and thus can increase user productivity.

Our tool for teaching programming is an example of systems which are based on conventional programming languages and is written on top of the Pascal language. It requires the user to represent his problem in the form of a flowchart to generate the corresponding Pascal code. Execution of the code is also visualized from the flowchart step by step, showing the values for variables involved in the flowchart.

Thus the tool presents the control flow of a program by means of flowcharts. It creates an environment for the user to gain programming logic, and relations between the control structures of the flowchart and Pascal language, also to visualize the program execution steps graphically.

Since the tool implements flowcharts as the graphical representation of programs, it carries all the advantages and disadvantages brought into action by use of flowcharts. First difficulty comes from the fact that a flowchart could not represent all the constructs of a structured programming language. For example, different sorts of iteration/loop/selection mechanisms of a structured language might only be represented by a decision symbol with different connection methods to other symbols, sometimes causing semantically incorrect flowcharts. For our system the reason for the choice of an unstructured tool to represent a structured environment is that, flowcharts are very familiar charts and are used almost by every one since the first appearance of assembly and high level programming languages.

For the future work, the design of a powerful graphical tool satisfying the

requirements of structured programming languages and use in programming environments may be worth studying. In this way semantic errors coming from the graphical representation of program steps can be removed totally, since every construct of a language is represented by a unique symbol. The second issue might be the application of a syntax directed editor for textual parts of such a tool during text entry to reduce translation errors coming from the code generation phase.

Also by the invention of new graphical tools for structured languages, two-fold editing of both graphical part and program code would be possible. In such a case, modification of graphical part, lets the program code to be modified automatically (as in our tool), and also editing of the source code changes the graphical part automatically. This new approach might also allow development of special purpose debugging tools for such a graphical environment.

Although our tool and many other systems implement programming with the graphical aids much work still needs to be done in this field. A lot of work, for example, still needs to be done in the area of representations of data types and data structures graphically. Computer programs are complex and abstract objects, and determining what form pictures with a minimal text representing them should take, is a complex task.

## A. USER'S MANUAL

### A.1 Generation of a Flowchart

#### A.1.1 Creating a New Flowchart

All the symbols and operations for flowchart creation are provided through a flowchart menu. It is displayed on the screen by a pressing the left mouse button from the border of the flowchart window or pressing the right button inside the drawing area of the flowchart window. The steps to go through for generation of a flowchart are box selection, box placement, filling the boxes in with textual information, and connection of boxes by arcs. Then follows the further editing/saving/loading operations for the flowchart, if necessary.

- Picking and placing boxes

Flowchart boxes are picked from the flowchart menu by pointing with the mouse. Depressing the right mouse button inside the drawing area, causes the flowchart menu to appear. Release of the button on one of the boxes selects that symbol to be placed on the screen. Clicking the left button inside the flowchart window places the selected box to the location where the mouse is pointing to (Figure A.1).

- Filling in the boxes

As soon as a box is placed on the drawing area, a small text entry window appears on the screen, waiting for the user to fill in textual data of the box. When the text is typed into the window, clicking the 'OK' button residing on top of the text entry window causes that text to be written line by line within the boundary of the current symbol. Text window then disappears (Figure A.2).



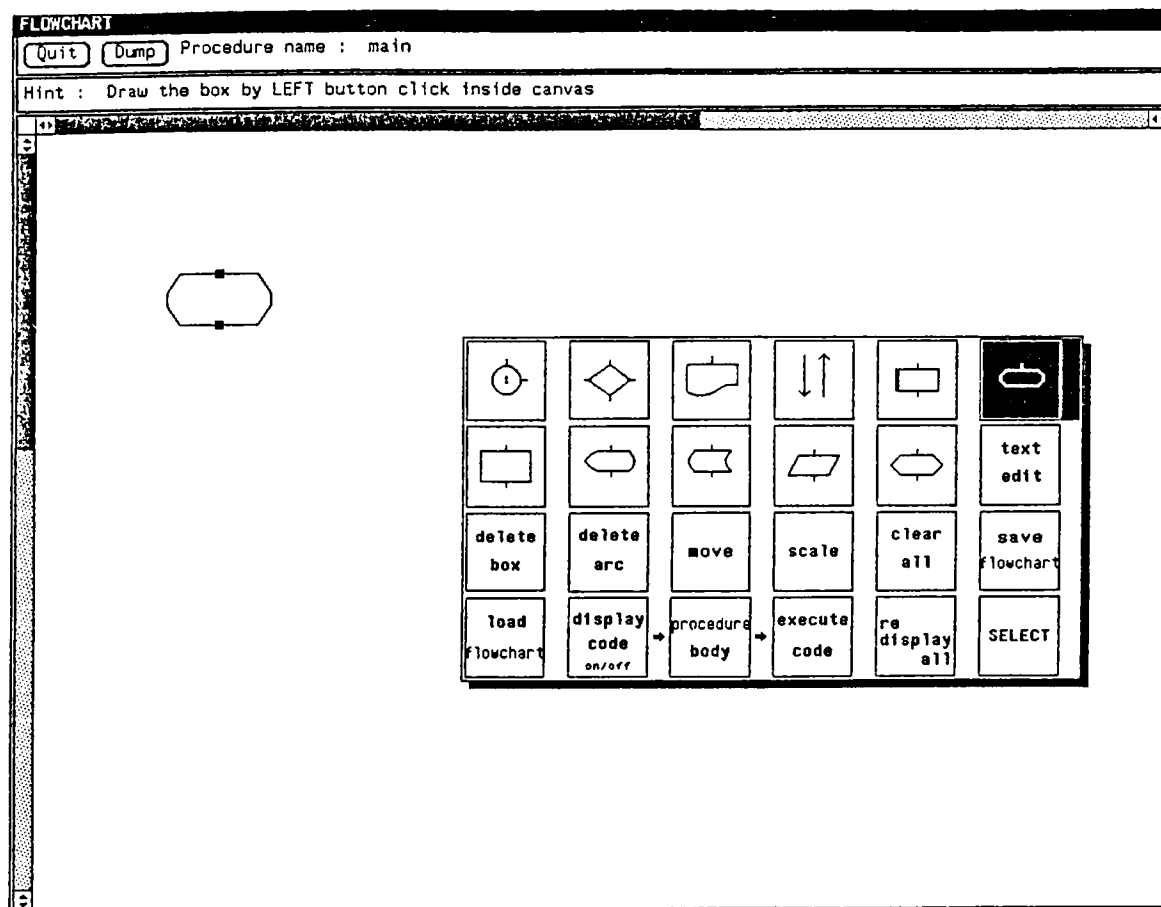


Figure A.1: Picking and placing boxes.

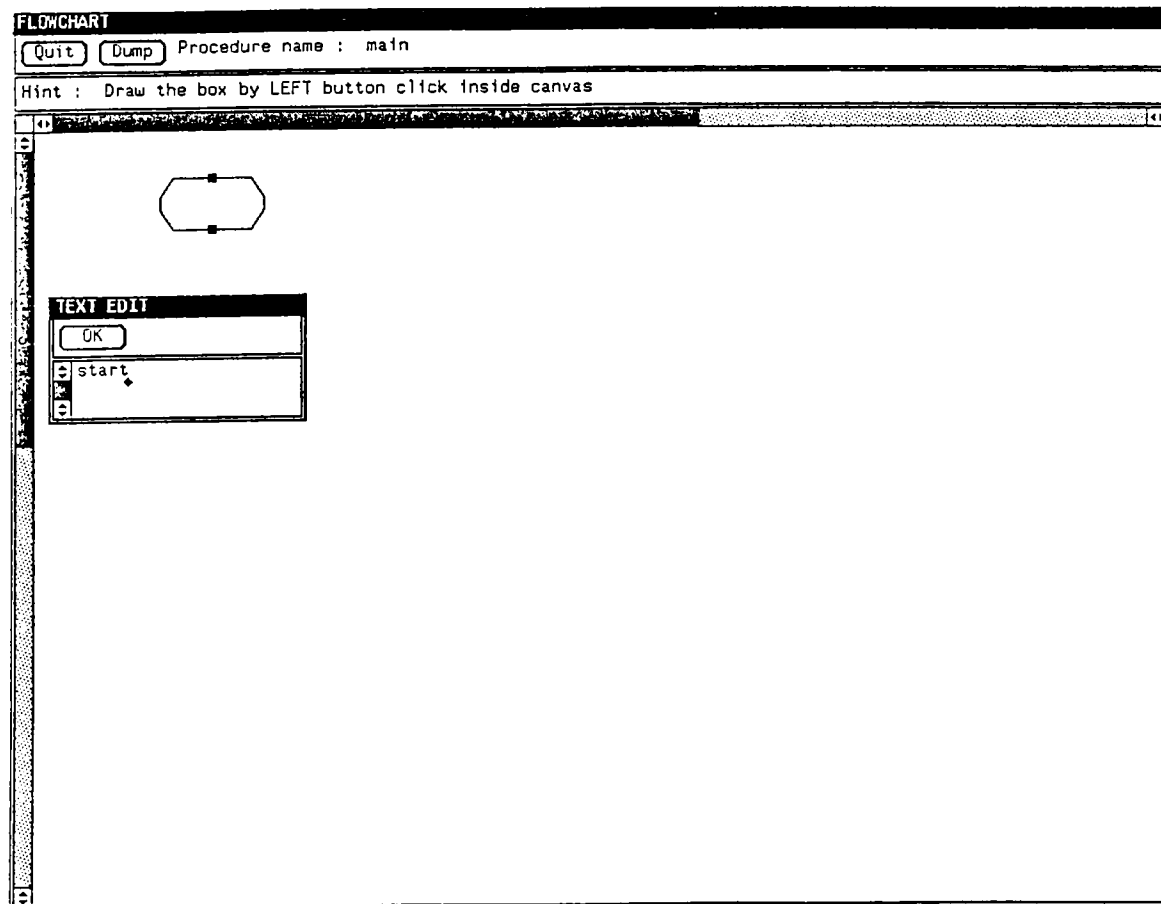


Figure A.2: Filling in the boxes.

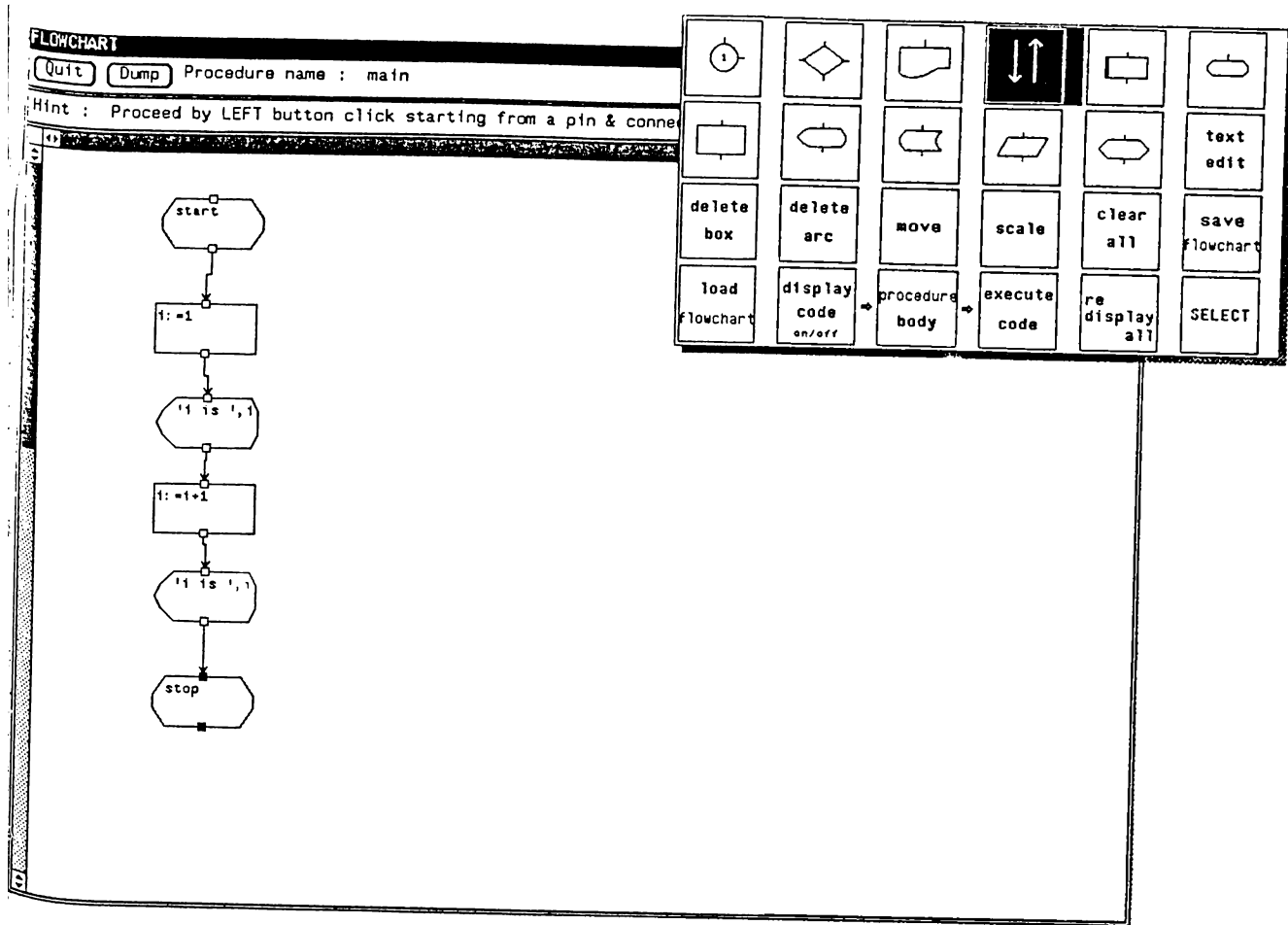


Figure A.3: Connection of boxes.

- Connection of boxes

Connection of flowchart boxes placed on the screen first requires the selection of arc symbol from the flowchart menu. Pressing the left mouse button near to a box selects the nearest pin of the box as the beginning pin of the arc in drawing area. As the mouse is moved on the drawing area, a horizontal or vertical path is drawn from the beginning pin to the cursor position dynamically. Releasing the button places the path to the drawing area permanently. After that, pressing the left button continues drawing the path from where it is left. Afterwards, clicking the middle button near a box causes the nearest pin to be the ending pin of the arc, and an arrow tip appears at the ending pin indicating the flow direction. An arc is drawn horizontally, if the movement of mouse in x direction is larger than that in y direction. It is drawn vertically, if vice versa. Since only horizontal and vertical arcs are possible, these arcs are called to be *Manhattan* (Figure A.3).

### A.1.2 Editing a Flowchart

Once we want to create a new flowchart or continue to work on a previously created one, a need for the operations to change some parts also arises. These operations should facilitate the identification of a box to operate on, sizing/movement/text editing/box deletion operations defined on the selected box, and arc deletion operation. No explicit insertion is required, since the tool always works in *insertion mode*.

- Selecting a box

Identification -selection- of a box on the drawing area is made possible by the 'SELECT' item from the flowchart menu. After choosing 'SELECT' item, clicking the left mouse button near the pins of a box selects the box to operate on and the box is highlighted. If there are nearby boxes, once the button is clicked the box having the nearest pin to the cursor position is selected.

- Deleting arcs

After choosing 'delete arc' item from the flowchart menu, clicking the left button near the beginning pin of an arc deletes the arc.

- Deleting a box

In order to delete a box from a flowchart together with its textual information, all the arcs beginning or ending at that box must be removed first. Next, the box to be deleted must be identified by 'SELECT' operation. Now, the box can be removed from the drawing area immediately by choosing the 'delete box' item from the menu.

- Moving a box

In order to move a box from its current position to another location, first, 'move box' item must be chosen from the flowchart menu. After that, as in the 'SELECT' operation pressing the left mouse button near one of its pins selects the box to be moved, and moving the mouse while the button is down moves the box with its text by the moving cursor. Releasing the button places the box permanently to that position. While moving the box, arcs attached to it changes according to position of the box, causing the vertical and horizontal arcs being damaged and inclined. This arcs can be made Manhattan (horizontal and vertical, only) by deleting those arcs and redrawing (Figure A.4).

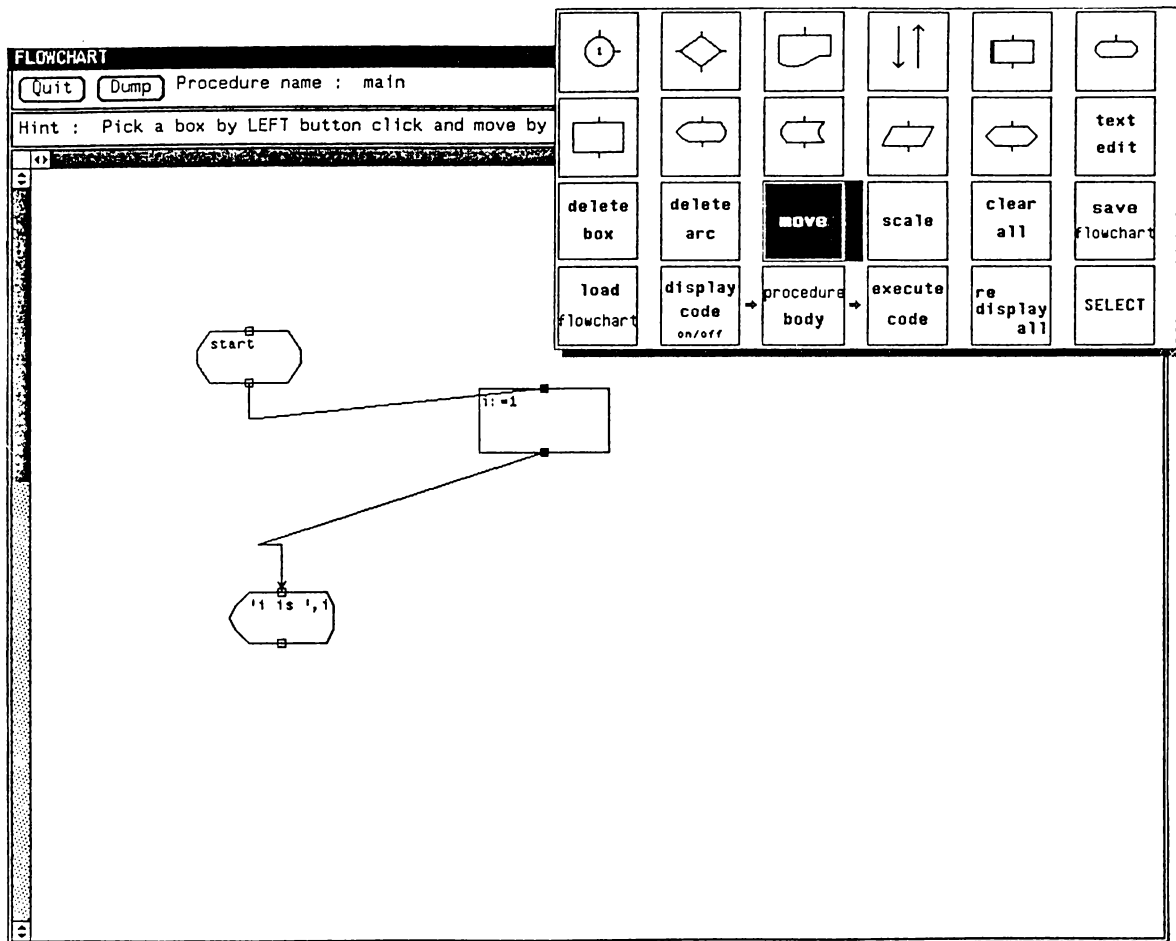


Figure A.4: Moving a box.

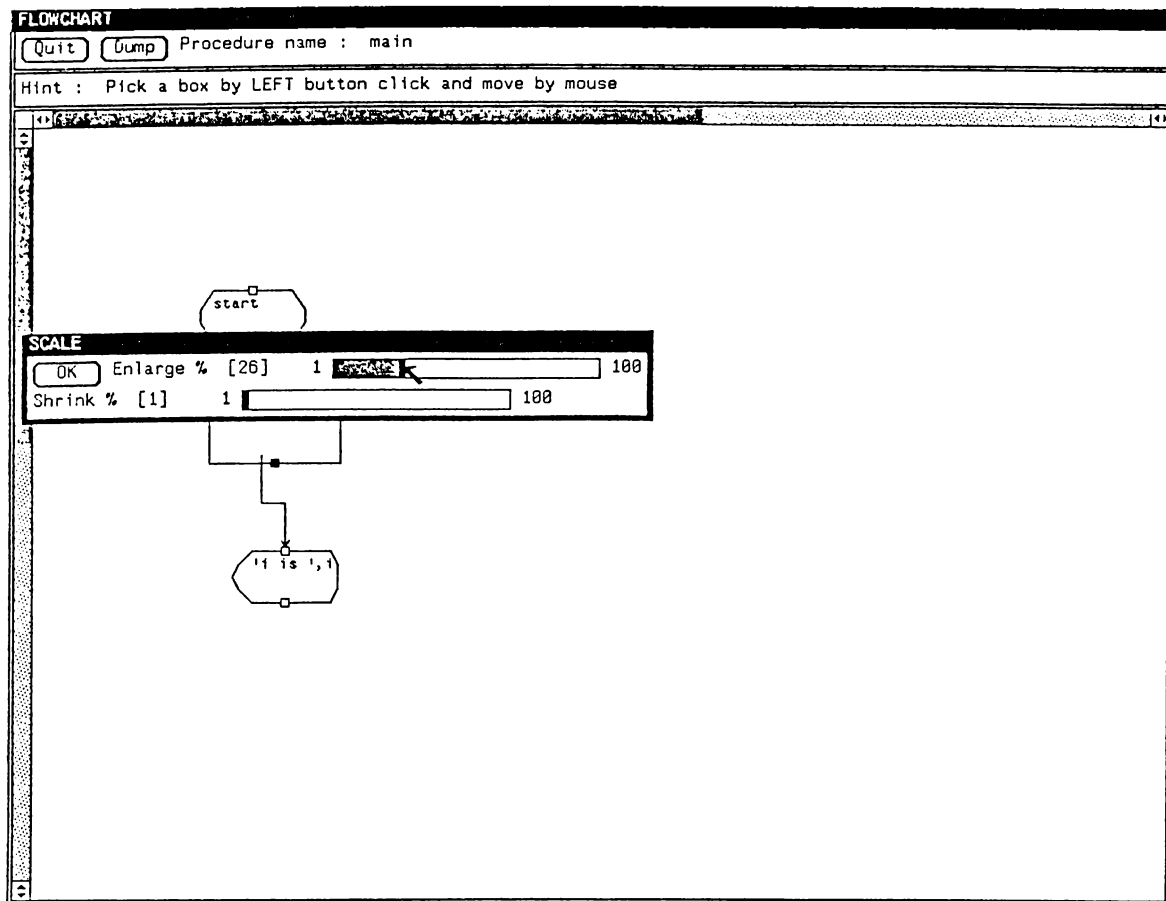


Figure A.5: Scaling a box.

- Scaling a box

Size of the boxes can be changed by choosing the 'scale' item from the flowchart menu, after 'SELECT'ing a box to be scaled. Then, immediately appears a scale window containing two sliders for percentage enlargement and percentage shrinking. Sliders are activated by moving the mouse over them, while the left mouse button is down. As soon as the button is released, the size of the box changes according to percentage of slider values (Figure A.5).

- Changing the text within a box

In order to edit the textual information contained in a box, first the box must be 'SELECT'ed. Then choosing 'text edit' item of the menu, a text entry window appears containing the textual information of the box -if full-, or empty -if no text-. The characters can be deleted and new ones can be added inside the text entry window. When 'OK' button of the window is clicked, edited text in the text entry window will go

into the box and fit its borders.

### **A.1.3 Redisplaying a Flowchart**

Clicking the left mouse button on the 'redisplay all' item of the menu first clears the drawing area and the whole flowchart is immediately redisplayed, i.e., drawing area is refreshed.

### **A.1.4 Saving a Flowchart to a File on the Disk**

While drawing and editing a flowchart or when one is finished, it can be saved onto a disk file together with its sub-flowcharts to be used later.

- Entering a file name to store the flowchart

When the 'save flowchart' item is selected to store a flowchart on the disk, a text entry window will appear immediately asking file name for saving the flowchart. After typing in the file name and clicking the 'OK' button on the text entry window, the flowchart on the drawing area and its sub-flowcharts are written to disk with the file name just read. If the file name already exists, it is overwritten. The work on the flowchart residing on the drawing area can still continue.

### **A.1.5 Loading a Flowchart from a File on the Disk**

A previously saved flowchart with its sub-flowcharts can be loaded from disk to operate on, if required.

- Entering a file name to get the previously saved flowchart from disk

After 'load flowchart' item of the menu is activated, a text entry window is displayed asking a file name to get the previously saved flowchart from disk. When the file name is entered and 'OK' button is pressed, the existing flowchart on the drawing area is cleared, and the new flowchart from the disk is displayed there -if exist-. If no such file exists, the operations can continue on the existing flowchart.

### **A.1.6 Erasing a Flowchart from the Screen**

A flowchart residing in the drawing area can be removed totally with its related sub-flowcharts by 'clear all' menu item. The drawing area is cleared, but it is not deleted from the disk, if saved before.

## **A.2 Code Generation**

When it is finished with the flowchart, its corresponding Pascal source code can be generated/displayed/printed.

### **A.2.1 Creating the Flowchart for Main Body of a Program**

When the tool starts running or a flowchart is loaded from the disk, it is initially ready to display the main body of the program that it is symbolizing. The name of the current flowchart -for initial case main flowchart, named 'main'- is displayed near 'procedure name:' message. If the main body contains no references to any sub-flowcharts, i.e. no 'call subroutine' box exists, it is finished and the corresponding Pascal code can be generated (Figure A.6).

### **A.2.2 Creating Flowcharts for Subprograms of a Program**

If the main body would reference -call- some sub-routines -indicated by 'call subroutine' box-, their flowcharts should be created by choosing 'proc body On' menu item. With 'proc body On', a text entry window immediately appears asking the name of the sub-flowchart. When name entry for the sub-flowchart is completed by pressing 'OK' button, the drawing area is cleared to create the sub-flowchart or it displays its existing flowchart when one is already created before. If this sub-flowchart also has some references to sub-subroutines, the same procedure applies again. For going up to one step, to the flowchart of calling routine -main- after completing the sub-flowchart, the 'proc body Off' item of the menu should be chosen (Figure A.7).



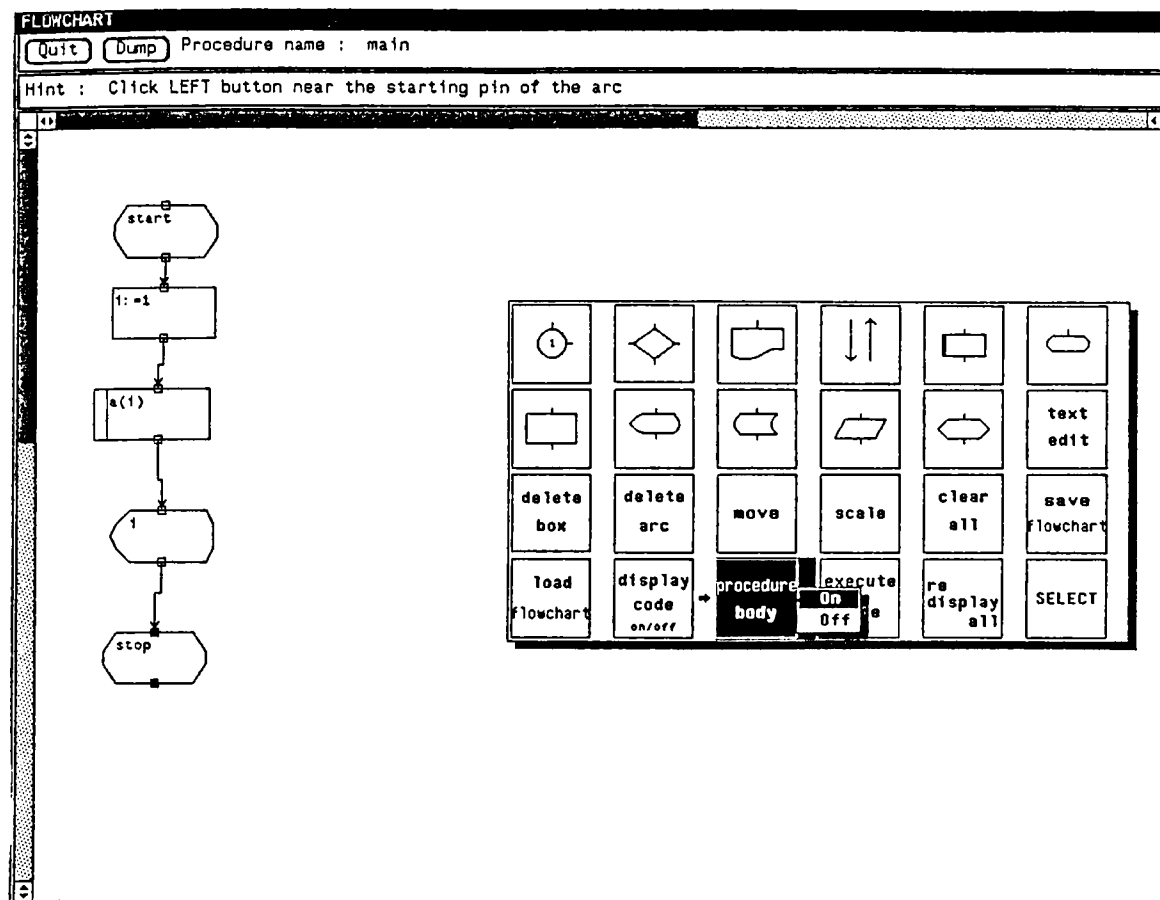


Figure A.6: Main flowchart.

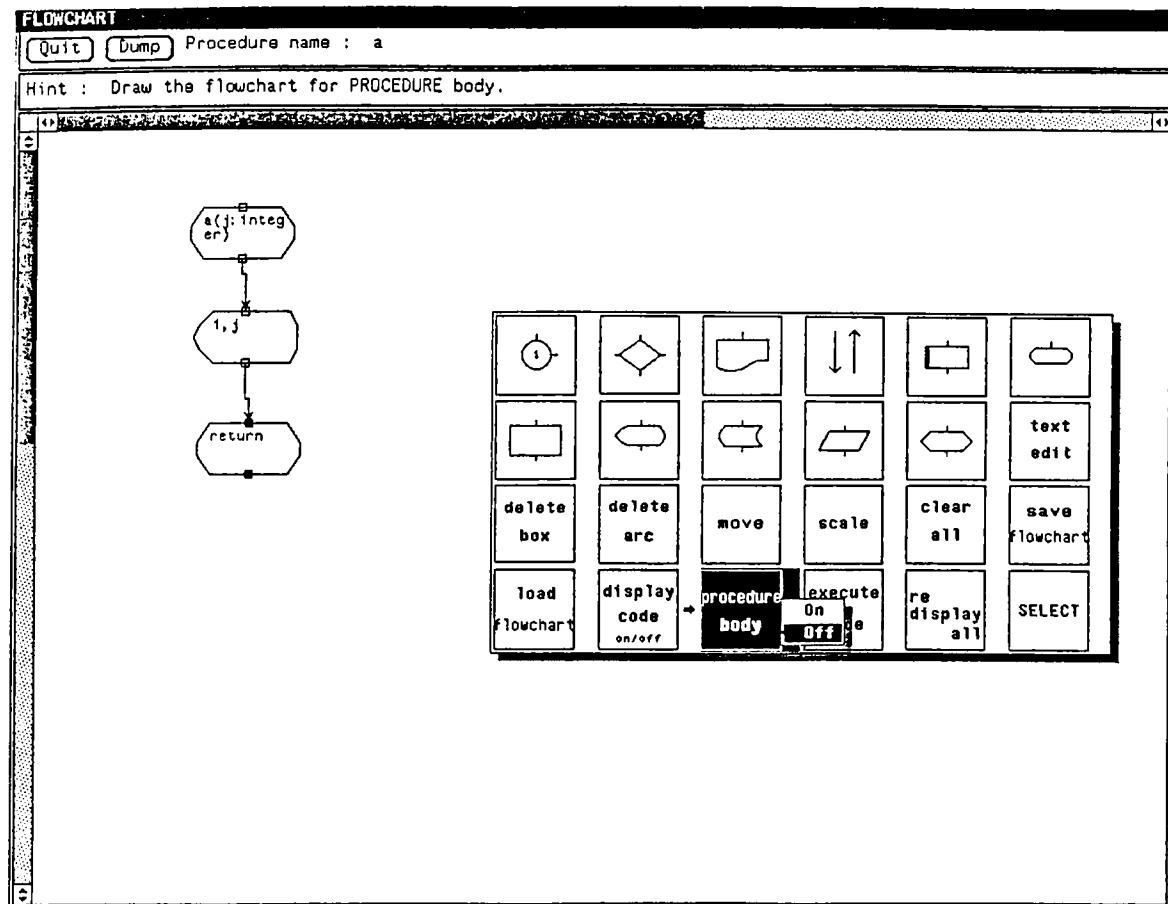


Figure A.7: Sub-flowchart.

If a routine has a sub-subroutine and that sub-subroutine also has its sub-sub-subroutine, each deeper one is created by going down through 'proc body On' item and drawing the flowchart of the routine. After the deepest routine, going back to the first calling routine is provided by clicking 'proc body Off' item that many times of 'Proc body On'. That is;

```
-main begin
|   -proc1 begin
|   |   -proc2 begin
|   |   |   -proc3 begin
|   |   |   |   .
|   |   |   |   .
|   |   |   |   .
|   |   |   -end proc3.
|   |   |
|   |   |   .
|   |   -end proc2.
|   |   .
|   |   .
|   -end proc1.
|   .
|   .
-end main.
```

If one routine has multiple sub-routines, however, first create one of them by 'proc body On', then return back to itself by 'proc body Off', then generate next subroutine by 'proc body On' and go back by 'proc body Off', and so on. That is;

```
-main begin
|   -proc1 begin
|   |   .
|   |   .
|   -end proc1.
|   -proc2 begin
|   |   .
|   |   .
|   -end proc2.
|   -proc3 begin
```

```

|      |
|      |      .
|      -end proc
|
|
|      .
-end main.

```

Each time the name of the working flowchart/sub-flowchart is displayed on the top of the drawing area near the 'Procedure name:' message.

### A.2.3 Generating the Pascal Code Corresponding to a Flowchart

The activation of 'display code On' item starts a series of operations to generate and display the Pascal source code corresponding to the flowchart on the drawing area (Figure A.8). These operations are giving a file name to write the Pascal source code, declaring the data types of the flowchart and sub-flowcharts, displaying the code etc.

- Entering a file name to write the Pascal source code on the disk

After code generation is started, a text entry window appears to get the file name for writing the Pascal source code of the flowchart. Text entry is completed by clicking on the 'OK' button (Figure A.9).

- Declaring data types in the main body of the flowchart

After the previous step, the declaration window immediately appears to get the types of the variables used in the main flowchart. Declaration is completed by clicking the 'OK' button on the declaration window (Figure A.10).

- Declaring data types in the sub-flowcharts

For each sub-flowchart, the tool asks for the declaration of its variables in the same way above. While the variables of each subroutine are being declared, the corresponding sub-flowchart is displayed at the same time with its name appearing near the 'Procedure name:' message (Figure A.11, A.12).

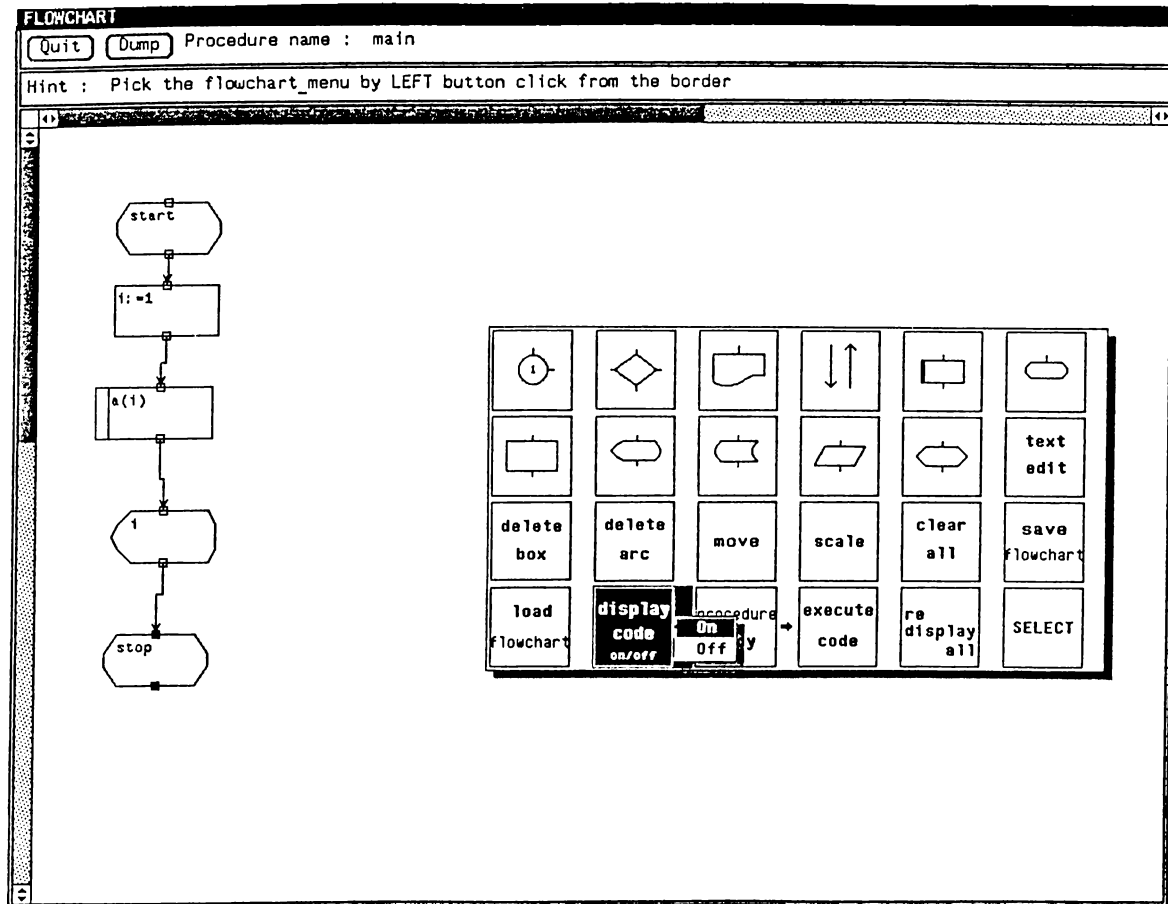


Figure A.8: Generating the Pascal code.

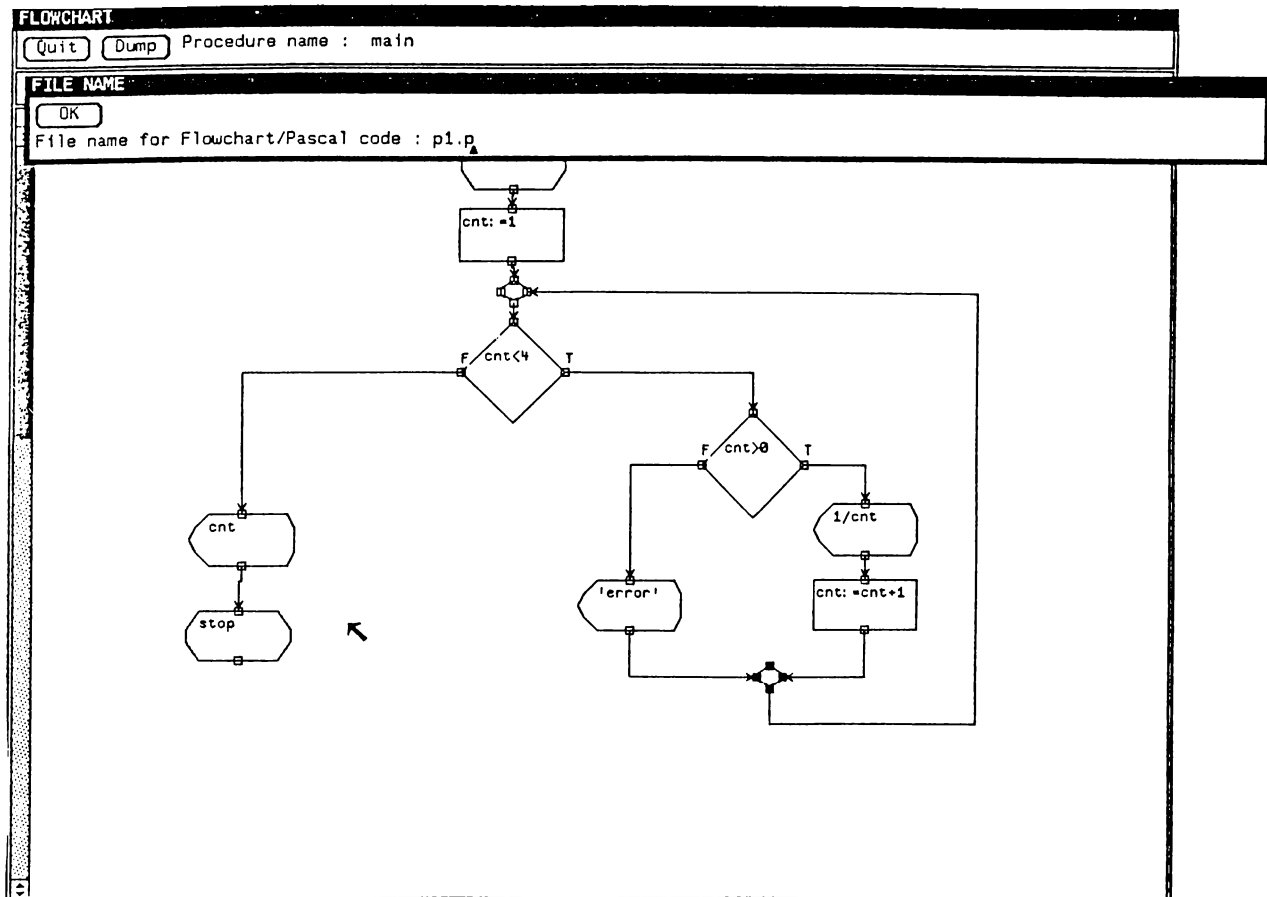


Figure A.9: A file name for the Pascal source code.

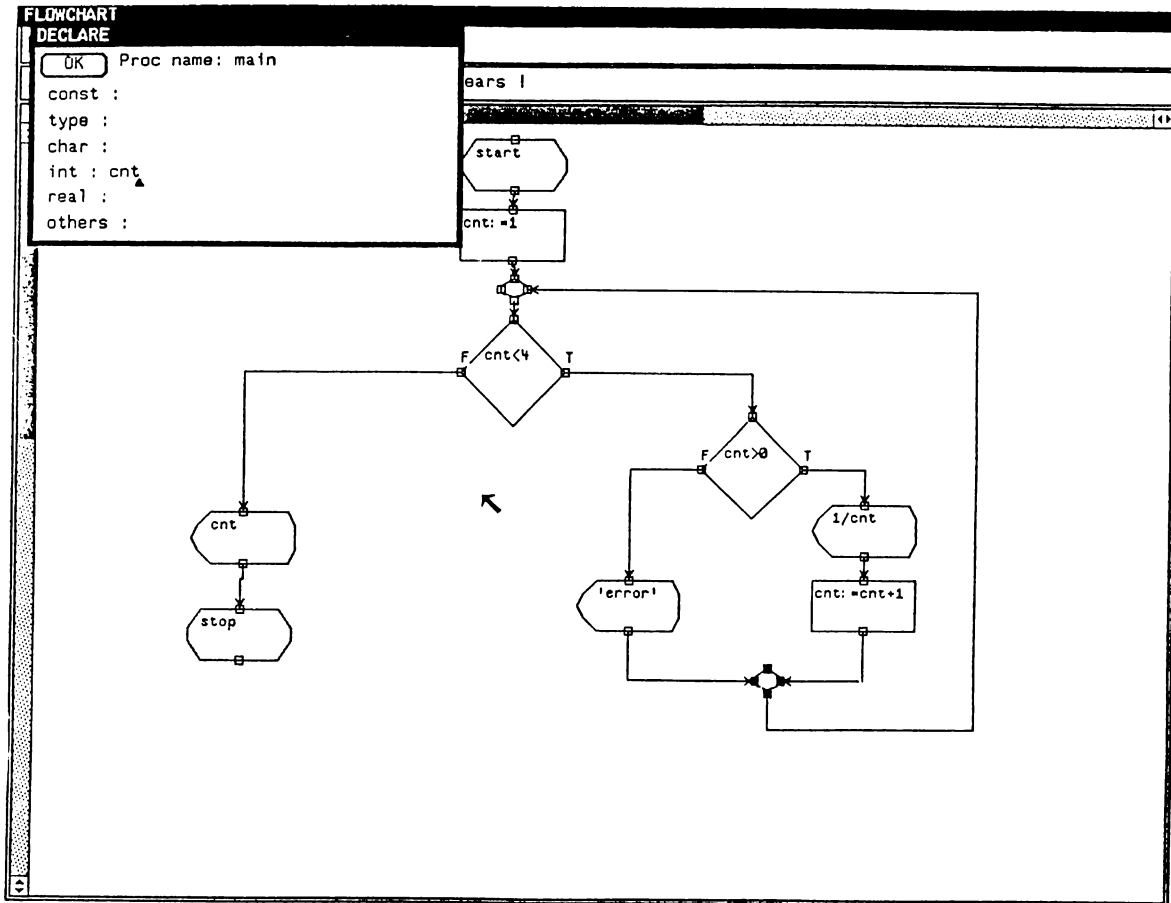


Figure A.10: Declaring data types in the main body.

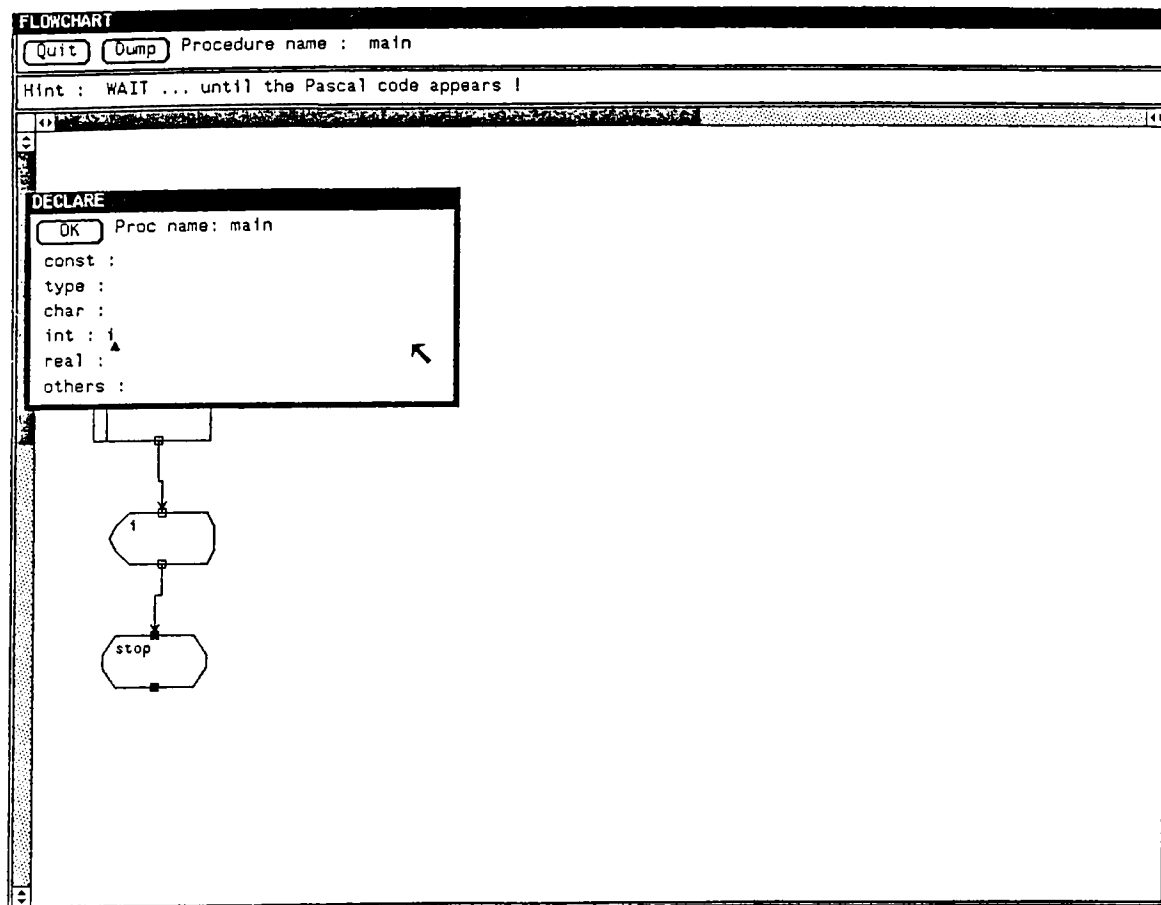


Figure A.11: Declaring data types in the main body.



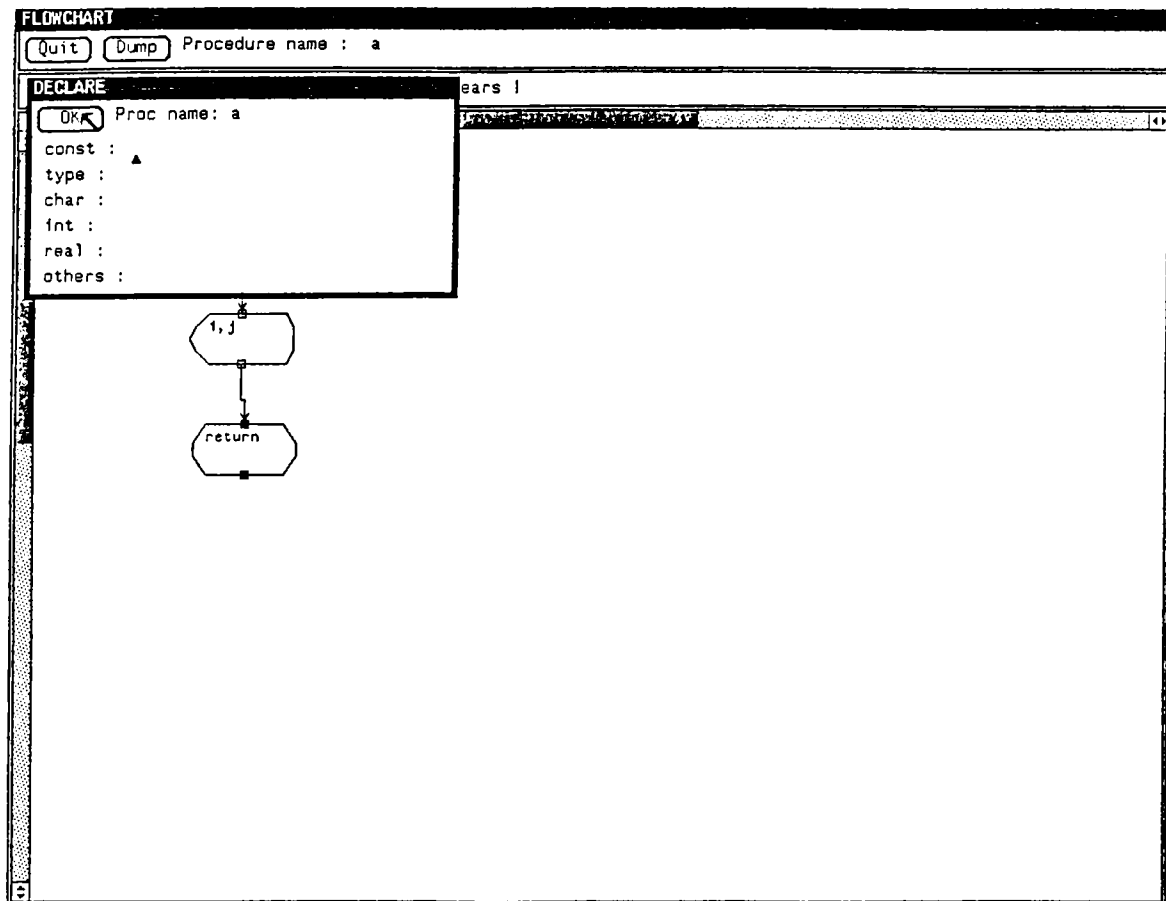


Figure A.12: Declaring data types of the sub-flowchart of the previous figure.

- Generating the code

Completion of declarations starts generation of Pascal code and writes the Pascal statements corresponding to the flowchart on a disk file name of which was just read.

- Displaying the Pascal source code on a display window page by page

After the source code is written to disk, a code display window appears to display Pascal source code of the flowchart. If the code is too long it can be scrolled up and down through the use of 'UP' and 'DOWN' buttons of the code display window (Figure A.13-a,b).

- Printing the Pascal source code from the printer

If wanted by the user, listing of the source Pascal code can be printed from the matrix printer connected to the system by activating 'PRINT' button.

- Removing the code display window

Once the code is generated and displayed in the code display window, the window can be destroyed in two ways. If the 'code generate Off' item is used from the flowchart menu, the code display window is permanently destroyed and next selection of the 'code display On' item causes the Pascal code of the flowchart to be generated from the beginning again. However, by the use of 'QUIT' button of the code display window, only the window will disappear temporarily. When the 'code generate On' item is used, the window will appear again and the Pascal code previously generated will be displayed. Therefore, the code will not be generated again, reducing manual operations like declarations and consume user's time.

### A.3 Execution of the Pascal Code for a Flowchart

Through the selection of the 'execute code' item of the flowchart menu, the Pascal source code corresponding to the flowchart on the drawing area will start to execute step by step under the user control, highlighting the flowchart box corresponding to the executing Pascal statement, and displaying the values of simple variables for each executing routine. When a subroutine is called during the execution, calling flowchart disappears and the sub-

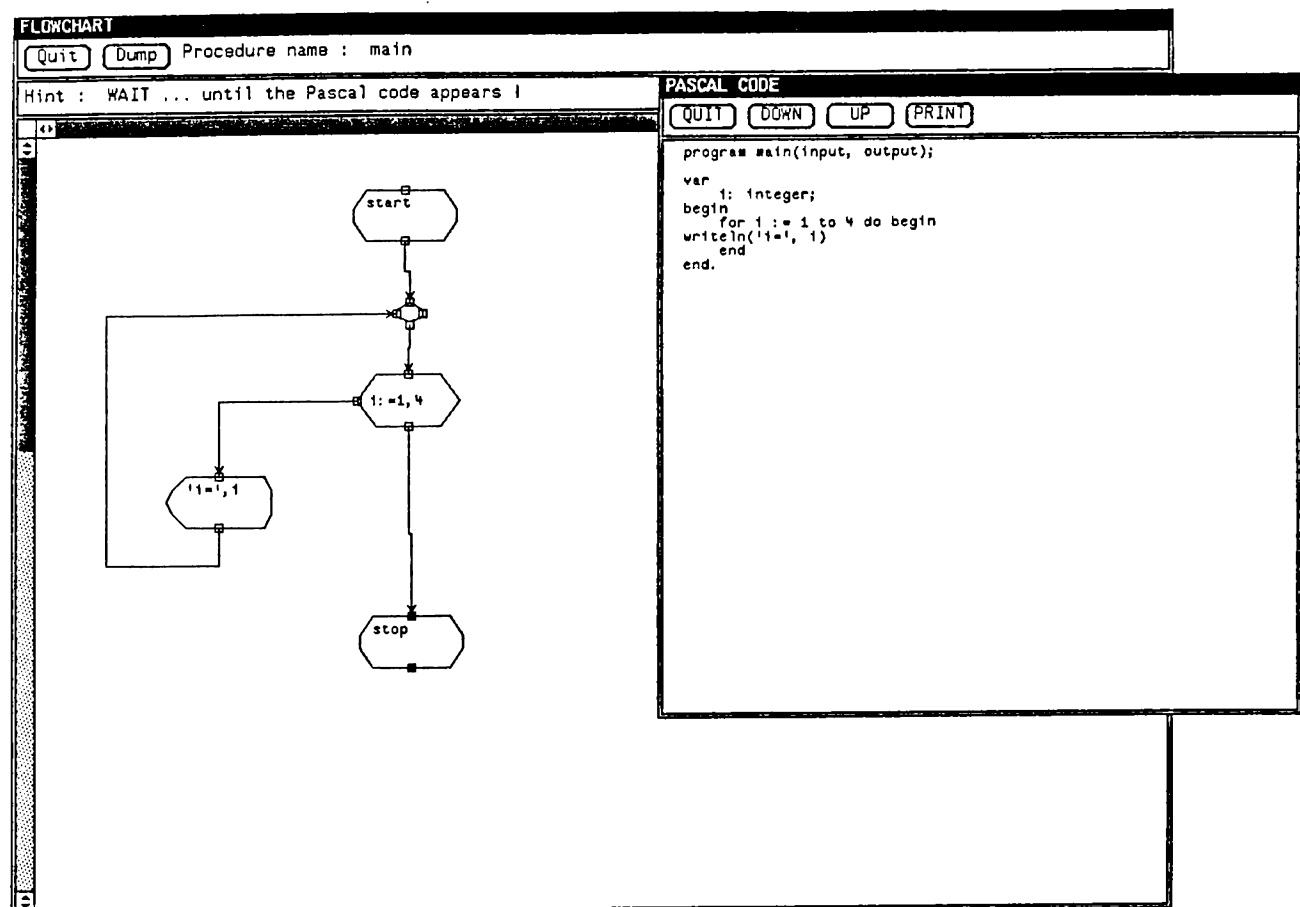
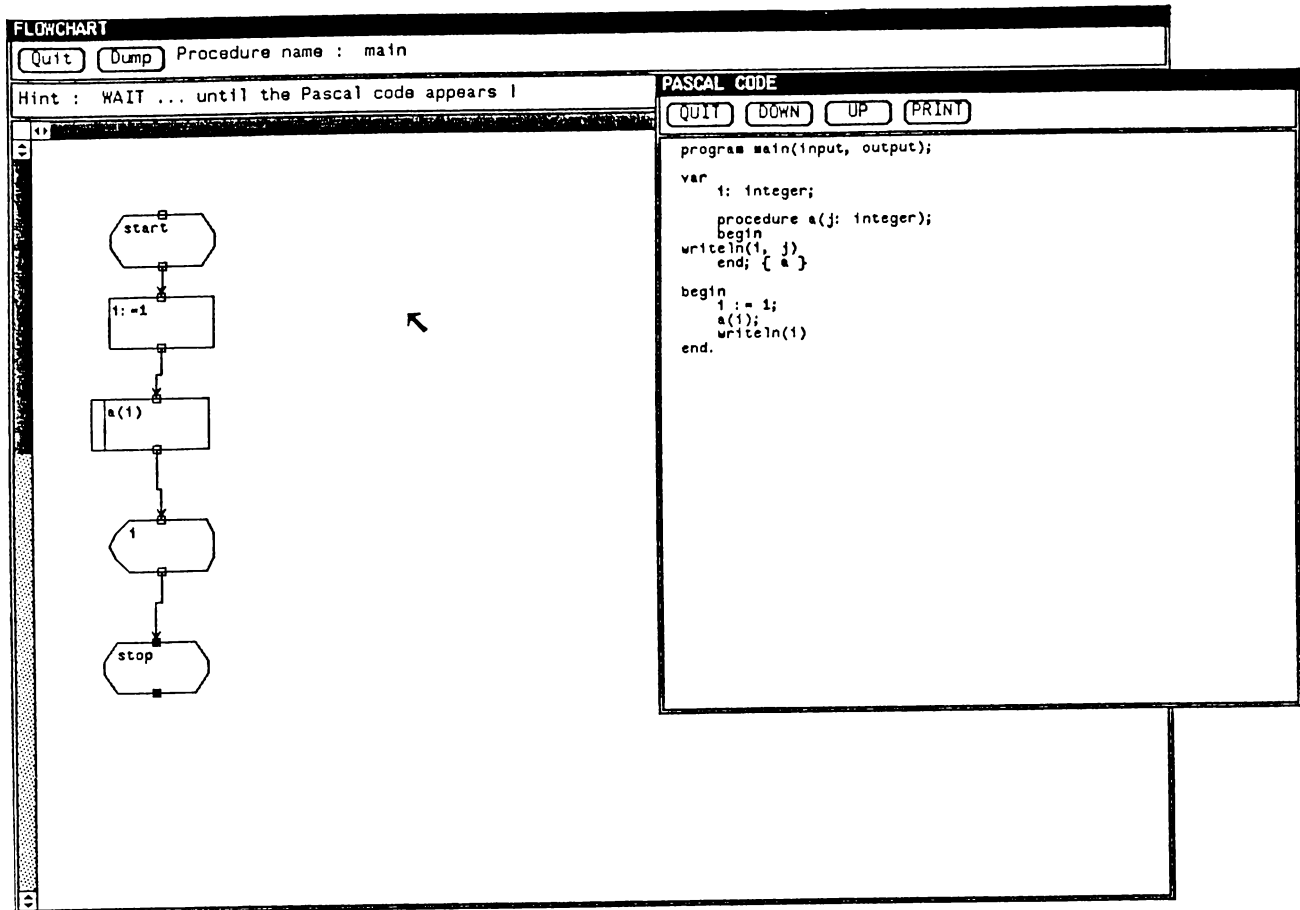


Figure A.13: Displaying the Pascal source code.

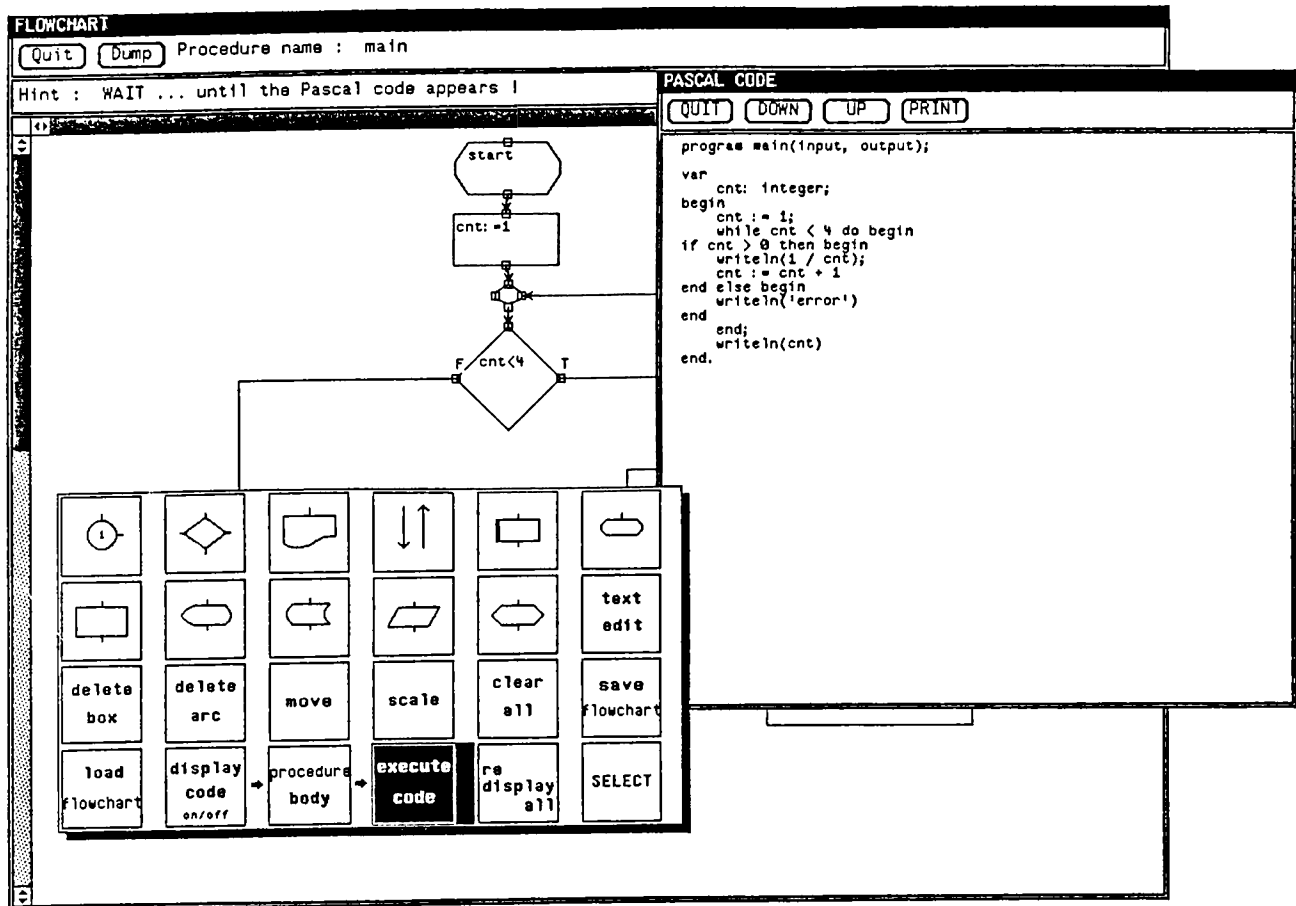


Figure A.14: Execution of the code.

flowchart appears while execution continues over it. Returning from the sub-flowchart causes the calling program to be redisplayed and continue execution (Figure A.14).

When the 'execute code' item is activated, if Pascal source code to be executed contains some compilation time errors, code execution does not start. Instead, in a display window, error messages of the Pascal compiler is displayed to the user to give some references to Pascal source code. According to the messages, it is required by the user to correct syntax/semantics errors of the flowchart, and redo the whole operation again (Figure A.15-a,b).

### A.3.1 Stepwise Execution of the Code

After the execution of the code starts from the first box -start box-, a small window containing a 'STEP' button appears and execution suspends waiting

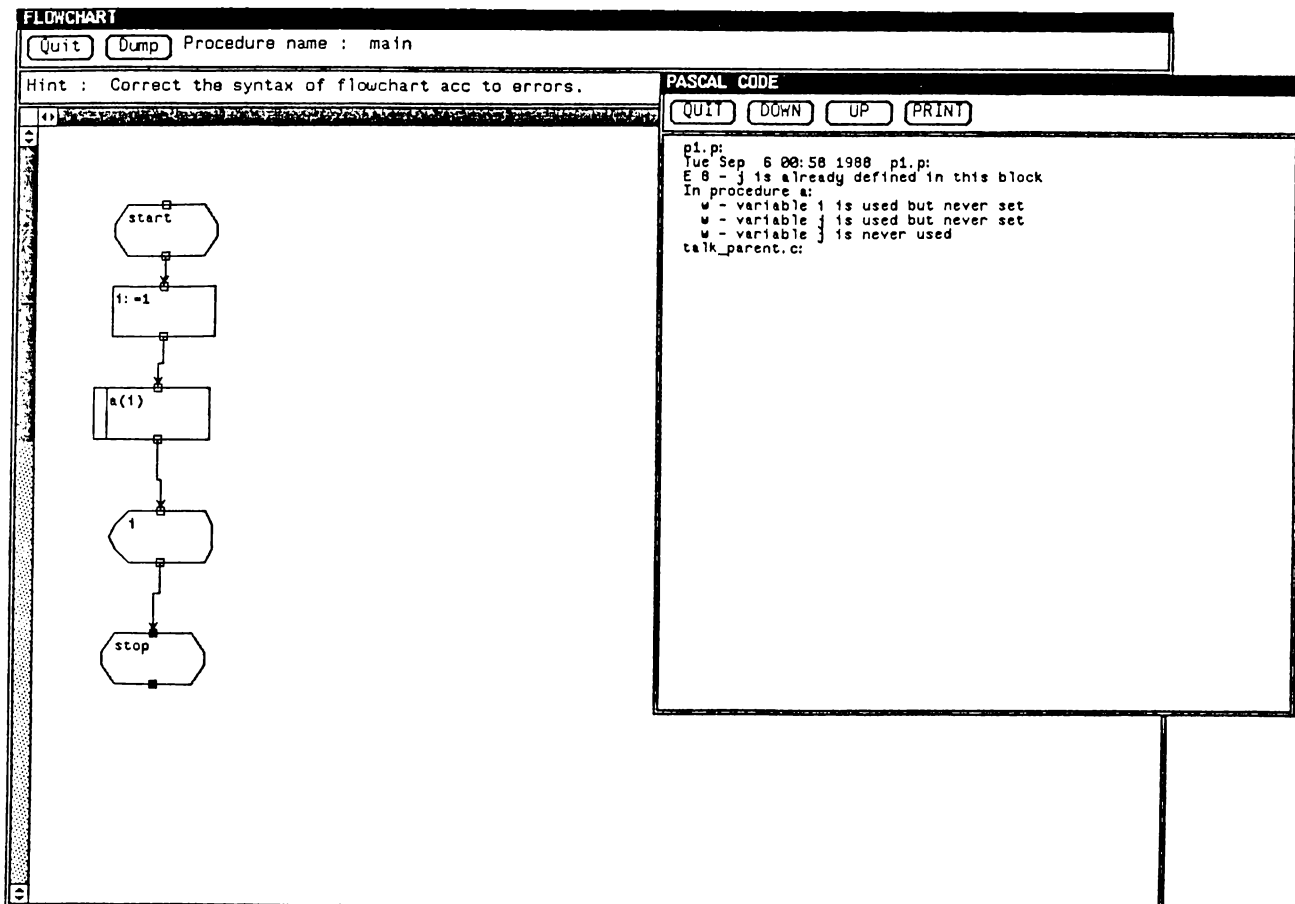
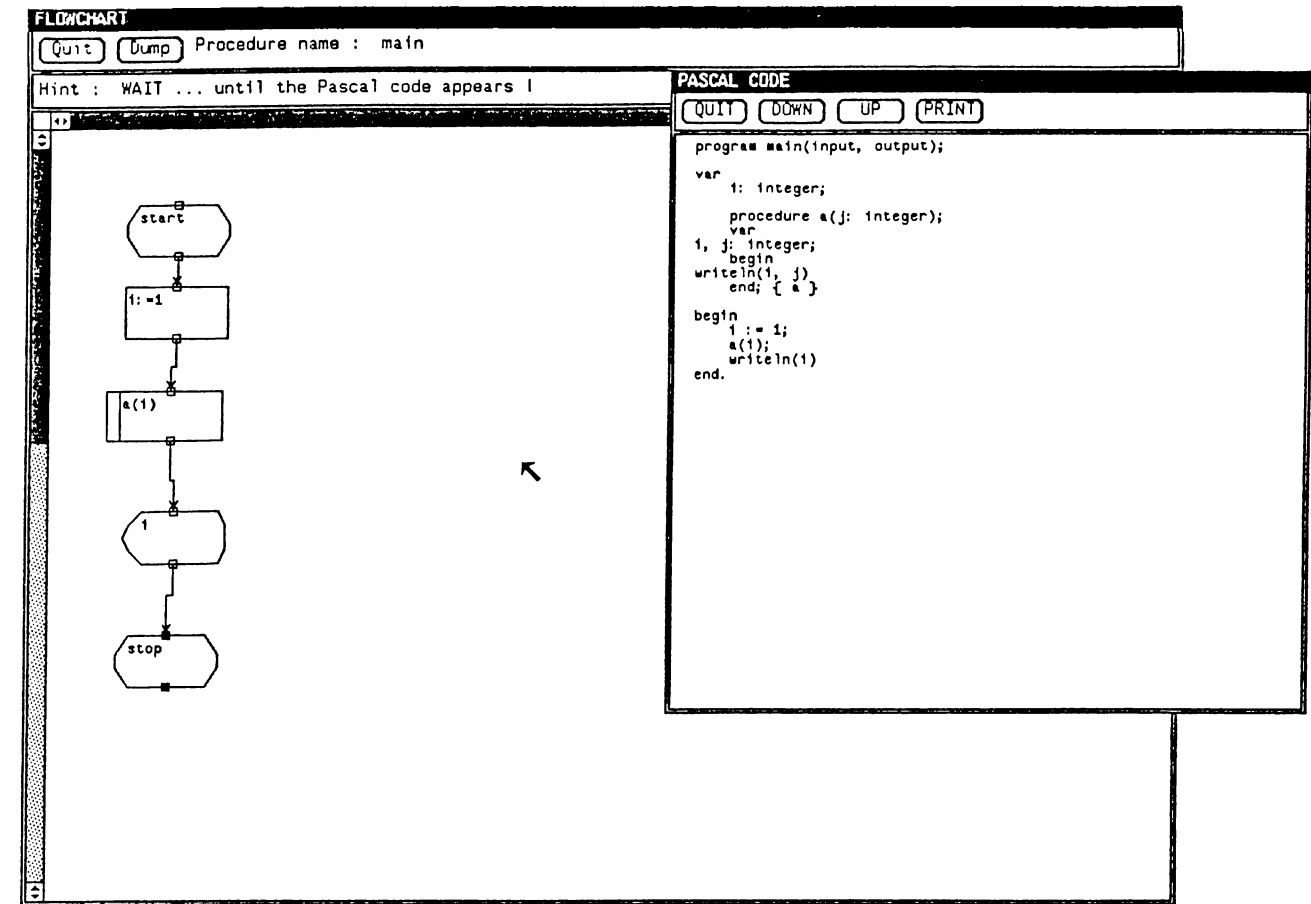


Figure A.15: Erroneous code and error messages.

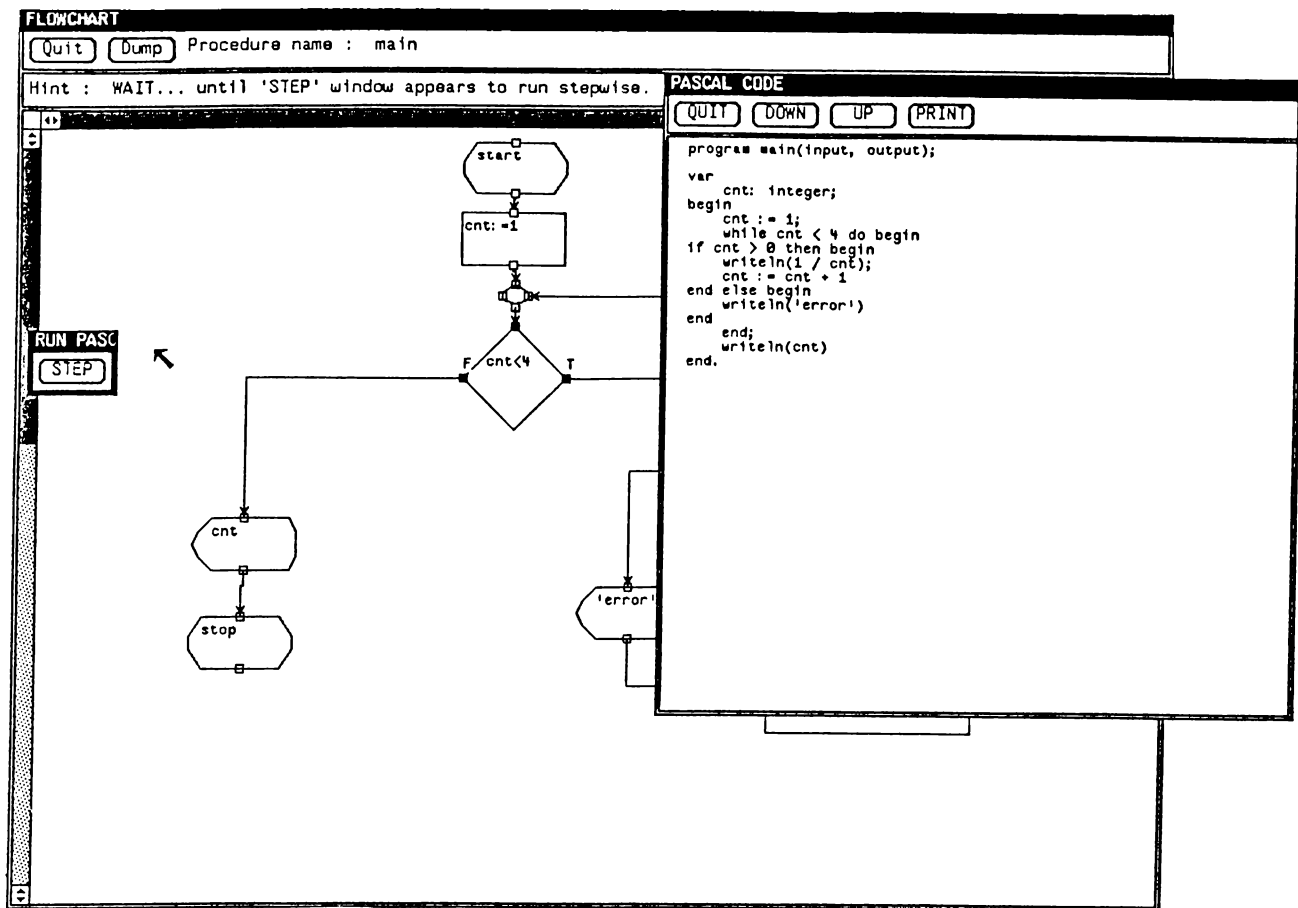


Figure A.16: 'STEP'ping.

the user to click on the 'STEP' button to continue execution at the next flowchart box. Each executing box waits for 'STEP'ping, and operation works in the same way until the last flowchart box -stop box- is encountered. Thus, the flowchart and its sub-flowcharts are traced through their boxes step by step under the user control (Figure A.16, A.17, A.18, A.19).

### A.3.2 At Each Step Highlighting the Corresponding Flowchart Box

While stepping through the symbols of the flowchart, each executing flowchart box is highlighted to show execution steps. During the subroutine calls, the sub-flowchart is displayed and execution continuous over it until a return to the calling routine is reached. When execution comes to the last symbol -stop box-, 'STEP'ping the last box completes the execution of the code.

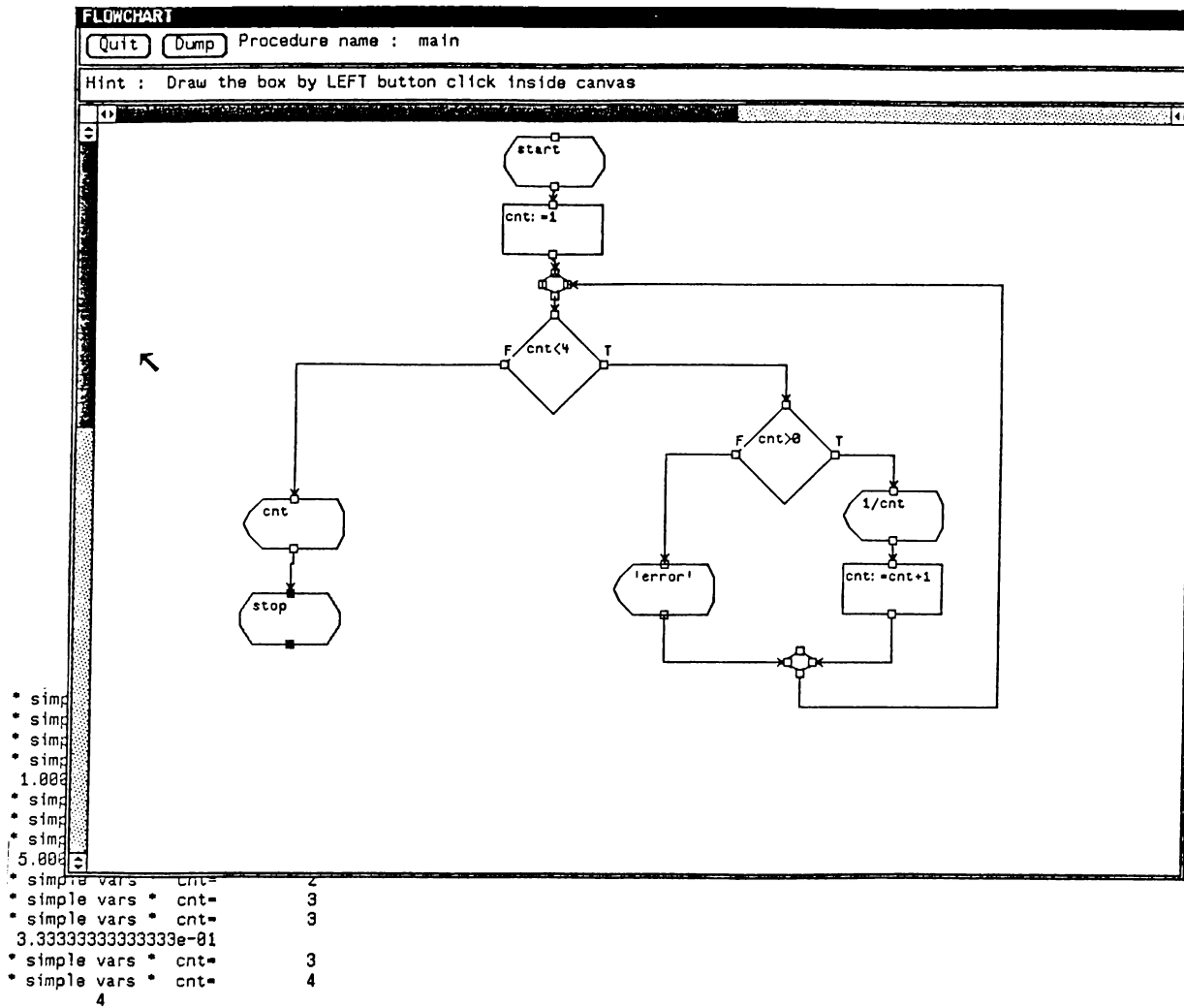


Figure A.17: End of 'STEP'ping while printing values of simple variables.

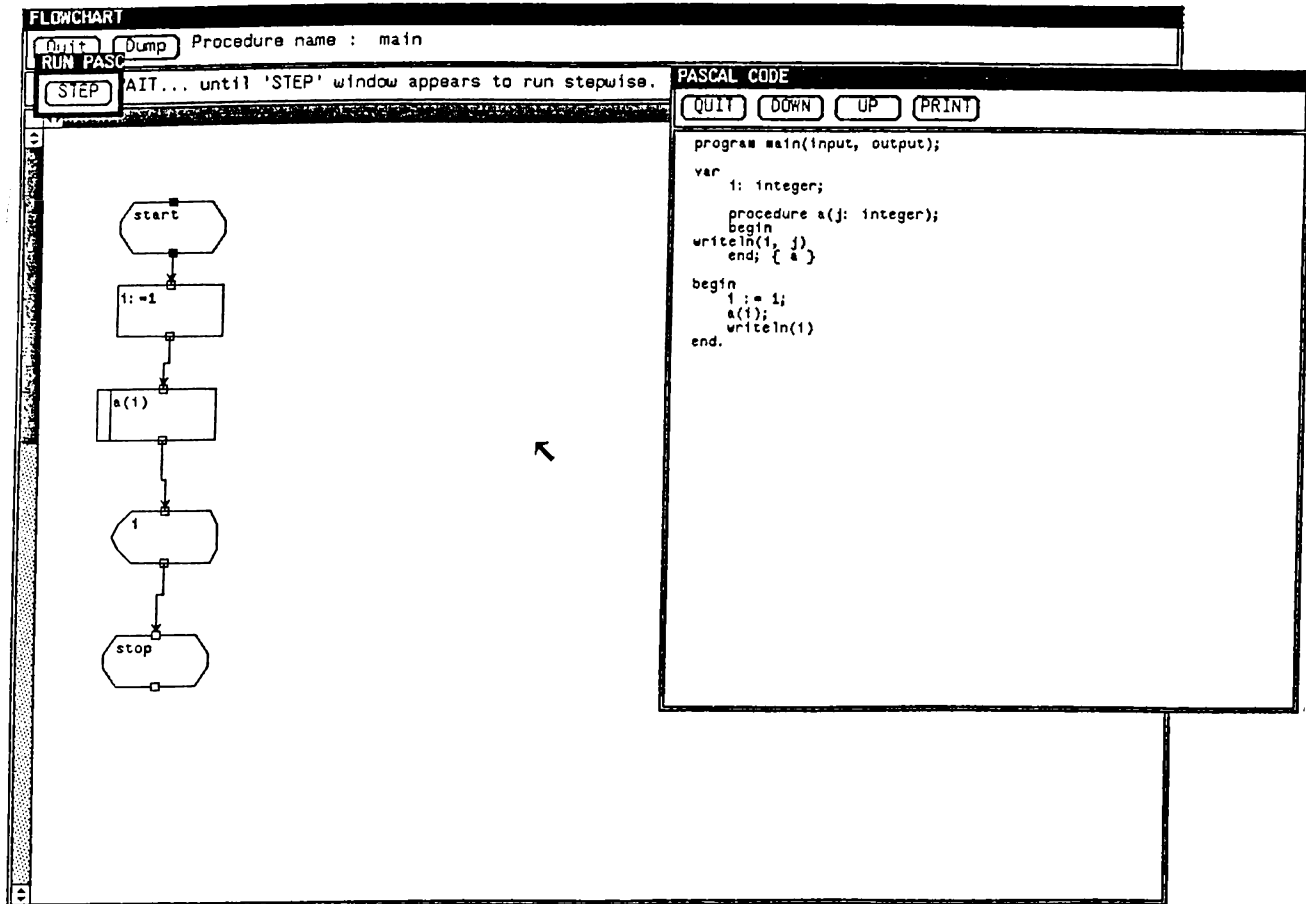
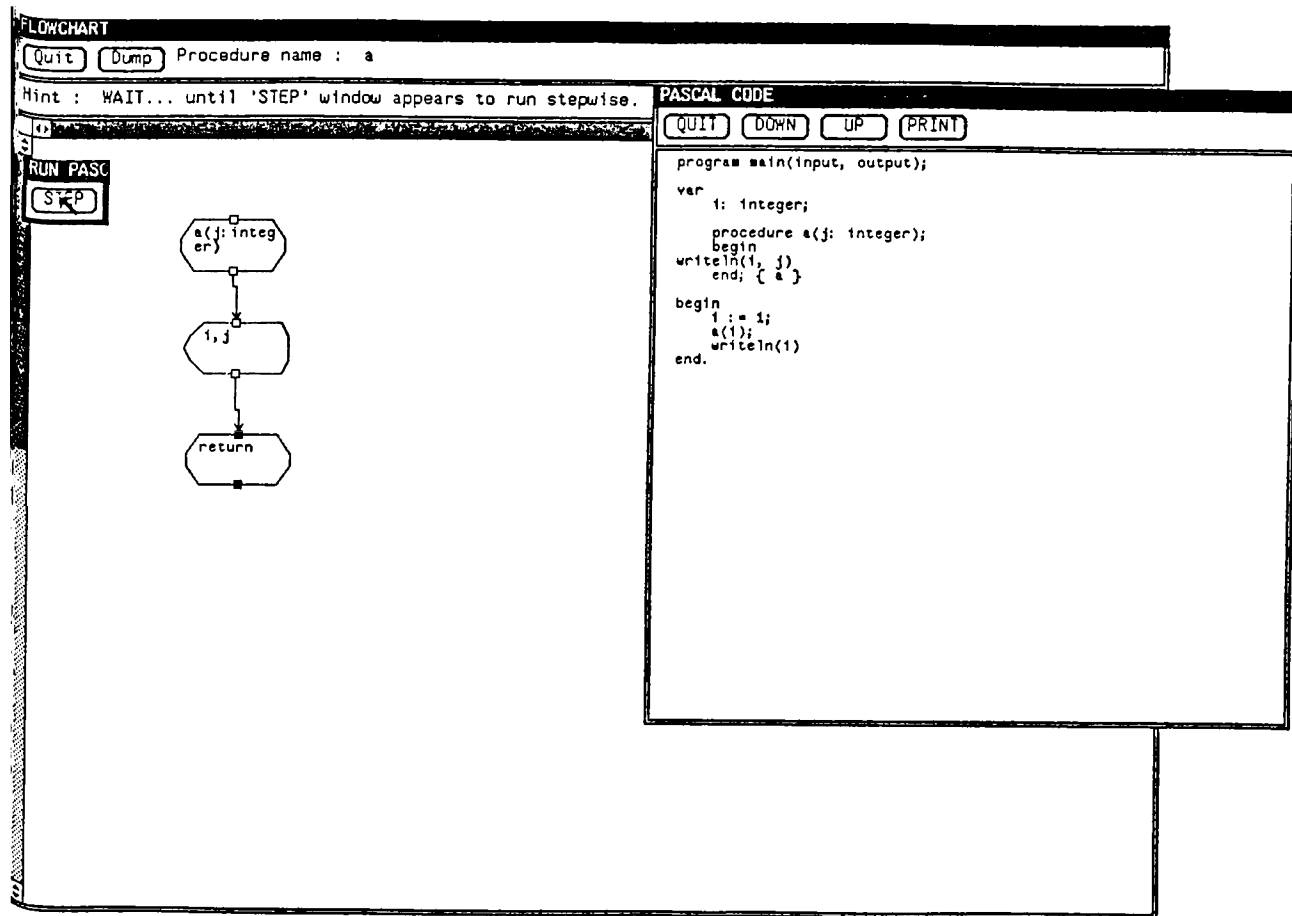


Figure A.18: 'STEP'ping in a main flowchart.





```

le vars * i=      0
le vars * i=      1
le vars * i=      1
1          1

```

Figure A.19: 'STEP'ping in a sub-flowchart.

### **A.3.3 At Each Step Displaying the Values of Simple Variables of the Currently Executing Routine**

While stepping through the boxes of the flowchart, at each step the values of simple variables of the executing routine are displayed to the user. These are the 'char', 'integer', and 'real' type variables.

### **A.4 Leaving the Tool**

After it is finished with the tool, the user can leave it by clicking the left mouse button on the 'QUIT' button over the drawing area.

### **A.5 Miscellaneous**

There exists some help tools and additional facilities of the tool, besides those listed above (Figure 3.1).

- **HINT**

For each item chosen from the menu, a help message describing what to do next for correct action guiding the user, appears on top of the drawing area.

- **Procedure Name**

During flowchart creation/editing/code generation/code execution phases, the name of the working flowchart/sub-flowchart whose chart appears on the screen is placed near the 'Procedure name:' message on top of the drawing area.

- **DUMP**

The 'DUMP' button dumps the image of the whole screen to a disk file in a specific format for a laser printer output.

- **SCROLLABLE drawing area**

The drawing area of the tool is not restricted by the screen or window size and can be scrolled left/right and up/down through the scrollbars attached to left and upper sides of the frame surrounding the drawing

area. The scrollbars are activated by the buttons of the mouse to scroll the drawing area.

## REFERENCES

- [1] Lodding, Kenneth N. *Iconic Interfacing*. IEEE CG&A, March/April, 11-20(1983).
- [2] Sibert, John L., Hurley, William D., Bleser, Teresa W. *An Object Oriented User Interface Management System*. ACM Siggraph '86 20(4), 259-268(Aug 18-22).
- [3] Newman, William N., Sproull, Robert F. **Principles of Interactive Computer Graphics**. McGraw-Hill, Tokyo(1981).
- [4] Özgüç, Bülent H. *Thoughts On User Interface Design For Multiwindow Environment*. Second International Symposium on Computer and Information Sciences, Istanbul. 477-488(1987).
- [5] Sun Microsystems Inc. *SunView<sup>TM</sup> Programmer's Guide*. (1982-86).
- [6] Banerjee, Jay., Chou, Hong-Tar., Garza, Jorge F., Kim, Won., Woelk, Darrel., Ballou, Nat., Kim, Hyoung-Joo. *Data Model Issues for Object Oriented Applications*. MCC Technical Report. (November 12, 1986).
- [7] Wisskirchen, Peter. *Towards Object Oriented Graphics Standards*. Comput. & Graphics 10(2), 183-187(1986).
- [8] Sun Microsystems, Inc. **Sun-3 Architecture**. A Sun Technical Report. (1985-86).
- [9] Tassel, Dennie Van. **Program Style, Design, Efficiency, Debugging, and Testing**. Prentice-Hall, Inc., Englewood Cliffs(1978).
- [10] Densmore, Owen M., Rosenthal, David S. H. *A User-Interface Toolkit in Object Oriented POSTSCRIPT*, Computer Graphics Forum 6, 171-180(1987).
- [11] Diederich, J., Milton, J. *Experimental Prototyping in Smalltalk*, IEEE SOFTWARE, 50-64(May 1987).

- [12] Tesler, L. *The Smalltalk Environment*, Byte, 90-147(Aug 1981).
- [13] Goldberg, A., Robson, D. **Smalltalk-80: The Language and Its Implementation**. Addison- Wesley, Reading, Massachusets(1983).
- [14] Reader, G. *A survey of Current Graphical Programming techniques*, Computer 18(8), 11-25(Aug. 1985).
- [15] Zloff, M. M. *Classification of visual programming languages*, IEEE Computer Society Workshop on Visual Languages, 232-235(1984).
- [16] Glinert, E. P. *Towards 'Second Generation' Interactive Graphical Programming Environments*, IEEE Workshop on Visual Languages 61-70(1986).
- [17] Reiss S. P. *PECAN: Program Development Systems that Support Multiple Views*, IEEE Transactions on Software Environments SE-11(3), 276-285(Mar. 1985).
- [18] Catteneo, G., Guercio, A., Levialdi, S., Tortora, G., *IconLisp: An example of a Visual Programming Language*, IEEE Computer Society Workshop on Visual Languages, 22-25(1986).
- [19] Glinert, S. P., Tanimoto, S. L., *PICT: An Interactive Graphical Programming Environment*, Computer 17(11), 7-25(Nov. 1984).
- [20] Yoshimoto, I., Monden, N., Hirakawa, M., Tanaka, M., Ichikawa, T. *Interactive Iconic Programming Facility in Hi-Visual*, IEEE Computer Society Workshop on Visual Languages, 34-41(1986).
- [21] Göktepe, M., Özgüç, B., Baray, M., *Design and Implementation of a Tool for Teaching Programming*, Computers and Education, (in print, 1988).