1-1-1998

Automatic flowchart displays for software visualisation

Paul R. Schurmann
Edith Cowan University

Edith Cowan University

Copyright Warning

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Automatic Flowchart Displays for Software Visualisation

By

Paul Robert Schurmann

Bachelor of Applied Science (Information Science)

A dissertation submitted in partial fulfilment of the

Requirements for the Award of

Master of Science

At the

Faculty of Science, Technology and Engineering,

Edith Cowan University

Date of Submission: May 1998

Abstract

Understanding large software projects and maintaining them can be a time consuming process. For instance, when changes are made to source code, corresponding changes have to be made to any related documentation. One large section of the documentation process is the creation and management of diagrams. Currently there are very few automated diagramming systems that can produce diagrams from source code, and the majority of these diagramming systems require a significant amount of time to generate diagrams.

This research aims at investigating the process of creating flowchart diagrams from source code and how this process can be fully automated. Automating the diagram creation process can save the developer both time and money. By saving the developer time we allow the developer to concentrate on more critical areas of their project.

This thesis will involve the design and implementation of a prototype software tool that will allow the user to quickly and easily construct meaningful diagrams from source code. The project will focus directly on the interpretation of the Pascal language into Flowcharts. The emphasis of the project will be on the arrangement of the flowchart, with a goal to create clear and understandable diagrams.

Acknowledgments

The author wishes to thank **Dr. Wei Lai** for his invaluable technical advice and expertise in the area of Software Visualisation. The author would also like to thank **Dr. James Millar** for his expert advice in the area of thesis content and presentation.

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in an institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signature

Date 11/12/98

Table of Contents

List of Figures

1. Introduction

1.1. The Aim

Over recent years computing power has increased considerably and as a result software projects have increased in size and complexity. In the past, it has been demonstrated that the majority of software costs are associated with the maintenance of these projects. Often work is doubled during the maintenance phase. For instance when a change is made to the source code, a corresponding change is required in the documentation. It is evident that we need to make this task simpler and more manageable. Automating this task can greatly simplify the developer's job. One large area of the documentation phase is the generation and maintenance of diagrams. The area of software visualisation addresses the problem of *creating diagrams from software automatically.*

This study aims to develop a working prototype of a Software Visualisation tool. Developing a Software Visualisation tool will help to gain an understanding into the problems of developing other Software Visualisation tools.

This study also contributes by developing routines that can be applied to other Software Visualisation projects. For instance, this project aims at generating flowcharts from the Pascal Language, we could then apply this to a project that generates flowcharts from C or ADA; Or a program that generates functional hierarchy diagrams from some programming language.

1.2. The Problem Addressed

The objective of this project is to address the problem of implementing a Software Visualisation system. This will be accomplished by developing a working prototype in which flowcharts will be generated from a source language such as Pascal. The process of creating flowcharts will be fully automated.

A number of problems or questions that will be identified during this research are as follows:

- Q1: What are the problems associated with automating the flowchart creation process? OR: What are the problems associated with implementing a Software Visualisation System?
- Q2: What are some possible solutions to displaying large complicated diagrams?
- Q3: How can we automatically display diagrams in a clear and uncluttered manner? OR: How do we address the graph drawing problem?
- Q4: How do we convert one-dimensional information into two-dimensional information. E.g. How do we convert Pascal source code into a Flowchart.

1.3. Advantages of Software Visualisation Systems

Below is a list of some advantages for using Software Visualisation systems.

- Eliminates the need for manually re-documenting software projects, as the computer generates all of the diagrams required.
- Creates a link between the source code and documentation/diagrams. This enforces consistency between source code and related documentation
- Programmers can view complex software quickly through generated diagrams, and easily gain a greater understanding of the software.

- The diagram creation process can be a difficult and often time-consuming activity. A good Software Visualisation system makes the diagram creation process simple no matter the size of the project.
- Software Visualisation systems can eliminate a large number of repetitive tasks.
- Provides the user with an instant source code diagramming system. Diagrams can be produced on demand with very little effort.
- Reduces time taken to comprehend foreign source code as the generated diagrams can be used to visually demonstrate the purpose of the program.
- Provides a diagramming system that can be adopted as a standard.
- Allows the developer to easily create documentation for systems that do not have any documentation.
- Learning curve for understanding the concepts of programming can be dramatically reduced. Eg. Beginner programmers can see pictorially the logic of their program.
- Time is saved producing documentation.
- Developer can quickly and easily produce high quality documentation for a system.
- Changes in system requirements can be documented quickly. Eg. Diagrams are produced directly from the source code so that the when the source code is changed so are the diagrams. This leads to a higher correlation between programs and documentation.
- Documentation generated from the Software Visualisation system can be compared to the initial system design. This helps to enforce consistency and to quality assure the product.

1.4. Contributions of this Thesis

Currently there are very few good Software Visualisation tools available, and the majority of these systems require a significant amount of input from the user or programmer to generate simple diagrams. This study is important, as it will develop a Software Visualisation tool that will *fully* automate the process of creating diagrams from source code. That is, exact geometric information such as symbol, line and text location for the diagram will not need to be specified from the programmer or user of the program.

This research is also important as it focuses on creating automatic drawing procedures for flowcharts. In current systems the user or designer needs to specify geometric information in detail, such as shape, size and location for every symbol in a flowchart component. Also this research looks at separating the logical part from the geometric part for a flowchart component by developing a solid model for the structural modelling of flowcharts.

1.5. The Structure of this Thesis

Chapter 2 gives a definition of Software Visualisation and the theory used for the generation of the prototype. The syntax of some Pascal statements is given to help understand the kind of statements that have to be converted into flowchart components. The related flowchart components for these Pascal statements are then described. This chapter also reviews two data-structures that will be used in the development of the prototype. This is then followed by a description of a layout adjustment routine called the force scan algorithm. In Chapter 2 we also see an analysis of existing software products and research done in the area of computer flowchart systems.

The design/theory of the prototype is given in Chapter 3. This chapter expands on the topics described in chapter 2 and describes the theory that will be used to construct the prototype. This chapter is broken down into a number of areas. The topics described in Chapter 3 relate to the process that a source file goes through to be converted into a flowchart diagram.

Chapter 4 uses the principles outlined in previous chapters to further develop source code examples written in the C-Language. Implementation issues such as speed and program representation are also examined during this chapter.

Chapter 5 illustrates the results from the development of the prototype. Output from the prototype will be demonstrated throughout this chapter by diagrams generated by the prototype. Sample Pascal code stubs will be used as test data for the prototype.

The results from chapter 5 lead to a discussion in Chapter 6 about the implementation and proven capabilities of the software prototype. The initial thesis questions will be re-addressed in this chapter and a discussion aimed at providing answers will be given.

The appendix section contains source code and sample output from the prototype. The appendix is used in conjunction with the results chapter to further demonstrate the capabilities of the prototype.

2. Literature Review

2.1. Definition of Software Visualisation

It is important that a distinction be made between Software Visualisation and Visual Programming. Myers illustrates the difference between Software Visualisation and Visual Programming by the definitions that follow:

Program.

"A 'program' is defined as a set of statements that can be submitted as a unit to some computer system and used to direct the behaviour of that system."

Oxford Dictionary (Cited in Myers, 1995)

Visual Programming

"Visual Programming (VP) refers to any system that allows the user to specify a program in a two (or more) dimensional fashion. Conventional textual languages are not considered two dimensional since the compilers or interpreters process them as long, one-dimensional streams. Visual Programming includes graphical programming languages and using conventional flowcharts to create programs. It does not include systems that use conventional (linear) programming languages to define pictures, such as, SKETCHPAD [Sut63], CORE, PHIGS, Postscript [Adobe 85], the Macintosh Toolbox [Apple 85], or X-11 Window Manager Tool-kit."

(Myers, 1995)

Program Visualisation. (Software Visualisation)

"Program Visualisation (PV) is an entirely different concept from Visual Programming. In Visual Programming, the graphics is the program itself, but in Program Visualisation, the program is specified in a conventional, textual manner, and the graphics is used to illustrate some aspect of the program or its run-time execution. Unfortunately, in the past, many Program Visualisation systems have been incorrectly labelled "Visual-Programming" (as in [GRAFTON 85]). Program Visualisation systems can be classified using two axes: whether they illustrate the code or the data of the program, and whether they are dynamic or static. "Dynamic" refers to systems that can show an animation of the program running whereas "static" systems are limited to snapshots of the program at certain points. If a program created using Visual Programming is to be displayed or debugged, clearly this should be done in a graphical manner, but this should not be considered Program Visualisation."

(Myers, 1995)

2.2. The Pascal Language

The Pascal language consists of a set of programming language statements. This section contains a sample of some commonly used programming language statements. These programming language statements can be compared with their related flowchart components shown in section 2.3.

2.2.1. IF Statement Syntax:

Form: *If Condition Then*

Statement $_T$

Else

Statement $_F$

Example: *If X >= 0.0 then*

Write ('Positive')

Else

Write ('Negative')

Interpretation: *If the condition evaluates to true, then Statement* $_T$ *executes; Otherwise, Statement* $_F$ *executes.*

(Koffman, 1989, p. 87)

Figure 1 IF Statement Syntax (Two Alternatives).

Form: *If Condition Then*

Statement $_T$

Example: *If X > 0.0 Then*

*PosProd := PosProd * X*

Interpretation: *If the condition evaluates to true, then Statement* $_T$ *executes; Otherwise, it does not execute.*

(Koffman, 1989, p. 87)

Figure 2 IF Statement Syntax (One Alternative)

2.2.2. *WHILE Statement Syntax*

Form: *While Expression Do*

Statement

Example: *CountStar := 0*

While CountStar < N do

Begin

Write ('');*

CountStar : = CountStar + 1

End {While};

Interpretation: *The Expression (a condition to control the loop process) is tested and if it is true, the statement is executed and the expression is re-tested. The statement is repeated as long as (while) the expression is true. When the expression is tested and found to be false, the while statement is executed.*

Note: *If the expression evaluates to false the first time it is tested, the statement is not executed.*

(Koffman, 1989, p. 121)

Figure 3 WHILE Statement Syntax

2.2.3. FOR Statement Syntax

Form: *For Counter := Initial to Final do*

Statement

For Counter := Initial downto Final do

Statement

Example: *For I:= 1 to 5 do*

Begin

ReadLn (InData, NextNum);

Sum : = Sum + NextNum

End;

For CountDown := 10 downto 0 do

Writeln (CountDown: 2)

Interpretation: *The statement that comprises the loop body is executed once for each value of counter between initial and final, inclusive, initial and final can be constants, variables, or expressions of the same ordinal type as counter.*

Note 1: *The value of counter cannot be modified in statement*

Note 2: *The value of final is computed once, just before loop entry. If final is an expression, any change in the value of that expression has no effect on the number of iterations performed.*

Note 3: *After loop exit, the value of counter is considered undefined.*

Note 4: *statement is not executed If initial is greater than final. (In the downto form, statement is not executed if initial is less than final.)*

(Koffman, 1989, p. 315)

Figure 4 FOR Statement Syntax

2.2.4. *REPEAT Statement Syntax*

Form: *Repeat*

Loop body

Until Termination Condition

Example: *Repeat*

Write ('Enter a digit: ');

Read (Ch)

Until ('0' <= Ch) ad (Ch <= '9')

Interpretation: *After each execution of loop body, termination condition is evaluated. If termination condition is True, loop exit occurs and the next program statement is executed. If termination condition is False, loop body is repeated.*

(Koffman, 1989, p. 317)

Figure 5 REPEAT Statement Syntax

2.2.5. CASE Statement Syntax

Form: *Case Selector of*

Label $_1$: Statement1;

Label $_2$: Statement2;

.

.

.

Label $_n$: Statement n

Example:

```
Case N of
    1, 2: Writeln ('Buckle my shoe');
    3, 4: Writeln ('Shut the door');
    5, 6: Writeln ('Pick up sticks')
End
```

Interpretation: *The selector expression is evaluated and compared to each case label. Each label $_i$ is a list of one or more possible values for the selector, separated by commas. Only one statement $_i$ is executed; if the selector value is listed in label $_1$, statement $_1$ is executed; if the selector value is listed in label $_i$, statement $_i$ s executed. Control is then passed to the first statement following end. Each statement $_i$ may be a single or a compound Pascal statement.*

Note 1: *If the value of the selector is not listed in any case label, an error message is printed and the program execution stops.*

Note 2: *A particular selector value may appear in, at most one case label.*

Note 3: *The type of each selector value must correspond to the type of the selector expression.*

Note 4: *Any ordinal data type is permitted as the selector type.*

(Koffman, 1989, p. 307)

Figure 6 CASE Statement Syntax

2.3. Flowcharts

Flowchart diagrams are created using a series of flowchart symbols. Individual programming language statements can be represented by an arrangement of flowchart symbols. An arrangement of flowchart symbols makes up a flowchart component. For example in section 2.2.1 the syntax of the IF Statement is given. A corresponding flowchart component in section 2.3.2.1 is used to represent this programming language statement. The flowchart component can be seen as a set of flowchart symbols organised in a set pattern/layout with the possibility of having sub-components.

Note: It has been found through research that a number of variations to the layout of flowchart components exist. This section illustrates a sample of some flowchart symbols and components.

2.3.1. Flowchart Symbols

Flowchart symbols are the building block for flowchart diagrams. Below is a sample of some ANSI flowchart symbols.

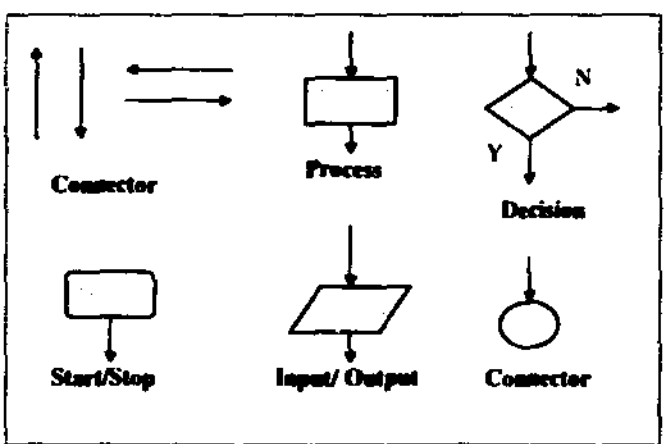


Figure 7 ANSI Flowchart Symbols

2.3.2. *Flowchart Components*

Flowchart Components are constructed using flowchart symbols. The figures to follow illustrate a sample of some flowchart components. E.g. IF, WHILE, REPEAT, FOR, and CASE Component. Each flowchart component described can contain one or more decision, process and connector symbols. For example, the IF Component contains 1 decision symbol, 2 process symbols, and a set of connectors.

2.3.2.1. *IF Component*

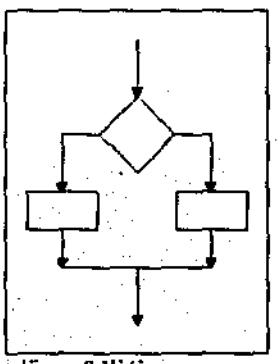


Figure 8 IF Component

2.3.2.2. *WHILE Component*

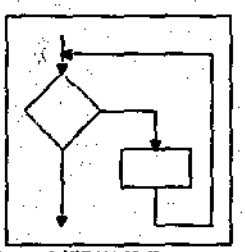


Figure 9 WHILE Component

2.3.2.3. *REPEAT Component*

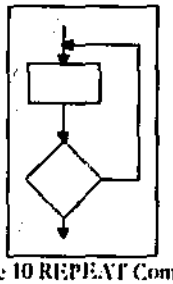


Figure 10 REPEAT Component

2.3.2.4. *FOR Component*

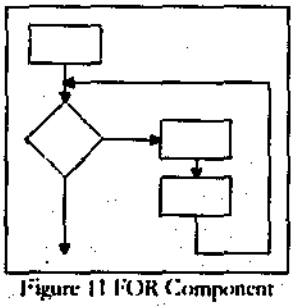


Figure 11 FOR Component

2.3.2.5. *CASE Component*

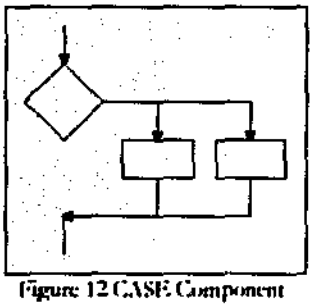


Figure 12 CASE Component

2.4. Data Structure

A data structure is required to store flowchart information. Potentially there are an infinite number of combinations that flowchart symbols can be joined together. This leads to the requirements of a *dynamic data structure* that can represent a flowchart. This section illustrates two data structures that can be used to store flowchart information. The linked list structure is suited to store relationships between flowchart components. The tree structure is suited to represent flowchart components.

2.4.1. Linked List Structure

(Shiftlet, 1990, pp. 215-326) describes the linked list data structure and the operations applied to the data structure. Although this is Pascal code, the code demonstrates the linked list structure and can easily be converted to other languages. The definition of the linked list structure is as follows:

```
Type
    Pointer = ^Node;
    Node = record
        Info : integer;
        Next : pointer
    End;
Var
    P, Q : Pointer
```

(Shiftlet, 1990, pp. 220)

Figure 13 Linked List Definition - Pascal Code.

2.4.2. *Tree Structure.*

The tree structure is used to store the nodes in a diagram. The tree structure is a suitable structure as it can represent diagrams that have a hierarchical structure. Figure 14 (a) demonstrates a node and its pointer to other nodes. Figure 14 (b) demonstrates a graphical representation of figure 14(a) in a tree structure form. Figure 14(c) demonstrates how this node can be used in combination with other nodes to form a hierarchical structure.

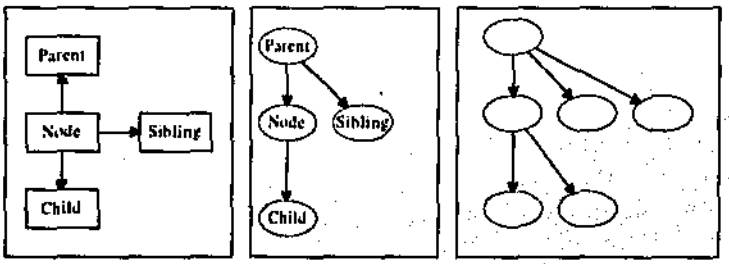


Figure 14 Tree Definition

```
Type
    Tree = ^Node;
    Node = record
        Parent, Child, Sibling: Tree;
        El: ElType;
    End
Var
    Diagram: Tree
```

Figure 15 Tree Definition -Pascal Code.

2.5. Layout Adjustment Routine

Layout adjustment algorithms can be applied to diagrams to improve the visual appearance of the diagram in some way. For example, a layout adjustment algorithm could be used to eliminate wasted space in a diagram. A number of algorithms exist to remove unwanted diagram characteristics as illustrated by (Misue, Eades, Lai, and Sugiyma, 1993, pp. 183-210). The layout adjustment algorithm used for this project is called the Force-Scan algorithm. The Force-Scan algorithm was chosen, as it is an extremely useful algorithm as it can be adapted to:

- Reduce wasted space.
- Eliminate overlap among diagram components.
- Produce a compact diagram while preserving the general shape of the diagram.

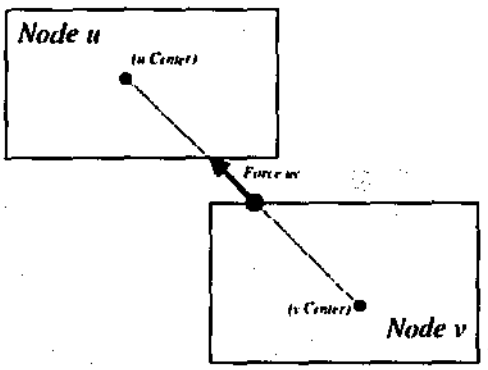


Figure 16 Force Scan Effect

The main idea of the force-scan algorithm is to choose a force between two nodes so that if the two nodes overlap, the force chosen will push one node away from the other. The force-scan algorithm can be extended to reduce wasted space between nodes by simply choosing a force that will push one node closer to the other so that there is no space left. This can be seen in figure 16 where a force *uv* is chosen to move *Node v* toward *Node u*. In effect, a *desired force* is chosen that will remove overlapping nodes and nodes separated by

too much distance. When dealing with more than two nodes, the force scan algorithm is broken down into two separate scans in the horizontal and vertical direction.

(Misue, Eades, Lai, and Sugiyama, 1995, p. 192) describe the Horizontal algorithm as below:

$i \leftarrow 1;$
while$(i < |V|)$
  Suppose that k is the largest integer such that
  $x_i = x_{i+1} = \ldots x_{ki}$
  $\delta \leftarrow \max_{i \le m \le k \le j \le |V|} \int_{m v_j}$;
  for $\leftarrow$ k + 1 to $|V|$ *do* $x_{v_j} := x_{v_j} + \delta$
  $i \leftarrow k + 1;$

Figure 17 Horizontal Scan Algorithm

The Vertical Scan algorithm is a similar algorithm, the main difference is that x and y are exchanged in the equation as can be seen below:

$i \leftarrow 1;$
while$(i < |V|)$
  Suppose that k is the largest integer such that
  $y_i = y_{i+1} = \ldots y_{ki}$
  $\delta \leftarrow \max_{i \le m \le k \le j \le |V|} \int_{m v_j}$;
  for $\leftarrow$ k + 1 to $|V|$ *do* $y_{v_j} := y_{v_j} + \delta$
  $i \leftarrow k + 1;$

Figure 18 Vertical Scan Algorithm

2.6. Approaches to Visualisation Systems.

There are many different approaches to visualisation systems. Some examples of visualisation systems include:

- PECAN (Reiss 1985) and the GARDEN (Reiss et al 1986).
- Pascal/HSD (Diaz-Herrera and Flude 1980) and FPL (Cunniff et al 1986) on the display of structured flowcharts.
- TRIP System. (Kamada 1989) The TRIP system is a general approach to visualisation.

Some research focus on the graph structure used to represent general diagram applications such as Higraph (Harel 1988) and Cigraphs (Lai 1993). The Cigraph model can be used to represent many diagram applications and it supports automatic layout. One of its examples is to represent flow charts (Lai and Eades 1996). Although they proposed a representation for flowchart components (Lai and Eades 1996) based on the Cigraph model, they didn't show further implementation to test it by translating a program source code to a flow chart. This research will utilise parts of the Cigraph model in the implementation of the prototype. For example, the Cigraph model includes the use of layout adjustment routines to improve the layout of the diagram. This project will also involve the application of layout adjustment routines such as the force-scan algorithm.

NOTE: The Cigraph model is a general structure used to represent many different diagrams. Although it can represent a flowchart, it may not be an efficient way for visualising flowchart applications.

2.7. Existing Software Projects/Products

There exist a number of software products available that can create flowcharts successfully. These products range in ability but can be classified into two distinct categories; those that act as drawing programs and those that create the drawings for you based on source code. These two categories are explained in the sections that follow:

2.7.1. Flowchart Drawing Programs

While these drawing programs cannot truly be classified as being Software Visualisation systems, they do however semi-automate the flowchart creation process. This is accomplished by giving the user a set of tools that help to create flowcharts. ABC Flowcharter is an example of such a program. ABC Flowcharter gives the user the ability to select from a range of drawing symbols to help create a flowchart. Also ABC Flowcharter allows the user to define diagram relationships. This means that when you shift a process symbol, all of the relationships displayed are also redirected toward this process symbol. Below is a list of some other flowchart drawing programs:

- Micrografx FlowCharter Version 7.0.
- Easy Flow by Haven Tree
- Flowchart Maker for the Mac
- Top Down Flowchart 5.0
- FlowView from Think and Do Software
- Mac Flow
- Win Flow
- Visio 3.0
- QA-Flow by Quality America

- Chartist by NovaGraph
- Flowchart Express V1.0 by Kaetron Software

The majority of software products that create flowcharts are based on these flowchart-drawing programs and it is evident that after using these products that it would be impractical to create flowcharts for small size software projects let alone large size projects. It is important that we develop a software tool that can automatically generate diagrams directly from the source code. The next section deals with existing products and the theory regarding the way they create flowchart diagrams.

2.7.2. *Automatic Flowcharting Programs.*

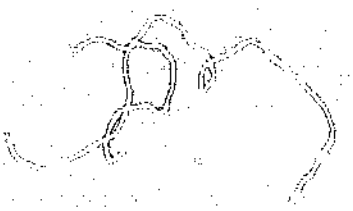
There are few systems available today that can successfully produce high quality flowcharts automatically from source code. Although current systems such as FlowView described by (Marlin and Jacobs, 1995) and the PECAN system described by (Reiss, 1985) can support the Multi-view of flowcharts, these systems cannot provide automatic drawing procedures for flowcharts. In these systems the graphical appearance of a flowchart component is pre-defined by the user or designer, that is, one needs to specify the geometric information in detail, such as shape, size and the location of every node image in a flowchart component. This is a tedious task! It is also important that we develop a good model that separates the logical part from the geometric part of a flowchart component.

Also within these system it is evident that the diagrams produced can have undesirable characteristics. For example, the article by (Reiss, 1985) shows screenshots taken from the PECAN system. These screen shots demonstrate that the PECAN system can possess 'undesirable diagram characteristics'. For instance (Reiss, 1985, p. 283, figure 7) is a screen shot of the PECAN system and clearly shows that it is possible to have lines overlapping throughout a flowchart diagram.

It is also evident that by examining these systems that other undesirable characteristics exist. E.g. Wasted space, Inconsistency, etc. It has been observed in the FlowView system that it is restricted by the assumption of flowchart symbols such as process and decision symbols are a fixed unit of 2x2. When you start to consider symbols as containing text then programming the layout process begins to become even more complicated. This is further complicated by the inclusion of other flowchart elements such as labels.

While there are a few systems capable of producing flowcharts, it is evident that we need to develop layout routines that do not deal with the geometric information such as size, location and shape. This is why this thesis deals with using abstract layout definitions that describe the general layout of a flowchart component, and to then apply a layout adjustment routine that re-arranges the diagram correctly to automatically give exact

geometric information. NOTE: The computer calculates exact geometric information automatically and not the user or programmer.



3. System Design

3.1. Introduction

The design of this prototype is broken down into a number of areas. These areas are demonstrated in figure 19.

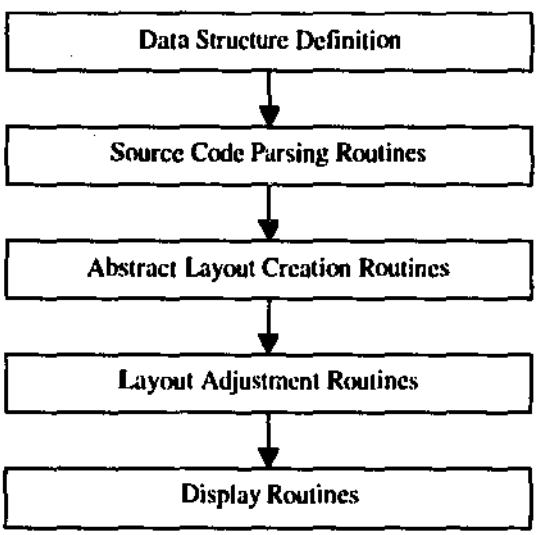


Figure 19 The Architecture.

The definition of the *data structure* provides a suitable storage structure to store diagram information.

Source code parsing routines read the source code information into the data structure. From there we can generate general diagram information through a series of *abstract layout creation routines*. The layout creation routines describe the general shape of the diagram without having to specify geometric information such as the exact shape, size and location. To further refine the diagram we need to apply a suitable layout adjustment routine. *Layout adjustment routines* shift diagram components around until the diagram can be displayed in a

clear and uncluttered manner. Once the diagram has its geometric information set, the diagram can then be displayed through a set of *display routines.*

3.2. Data Structure Definition

A data structure is required for storing diagram information. Two data structures will be used in combination to store this information. The tree structure stores node information while the list structure stores edge information.

3.2.1. General Node Definition

The general node definition given in the figure below demonstrates the hierarchical nature of a node. For instance one node can have a Sibling and a Child node. The sibling and child node can each have a child and sibling node, and so on.

```
struct Node
 {
    /* Data Definition */
    struct Node *Sibling, *Child;
};
```

Figure 20 General Node Definition.

To represent an entire flowchart diagram a root node is used. Nodes are used to represent individual flowchart components and flowchart symbols.

3.2.2. Edge Table Definition

The edge table definition is essentially a list of edges. An edge by the definition given in the figure below defines a relationship between two nodes *Node1 and Node2*. In this example, pointers are used to reference two different nodes. Style is a variable used to describe the arrow style used to connect the two nodes. This is the visual relationship displayed on the completed flowchart and can be represented by a single line, single arrowhead or a double arrowhead.

```
struct edge
 {
  struct Node *Node1;
  struct Node *Node2;
  int Style;
  struct edge *Next;
 };
typedef struct edge EDGE;
```

Figure 21 Edge Definition

3.2.3. Extended Node Definition

The node definition given previously is a general definition and can be extended to include data. We can further expand the definition to include an edge table for each node. The topology used in the design of the prototype is that each component of the diagram will be represented by one node. Each component of the diagram may consist of sub-components. Each sub-component may consist of sub-sub-components and so on as per the definition of a node. To represent the relationship between components, each node

will have and edge table that describes the relationship between the node's immediate child nodes.

```
struct Node
 {
  int ID;
  String Statement, Colour;
  int X, Y, Type, Expanded, Width1, Width2, Height1, Height2, DX, DY;
  struct Node *Sibling, *Child, *Parent;
  EDGE *Edge;
 };
typedef struct Node NODE;
```

Figure 22 Extended Node Definition

A number of routines are needed to operate on the main data structure. For example to create a node, the following code would be used:

```
NODE *
Create_Node (String Statement)
{
    NODE *Temporary_Node;
    Temporary_Node = malloc (sizeof (NODE));
    /* Initialize Data Values Here. */
    Temporary_Node->Sibling = Temporary_Node->Child =
    Temporary_Node->Parent = NULL;
    return (Temporary_Node);
}
```

Other routines used include :

- Finding a node based on a criteria such as X, Y Value.
- Deleting a node and its children for memory management.
- Setting data values for one given nodes.
- Adding child nodes.

3.3. Source Code Parsing

Source Code Parsing routines are used to generate and store the information about the source code. Each statement from a source code file is stored as one node in the data structure. Below in figure 23 and 24 is a simplified source code parsing routine used in this project to create the flowchart diagram and store values. Although the source code parsing routine is limited, it can be easily adapted to cater for a more precise syntax. It can also be easily adapted to cater for other programming languages.

Routine: *Parse Source Code*

Algorithm:

```
Last Statement = NONE
While not (EOF) do
Begin
    Read Next Statement (File, Statement)
    Process Statement (Statement, Last Statement)
    Last Statement = Statement
End
```

Figure 23 Sample Source Code Parsing Routine

Routine : *Process Statement (Statement, Last Statement)*

Algorithm:

```
Case Statement Type (Statement) of
```

```
'If Statement':
        Current Component = Create IF Component (Statement)
'While Statement':
        Current Component = Create WHILE Component (Statement)
'For Statement':
        Current Component = Create FOR Component (Statement)
'Repeat Statement':
        Current Component = Create REPEAT Component (Statement)
'Case Statement':
        Current Component = Create CASE Component (Statement)
Other Statement:
        Create General Component
End
Parent Component = Determine Parent Component (Graph, Last Statement)
  /* Determines the parent Component based on the last statement */
Add Component (Graph, Current Component, Parent Component).
  /* Adds the component to the Graph Structure */
```

Figure 24 Sample Statement Processing Routine

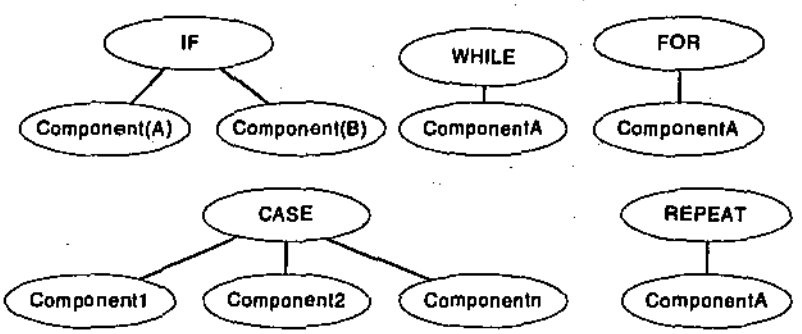


Figure 25 Sample Tree Structures for components

In the figure 25 we see a graphical representation for some common programming language statements. For example: An if statement can have two directions based on a

condition. In other words, an IF component can consist of two sub components. This graphical representation is used to give an example of how the *Source Code Parsing Routine* would process a piece of Pascal source code and store it internally.

Below is an example of how the *Source Code Parsing Routine* would process a piece of Pascal source-code and store it internally. Source code shown in Figure 26 would have a tree structure demonstrated in Figure 27.

```
If (A=B) then
    Case Value of
        1: Writeln ('1')
        2: Writeln ('2');
        3: Writeln ('3');
    End
Else
    Writeln ("A<>B");
```

Figure 26 Sample Program

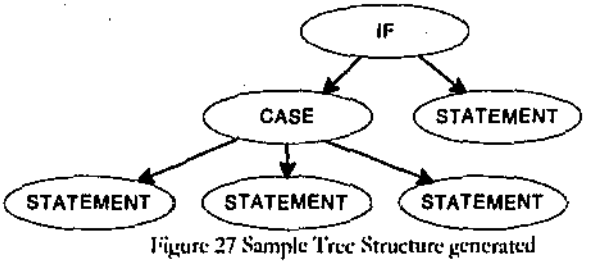


Figure 27 Sample Tree Structure generated

3.4. Automatic Layout

3.4.1. Undesired Diagram Attributes

One aspect of this thesis is the ability to automatically generate a diagram that has a 'good layout'. To define such a diagram we need to examine undesirable characteristics of a diagram and to then avoid them when creating a diagram. Below are some examples of undesired diagram attributes:

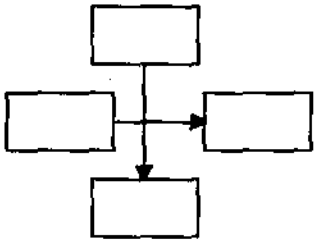


Figure 28 Undesired Diagram Attribute - Overlapping Edges.

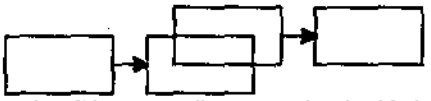


Figure 29 Undesired Diagram Attribute - Overlapping Nodes.

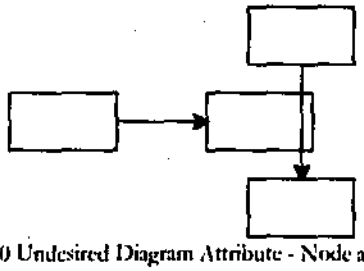


Figure 30 Undesired Diagram Attribute - Node and Edge Overlaps.

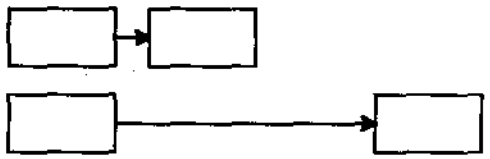


Figure 31 Undesired Diagram Attribute - Wasted Space.

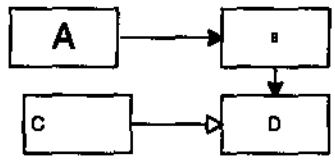


Figure 32 Undesired Diagram Attributes - Inconsistency.

The undesired diagram attributes can be avoided by implementing the following:

- Good Layout Routines
- An Effective Layout Adjustment Routine

Using "Good Layout Routines" can effectively avoid all of the undesired diagram characteristics. Using a layout adjustment routine such as the force scan algorithm can further refine the diagram by minimising node overlaps and reducing wasted space on a diagram. The force-scan algorithm is a particularly good algorithm, as it does not distort the shape of the diagram components. That is, after applying the force-scan algorithm the diagram has a similar shape to the original layout but at the same time eliminates some undesirable diagram attributes.

3.4.2. *Vertical & Horizontal Layout.*

Flowchart components can have only one entry point and only one exit point. However, the entry and exit points can be directed at any angle to the component as demonstrated in Figures 33, 34, and 35. This can create a large number of variations that a flowchart can have regarding the entry and exit points. Due to the large number of variations in entry and exit points there exists the problem of trying to connect components together and insert components into other components as illustrated by figure 36.

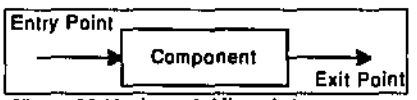


Figure 33 Horizontal Aligned Component

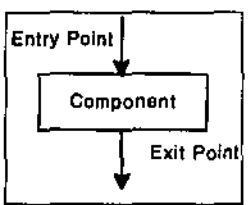


Figure 34 Vertical Aligned Component

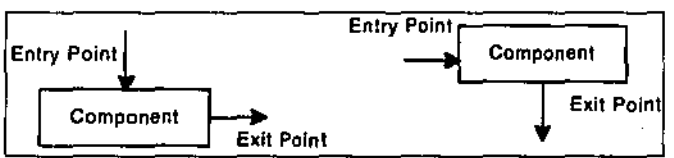


Figure 35 Horizontal-Vertical Aligned Component

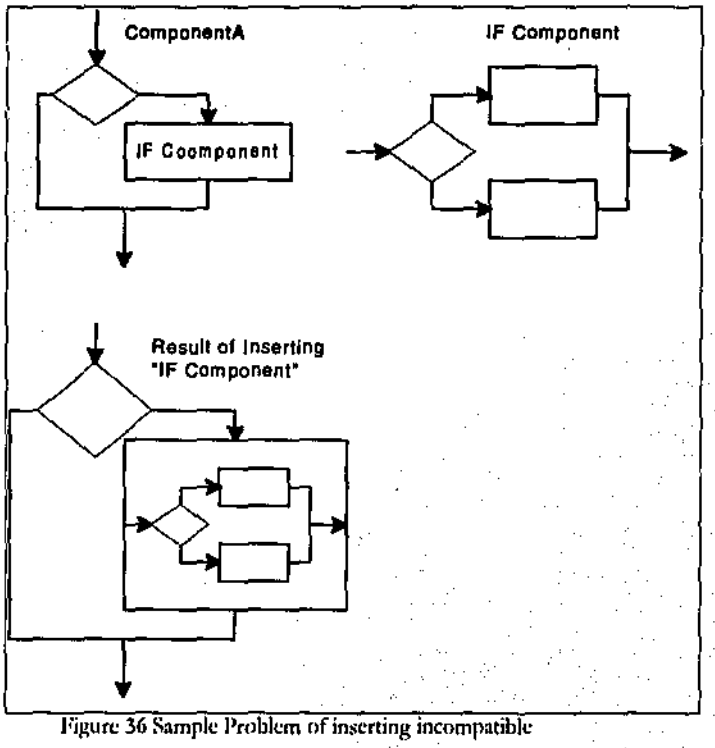


Figure 36 Sample Problem of inserting incompatible components.

As can be seen in figure 36, inserting an "IF Component" into "Component A" has a result of having the entry and exit points of the "IF Component" out of alignment with the parent component *A*. To join the arrows of the two components would result in a poor diagram.

To greatly simplify this problem, the selection of one style of component alignment needs to be chosen. The prototype will implement a vertical alignment for components in which the entry point for a component is from the top, and the exit point is from the bottom as displayed in the "vertically aligned component" figure 34. The layout for a range of components is described in the *Layout Creation Process.*

3.4.3. *Layout Creation Routines*

To this point, the only information stored so far about the diagram is information extracted through the source code parsing routine. A tree structure has been generated but no geometric information has yet been defined. Traditional layout algorithms can be applied to each component/node on this tree structure. By knowing the component type a corresponding layout creation routine can be applied. The component layout routines generate a series of extra nodes for each component. The extra nodes consist of drawing information. For example, the *IF Component Layout* process would generate an IF Symbol node, one label node, and two dummy nodes. This is illustrated in the figure below:

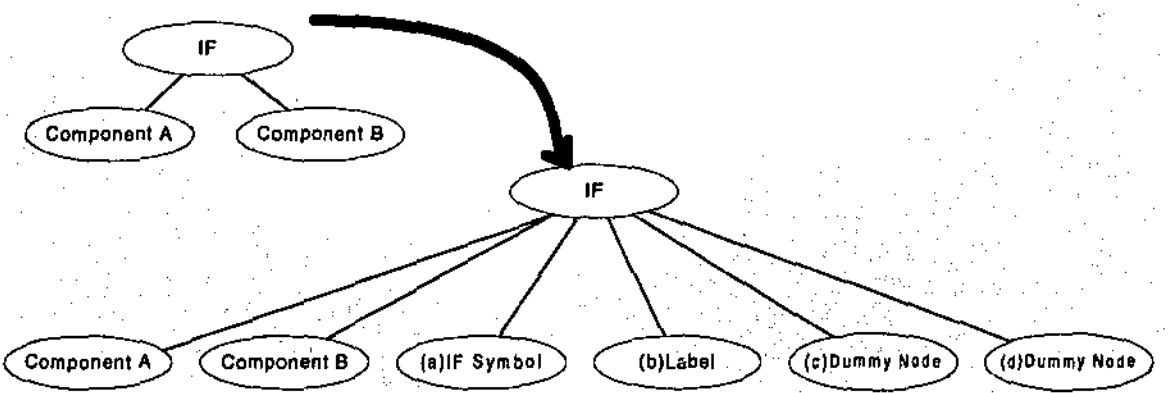


Figure 37 Result of Applying Layout Creation Routine

Layout Creation Routines generate the shape of individual flowchart components. This section deals with the following component layout descriptions :

- BLOCK component
- IF component
- WHILE component
- FOR component
- REPEAT component

- CASE component

Each component description will consist of:

- Programming language syntax.
- Component Shape Diagram.
- Component Layout Diagram.
- Edge Table.
- & Node Position Table.

By examining the syntax of a programming language statement, we can see how information can be extracted and then placed into a component. This can be observed by looking at the *Programming language syntax* and then the *Component Shape Diagram.* The *Component Layout Diagram* illustrates what a component is made up of. E.g. IF Symbol, dummy nodes, sub-components, case symbols, etc. Observing the *Component Layout Diagram* leads us to defining node positions in the *Node Position Table.* It is important to note that the positions specified in the 'Node Position Table' are only *relative positions*; that is, these positions do not describe exact positions for each node. These relative positions tell us which nodes are in the same row or same column and where nodes are placed in relation to other nodes. Exact geometric information is generated by the application of the force scan algorithm. The *Edge Table* defines the relationships between the nodes, this also can be observed from the *Component Layout Diagram.* This is demonstrated by the example figures 38 through to 55.

Note: In figure 45 there is an extra use node called a dummy node named Node f. The use of placing this dummy node allows the line through nodes a, b, i, h, and A to be redirected around component A. This redirection is possible, as the Force-Scan algorithm will push node f to the outside of component A. As node f is forced to the right, so to are nodes b and i, as they are in the same column.

3.4.3.1. Block Component Layout

Begin

Component$_0$

Component$_1$

...

Component $_n$

End

Figure 38 BLOCK Component Syntax

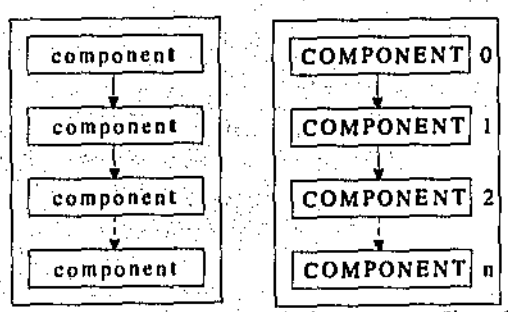


Figure 39 Block Component Shape & Layout

Node Positioning:

Component$_0$ = (x, y), *Component$_1$ = (x, y-1),* ...

Component $_n$ = (x, y - n)

Edge Table:

(Component$_0$→Component$_1$), (Component$_1$→Component$_2$) ...

(Component$_{n-1}$→Component $_n$)

Figure 40 BLOCK Component: Node Position, & Edge Table.

3.4.3.2. IF Component Layout

If <condition> then

Component

Else

Component

Figure 41 IF Component Syntax

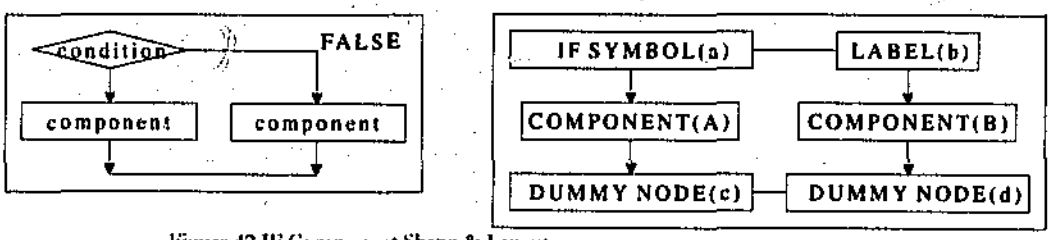


Figure 42 IF Component Shape & Layout

Node Positioning:

$A = (x, y-1)$ $B = (x+1, y-1)$

$a = (x, y)$ $b = (x+1, y)$

$c = (x, y-2)$ $d = (x+1, y-2)$

Edge Table:

(a-b), (c-d), (a→A), (A→c), (b→B), (B→d)

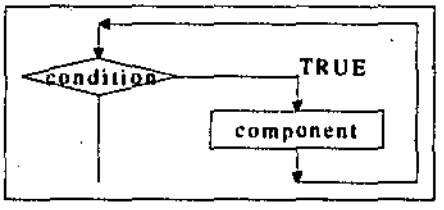
Figure 43 IF Component: Node Position, & Edge Table.

3.4.3.3. *WHILE Component Layout*

While <condition> do

Component

Figure 44 WHILE Component Syntax



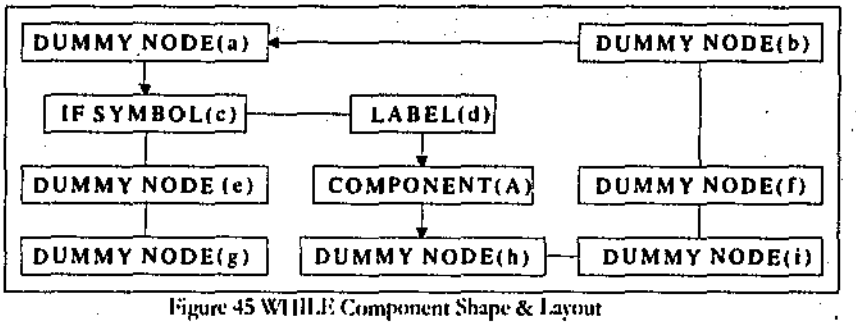


Figure 45 WHILE Component Shape & Layout

Node Positioning:

$A = (x+1, y-2)$ $a = (x, y)$

$b = (x+2, y)$ $c = (x, y-1)$

$d = (x+1, y-1)$ $e = (x, y-2)$

$f = (x+2, y-2)$ $g = (x, y-3)$

$h = (x+1, y-3)$ $i = (x+2, y-3)$

Edge Table:

(c-e), (e-g), (h-i), (i-f), (f-b), (c-d), (a→c), (d→A), (A→h), (b→a)

Figure 46 WHILE Component: Node Position, & Edge Table.

3.4.3.4. FOR Component Layout

For Value := Initial Value to Final Value do

Component

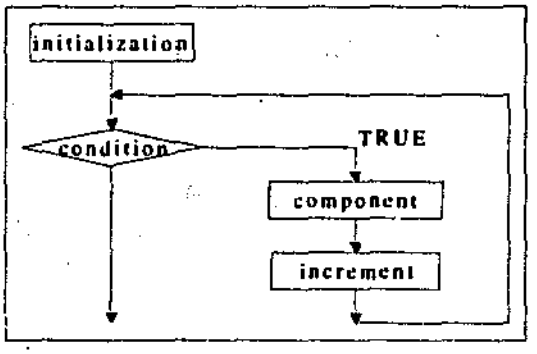
Where :

initialization = "Value := Initial Value"

condition = "Value<=Final Value"

increment = "Value := Value + 1"

Figure 47 For Component Syntax



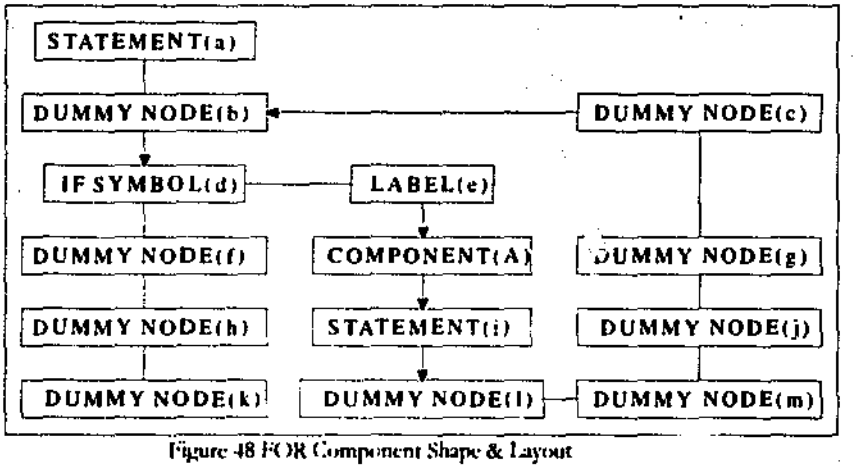


Figure 48 FOR Component Shape & Layout

Node Positioning:

A = (x+1, y-3)	*a = (x, y)*
b = (x, y-1)	*c =(x+2, y-1)*
d = (x, y-2)	*e = (x+1, y-2)*
f = (x, y-3)	*g = (x+2, y-3)*
h = (x, y-4)	*i = (x+1, y-4)*
j = (x+2, y-4)	*k = (x, y-5)*
l = (x+1, y-5)	*m = (x+2, y-5)*

Edge Table:

(a-b), (d-f), (f-h), (h-k), (d-e), (l-m), (m-j), (j-g), (g-c),

(b→d), (e→A), (A→i), (i→l), (c→b)

Figure 49 FOR Component: Node Position, & Edges Table

3.4.3.5. REPEAT Component Layout

Repeat

Component

Until <condition>

Figure 50 REPEAT Component Syntax

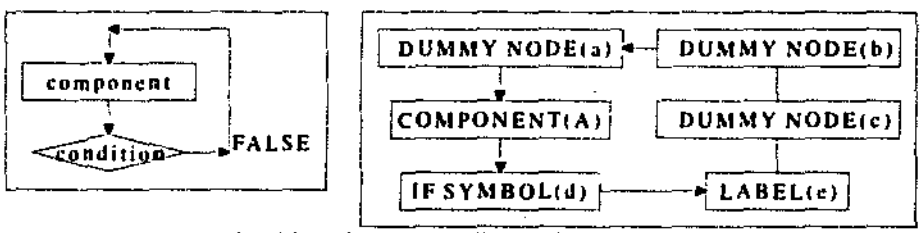


Figure 51 REPEAT Component Shape & Layout

Node Positioning:

A = (x, y-1)

a = (x, y)

b = (x+1, y)

c =(x+1, y-1)

d = (x, y-2)

e = (x+1, y-2)

Edge Table:

(e-c), (c-b), (a→A), (A→d), (b→a) , (d→e)

Figure 52 REPEAT Component Node Position, & Edge Table

3.4.3.6. CASE Component Layout

Case of

1: component

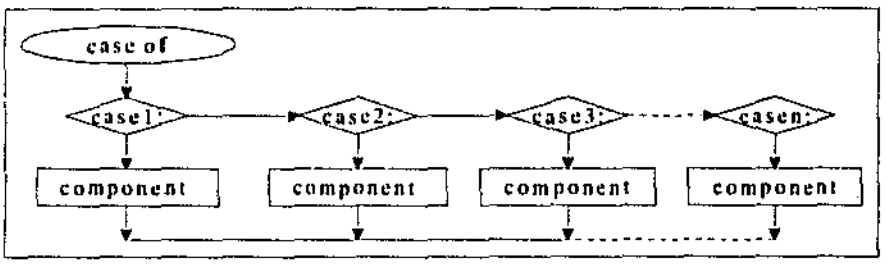
2: component

3: component

n: component

end

Figure 53 CASE Component Syntax



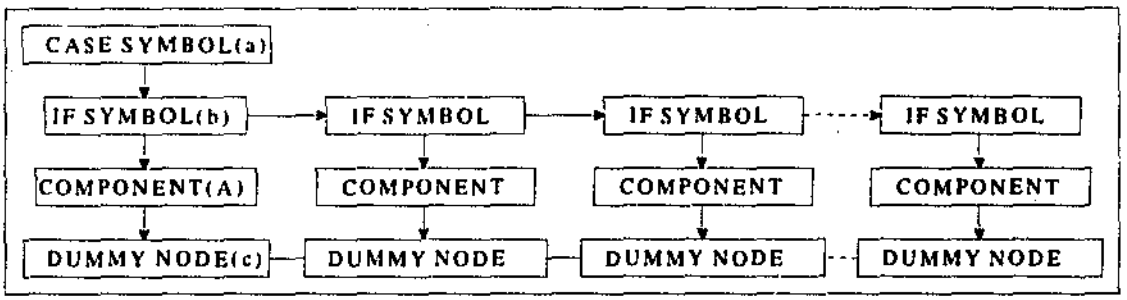


Figure 54 CASE Component Shape & Layout

Node Positioning:

$A_0 = (x, y-2)$, $A_1 = (x+1, y-2)$, $A_2 = (x+2, y-2) \ldots A_n = (x+n, y-2)$

$a = (x, y)$

$b_0 = (x, y-1)$, $b_1 = (x+1, y-1)$, $b_2 = (x+2, y-1) \ldots b_n = (x+n, y-1)$

$c_0 = (x, y-3)$, $c_1 = (x+1, y-3)$, $c_2 = (x+2, y-3) \ldots c_n = (x+n, y-3)$

Edge Table:

$\{(a \rightarrow b)$,

$((b_0 \rightarrow A_0) \ldots (b_n \rightarrow A_n))$,

$((A_0 \rightarrow c_0) \ldots (A_n \rightarrow c_n))$,

$((b_0 \rightarrow b_1), (b1 \rightarrow b_2), \ldots (b_{n-1} \rightarrow b_n))$

$((c_0 \rightarrow c_1), (c1 \rightarrow c2), \ldots (c_{n-1} \rightarrow c_n))\}$

Figure 55 CASE Component: Node Position, & Edge Table

3.5. Layout Adjustment Routine - Force Scan Algorithm

To this point we have dealt with giving the flowchart 2-dimension information. Relative coordinates have been specified but we cannot yet display the diagram, as the components have no real size or location yet defined. To obtain exact coordinates, the force scan algorithm is applied to each branch of the tree structure until it reaches the root node. However, before the force-scan algorithm can be applied we need to first set node sizes for the set of nodes that have a definable size. The set of nodes includes IF Symbols, CASE Symbols, General Statements and other similar symbols.

For example to find the width of a general statement as shown below we would call a function *TEXT_WIDTH* that would determine the width of a specified text string. The width returned would be in units relative to the output device such as Pixels, and the width would also be based on the font type. Once the text width is calculated, the node size can then calculated by adding a node boundary value. A similar calculation is used to determine the statement's height, except that the function *TEXT_HEIGHT* does not require a text string as a parameter:

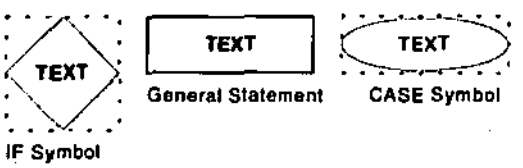


Figure 56 IF Symbol, General Statement, & CASE Symbol

$$Eg.\ Width = TEXT\_WIDTH\ ("TEXT") + 2 \times NODE\_BOUNDARY\_WIDTH$$

$$Height = TEXT\_HEIGHT + 2 \times NODE\_BOUNDARY\_HEIGHT$$

Similarly the sizes for the IF Symbol and CASE Symbol can be determined. This is simplified by assuming that the two symbols are surrounded by an imaginary boundary as displayed by the rectangles surrounding each symbol in figure 56. This boundary is assumed to be rectangular in shape as it simplifies the programming process. We could

extend this process later to include other shapes for boundaries to further reduce space wasted inside diagrams.

After generating the node sizes, we can then apply the force scan algorithm to each node. This process is done by first processing the farthest descendants of the root node and working the way up to the root node. As each node of the tree structure is processed a size for that node/component is set.

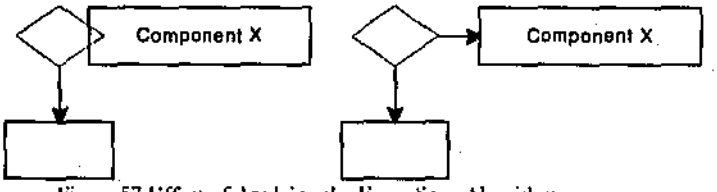


Figure 57 Effect of Applying the Force Scan Algorithm

In figure 57 we can see the effect of the force scan algorithm. *Component X* is pushed away from the IF symbol thereby removing the node/component overlap.

3.6. Display Routines

To display the flowchart a set of related drawing routines are required. Below is a sample of some drawing routines that are required for implementation:

- Draw_Flowchart_Component
- Draw_Edges
- Draw_IF_Symbol
- Draw_CASE_Symbol
- Draw_General_Statement.
- Draw_Rectangle
- Draw_Ellipse
- Draw_Line
- Draw_Arrow_Symbol
- Draw_Text

3.7. User Interface Design

Effectively all that is required for the user interface is a display area and a method for scrolling around the generated diagram. The Prototype will consists of a display area for the diagram and a set of menu options with a scrollable display area. The menu options are described in the figure below:

File

Open	*{Opens a dialog box for selecting source code files}*
Save	*{Opens a save dialog box that allows the diagram to be saved.}*
Exit	*{Exits the application}*

Diagram

Settings	*{Opens a dialog box with the diagram settings displayed}*
Collapse All	*{Sets the diagram to a collapsed state}*
Expand All	*{Expands every component of the diagram}*
Display Grid	*{Displays a grid used to determine diagram spacing.}*
Display Boundaries	*{Displays boundaries around every flowchart component}*
Display Edges	*{Toggles the visibility of the diagram's edges}*
Display Components	*{Toggles the visibility of the diagram Components}*

Zoom

In	*{Zooms in by a set factor}*
Out	*{Zooms out by a set factory}*
Default Zoom	*{Returns to the default zoom amount}*

Help

About	*{Display information about the program version}*
Program	*{Displays help information about the program}*

Figure 58 User Interface

The user interface will allow the user to browse a flowchart diagram. The user will be able to collapse all of the components down and selectively expand the desired components. This is extremely useful when dealing with a large diagram as only the selected information is displayed. The user interface will read a configuration file that will change diagram attributes and the visual appearance of the generated diagrams. A sample of some diagram attributes include:

- Node spacing distance
- Display Colours
- Arrow Style
- Diagram boundary area
- Visibility of diagram elements.

4. Implementation

The implementation for this prototype spans quite a few pages and as a result the majority of the source-code for this prototype is contained in the appendix section of this thesis.

4.1. Development Environment

The development environment chosen for this project is a Unix based operating system using a C compiler with Motif extensions for the user interface development. The windowing system used for this project is the X-Window system. Due to the multi-tasking abilities of the operating system it is possible to display multiple views of a program through multiple windows.

The computer system used to implement this project has the following specifications:

- Pentium 166, 32 M RAM., 2 MB Video card. 1 GB IDE HDD.

4.2. Coordinate System.

The coordinate system used for displaying the flowchart to the output device is a different system to that used in the component layout process and as a result the y-axis information needs to be inverted. Figure 59 demonstrates the different coordinate system.

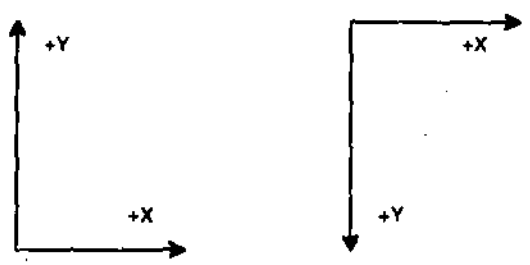


Figure 59 Coordinate System Used for Layout Generation; & Coordinate system used for displaying the flowchart.

4.3. Configuration File

A configuration file allows the prototype to generate different diagram appearances. Using a configuration file is a simple way of initialising program variables without having to re-compile the program. Diagram settings can be changed on the fly as required. This is useful as it allows the user of the program to experiment with the diagram layout without requiring any knowledge of the programs coding. Below is a sample configuration file:

NODE_SEPERATION_X = 8
NODE_SEPERATION_Y = 8
BORDER_DISTANCE_X = 10
BORDER_DISTANCE_Y = 10
NODE_BOUNDARY_WIDTH = 20
NODE_BOUNDARY_HEIGHT = 1
ARROW_SIZE = 9
GRID_SIZE = 5
DISPLAY_DEPTH = 0
BACKGROUND_COLOUR = white
FOREGROUND_COLOUR = black
IF_SYMBOL_COLOUR = black
LABEL_TEXT_COLOUR = blue
PROCEDURE_CALL_COLOUR = green
GENERAL_STATEMENT_COLOUR = blue
EDGE_COLOUR = blue
GRID_COLOUR = lightgrey
BOUNDARY_COLOUR = lightgrey
DISPLAY_GRID = 0
DISPLAY_COMPONENTS = 1
DISPLAY_EDGES = 1
DISPLAY_BOUNDARIES = 0

Figure 60 Sample Configuration File

4.4. Code Samples

4.4.1. Layout Creation Routines

The layout creation process described in the previous chapter can be compared with he actual implementation in figure 61. *See appendix section for further source code.*

4.4.2. Block Component Layout

```
void
Create_BLOCK_Component_Layout (NODE * Current_Node)
{
    NODE *Cursor;
    int X, Y;
    int Index;
    X = Current_Node->X;  Y = Current_Node->Y;
    Index = 0;
    Cursor = Current_Node->Child;
    while (Cursor != NULL)
        {
        Cursor->X = X;    Cursor->Y = Y - Index;
        Index ++;
        Cursor = Cursor->Sibling;
        }
    Cursor = Current_Node->Child;
    while (Cursor->Sibling != NULL)
        {
        Add_Edge_To_Node (Current_Node, Cursor, Cursor->Sibling,
            SINGLE_ARROWHEAD1);
        Cursor = Cursor->Sibling;
        }
}
```

Figure 61 Code Sample: Block Component Layout

```
void Adjust_Node_Layout (NODE * Current_Node)
{
    NODE_LIST *Temporary_Node_List;
    NODE *Reference_Node;
    if (Current_Node != NULL)
        {
        if (Current_Node->Expanded == TRUE)
            Adjust_Node_Layout (Current_Node->Child);
        Adjust_Node_Layout (Current_Node->Sibling);
        if (Current_Node->Child != NULL)
            {
            Reference_Node = Create_Node ("Reference Point",
                KEYWORD_DUMMY_NODE);
            Reference_Node->X = Current_Node->X;
            Reference_Node->Y = Current_Node->Y;
            Temporary_Node_List = Create_Node_List (Current_Node,
                    Reference_Node);
            Sort_Node_List_By_X_Value (Temporary_Node_List);
            Set_Accumulative_Force_X (Temporary_Node_List);
            Sort_Node_List_By_Y_Value (Temporary_Node_List);
            Set_Accumulative_Force_Y (Temporary_Node_List);
            Shift_Nodes_In_List (Temporary_Node_List,
                    Reference_Node->DX, Reference_Node->DY);
            Destroy_Node_List (Temporary_Node_List);
            }
        Set_Node_Size (Current_Node);
}}
```

Figure 62 Code Sample: Force Scan Algorithm

The Layout Adjustment Routine operates on every node of the diagram individually. It is a recursive routine that processes every child and sibling node as can be seen by the following code stub:

```
void Adjust_Node_Layout (NODE * Current_Node)
{
    ...
        Adjust_Node_Layout (Current_Node->Child);
    Adjust_Node_Layout (Current_Node->Sibling);
  /* Node Adjustment Layout Code Here */
```

Figure 63 Sample Code Stub illustrates recursion.

The layout adjustment routine uses a reference point for the layout process. The use of the reference point helps to restore the current nodes original position after the layout process has been applied to its children nodes.

The layout process involves creating a list of the current nodes child nodes. The list is sorted by X value and the individual forces in the X-direction are set by the function *Set_Accumulative_Force_X*. A similar process is applied in the Y-direction and then the child nodes are shifted according to the predetermined forces calculated by the functions *Set_Accumulative_Force_X* and *Set_Accumulative_Force_Y*.

The *Set_Accumulative_Force_X* function breaks the list into a number blocks grouped by X value. Each block of nodes is compared with its immediate neighbouring block and a force is determined. For a set of *n* blocks, the following would apply:

Block(0), Block(1), Block(2)...Block(n)

Where the forces are calculated between:

Block(0) & Block(1),

Block(1) & Block(2)

...

Block(n-1) & Block(n)

Noting:

Block 0 remains where it is.

Block 1 would be moved first according to its position in relation to block 0.

Block n would be moved last according to its position in relation to block n-1.

4.5. Static And Dynamic Program Execution Representation

The definition of program visualisation in chapter 2 describes static and dynamic representation of software. Dynamic representation involves showing an animation of the running program, whereas the static representation is simply a snapshot of the program at a certain point in time. This implementation of this project would be classified as using static representation. To extend this project to a dynamic representation is not conceptually a difficult process, however it does involve a significant amount of knowledge about the software compiler/interpreter specifications. That is, as a program executes a trace must be kept of where the program is currently executing so that it can be graphically represented.

4.6. Zoom Factor & Font Size.

Modifying the zoom factor is not a difficult process. One of the simplest ways to increase the zoom factor is to increase the font size. By increasing the font size, you also increase the flowchart symbol sizes and as a result the entire diagram is increased by some size. The

effect of changing the font size can be seen by the sample figures. Note: A proportional scale factor would normally be applied to the component separation and the width of the lines would also be changed by this factor.

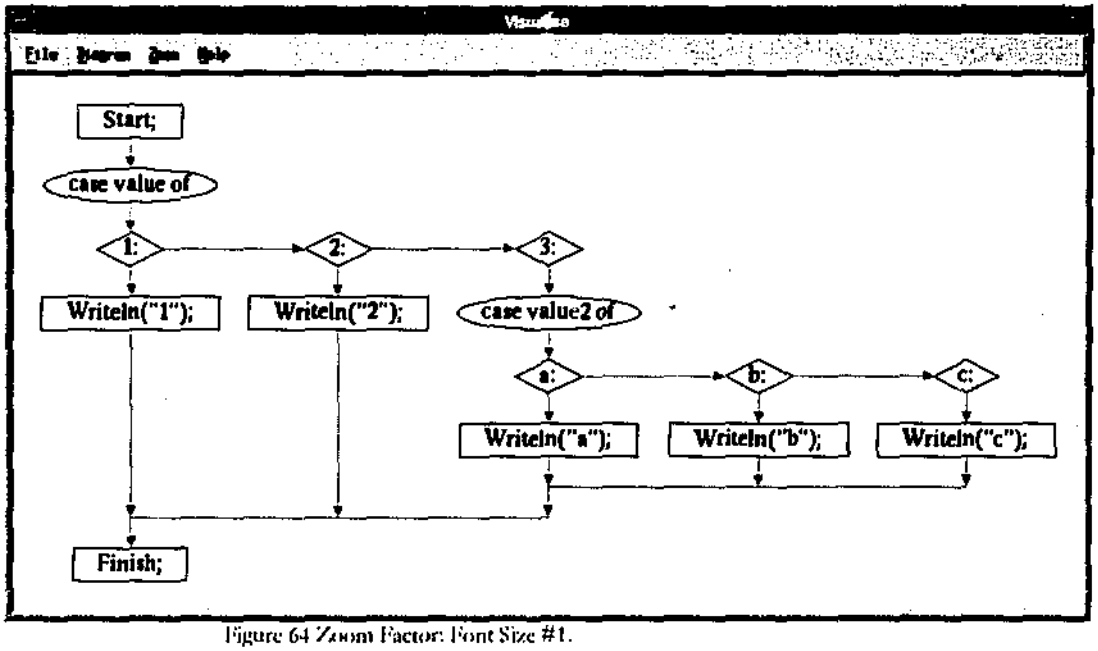


Figure 64 Zoom Factor: Font Size #1.

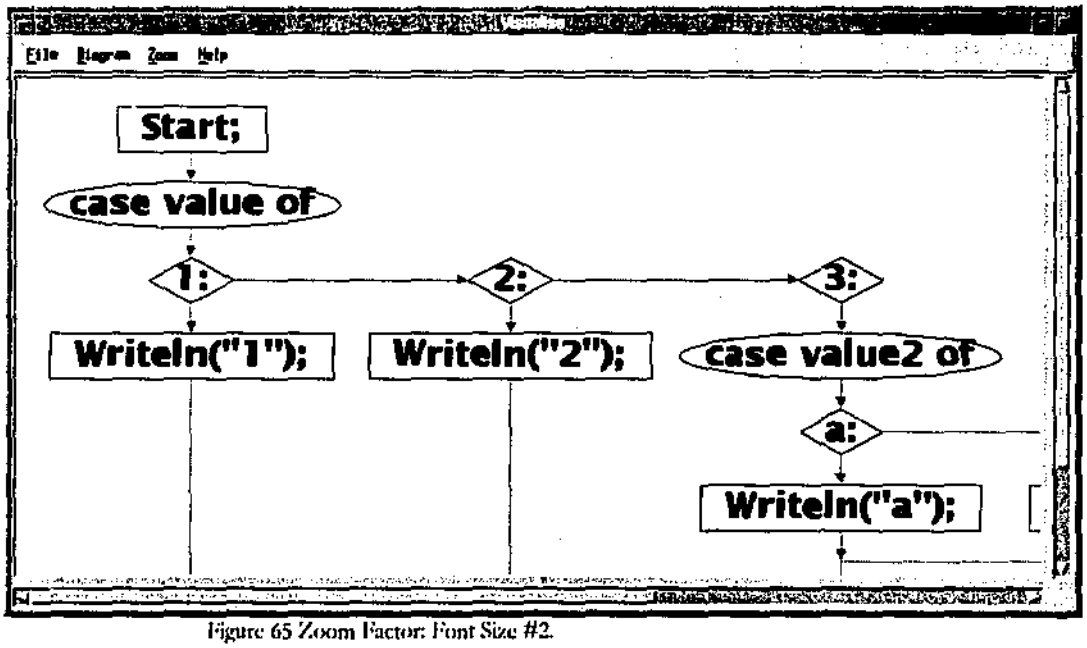


Figure 65 Zoom Factor: Font Size #2.

4.7. Output to X-FIG format

The process of creating an X-Figure file format was examined for this thesis so that a snapshot of the generated diagrams could be saved. X-FIG files could then imported into this document. The advantage of being able to generate this file format is that X-FIG is a popular drawing package. Under LINUX, the source code for XFIG is freely available over the Internet. Another advantage of converting to this format is that it is possible to produce high quality printouts. These files can also be easily ported to many different Unix platforms. XFIG also allows free editing of the diagrams. That is, objects can be moved around and text can be edited easily.

5. Results

This chapter focuses on the output from the prototype. All diagrams for this chapter were created using the software visualisation prototype tool. The process involved in creating the diagrams for this chapter are outlined in the steps below:

1. Create a stub of code and save as a text file.
2. Execute command line statement to invoke the program. e.g. Visualise *filename.pas*
3. Program generates all flowchart information transparent to the user of the program.
4. Using the program's menu system a snapshot of the diagrams can be saved as an X-FIG file and then using the X-FIG program the file can be exported into other formats such as postscript.

In step 3 the program generates the flowchart transparently to the user of the prototype. All that is seen by the user after pressing the <enter> key is the flowchart displayed on the screen. Step 3 is further expanded below:

3.1 Parse source code file and store information about the source code.

3.2 Add extra diagram information such as diagram symbols and edges.

3.3 Apply abstract layout information to each node in the diagram according to the type of node. For example, if the node is an if-statement then apply the abstract layout routine for the if-statement. This process is applied to every node within the data structure.

3.4 Assign node size information to the set of nodes that have a definable size such as symbols.

3.5 Apply Force-Scan algorithm to add geometric information and reduce unwanted diagram characteristics. The Force-Scan algorithm is applied to every node of the diagram starting at the farthest descendant of the root node and is then applied in the direction towards the root node.

3.6 Use display routines to display the diagram.

Important points to note about the results:

- All diagrams for this chapter were created using the software visualisation prototype tool.
- All diagrams generated were created faster than typing the command line statement that runs the prototype. To draw these diagrams manually would be significantly slower and would not produce the same quality diagrams.
- The program demonstrated the ability of component to have sub-components. This is demonstrated by generating flowcharts for source code that has nested statements.
- All flowcharts generated have the desired visual appearance.
- High quality diagrams were generated.

5.1. Sample Program Output

This section contains stubs of code with the corresponding flowchart generated by the prototype. For a full explanation of the translation process and a worked example see *appendix S*. The process described in this appendix can be applied to each of the sample flowcharts generated in this section.

5.1.1. BLOCK Component

```
Begin
    Writeln ("A");
    Writeln ("B");
    Writeln ("C");
    Writeln ("D");
End
```

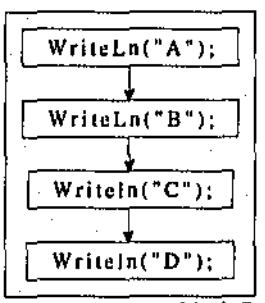


Figure 66 Program Output: Block Component

Start;

If c then

Writeln ("c = true");

Else

Writeln ("c = false");

Finish;

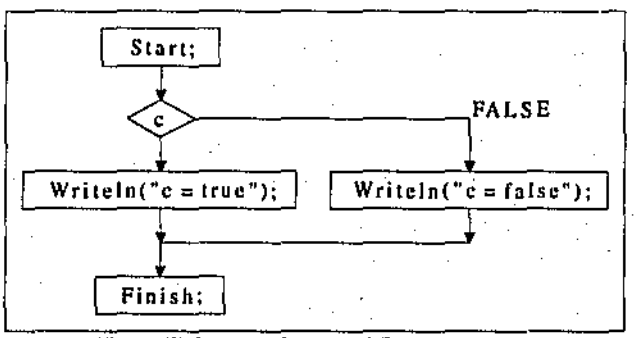


Figure 67 Program Output: If Component

5.1.3. WHILE Component

Start;

While c=0 do

Writeln ("Enter value for C>");

ReadLn(C);

Finish;

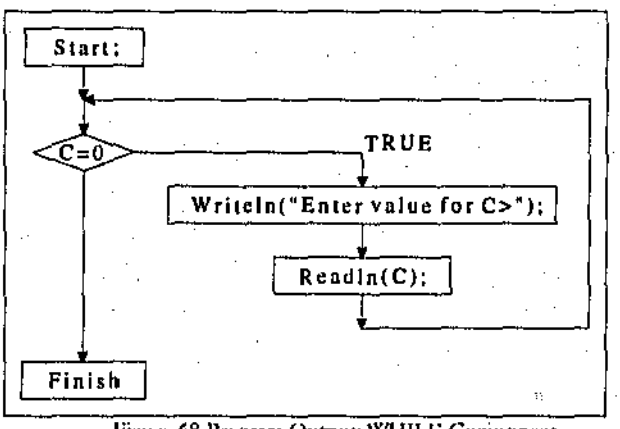


Figure 68 Program Output: WHILE Component

```
Start;
Repeat
    Writeln ('Enter value for C>');
    ReadLn ('C');
Until c=0
Finish;
```

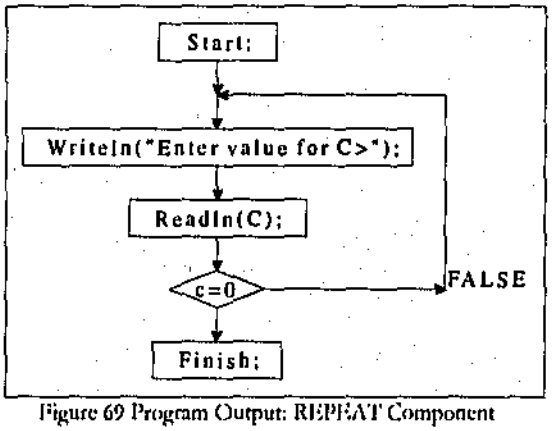


Figure 69 Program Output: REPEAT Component

5.1.5. FOR Component

Start;

For value := 1 to 100 do

Writeln (value);

Finish;

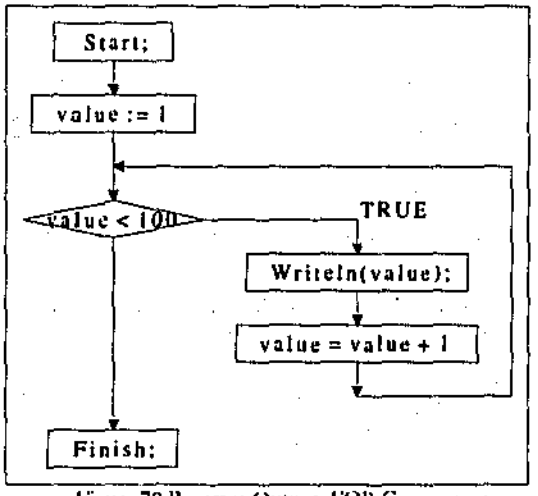


Figure 70 Program Output: FOR Component

Start;

Case value of

1: Writeln ("1");

2: Writeln ("2");

3: Writeln ("3");

4: Writeln ("4");

End

Finish;

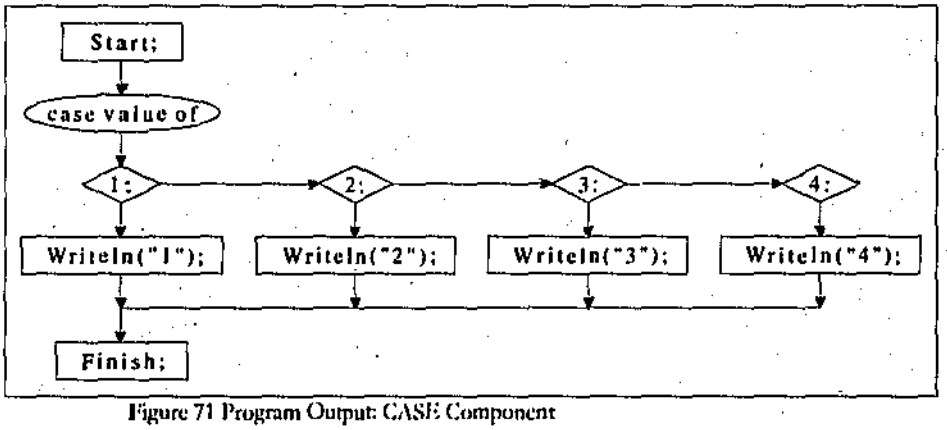


Figure 71 Program Output: CASE Component

5.2. Nested Statements

5.2.1. Nested If Component

Start;

If c1 then

Begin

Writeln ("c = true")

If C2 then

Writeln ("c2 = true");

Else

Writeln ("c2 = false");

End

Else

Writeln ("c = false");

Finish;

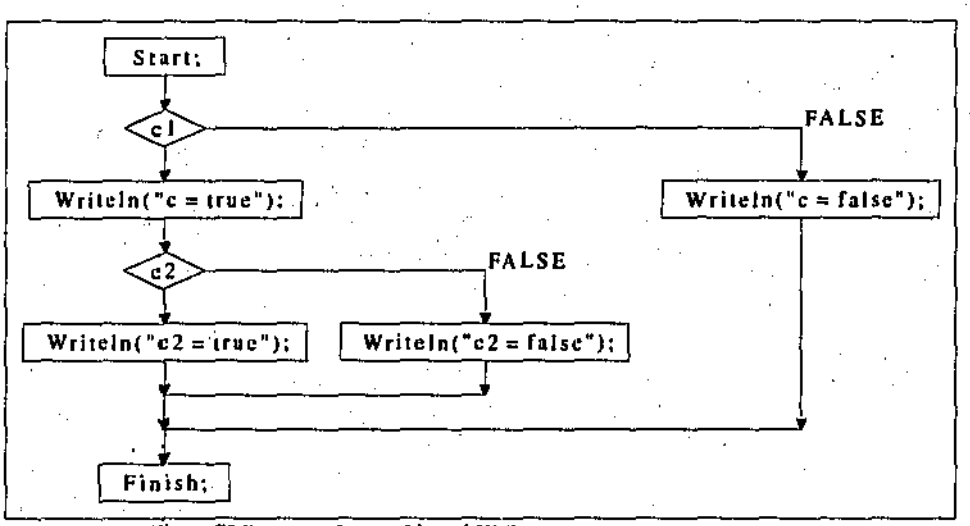


Figure 72 Program Output: Nested IF Component

5.2.2. Nested WHILE Component

```
Start;
While c1=0 do
begin
    Writeln ("Enter value for C1>");
    ReadLn (C1);
    While c2=0 do
        Begin
            Writeln ("Enter value for C2>");
            ReadLn(C2);
        End;
End;
Finish;
```

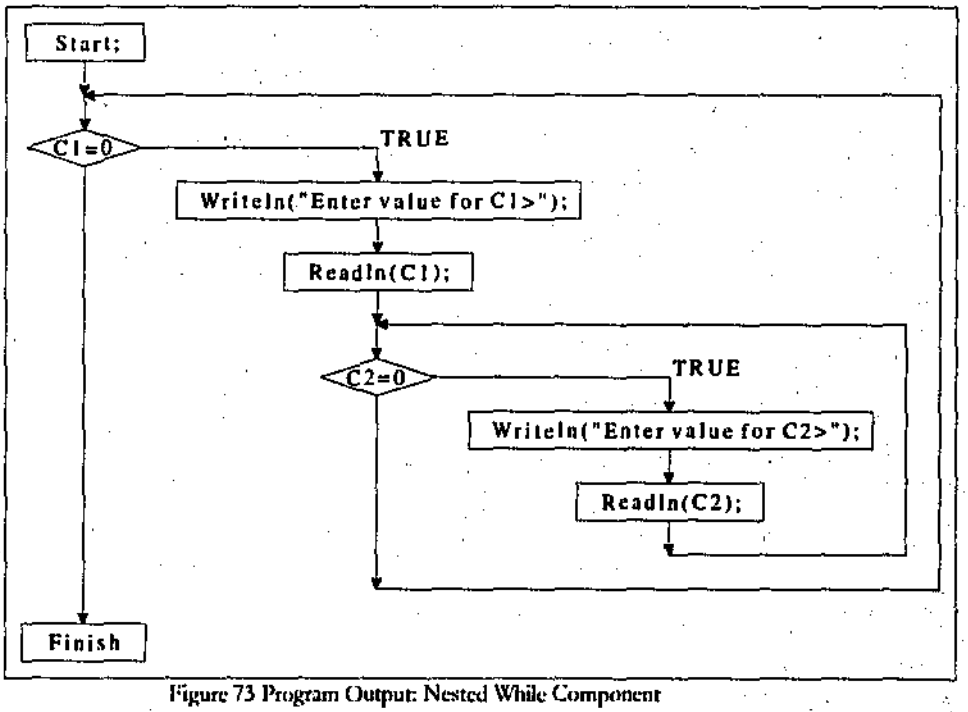


Figure 73 Program Output: Nested While Component

5.2.3. Nested REPEAT Component

```
Start;
c1 := -1;
c2 := -1;
Repeat
    Writeln ("Enter value for c2>");
    ReadLn (c2);
    Repeat
        Writeln ("Enter value for c1>");
        ReadLn (c1);
    Until c1=0;
Until c2= 0;
Finish;
```

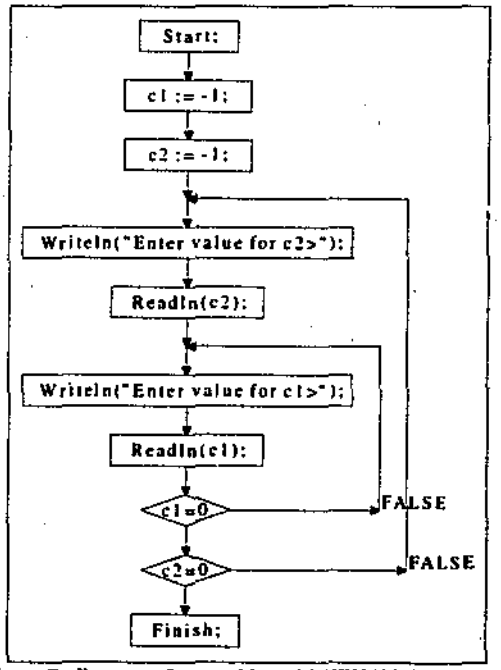


Figure 74 Program Output: Nested REPEAT Component

5.2.4. *Nested FOR Component*

```
Start;
For value :=1 to 100
    Begin
        Writeln (value);
        For value2 := 1 to 20 do
            Writeln (value2);
Finish;
```

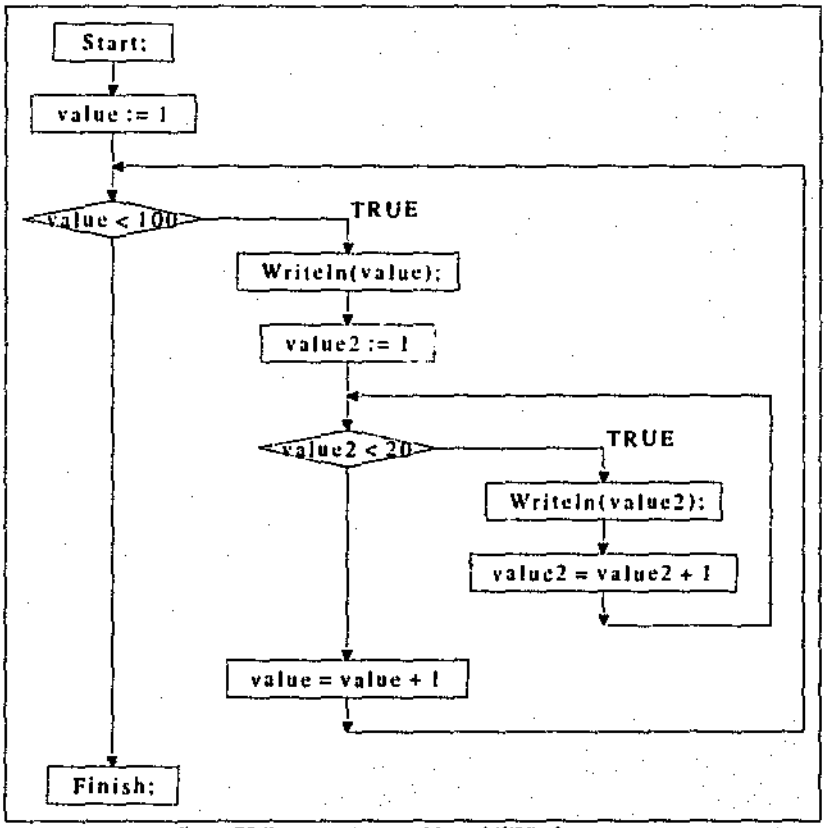


Figure 75 Program Output: Nested FOR Component

5.2.5. Nested CASE Component

```
Start;
Case value of
    1: Writeln ("1");
    2: Writeln ("2");
    3:    case value2 of
                a: Writeln ("1");
                b: Writeln ("2");
                c: case value2 of
          End
End
Finish;
```

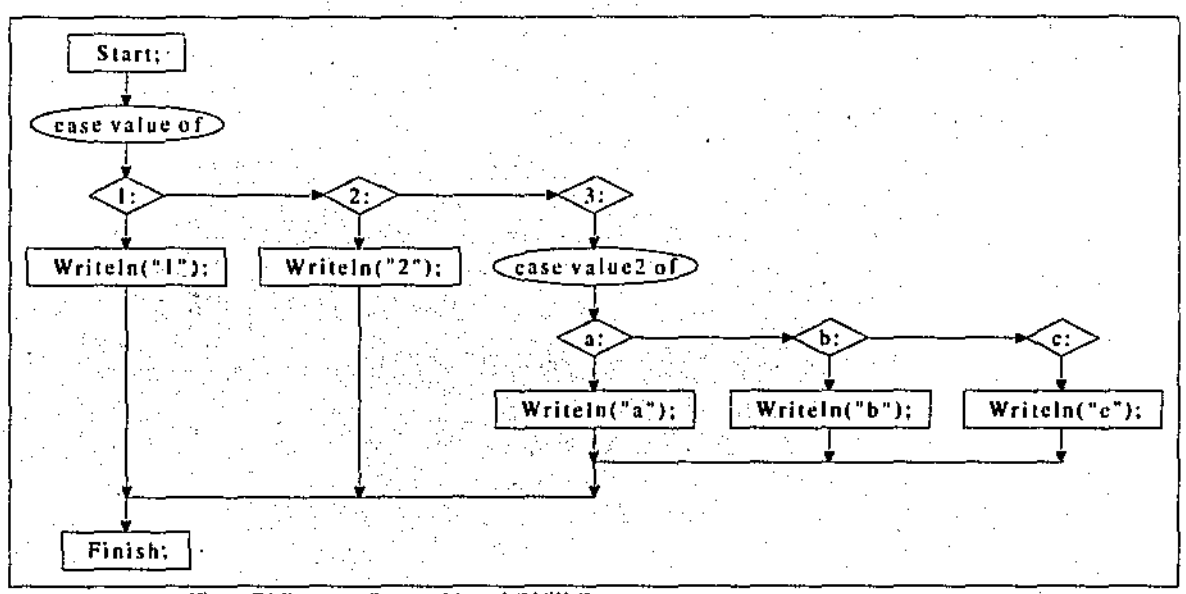


Figure 76 Program Output: Nested CASE Component

5.3. Effect of Changing Diagram Attributes

A diagram can consist of a number of attributes. Changing an attribute of a diagram can affect the overall appearance of the diagram. For example, font size, font type, arrow size, arrow type, drawing colours, and line width can each affect the final diagrams appearance.

The figures included, demonstrate the effect of changing diagram attributes:

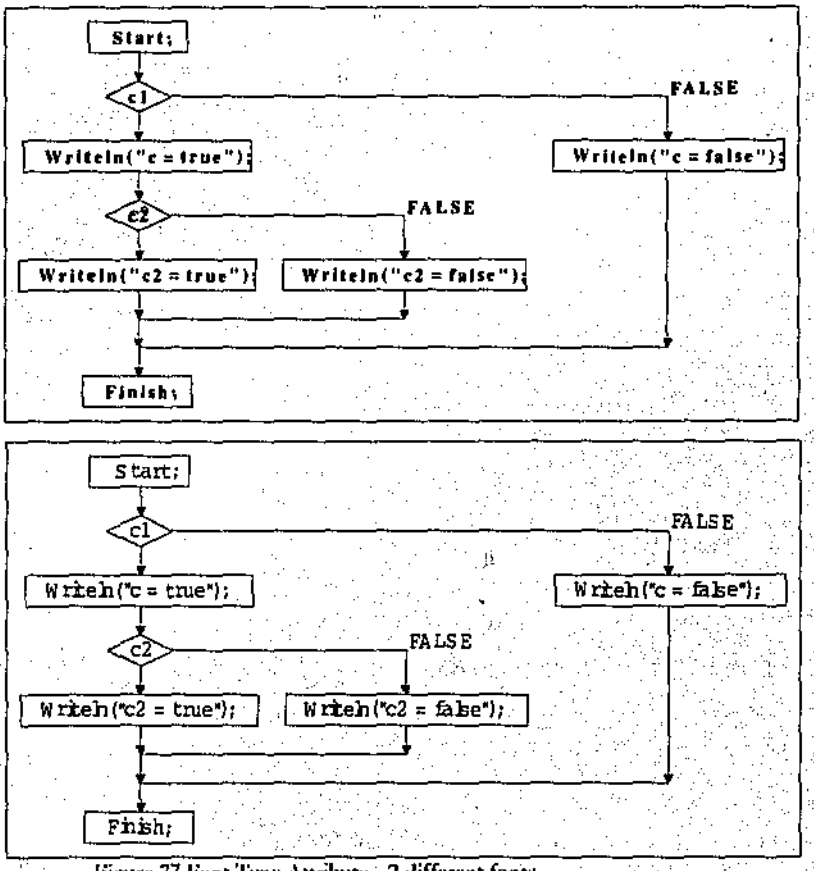


Figure 77 Font Type Attribute - 2 different fonts.

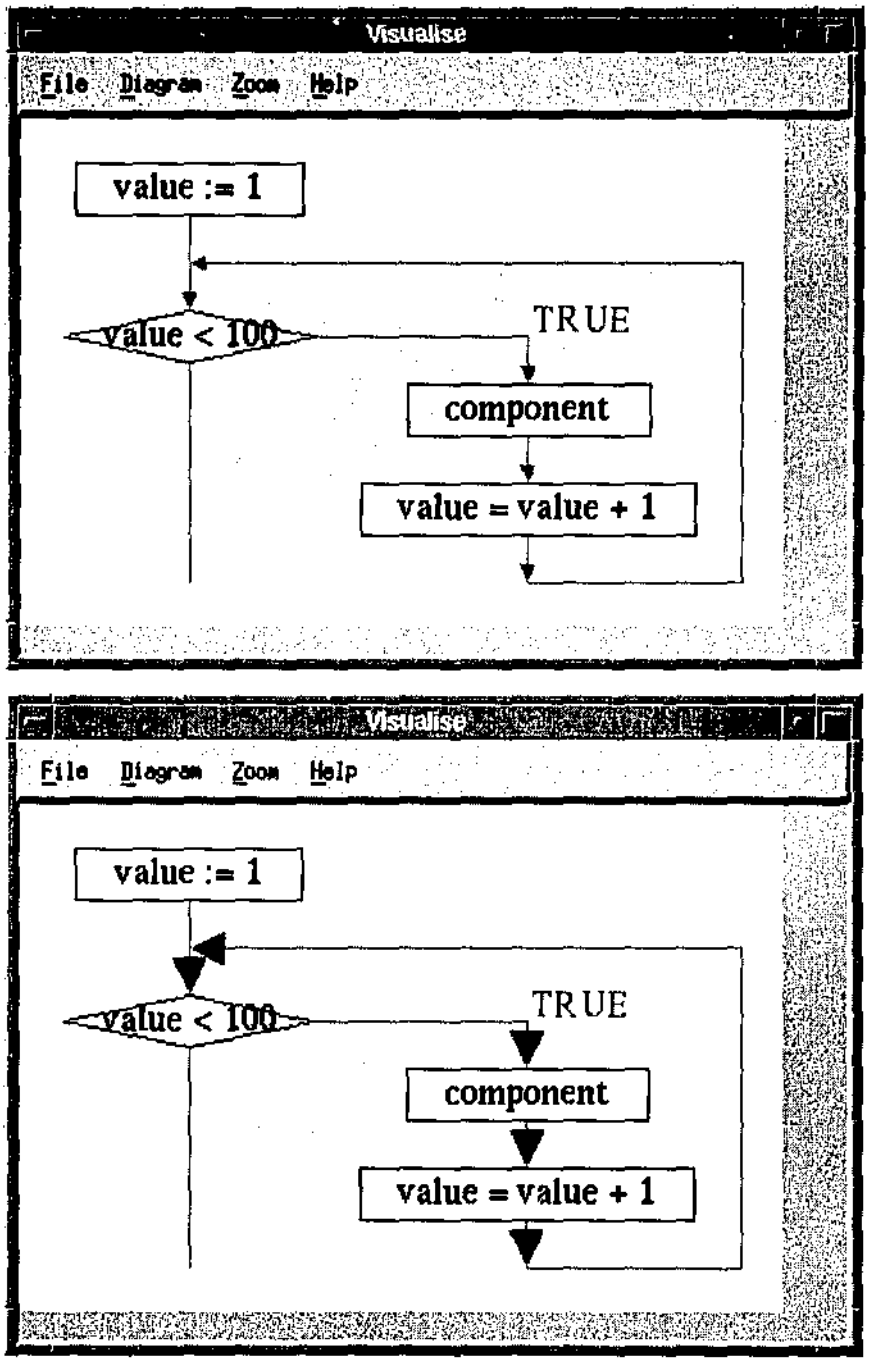


Figure 78 Effect of Changing the Arrow Size Attribute

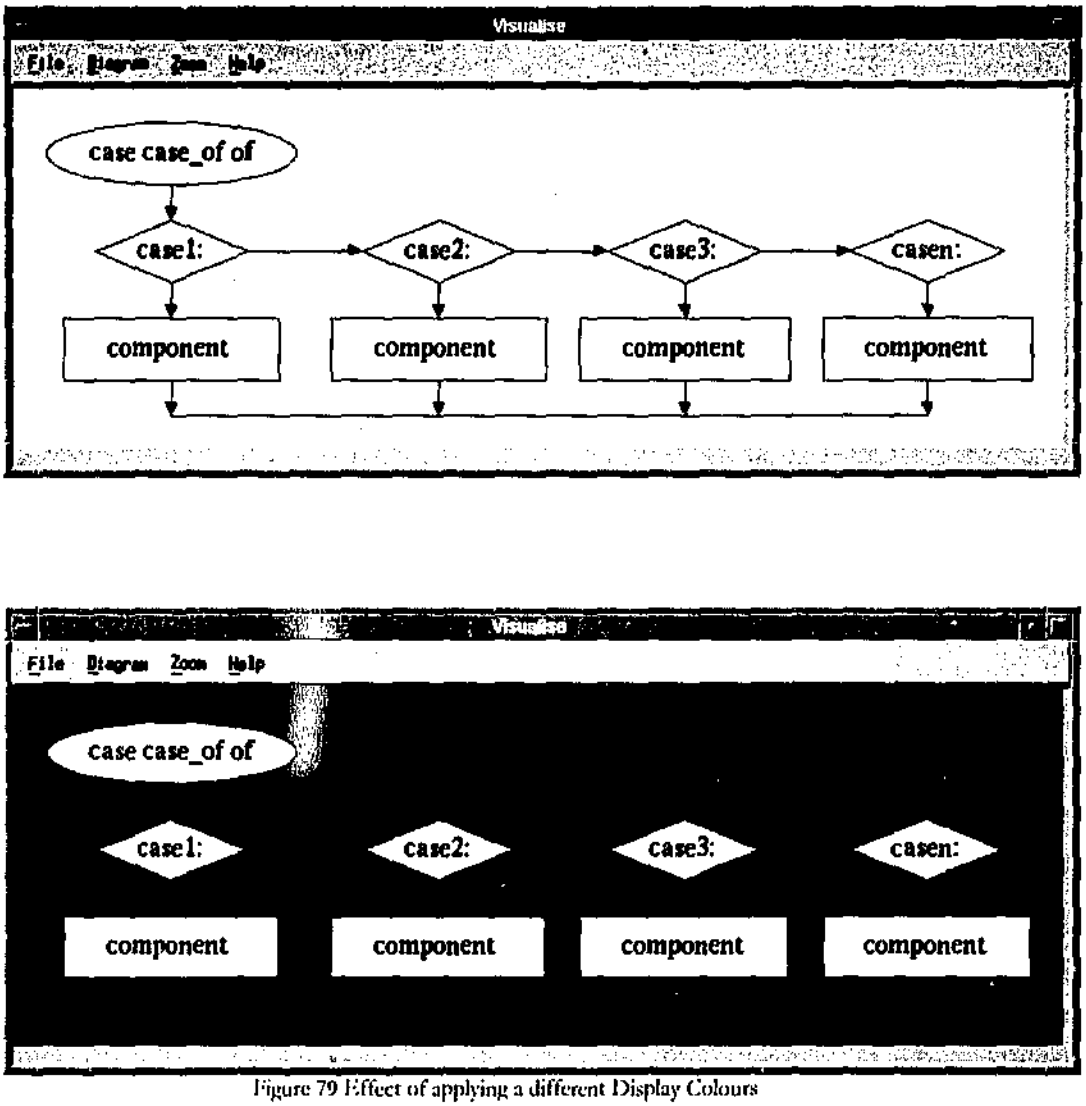


Figure 79 Effect of applying a different Display Colours

Changing the display colours can help us focus on specific areas of a program. Colour coding diagram components can help us to quickly distinguish between diagram components.

Note: The poor choice of display colours can make the diagram difficult to read.

Diagram nodes are separated horizontally and vertically. Changing the amount of separation between nodes can change the appearance of the diagram. To illustrate how changing the component spacing can effect the diagram, a sample *FOR Statement* is considered below:

Note: On figure 80 that the diagram on the left is more compact than the diagram on the right. Also note that the size of each flowchart component is shown by a rectangle around that component. Normally the component boundary would not be visible, but is displayed to help understand the effect of changing the components spacing.

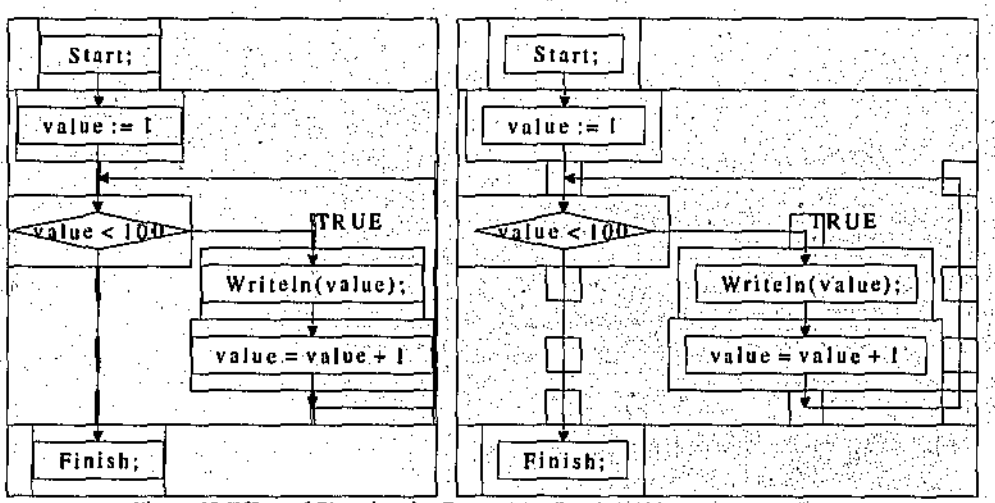


Figure 80 Effect of Changing the Component Spacing with Node boundaries displayed.

5.4. Sample Pascal Programs And Program Output

To test the capabilities of the Prototype in creating flowcharts a sample of Pascal programs were examined from Koffman (1989). The source for the flowcharts can be located in the appendix section of this thesis.

The diagrams generated were consistent with other diagrams produced. Due to the limitation of the amount of information you can fit onto an A4 page, some scaling was used on the diagrams and as a result some diagrams may appear slightly different. Figure 89 *Program ShowDiff* is a good example of the program's ability to generate a high quality flowchart diagram.

5.4.1. Sample Pascal Program 1 – Program Cryptogram

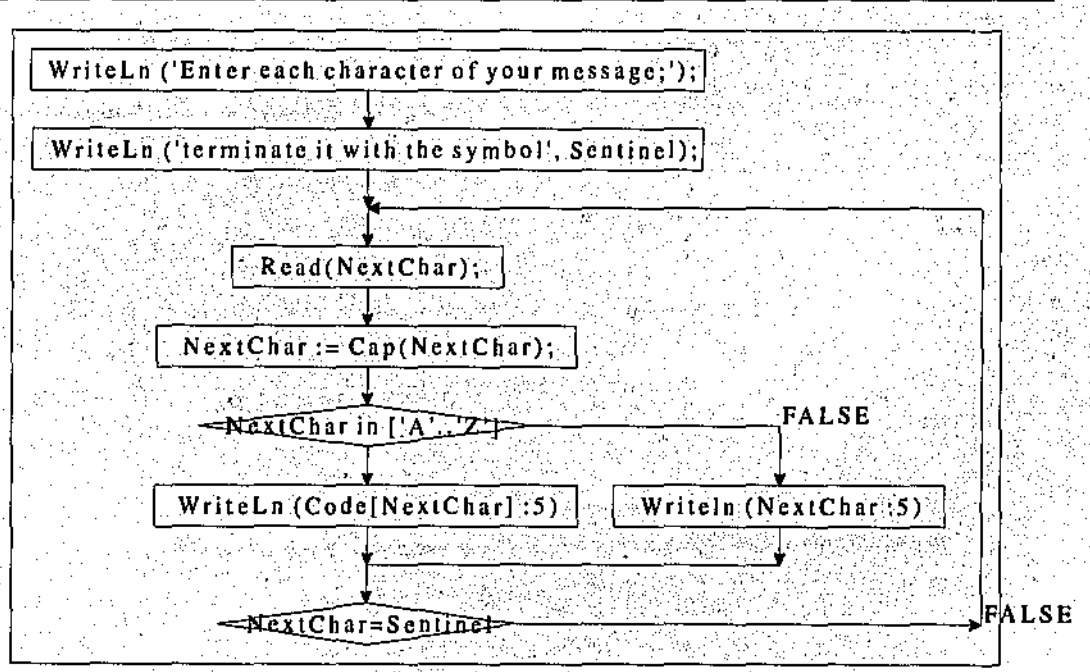


Figure 81 Procedure ReadCode

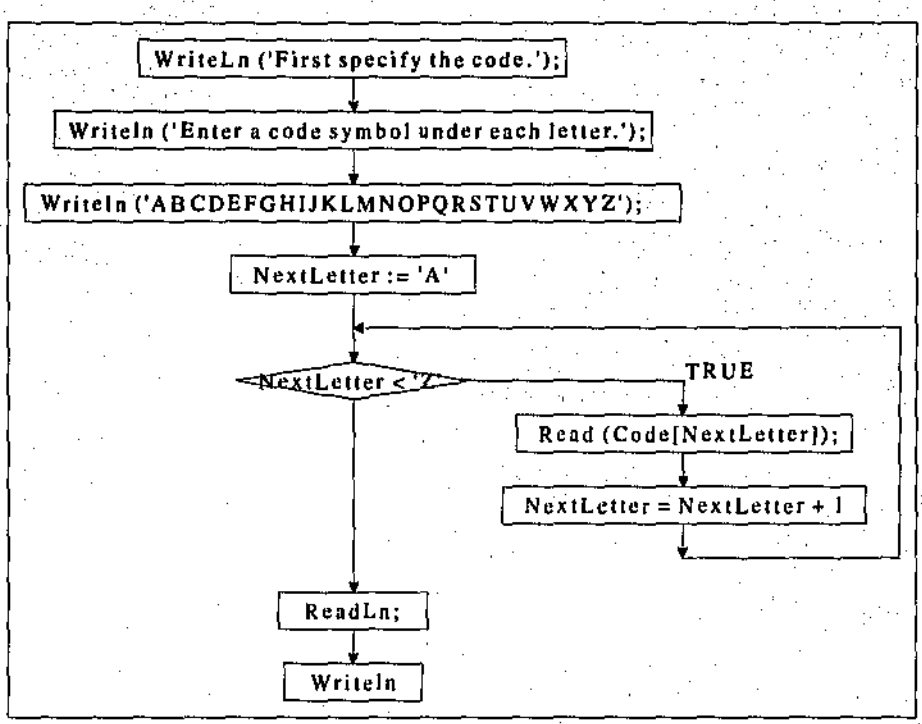


Figure 82 Procedure Encrypt

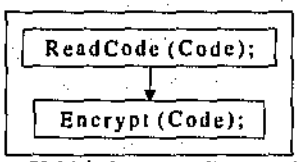


Figure 83 Main Program - Program Cryptogram

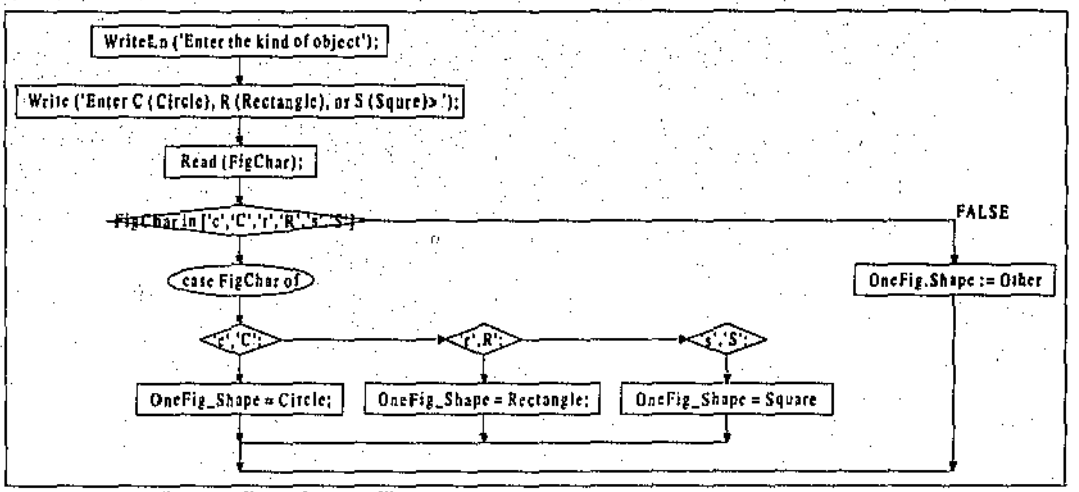


Figure 84 Procedure GetFigure

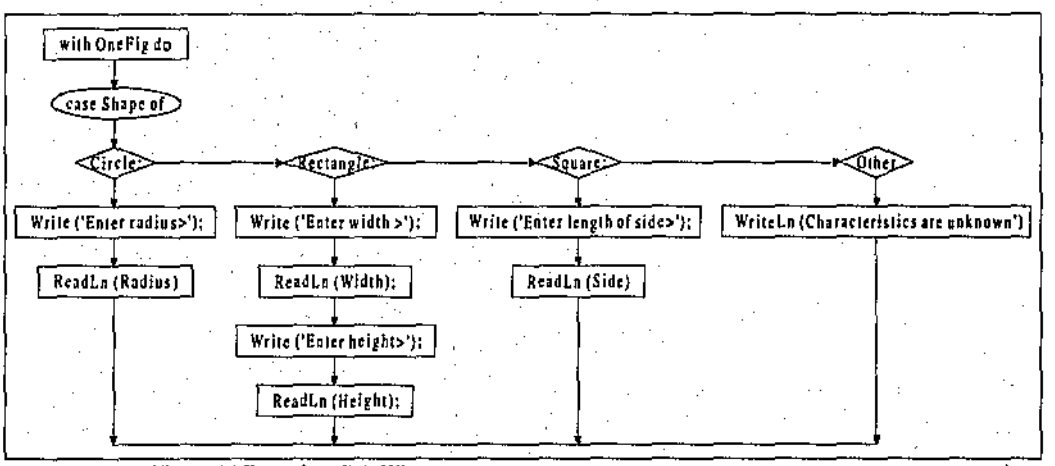


Figure 85 Procedure ReadFigure

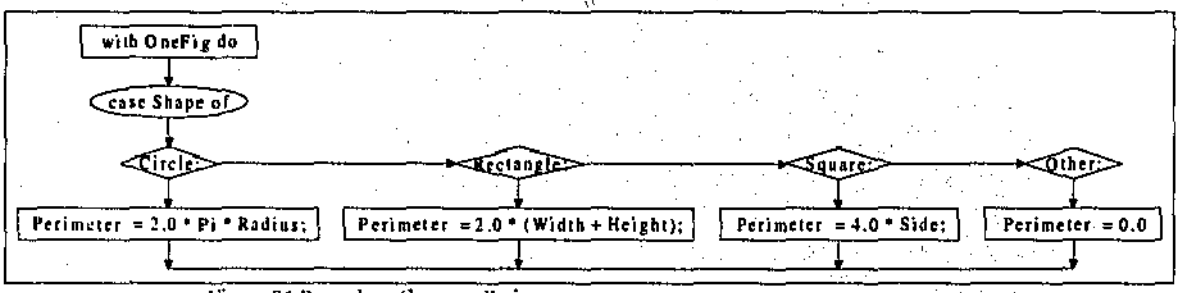


Figure 86 Procedure ComputePerim

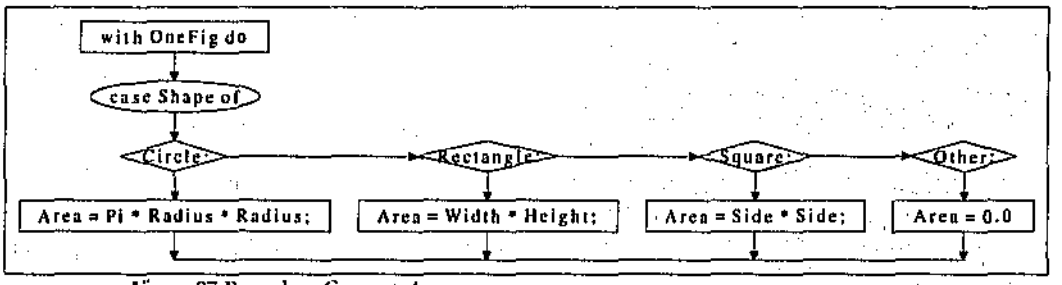


Figure 87 Procedure ComputeArea

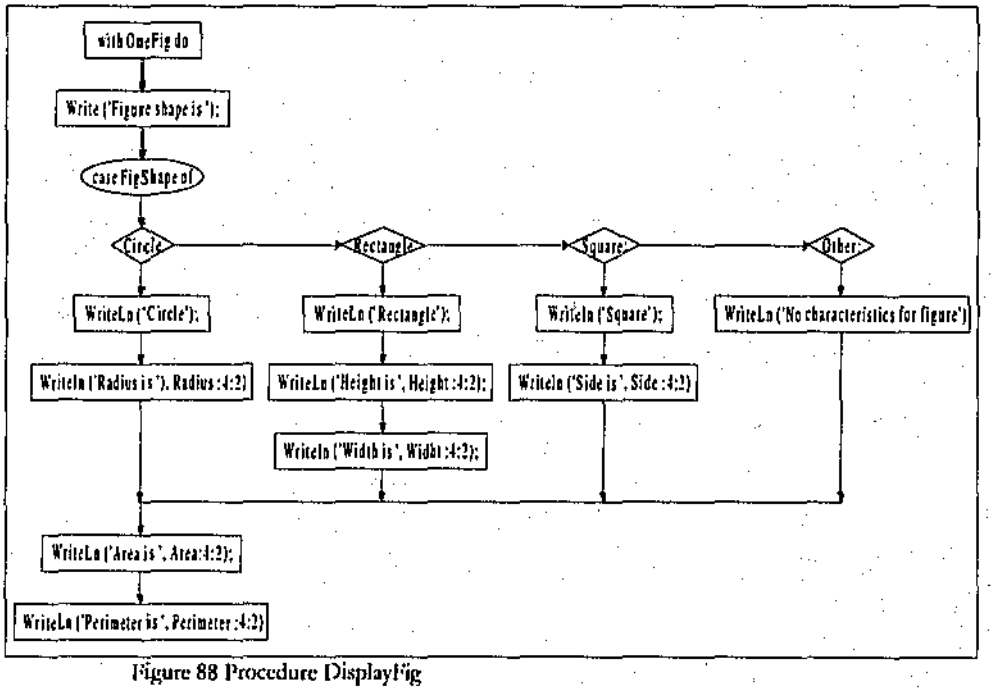


Figure 88 Procedure DisplayFig

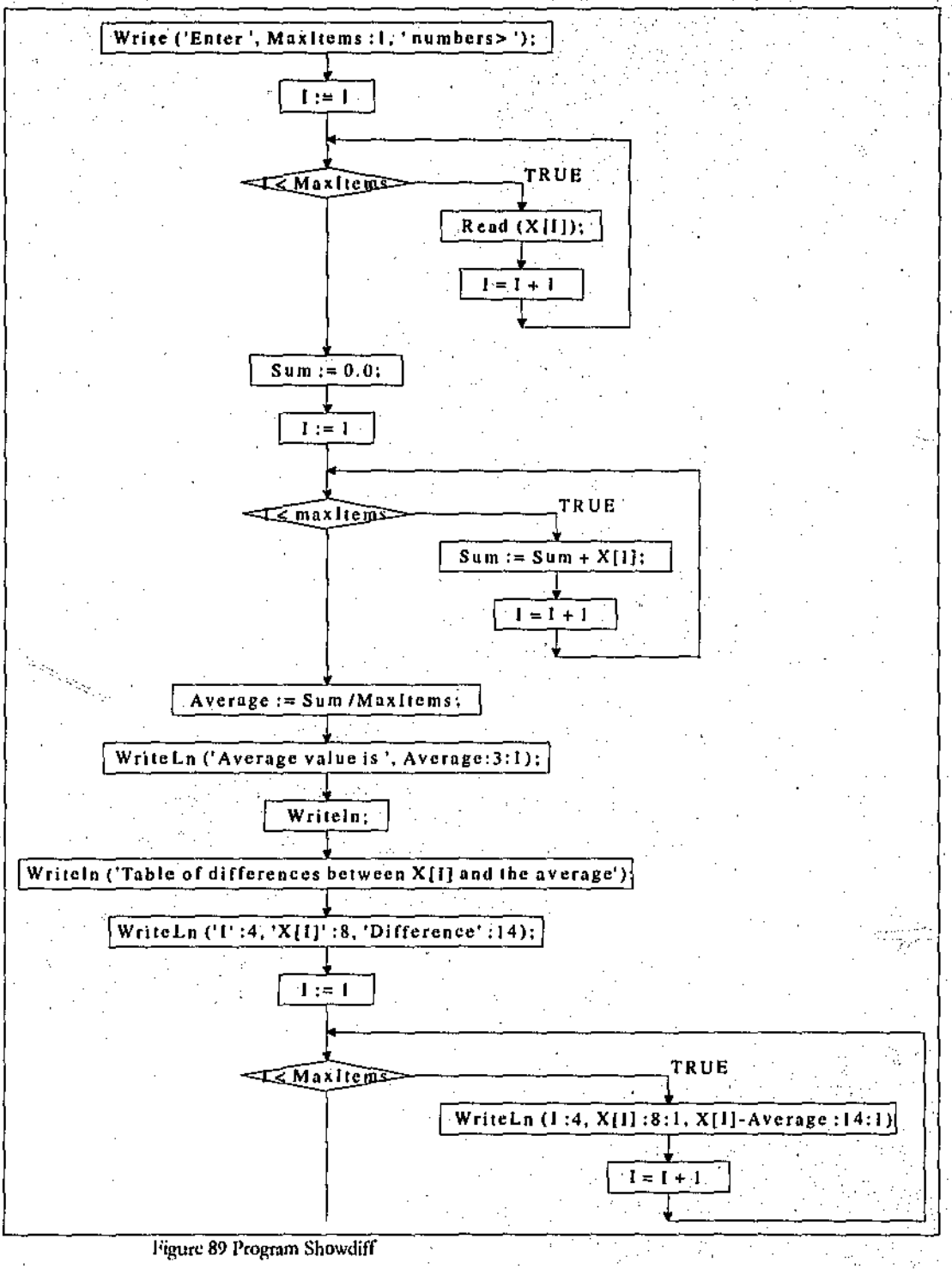


Figure 89 Program Showdiff

5.5. Information Hiding

Restricting the amount of visual information displayed can greatly simplify a diagram. The prototype allows the user to selectively expand/collapse a flowchart component with a single mouse button click. The Pascal program and figures to follow demonstrate the effect of selectively expanding a flowchart. The effect of selectively collapsing a flowchart is done in a similar process where a mouse click in the outer boundary of a flowchart component will collapse that component. The Pascal program and figures to follow demonstrate the effect of selectively expanding a flowchart.

```
begin
    statement1;
    if c1 then
        begin
            statement2;
            statement3;
            statement4;
        end;
    else
        while c2 do
            begin
                statement5;
                statement6;
                statement7;
            end
    statement8;
    statement9;
end;
```

Figure 90 Sample Pascal Program

Figure 91 Fully Collapsed Flowchart Figure

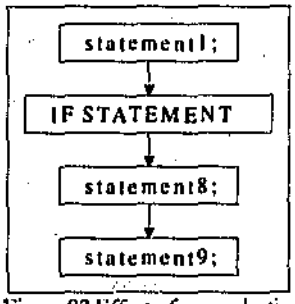


Figure 92 Effect of user selecting the BLOCK STATEMENT

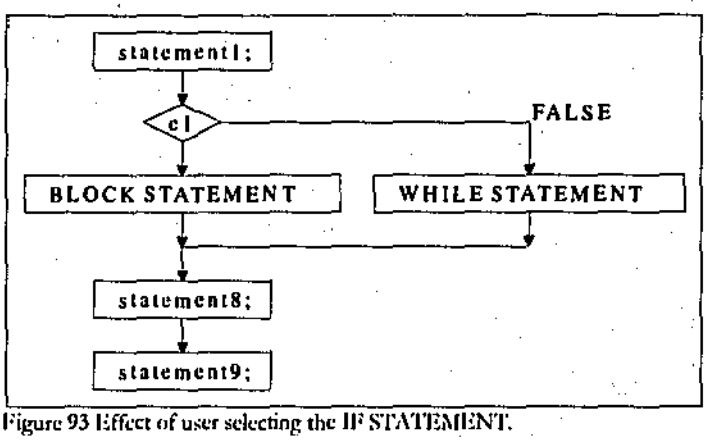


Figure 93 Effect of user selecting the IF STATEMENT.

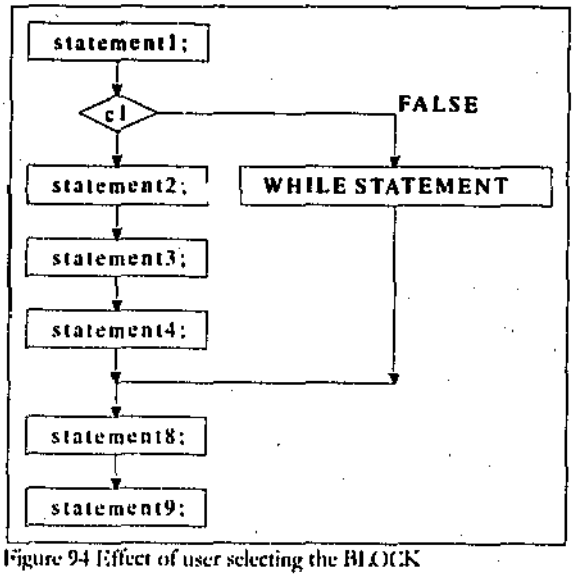


Figure 94 Effect of user selecting the BLOCK

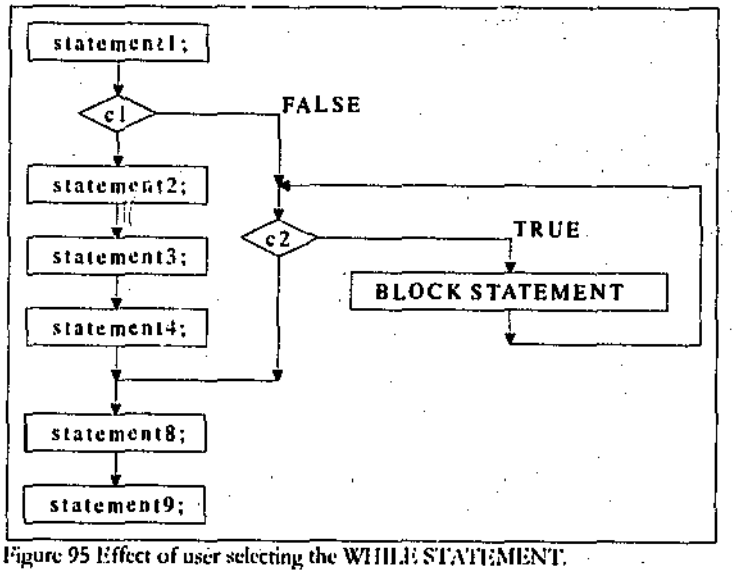


Figure 95 Effect of user selecting the WHILE STATEMENT.

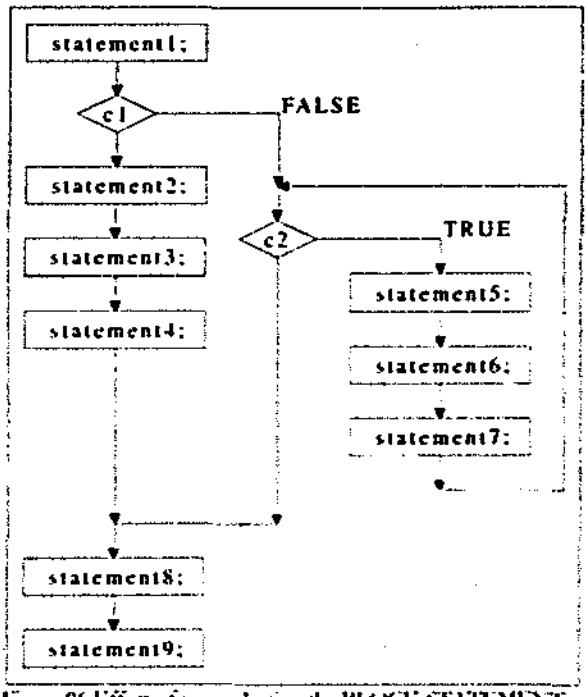


Figure 96 Effect of user selecting the BLOCK STATEMENT

5.6. Application Of The Prototype To Other Diagrams

It is important that the prototype can be applied to other types of diagrams. To do this all that is required is to develop a set of layout routines that describe the new component's layout. As an example of this, the prototype was extended to include Hierarchy diagram. The hierarchy diagram generated can be used to demonstrate the internal storage of the flowchart diagram. The function used to generate the Hierarchy diagram layout is called *Create_Hierarchy_Diagram_Layout* and the source code is listed in the appendix section of this thesis. To allow this program to create a Functional Hierarchy diagram all that would be required is to create a new Source Code parsing routine and a set simple Layout Creation Routine. The figure below illustrates a sample Hierarchy diagram produced from the system.

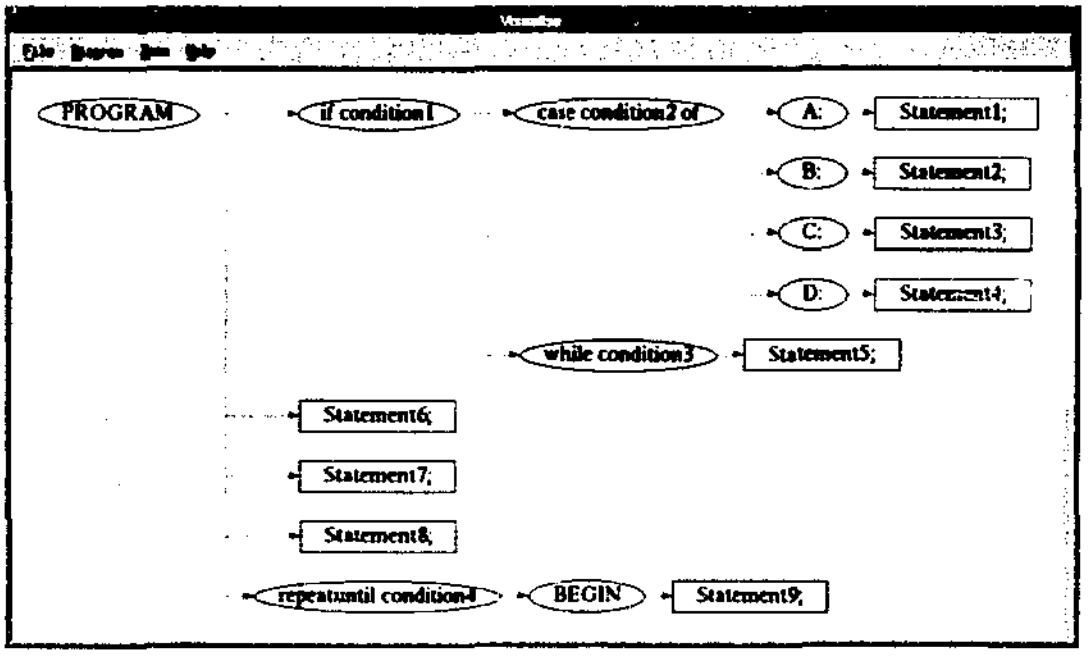


Figure 97 Sample Hierarchy Diagram

5.7. Large Diagram Representation

Many different things can be done to display large diagrams. For example, scroll-bars can be used when the diagram exceeds the display area.

As demonstrated previously, breaking the diagram down into different levels of abstraction can help to reduce the amount of information that a user sees of the flowchart. Extremely large and complicated flowcharts tend to imply that the coding involved really needs to become more structured. If a procedure is broken down into smaller procedures, then the overall effect is a set of smaller flowcharts. Unfortunately some diagrams can be excessively large and the only thing left to do is display the diagram onto many different sections.

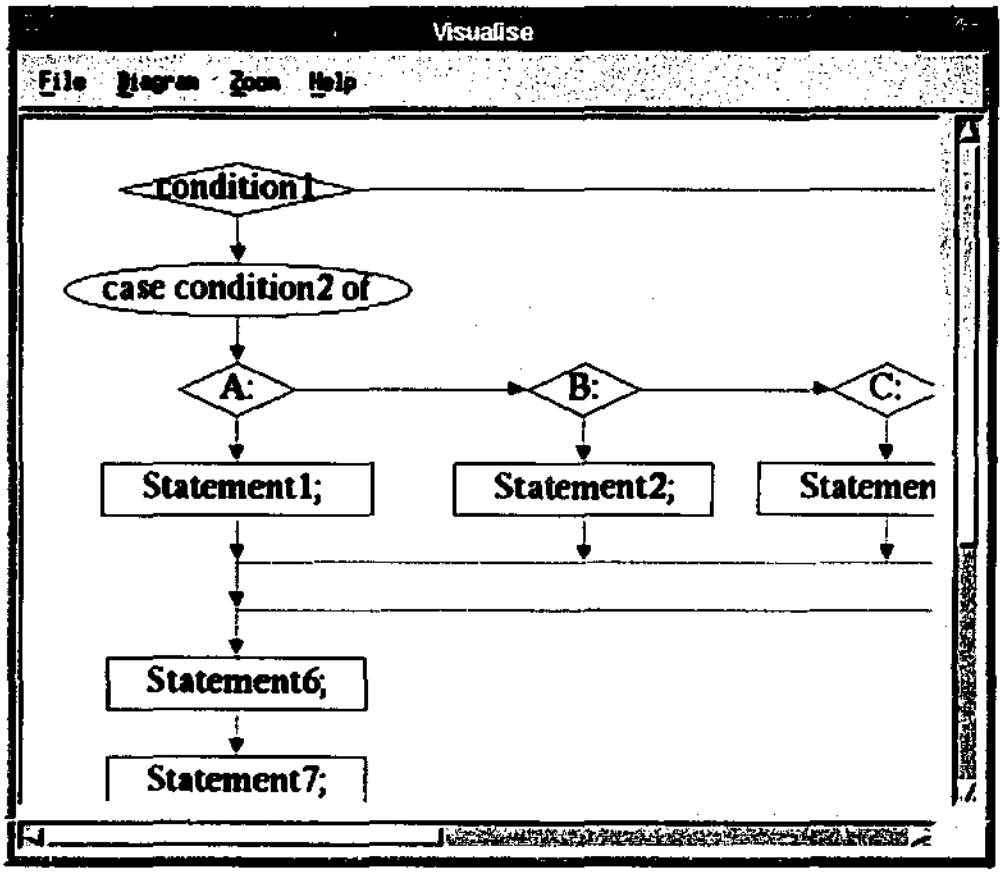


Figure 98 Scroll-Bars

Changing the zoom factor can also shrink the diagram to fit the display and sections of the diagram can be individually zoomed into.

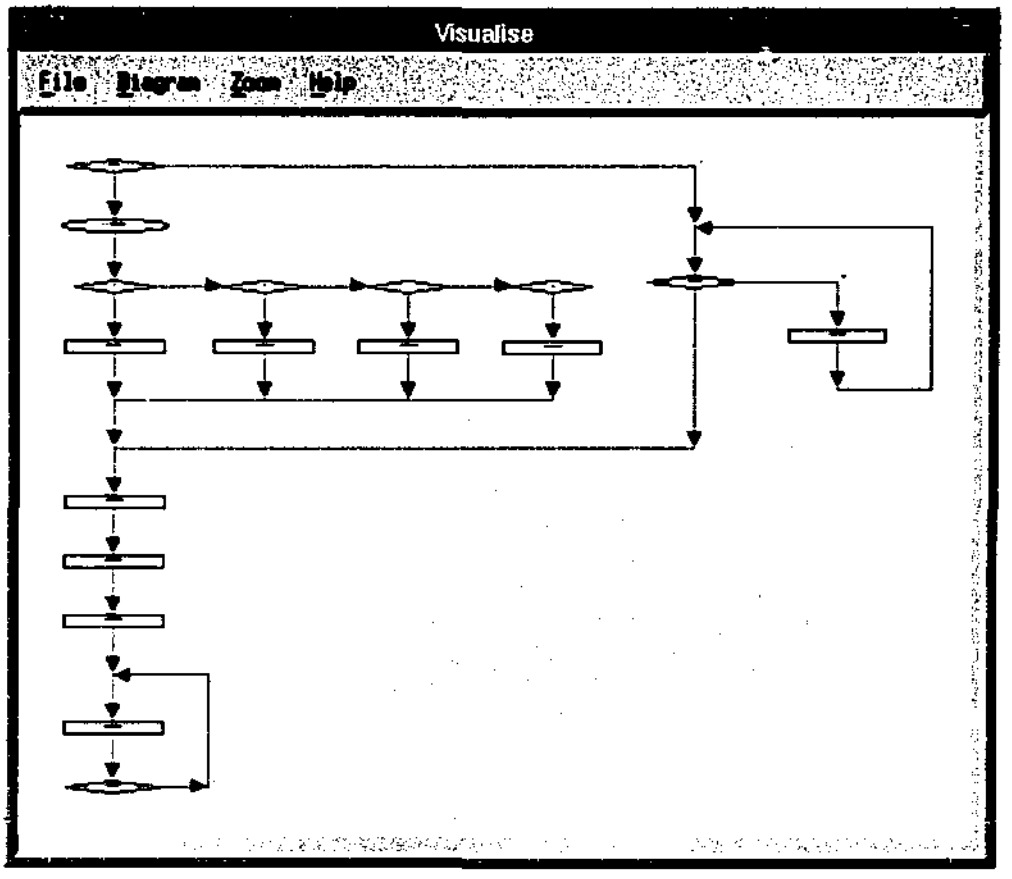


Figure 99 Full diagram display by zooming out

5.8. Extension To The Force-Scan Algorithm

One problem with assuming that diagram component boundaries are rectangular in shape is that the force-scan algorithm can produce a diagram that has wasted space. Extending the force-scan algorithm to work with polygons can further reduce space wastage. The disadvantage of using polygons to represent component boundaries is that a more complicated force-scan algorithm is required and as a result more processing time is required to adjust the diagrams layout. The amount of time taken to process polygons is not considered to be too much of an overhead, as the overall time taken to produce a flowchart would still be significantly faster than having to manually draw the same diagram. Also computer-processing speeds are improving and becoming cheaper.

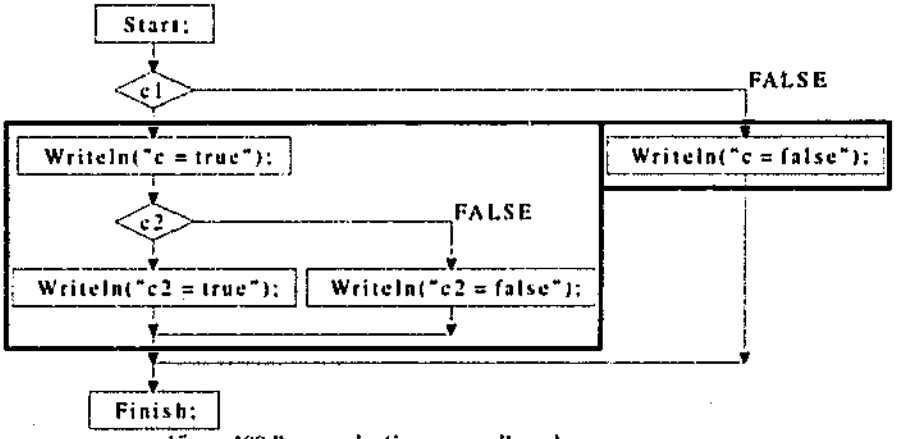


Figure 100 Rectangular Component Boundary.

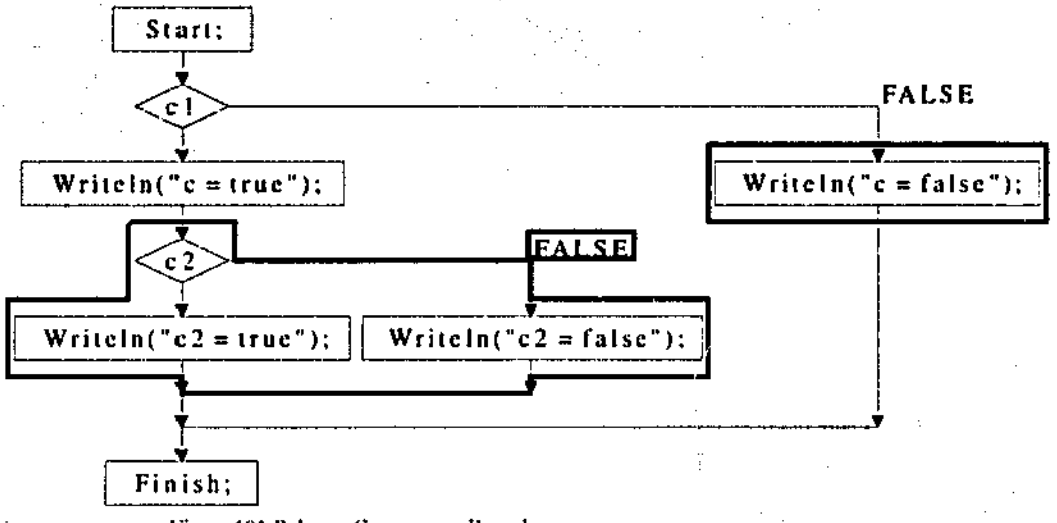


Figure 101 Polygon Component Boundary

5.9. Multiple Flowchart Views

The ability to browse a flowchart diagram with multiple views of a source code file is extremely useful. It is possible to run the prototype many times and obtain different views of the source-code. This is demonstrated by the screen dump in following figure:

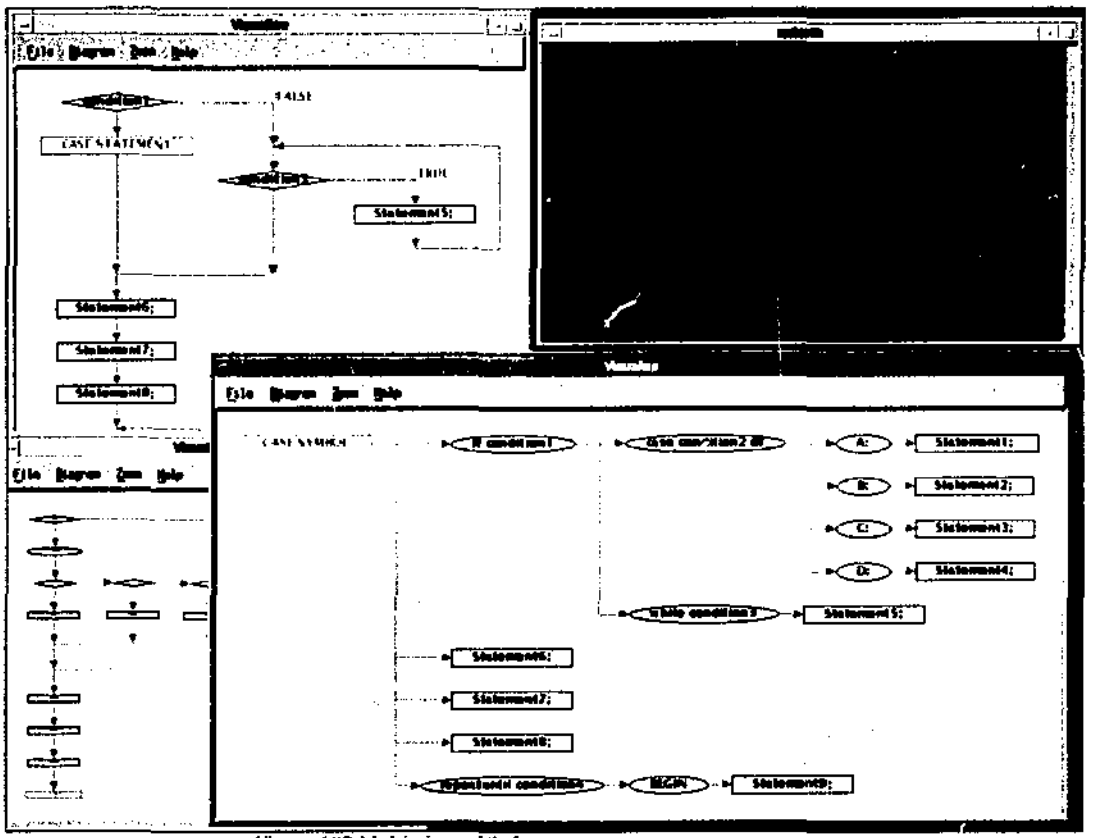


Figure 102 Multi-view of Software

5.10. Edge Overlap Removal

Creating extra nodes in the layout creation process solves the problem of edge overlaps. This is a result of applying the force-scan algorithm. The force algorithm pushes these extra nodes to the desired position and as a result the edges are also redirected. This is illustrated in the figures to follow:

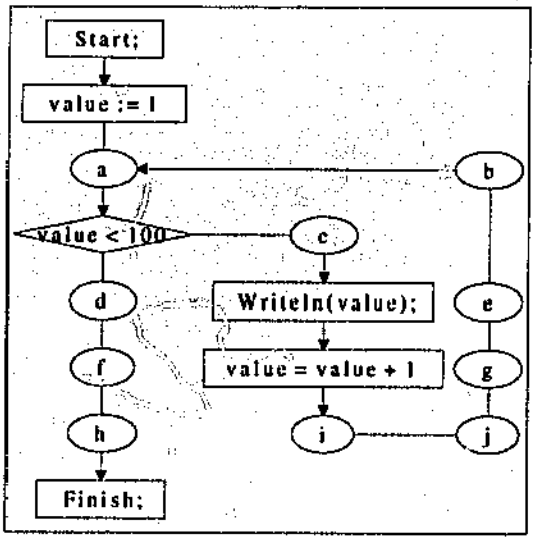


Figure 103 Component Layout for Edge Overlap removal

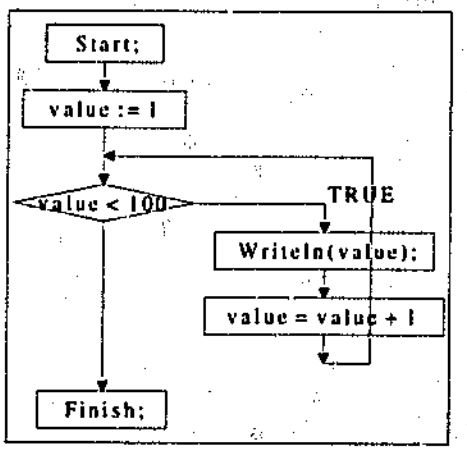


Figure 104 Result of omitting dummy nodes (e) & (g) from previous figure causing Edge overlap.

5.11. Speed Issues

The overall speed of the application is notably fast. If required the application could be enhanced with further research in these areas:

- Use Of Look-up Tables To Speed Up Searches
- Faster Sorting Routines.
- Adjustment to Diagram.

5.12. Summary Of Results

As a general summary of the results, the majority of the diagrams produced were generated quickly, and the output was clear and concise. The following problems were encountered with the Prototype:

- One problem encountered was that when a general statement was too long. For example, to represent the statement:
  Writeln("01234567891011121314151617181920212223242 5");
  would result in a large amount of wasted space in the horizontal direction. A solution to this may be to truncate this statement to smaller length or to make the statement multi-line. By making this statement multi-line will reduce the overall length of the statement.
- Although expected, another problem was encountered where sometimes space was wasted due to the assumption that the diagram components are surrounded by a rectangular boundary. The wasted space can be reduced in size by extending the force scan algorithm to work with polygons as diagram component boundaries.

This results from this chapter proved:

- It is possible to represent a wide range of flowchart components. Eg. IF, FOR, WHILE, REPEAT, and CASE.
- It is possible to represent nested flowchart components. Eg. Nested IF, Nested FOR, Nested WHILE, Nested REPEAT, and Nested CASE.
- Different Diagram attributes affect the diagram. Eg. Font, arrow, display colour, an node space attributes.
- It is possible to represent an entire program through using the prototype.
- Information hiding can reduce the diagram complexity by selectively collapsing and expanding nodes/components as demonstrated in figures 91-96.
- Large diagrams can be displayed through this prototype via different techniques such as scrollbars and zoom-in/out facilities.
- Diagrams can be further compacted by applying the force-scan algorithm by using polygon component boundaries instead of rectangular boundaries. While the polygon approach is a slower process it can significantly improve some diagrams.
- Multiple flowchart views can be used.
- Edge overlap is removed automatically by the addition of extra dummy nodes.

6. Conclusion and Future Work.

This chapter identifies the questions raised by the thesis and provides a discussion to each question. A discussion is given for other important topics. This is then followed by a conclusion and possible future work.

6.1. Question 1

What are the problems associated with automating the flowchart creation process? OR: What are the problems associated with implementing a Software Visualisation System?

There are many different problems associated with automating the process of flowchart creation. Below is a sample of some of the problems encountered and overcome during this project:

- What data structure is suitable for displaying flowcharts and other diagrams? Answer: A range of data structures can be used for displaying flowcharts and other diagrams. The most effective one found was a combination of the tree structure and linked list structure.

- How do you to interpret a source code file and store it internally so that it can be represented as a structured diagram? Answer: Through the use of source code parsing routines.

- How do you generate two-dimensional information for a flowchart diagram with only one-dimensional information such as source code? How can you improve the diagrams overall appearance? NOTE: This question is answered by section 6.4 Question 4.

- What are some possible solutions for displaying large complicated diagrams? NOTE: This question is answered by section 6.2 Question 2.

- How do you display a diagram in a clear and uncluttered manner? NOTE: This question is answered by section 6.3 Question 3.

6.2. Question 2

What are some possible solutions to displaying large complicated diagrams?

The phrase "You cannot see the forest for the trees" is true when it comes to displaying a large diagram. A large diagram can often over-complicate a simple fact that the diagram was meant to illustrate. Changing the zoom factor can also display the entire diagram and thereby give an overall appearance of the flowchart. This can be effective when the size of the diagram is not too big.

Most two-dimensional computer diagramming packages provide scrollbars that allow the diagram to be scrolled around and viewed. This is a useful tool to view diagrams that exceed the size of the display area.

Providing different levels of abstraction can also help to reduce a diagram's complexity. For example, this project created a program that allows individual components of a diagram to be collapsed or expanded with the response to a mouse button click. The menu system allows the user to collapse the diagram and then expand the sections of the diagram that user wanted to view. Providing different levels of abstraction is a useful tool for information hiding. Ideally the programmer would use a structured programming methodology in which their code uses functions and procedures as much as possible. This would improve the appearance of the flowcharts created. Diagrams can be produced for each of the functions and procedures separately. The design of the prototype was to allow the user to browse around a program freely. Although not yet implemented, the prototype was to include a feature to allow the user to select a procedure call and then the program would create a new window with the related procedure's flowchart displayed within. This feature would be useful when browsing a large program.

6.3. Question 3

How can we automatically display diagrams in a clear and uncluttered manner? OR: How do we address the graph drawing problem?

To automatically display diagrams, a set of layout generation routines are required. To display the diagram in a clear and uncluttered manner highly depends on the layout generation routine used.

Layout adjustment routines can be applied to the generated diagram to filter out undesired characteristics from the diagram. The main problem with applying a layout adjustment routine is that the layout adjustment routine may distort the diagram from its original layout. The layout adjustment routine chosen for this project was the force-scan algorithm. The reason for choosing the force-scan algorithm is that it can create a very compact diagram and does not distort the appearance of the diagram.

Understanding how the force-scan algorithm works also helped in creating layout generation routines that produced diagrams that did not have node and edge overlaps. Strategically placing a set of dummy nodes in a flowchart component definition ensured that diagram's edges were redirected around components. Without this knowledge a new problem of how to re-route diagram edges would need to be addressed.

6.4. Question 4

How do we convert one-dimensional information into two-dimensional information. E.g. How do we convert Pascal source code into a Flowchart.

This was addressed by the thesis by first breaking the programming language into distinct statement types. Information from the programming language statements was then converted into flowchart components. These components were assigned relative positions based on the definition of the flowchart components. Layout adjustment routines were then applied to tidy the appearance of the diagram.

The process of generating two-dimensional information from one-dimensional information is as follows:

- Break programming language into statements/components.
- Break diagramming system into components.
- Use a set of abstract layout creation routines for diagram components.
- Utilise a layout adjustment routine for the entire diagram.

6.5. Limitations

There were no real limitations encountered with this project. All test of the prototype produced diagrams as expected and the quality of the flowcharts was high. The only problem with creating flowcharts seemed to be the notation used to draw flowcharts. For example, to draw an if symbol you draw some text and then you draw a diamond around the text. Drawing a diamond around text can cause a large symbol to be created when the text string is lengthy. A solution to this problem would be to adopt a slight different diagramming notation.

6.6. Abstract Flowchart Definition

The programming required no geometric information such as location, size, etc. to be specified by the programmer, as this was all determined by the application of the force-scan algorithm. Note: Only relative positions were required from the programmer. It was noted after the implementation that it would be possible to further abstract the layout creation process. Function ROW and COLUMN could be used to define columns and rows of flowchart component nodes. For example, the Node Positions table defined in figure 46 defines the node positions as:

A = (x+1, y-2)

a = (x, y)

b = (x+2, y)

c = (x,y-1)

d = (x+1, y-2)

e = (x, y-2)

f = (x+2, y-2)

g = (x, y-3)

h = (x+1, y-3)

i = (x+2, y-3)

Using the ROW and COLUMN function we could define the node positions by:

ROW (a, b)

ROW (c, d)

ROW (e, A, f)

ROW (g, h, i)

COLUMN (a, c, e, g)

COLUMN (d, A, b)

COLUMN (b, f, i)

Note: The columns and rows can be seen by examining figure 45. Also note that the order in which ROW and COLUMN are called. For instance, by looking at the series of calls to ROW we can tell that the first ROW is ROW a-b, followed by c-d, and so on. We can also tell that ROW e-A-f is between ROW c-d and ROW g-h-i. This description allows us to define the layout without having to specify numeric information. This description is also shorter and easier to code.

6.7. Ease of Use.

During the initial writing of this thesis, all of chapter 3's diagrams were sketched out on paper and then converted to electronic format through a number of drawing packages. This process was time consuming and the end product took some effort to ensure consistency between the diagrams. After the prototype was finished it took very little time to regenerate the same diagrams with very little effort. All of the component shape and layout diagrams in Chapter 3 were created using the prototype. There were some changes made to the text in these diagrams, however the layout of the diagrams remained the same. All diagrams produced in Chapter 4 were solely produced from the prototype. The main advantage of using the prototype was the speed, quality and consistency of the diagrams

produced. To demonstrate how easy it is to generate an entire flowchart, the following Unix command created the flowchart seen in figure 75:

Visualise nestedfor.pas

To create a Hierarchy diagram was equally as easy:

Visualise nestedfor.pas –HIERARCHY

This could also be simplified further by allowing the user of the program to select from a list of source code file names. When the user selects this file a diagram is produced. The user could further select from a set of diagram types to generate a different type of diagram.

Typing one command to generate a flowchart has obvious benefits over manually creating the same diagram by hand. As the generated diagrams are consistent we find that a programmer can view other peoples code even though their coding style may be completely different. This could be extended further to include the possibility of interpreting different programming languages. Eg. Code written in two different languages that do the same thing would produce identical flowcharts. This would be useful as a programmer could view any source code through the generated flowchart without having to learn that specific programming language.

6.8. Pseudo-code and Flowcharts.

The traditional approach for creating flowcharts is to manually draw them. Creating simple flowcharts is time consuming as time is spent moving diagram symbols around until they appear to have a good layout. Making changes to such a diagram can become a time consuming task and in some cases it is easier to recreate the entire diagram from scratch. Clearly this is slow way of producing flowcharts. It is evident from writing the Software Visualisation Prototype that we could change the prototype to interpret a version of pseudo code that would contain a set of programming language constructs such as IF, WHILE, REPEAT, FOR, etc. This pseudo code would form a flowcharting language that would allow someone to create a flowchart from a basic textual description. For example the statement below would create an IF flowchart component automatically:

IF "Debt is greater than $500 and the last payment is overdue" THEN

    "Send the customer an overdue notice";

    "Store record of payment notice into the computer";

END IF

This method of creating flowcharts is far more effective. We can also further refine the pseudo code to create the actual computer programming language statements.

6.9. Conclusion

Software Visualisation is a topic that is rapidly growing in popularity. There are few software visualisation tools that can effectively generate flowchart diagrams. This thesis demonstrated the implementation of a flowcharting program that is capable of automatically generating flowcharts quickly and effectively.

Other points of interest include:

- The ability to generate a number of different diagrams was proven. For example, In this project flowchart and hierarchy diagrams were generated.
- All diagrams produced were generated quickly and in all cases, the flowcharts were created faster than it took to type in the Unix command!
- All diagrams generated produced a high quality output.
- All diagrams produced required NO user input. This was an important problem to solve as was addressed.
- The prototype demonstrated that creating a visualisation tool is a highly effective way of creating diagrams from source code.
- It is possible to hide excess information through various techniques such as collapsing diagram components.
- It is possible to create an entire diagram by only specifying a set of simple "abstract layout creation routine." that only define the general shape of the diagram components without having to specify exact geometric information such as size, location, etc.
- The majority of the diagrams in this thesis were generated through the prototype.
- Initial thesis questions were answered.

6.10. Future Work

The prototype was successful in generating diagrams from source-code. The layouts for the diagrams were created quickly. After using the prototype it is clear that this is a far more effective way of generating diagrams than traditional methods. As this was only a prototype future work could be done to increase the functionality of the prototype. Some of this functionality could include:

- Development of routines to enhance the force-scan algorithm to interpret node boundaries as polygons. This would further reduce the size of some diagrams.

- Create a set of modules to interpret a wider range of programming languages. The program could then identify the source codes language and then automatically create flowcharts by using the selected language module.

- Develop a flowcharting language based on pseudo-code. This would allow the user to quickly develop flowcharts without having to specify geometric information. This would be an extremely effective way of generating flowchart diagrams over the traditional approach of drawing of them.

- Develop a set of different layout routines to generate a wider range of diagrams.

- Develop a large set of layout routines for each component. Using this range of component layouts, the computer could then try different combinations of these components to generate a range of diagrams. The computer could then identify the diagram with the smallest size for the user. This could be further extended by using artificial intelligence to identify the best combinations of components to generate a diagram with the smallest layout possible.

7. Reference List

Cunniff, N., Taylor, R., Black, J. (April 1986) *Does Programming Language Affect the Type of Conceptual Bugs in Beginners' Programs? A Comparison of FPL and Pascal.* Proceedings of CHI'86. April 1986. Pp. 175-182.

Diaz-Herrera, J., Fludc, R. (1980) *Pascal/HSD: A grephical programming System.* Proceedings of the IEEE COMPSAC' 80. Pp. 723-728.

Jacobs, D., Marlin, C. (1995) *Unparsing Flowcharts from Abstract Syntax Trees in A Multiple View Software Development Environment.* Australian Computer Science Communications, Vol. 17, No. 1.

Kamada, T. (1989) *Visualizing Abstract Objects and Relations.* Computer Science. World Scientific. Volume 5. Singapore, New Jersey, London, Hong Kong.

Koffman, E, B. (1989) *Pascal 3rd Edition.* Addison-Wesley Publishing Company, Inc.

Harel, D. (1988) *On Visual Formalisms.* Communications of the ACM. Vol. 31. No. 5. Pp 514-530.

Lai, W. (1993) *Building Interactive Diagram Applications.* Department of Computer Science, University of Newcastle. PhD. Thesis.

Lai, W., Eades, P. (1996) *Structural modelling of flowcharts.* World Scientific Publishing. Pp. 232-243.

Myers, B. (1995) *The State of the Art in Visual Programming and Program Visualization.* Conference Proceedings. Computer Science Department Carnegie Mellon University Pittsburgh USA.

Misue, M., Eades, P., Lai, W., Sugiyama, K. (1995) *Layout Adjustment and the Mental Map.* Journal of Visual Languages and Computing. Volume 6, pp. 183-210

Reiss,. S. (March 1985) *PECAN: Program Development Systems that Support Multiple Views.* IEEE Transactions on Software Engineering. Vol. SE-11, No 3. Pp. 276-284.

Reiss, S. (May 1984) *Graphical Program Development with PECAN Program Development Systems.* Communications of the ACM. Vol. 19. No5. Pp. 30-41.

Reiss, S. (March 1985) *PECAN: Program Development Systems That Support Multiple Views.* IEEE Transaction on Software Engineering. Vol. 11. No. 3. Pp. 276-285.

Reiss, S., Golin, E., Rubin, R. (June 1986) *Prototyping Visual Languages with the GARDEN System.* Proceedings of the IEEE Workshop on Visual Languages. Pp. 81-90.

Shiftlet, A, B. (1990) *Elementary Data Structures with Pascal.* West Publishing Company. St. Paul New York, Los Angeles, San Francisco.

N.B. References embedded in direct quotes have been presented in their original format.

[OXFORD 83] *Dictionary of Computing.* Oxford: Oxford University Press, 1983.

[Sut63] Ivan E. Sutherland. "SketchPad: *A Man-Machine Graphical Communication System,*" AFIPS Spring Joint Computer Conference. 23 1963. Pp. 329-346

[Adobe85] Adobe Systems, Inc. *Postscript Language Reference Manual.* Addison-Wesley, 1985

[Apple 85] Apple Computer, Inc. *Inside Macintosh.* Addison-Wesley, 1985.

[GRAFTON85] Robert B. Grafton and Tadao Ichikawa, eds. IEEE Computer, Special Issue on Visual Programming 18(8) Aug. 1985.

A. "Visualise.c"

```
/*****************************************************************
 * File : Visualise.c                                            *
 * Main program for the Software Visualisation Prototype.        *
 * Development Environment:                                      *
 *    X-Windows                                                  *
 *    Linux 1.2.13                                               *
 *      Motif extensions                                         *
 * Compiled with the command :                                   *
 *    cc Visualise.c -o Visualise  -lXm -lXt -lX11               *
 *               -L/usr/X11R6/lib/  -L/usr/include/              *
 *               -L/usr/X11R6/include/X11/                       *
 *          -lm -ffast-math  -Wunused                            *
 *****************************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <Xm/MainW.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/DrawingA.h>
#include <Xm/CascadeBG.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>

#include "Global_Definitions.h"
#include "File_Operations.h"
#include "Diagram_Drawing_Routines.h"
#include "Node_Operations.h"

NODE *The_Diagram;

#include "File_Reading_Routines.h"
#include "Geometry_Operations.h"
#include "Display_Routines.h"
#include "Node_List_Sorting_Routines.h"
#include "Force_Scan_Routines.h"
#include "Menu_Selection_Routines.h"
#include "Menu_Generation_Routines.h"
#include "Mouse_Events.h"
#include "User_Interface.h"

void
main (Argument_List_Count, Argument_List)
     int Argument_List_Count;
     char *Argument_List[];
{
  Read_Configuration_File (CONFIGURATION_FILE_NAME);
  The_Diagram = Create_Node ("PROGRAM", KEYWORD_UNKNOWN);
  if (Argument_List_Count == 1)
```

```
    {
      printf ("%s : No input file specified.\n", Argument_List[0]);
      printf ("USAGE :  %s filename [-Hierarchy]\n",
Argument_List[0]);
      printf ("PARAMETERS :  filename = pascal file to be displayed
\n");
      printf (                -Herarchy forces a hierarchical display
\n");
      exit (0);
    }
  else
    Parse_Source_Code_File (Argument_List[1], The_Diagram);
  if (errno != 0)
    printf ("%s : No such file.\n", Argument_List[0]);
  else
    {
      if (Argument_List_Count >= 3)
      {
        if (strcmp (Argument_List[2], "-Hierarchy") == 0)
          Create_Hierarchy_Diagram_Layout (The_Diagram);
        else
          Create_Flowchart_Diagram_Layout (The_Diagram);
      }
      else
      Create_Flowchart_Diagram_Layout (The_Diagram);

      Map_Coordinates_To_Display (The_Diagram);
      /* Adjust_Diagram_Layout (The_Diagram); */
      Create_User_Interface (Argument_List_Count, Argument_List);
    }
}
```

B. "Global_Definitions.h"

```
/*************************************************************
 * File : Global_Definitions.h                               *
 *************************************************************/

#define UP_ARROW 0
#define DOWN_ARROW 1
#define LEFT_ARROW 2
#define RIGHT_ARROW 3

#define NO_ARROWHEAD 0
#define SINGLE_ARROWHEAD1 1
#define SINGLE_ARROWHEAD2 2
#define DOUBLE_ARROWHEAD 3

#define XFIG_FILE_NAME "XFIG.fig"

FILE *XFIG_File_Pointer;

/* A list of some system fonts
   #define FONT_NAME "vga"
   #define FONT_NAME "-adobe-helvetica-medium-r-normal--0-0-75-75-p-
0-iso8859-1"
   #define FONT_NAME "5x7"
   #define FONT_NAME "lucidasans-8"
   #define FONT_NAME "kana14"
   #define FONT_NAME "lucidasanstypewriter-12"
   #define FONT_NAME "nil2"
   #define FONT_NAME "olcursor"
   #define FONT_NAME "12x24romankana"
   #define FONT_NAME "lucidasanstypewriter-bold-10"
   #define FONT_NAME "-schumacher-*-*-*-*-*-*-*-*-*-*-*-*-*"
   #define FONT_NAME "-misc-*-*-*-*-*-12-*-*-*-*-*-*-*-*"
   #define FONT_NAME "-misc-fixed-*-r-*-*-12-*-*-*-*-*-*-*-*"
   #define FONT_NAME "-*-clean-*-*-*-*-10-*-*-*-*-*-*-*"
   #define FONT_NAME "-b&h-*-*-r-*-*-10-*-*-*-*-*-*-*"
*/

/* Default Viewing font
  #define XFIG_SCALEX 20
  #define XFIG_SCALEY 20
  #define FONT_NAME "fixed"
 */
/* XFIG Font Format */
  #define XFIG_SCALEX 15
  #define XFIG_SCALEY 15
  #define FONT_NAME "-adobe-times-medium-r-normal--20-140-100-100-p-
96-iso8859-1"


#define CONFIGURATION_FILE_NAME "Visual.cfg"
#define MAXIMUM_STRING_LENGTH 80

#define KEYWORD_IF 0
#define KEYWORD_FOR 1
```

```
#define KEYWORD_WHILE 2
#define KEYWORD_CASE 3
#define KEYWORD_BEGIN 4
#define KEYWORD_END 5
#define KEYWORD_REPEAT 6
#define KEYWORD_UNTIL 7
#define KEYWORD_ELSE 8
#define KEYWORD_STATEMENT 9
#define KEYWORD_PROCEDURE 10
#define KEYWORD_PROGRAM 11
#define KEYWORD_STATEMENT_BLOCK 12
#define KEYWORD_DUMMY_NODE 13
#define KEYWORD_FUNCTION 14
#define KEYWORD_GENERAL 15
#define KEYWORD_UNKNOWN  16
#define IF_SYMBOL 17
#define LABEL_SYMBOL 18
#define CASE_SYMBOL 19

String Statement_Table[] =
{
  "IF STATEMENT",
  "FOR STATEMENT",
  "WHILE STATEMENT",
  "CASE STATEMENT",
  "BLOCK STATEMENT",
  "END STATEMENT",
  "REPEAT STATEMENT",
  "UNTIL STATEMENT",
  "ELSE STATEMENT",
  "STATEMENT",
  "PROCEDURE",
  "PROGRAM",
  "STATEMENT BLOCK",
  "DUMMY NODE",
  "FUNCTION",
  "STATEMENT",
  "KEYWORD_UNKNOWN",
  "IF SYMBOL",
  "LABEL",
  "CASE SYMBOL"
};

typedef char String_Type[MAXIMUM_STRING_LENGTH];

Widget draw_area;
Display *display;
Screen *screen;
Window draw_window;
GC theGC;

int DISPLAY_GRID, DISPLAY_COMPONENTS, DISPLAY_EDGES,
DISPLAY_BOUNDARIES,
  NODE_SEPERATION_Y, BORDER_DISTANCE_X, BORDER_DISTANCE_Y,
NODE_BOUNDARY_WIDTH,
  NODE_BOUNDARY_HEIGHT, GRID_SIZE, ARROW_SIZE, DISPLAY_DEPTH,
NODE_SEPERATION_X,
```

```
  NODE_SEPERATION_Y, X, Y, FOUND;

String_Type BACKGROUND_COLOUR, FOREGROUND_COLOUR, IF_SYMBOL_COLOUR,
LABEL_TEXT_COLOUR,
  LABEL_TEXT_COLOUR, PROCEDURE_CALL_COLOUR, GENERAL_STATEMENT_COLOUR,
EDGE_COLOUR,
  GRID_COLOUR, BOUNDARY_COLOUR;

Dimension Screen_Width, Screen_Height;
Pixmap Users_Bitmap;
Widget Top_Level, Main_Window, Drawing_Area;
GC The_Graphics_Context;
XtAppContext The_Application;
XGCValues Graphics_Context_Values;
XFontStruct *Font_Structure;
int Zoom_Factor = 10;
String translations =
"<Btn1Down>: draw(down) ManagerGadgetArm()\n\
 <Btn1Down>: draw(down) ManagerGadgetActivate()";
int Output_To_FIG = FALSE;
```

C. "File_Operations.h"

```
/*****************************************************************
 * File : File_Operations.h                                      *
 *****************************************************************/

/*****************************************************************
 * Function : Copies all diagram variables with an initial
 * value. This is only required when the default configuration
 * file is not available.
 *****************************************************************/
void
Set_Default_Values (void)
{
  NODE_SEPERATION_X = 10;
  NODE_SEPERATION_Y = 10;
  BORDER_DISTANCE_X = 10;
  BORDER_DISTANCE_Y = 10;
  NODE_BOUNDARY_WIDTH = 10;
  NODE_BOUNDARY_HEIGHT = 4;
  GRID_SIZE = 50;
  ARROW_SIZE = 9;
  DISPLAY_DEPTH = 0;
  strcpy (BACKGROUND_COLOUR, "white");
  strcpy (FOREGROUND_COLOUR, "black");
  strcpy (IF_SYMBOL_COLOUR, "black");
  strcpy (LABEL_TEXT_COLOUR, "blue");
  strcpy (PROCEDURE_CALL_COLOUR, "green");
  strcpy (GENERAL_STATEMENT_COLOUR, "blue");
  strcpy (EDGE_COLOUR, "blue");
  strcpy (GRID_COLOUR, "lightgrey");
  strcpy (BOUNDARY_COLOUR, "lightgrey");
  DISPLAY_GRID = TRUE;
  DISPLAY_COMPONENTS = TRUE;
  DISPLAY_EDGES = TRUE;
  DISPLAY_BOUNDARIES = FALSE;
}

/*****************************************************************
 * Function : After reading the configuration file the
 * variables are set. These variables effect the visual and
 * overall appearance of the diagram.
 *****************************************************************/
void
Set_Variable_Values (String_Type Variable, String_Type Value)
{
  if (strcmp (Variable, "NODE_SEPERATION_X") == 0)
    sscanf (Value, "%d", &NODE_SEPERATION_X);
  if (strcmp (Variable, "NODE_SEPERATION_Y") == 0)
    sscanf (Value, "%d", &NODE_SEPERATION_Y);
  if (strcmp (Variable, "BORDER_DISTANCE_X") == 0)
    sscanf (Value, "%d", &BORDER_DISTANCE_X);
  if (strcmp (Variable, "BORDER_DISTANCE_Y") == 0)
    sscanf (Value, "%d", &BORDER_DISTANCE_Y);
  if (strcmp (Variable, "NODE_BOUNDARY_WIDTH") == 0)
    sscanf (Value, "%d", &NODE_BOUNDARY_WIDTH);
```

```
  if (strcmp (Variable, "NODE_BOUNDARY_HEIGHT") == 0)
    sscanf (Value, "%d", &NODE_BOUNDARY_HEIGHT);
  if (strcmp (Variable, "GRID_SIZE") == 0)
    sscanf (Value, "%d", &GRID_SIZE);
  if (strcmp (Variable, "ARROW_SIZE") == 0)
    sscanf (Value, "%d", &ARROW_SIZE);
  if (strcmp (Variable, "DISPLAY_DEPTH") == 0)
    sscanf (Value, "%d", &DISPLAY_DEPTH);
  if (strcmp (Variable, "BOUNDARY_COLOUR") == 0)
    sscanf (Value, "%s", &BOUNDARY_COLOUR);
  else if (strcmp (Variable, "BACKGROUND_COLOUR") == 0)
    sscanf (Value, "%s", &BACKGROUND_COLOUR);
  else if (strcmp (Variable, "FOREGROUND_COLOUR") == 0)
    sscanf (Value, "%s", &FOREGROUND_COLOUR);
  else if (strcmp (Variable, "IF_SYMBOL_COLOUR") == 0)
    sscanf (Value, "%s", &IF_SYMBOL_COLOUR);
  else if (strcmp (Variable, "LABEL_TEXT_COLOUR") == 0)
    sscanf (Value, "%s", &LABEL_TEXT_COLOUR);
  else if (strcmp (Variable, "PROCEDURE_CALL_COLOUR") == 0)
    sscanf (Value, "%s", &PROCEDURE_CALL_COLOUR);
  else if (strcmp (Variable, "GENERAL_STATEMENT_COLOUR") == 0)
    sscanf (Value, "%s", &GENERAL_STATEMENT_COLOUR);
  else if (strcmp (Variable, "EDGE_COLOUR") == 0)
    sscanf (Value, "%s", &EDGE_COLOUR);
  else if (strcmp (Variable, "GRID_COLOUR") == 0)
    sscanf (Value, "%s", &GRID_COLOUR);
  else if (strcmp (Variable, "DISPLAY_GRID") == 0)
    sscanf (Value, "%d", &DISPLAY_GRID);
  if (strcmp (Variable, "DISPLAY_COMPONENTS") == 0)
    sscanf (Value, "%d", &DISPLAY_COMPONENTS);
  if (strcmp (Variable, "DISPLAY_EDGES") == 0)
    sscanf (Value, "%d", &DISPLAY_EDGES);
  if (strcmp (Variable, "DISPLAY_BOUNDARIES") == 0)
    sscanf (Value, "%d", &DISPLAY_BOUNDARIES);
}

/*************************************************************
 * Function : Dumps all configuration information to the
 * configuration file.  If any changes have been made to
 * the configuration variables during program execution, then
 * the changes are recorded. If no configuration file is
 * present then variables are given default values.
 *************************************************************/
void
Save_Configuration_File (String_Type File_Name)
{
  FILE *File_Pointer2;
  File_Pointer2 = fopen (File_Name, "w");

  fprintf (File_Pointer2, "NODE_SEPERATION_X = %d\n",
NODE_SEPERATION_X);
  fprintf (File_Pointer2, "NODE_SEPERATION_Y = %d\n",
NODE_SEPERATION_Y);
  fprintf (File_Pointer2, "BORDER_DISTANCE_X = %d\n",
BORDER_DISTANCE_X);
  fprintf (File_Pointer2, "BORDER_DISTANCE_Y = %d\n",
BORDER_DISTANCE_Y);
```

```
  fprintf (File_Pointer2, "NODE_BOUNDARY_WIDTH  = %d\n",
NODE_BOUNDARY_WIDTH);
  fprintf (File_Pointer2, "NODE_BOUNDARY_HEIGHT = %d\n",
         NODE_BOUNDARY_HEIGHT);
  fprintf (File_Pointer2, "ARROW_SIZE = %d\n", ARROW_SIZE);
  fprintf (File_Pointer2, "GRID_SIZE = %d\n", GRID_SIZE);
  fprintf (File_Pointer2, "DISPLAY_DEPTH = %d\n", DISPLAY_DEPTH);
  fprintf (File_Pointer2, "BACKGROUND_COLOUR = %s\n",
BACKGROUND_COLOUR);
  fprintf (File_Pointer2, "FOREGROUND_COLOUR = %s\n",
FOREGROUND_COLOUR);
  fprintf (File_Pointer2, "IF_SYMBOL_COLOUR = %s\n",
IF_SYMBOL_COLOUR);
  fprintf (File_Pointer2, "LABEL_TEXT_COLOUR = %s\n",
LABEL_TEXT_COLOUR);
  fprintf (File_Pointer2, "PROCEDURE_CALL_COLOUR = %s\n",
         PROCEDURE_CALL_COLOUR);
  fprintf (File_Pointer2, "GENERAL_STATEMENT_COLOUR = %s\n",
         GENERAL_STATEMENT_COLOUR);
  fprintf (File_Pointer2, "EDGE_COLOUR = %s\n", EDGE_COLOUR);
  fprintf (File_Pointer2, "GRID_COLOUR = %s\n", GRID_COLOUR);
  fprintf (File_Pointer2, "BOUNDARY_COLOUR = %s\n",
         BOUNDARY_COLOUR);

  fprintf (File_Pointer2, "DISPLAY_GRID = %d\n", DISPLAY_GRID);
  fprintf (File_Pointer2, "DISPLAY_COMPONENTS = %d\n",
         DISPLAY_COMPONENTS);
  fprintf (File_Pointer2, "DISPLAY_EDGES = %d\n", DISPLAY_EDGES);
  fprintf (File_Pointer2, "DISPLAY_BOUNDARIES = %d\n",
         DISPLAY_BOUNDARIES);
  fclose (File_Pointer2);
}

/*************************************************************
 * Function : Routine which interprets a configuration file.
 * The format of the config file is :
 *    VARIABLE = VALUE
 * The configuration file contains a list of variable assignments
 * ;where VARIABLE = DISPLAY_GRID | GRID_COLOUR | DISPLAY_EDGES, etc.
 *        VALUE = NUMERIC / COLOURSTRING   eg. 1, BLUE, GREEN, etc.
 *************************************************************/
void
Read_Configuration_File (String_Type File_Name)
{
  FILE *File_Pointer2;
  int i;
  String_Type Variable, Value;
  char Assignment[5];
  Set_Default_Values ();
  strcpy (Variable, "");
  strcpy (Assignment, "");
  strcpy (Value, "");
  File_Pointer2 = fopen (File_Name, "r");
  while (i != EOF)
    {
      i = fscanf (File_Pointer2, "%s%s%s", Variable, Assignment,
            Value);
```

```
      Set_Variable_Values (Variable, Value);
    }
  fclose (File_Pointer2);
}
```

D. "Diagram_Drawing_Routines.h"

```
/*****************************************************************
 * File : Diagram_Drawing_Routines.h                             *
 *****************************************************************/

/*****************************************************************
 * Function : Converts degrees into radians.
 *****************************************************************/
double
Radian_Equivalent (double Angle)
{
  return (Angle * 3.142 / 180);
}

/*****************************************************************
 * Function : Rotates a number of points by a specified angle,
 * around a specified location (X_Center, Y_Center).
 *****************************************************************/
void
Rotate_Points (XPoint * Points, int Number_Of_Points,
               int X_Center, int Y_Center, double Angle)
{
  XPoint Temporary_Point;
  int Index;
  double Point1, Point2;
  double Cos_Theta, Sin_Theta;
  Cos_Theta = cos (Radian_Equivalent (Angle));
  Sin_Theta = sin (Radian_Equivalent (Angle));
  for (Index = 0; Index < Number_Of_Points; Index++)
    {

      Temporary_Point.x = Points[Index].x - X_Center;
      Temporary_Point.y = Points[Index].y - Y_Center;
      Point1 =
      Temporary_Point.x * Cos_Theta -
      Temporary_Point.y * Sin_Theta + X_Center;
      Point2 =
      Temporary_Point.y * Cos_Theta +
      Temporary_Point.x * Sin_Theta + Y_Center;
      Points[Index].x = Point1;
      Points[Index].y = Point2;
    }
}

void Draw_XFIG_Rectangle(int X1, int Y1, int X2, int Y2)
{
  if (Output_To_FIG == TRUE)
    {
      fprintf (XFIG_File_Pointer, "2 2 0 1 -1 7 0 0 -1 0.000 0 0 0 0
0 5\n");
      fprintf (XFIG_File_Pointer, "         %d %d %d %d %d %d %d %d
%d %d\n",
               X1, Y1 , X2 , Y1 , X2 , Y2 , X1 , Y2 , X1 , Y1 );
    }
}
```

```
/*************************************************************
 * Function : Routine which displays a rectangle to the
 * drawing area.
 *************************************************************/
void
Draw_Rectangle (int X1, int Y1, int X2, int Y2, int FILLED)
{
  int Width, Height;

  Width = abs (X2 - X1);
  Height = abs (Y2 - Y1);
  if (FILLED == TRUE)
    XFillRectangle (XtDisplay (Drawing_Area), Users_Bitmap,
                The_Graphics_Context, X1, Y1, Width, Height);
  else
    XDrawRectangle (XtDisplay (Drawing_Area), Users_Bitmap,
                The_Graphics_Context, X1, Y1, Width, Height);
  Draw_XFIG_Rectangle(X1*XFIG_SCALEX, Y1*XFIG_SCALEY,
                  X2*XFIG_SCALEX, Y2*XFIG_SCALEY);

}

/*************************************************************
 * Function : Displays a line to the drawing area.
 *************************************************************/
void
Draw_Line (int X1, int Y1, int X2, int Y2)
{
  XDrawLine (XtDisplay (Drawing_Area), Users_Bitmap,
          The_Graphics_Context, X1, Y1, X2, Y2);
}

/*************************************************************
 * Function : Displays a series of lines to the drawing area.
 *************************************************************/
void
Draw_Lines (XPoint * Points, int Number_Of_Points)
{
  int Index;
  XDrawLines (XtDisplay (Drawing_Area), Users_Bitmap,
            The_Graphics_Context, Points, Number_Of_Points,
            CoordModeOrigin);
  if (Output_To_FIG == TRUE)
    {
      fprintf (XFIG_File_Pointer,
      "2 3 0 1 -1 7 0 0 -1 0.000 0 0 0 0 0 %d\n", Number_Of_Points +
1);
      for (Index = 0; Index < Number_Of_Points; Index++)
      fprintf (XFIG_File_Pointer, "%d %d ", Points[Index].x *
XFIG_SCALEX,
            Points[Index].y * XFIG_SCALEY);
      fprintf (XFIG_File_Pointer, "\n");
      fprintf (XFIG_File_Pointer,
      "     %d %d ", Points[0].x * XFIG_SCALEX, Points[0].y *
XFIG_SCALEY);
      fprintf (XFIG_File_Pointer,"\n");
```

```
    }
}

/*************************************************************
 * Function : Draws a circle to the drawing area. The X1, Y1,
 * X2, Y2 specify a rectangle in which the circle will be
 * drawn.
 *************************************************************/
void
Draw_Ellipse (int X1, int Y1, int X2, int Y2, int FILLED)
{
  int Width, Height;
  int XC, YC, RadiusX, RadiusY;
  Width = X2 - X1;
  Height = Y2 - Y1;
  if (FILLED == TRUE)
    XFillArc (XtDisplay (Drawing_Area), Users_Bitmap,
              The_Graphics_Context, X1, Y1,
              Width, Height, 0, 360 * 64);
  else
    XDrawArc (XtDisplay (Drawing_Area), Users_Bitmap,
              The_Graphics_Context, X1, Y1,
              Width, Height, 0, 360 * 64);
  XC = (X1 + (Width / 2)) * XFIG_SCALEX;
  YC = (Y1 + (Height / 2)) * XFIG_SCALEY;
  RadiusX = Width / 2 * XFIG_SCALEX;
  RadiusY = Height / 2 * XFIG_SCALEY;
  if (Output_To_FIG == TRUE)
    {
      fprintf (XFIG_File_Pointer,
      "1 2 0 1 -1 7 0 0 -1 0.0000000 1 0.000 %d %d %d %d 2475 2100
6450 3450\n",
       XC, YC, RadiusX, RadiusY);
    }
}

/*************************************************************
 * Function : Outputs a text string to a specified location .
 * Text is left aligned.
 *************************************************************/
void
Draw_Text (int X1, int Y1, char Text[80])
{
  XDrawImageString (XtDisplay (Drawing_Area), Users_Bitmap,
                    The_Graphics_Context, X1, Y1, Text,
                    strlen (Text));
  if (Output_To_FIG == TRUE)
    {
      fprintf (XFIG_File_Pointer,
      "4 0 -1 0 0 0 20 0.0000000 4 135 1320 %d %d %s\\001\n",
      X1 * XFIG_SCALEX, Y1 * XFIG_SCALEY, Text);
    }

}

/*************************************************************
 * Function :  Outputs a text string to a specified location.
```

```
 * Text is centered; where X1, Y1 is the center.
 ************************************************************/
void
Draw_Text_Centered (int X1, int Y1, char Text[80])
{
  int X_Center, Y_Center;
  X_Center = X1 - Font_Width (Text) / 2;
  Y_Center = Y1 + Font_Height (Text) / 4;
  XDrawImageString (XtDisplay (Drawing_Area), Users_Bitmap,
                The_Graphics_Context, X_Center, Y_Center,
                Text, strlen (Text));
  if (Output_To_FIG == TRUE)
    {
      fprintf (XFIG_File_Pointer,
            "4 1 -1 0 0 0 20 0.0000000 4 135 1320 %d %d %s\\001\n",
            X1 * XFIG_SCALEX,
            (Y1 + Font_Height (Text) / 4) * XFIG_SCALEY, Text);
    }
}

/************************************************************
 * Function : Accepts a number of points as input and draws
 * a polygon which is filled or not filled.
 ************************************************************/
void
Draw_Polygon (XPoint * Points, int Number_Of_Points, int FILLED)
{
  if (FILLED == TRUE)
    XFillPolygon (XtDisplay (Drawing_Area), Users_Bitmap,
              The_Graphics_Context,
              Points, Number_Of_Points,
              Convex, CoordModeOrigin);
  else
    Draw_Lines (Points, Number_Of_Points);

}

/************************************************************
 * Function : Redraws the users bitmap back onto the screen
 ************************************************************/
void
Redraw_Bitmap ()
{
  XCopyArea (XtDisplay (Drawing_Area),
           Users_Bitmap,
           XtWindow (Drawing_Area),
           The_Graphics_Context, 0, 0,
           Screen_Width, Screen_Height, 0, 0);
}
```

E. "Node_Operations.h"

```
/*****************************************************************
 * File : Node_Operations.h                                      *
 *****************************************************************/

/*****************************************************************
 * Defines an EDGE table.
 *****************************************************************/
struct edge
  {
    struct Node *Node1;
    struct Node *Node2;
    int Arrow_Style;
    struct edge *Next;
  };
typedef struct edge EDGE;

/*****************************************************************
 * Defines a node structure.
 *****************************************************************/
struct Node
  {
    int ID;
    String_Type Statement, Colour;
    int X, Y, Type, Expanded, Width1, Width2, Height1, Height2, DX,
DY;
    struct Node *Sibling, *Child, *Parent;
    EDGE *Edge;
  };
typedef struct Node NODE;

/*****************************************************************
 * Defines a list of nodes.
 *****************************************************************/
struct Node_List
  {
    NODE *Node_Pointer;
    struct Node_List *Next;
  };
typedef struct Node_List NODE_LIST;

/*****************************************************************/
NODE *Found_Value;
int Number_Of_Nodes;
int MINY, MINX, MAXY, MAXX;

/*****************************************************************
 * Function : Removes Node list from memory (Memory management
 * routine).
 *****************************************************************/
Destroy_Node_List (NODE_LIST * Current_Node)
{
  NODE_LIST *Cursor, *Next;
  Cursor = Current_Node;
  if (Cursor != NULL)
```

```
    {
      Next = Cursor->Next;
      while (Cursor != NULL)
      {
        free (Cursor);
        Cursor = Next;
        if (Cursor != NULL)
          Next = Cursor->Next;
      }
    }
}

/*****************************************************************
 * Function : Defines a Node List and fills it with the child
 * nodes of the Current_Node.
 *****************************************************************/
NODE_LIST *
Create_Node_List (NODE * Current_Node, NODE * Reference_Node)
{
  NODE_LIST *Temporary_Node, *The_List, *Last_Node;
  NODE *Cursor;

  Cursor = Current_Node->Child;
  if (Cursor != NULL)
    {
      The_List = malloc (sizeof (NODE_LIST));
      The_List->Node_Pointer = Cursor;
      Last_Node = The_List;

      if (Cursor != NULL)
      {
        Cursor = Cursor->Sibling;
        while (Cursor != NULL)
          {
            Temporary_Node = malloc (sizeof (NODE_LIST));
            Temporary_Node->Node_Pointer = Cursor;
            Last_Node->Next = Temporary_Node;
            Last_Node = Temporary_Node;
            Cursor = Cursor->Sibling;
          }
        Temporary_Node = malloc (sizeof (NODE_LIST));
        Temporary_Node->Node_Pointer = Reference_Node;
        Last_Node->Next = Temporary_Node;
        Last_Node = Temporary_Node;
        Last_Node->Next = NULL;
      }
    }
  else
    return NULL;
  return The_List;
}

/*****************************************************************
 * Function : Determines the maximum extent for the Current
 * node.
 *****************************************************************/
int
```

```
Maximum_X_Value (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      if ((Current_Node->X + Current_Node->Width2) > MAXX)
      MAXX = Current_Node->X + Current_Node->Width2;
      Maximum_X_Value (Current_Node->Child);
      Maximum_X_Value (Current_Node->Sibling);
      return MAXX;
    }
}

/*************************************************************
 * Function : Determines the maximum
 *    extent for the Current_Node
 *************************************************************/
int
Maximum_Y_Value (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      if ((Current_Node->Y +
         Current_Node->Height2) > MAXY)
      MAXY = Current_Node->Y + Current_Node->Height2;
      Maximum_Y_Value (Current_Node->Child);
      Maximum_Y_Value (Current_Node->Sibling);
      return MAXY;
    }
}
/*************************************************************
 * Function : Determines the extent for the current Node.
 *************************************************************/
int
Minimum_X_Value (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      if ((Current_Node->X - Current_Node->Width1) < MINX)
      MINX = Current_Node->X - Current_Node->Width1;
      Minimum_X_Value (Current_Node->Child);
      Minimum_X_Value (Current_Node->Sibling);
      return MINX;
    }
}

/*************************************************************
 * Function : Determines the extent of the Current_Node.
 *************************************************************/
int
Minimum_Y_Value (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      if ((Current_Node->Y - Current_Node->Height1) < MINY)
      MINY = Current_Node->Y - Current_Node->Height1;
      Minimum_Y_Value (Current_Node->Child);
      Minimum_Y_Value (Current_Node->Sibling);
```

```
      return MINY;
    }
}

/*****************************************************************
 * Function : Adds an edge to the "Current_Node".
 *****************************************************************/
void
Add_Edge_To_Node (NODE * Current_Node,
                  NODE * Node1,
                  NODE * Node2,
                  int Arrow_Style)
{
  EDGE *Cursor, *Temporary_Edge;

  Temporary_Edge = malloc (sizeof (EDGE));
  Temporary_Edge->Node1 = Node1;
  Temporary_Edge->Node2 = Node2;
  Temporary_Edge->Arrow_Style = Arrow_Style;

  if (Current_Node->Child->Edge == NULL)
    Current_Node->Child->Edge = Temporary_Edge;
  else
    {
      Cursor = Current_Node->Child->Edge;
      while (Cursor->Next != NULL)
      Cursor = Cursor->Next;
      Cursor->Next = Temporary_Edge;
    }
}
/*****************************************************************
 * Function : Determines the statement type and returns an
 * identifier for the selected statement.
 *****************************************************************/
int
Statement_Type (String_Type Statement)
{
  int Index;
  String_Type Temporary_String;
  strcpy (Temporary_String, Statement);
  for (Index = 0; Index != MAXIMUM_STRING_LENGTH; Index++)
    Temporary_String[Index] = toupper (Statement[Index]);
  if (strncmp (Temporary_String, "IF", 2) == 0)
    return KEYWORD_IF;
  else if (strncmp (Temporary_String, "ELSE", 4) == 0)
    return KEYWORD_ELSE;
  else if (strncmp (Temporary_String, "FOR", 3) == 0)
    return KEYWORD_FOR;
  else if (strncmp (Temporary_String, "WHILE", 5) == 0)
    return KEYWORD_WHILE;
  else if (strncmp (Temporary_String, "UNTIL", 5) == 0)
    return KEYWORD_UNTIL;
  else if (strncmp (Temporary_String, "REPEAT", 6) == 0)
    return KEYWORD_REPEAT;
  else if (strncmp (Temporary_String, "BEGIN", 5) == 0)
    return KEYWORD_BEGIN;
  else if (strncmp (Temporary_String, "END", 3) == 0)
```

```
    return KEYWORD_END;
  else if (strncmp (Temporary_String, "PROCEDURE", 9) == 0)
    return KEYWORD_PROCEDURE;
  else if (strncmp (Temporary_String, "PROGRAM", 7) == 0)
    return KEYWORD_PROGRAM;
  else if (strncmp (Temporary_String, "CASE", 4) == 0)
    return KEYWORD_CASE;
  else
    return KEYWORD_STATEMENT;
  return 0;
}

/*************************************************************
 * Function : Search routine which returns a pointer to the
 * node that is being searched for. Search is based on the
 * nodes ID.
 *************************************************************/
NODE *
Locate_Node (int Node_ID, NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      if (Current_Node->ID == Node_ID)
      {
        Found_Value = Current_Node;
        return (Current_Node);
      }
      else
      {
        if (Current_Node != NULL)
          {
            Locate_Node (Node_ID, Current_Node->Child);
            Locate_Node (Node_ID, Current_Node->Sibling);
          }
      }
    }
  return Found_Value;
}

/*************************************************************
 * Function : Adds a child node to a parent node.
 *************************************************************/
void
Add_Child_Node (NODE * The_Diagram, int ParentID, NODE * Child)
{
  NODE *Current_Node;
  NODE *Parent_Node;

  Current_Node = NULL;
  Current_Node = Locate_Node (ParentID, The_Diagram);
  Parent_Node = Current_Node;

  if (Current_Node->Child == NULL)
    {
      Current_Node->Child = Child;
      Child->Parent = Current_Node;
    }
```

```
  else
    {
      Current_Node = Current_Node->Child;
      while (Current_Node->Sibling != NULL)
      {
        Current_Node = Current_Node->Sibling;
      }
      Current_Node->Sibling = Child;
      Child->Parent = Parent_Node;
    };
}
NODE *SearchNode;

/*************************************************************
 * Function : A coordinate is given (X, Y). This routine
 * determines which node was selected based on the coordinate.
 *************************************************************/
Node_Selected (int X, int Y, NODE * Current_Node)
{
  NODE *Cursor;
  if (Current_Node != NULL)
    {
      Cursor = Current_Node->Child;
      while (Cursor != NULL)
      {
        if ((X >= Cursor->X - Cursor->Width1) &&
            (X <= Cursor->X + Cursor->Width2) &&
            (Y >= Cursor->Y - Cursor->Height2) &&
            (Y <= Cursor->Y + Cursor->Height1))
          {
            FOUND = TRUE;
            SearchNode = Cursor;
            if (Cursor->Expanded == TRUE)
            Node_Selected (X, Y, Cursor);
          }
        Cursor = Cursor->Sibling;
      }
    }
}

/*************************************************************
 * Function : Allocates memory and initializes node information.
 *************************************************************/
NODE *
Create_Node (String_Type Statement, int Type)
{

  NODE *Temporary_Node;
  Temporary_Node = malloc (sizeof (NODE));
  Temporary_Node->ID = Number_Of_Nodes++;

  Temporary_Node->Width1 = Temporary_Node->Width2 =
    Temporary_Node->Height1 = Temporary_Node->Height2 =
    Temporary_Node->DX = Temporary_Node->DY =
    Temporary_Node->X = Temporary_Node->Y = 0;

  Temporary_Node->Expanded = TRUE;
```

```
  Temporary_Node->Sibling = Temporary_Node->Child =
    Temporary_Node->Parent = NULL;
  strcpy (Temporary_Node->Colour, "BLACK");
  if (Type == KEYWORD_UNKNOWN)
    Temporary_Node->Type = Statement_Type (Statement);
  else
    Temporary_Node->Type = Type;
  strcpy (Temporary_Node->Statement, Statement);

  return (Temporary_Node);
}


/*************************************************************
 * Function : Prints details about the entire data structure
 * starting from the Current_Node pointer.
 *************************************************************/
void
Print_Tree_Details (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      printf ("(ID=%d TYPE=%d", Current_Node->ID, Current_Node-
>Type);
      if (Current_Node->Parent != NULL)
      printf (" PID = %d      X=%d Y=%d) ",
            Current_Node->Parent->ID,
            Current_Node->X, Current_Node->Y);
      else
      printf ("   X=%d Y=%d) ", Current_Node->X, Current_Node->Y);
      printf ("[%s]\n", Current_Node->Statement);

      Print_Tree_Details (Current_Node->Child);
      Print_Tree_Details (Current_Node->Sibling);
    }
}
/*************************************************************
 * Function : Frees memory (Memory management routine)
 *************************************************************/
Destroy_Edges (EDGE * CurrentEdge)
{
  EDGE *Cursor, *Next;
  Cursor = CurrentEdge;
  if (Cursor != NULL)
    {
      Next = Cursor->Next;
      while (Cursor != NULL)
      {
        free (Cursor);
        Cursor = Next;
        if (Cursor != NULL)
          Next = Cursor->Next;
      }
    }
}

/*************************************************************
```

```
 * Function : Destroys the flowchart starting from the
 * Current_Node. (Memory management routine).
 ************************************************************/
Destroy_Tree (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      Destroy_Tree (Current_Node->Child);
      Destroy_Tree (Current_Node->Sibling);
      Destroy_Edges (Current_Node->Edge);
      free (Current_Node);
    }
}

/************************************************************
 * Function : Sets the EXPANDED attribute to false on
 * all nodes which contain child nodes.
 ************************************************************/
void
Collapse_All_Nodes (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      Collapse_All_Nodes (Current_Node->Child);
      Collapse_All_Nodes (Current_Node->Sibling);
      switch (Current_Node->Type)
      {
      case (KEYWORD_STATEMENT_BLOCK):
      case (KEYWORD_CASE):
      case (KEYWORD_FOR):
      case (KEYWORD_BEGIN):
      case (KEYWORD_IF):
      case (KEYWORD_WHILE):
      case (KEYWORD_REPEAT):
        Current_Node->Expanded = FALSE;
        break;
      }
    }
}
/************************************************************
 * Function : Sets the EXPANDED attribute to true.
 * This makes the drawing routines display the entire
 * flowchart starting from the Current_Node.
 ************************************************************/
void
Expand_All_Nodes (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      Expand_All_Nodes (Current_Node->Child);
      Expand_All_Nodes (Current_Node->Sibling);
      Current_Node->Expanded = TRUE;
    }
}
```

F. "File_Reading_Routines.h"

```
/*****************************************************************
 * File : File_Reading_Routines.h                                *
 *****************************************************************/

FILE *File_Pointer;
int File_Index;
String_Type Statement, Buffer;
char String_Value[2];
int The_Buffer_Is_Filled;

/*****************************************************************
 * Function : Wipes leading ' ' and tab characters from a
 * statement.
 *****************************************************************/
void
Remove_Leading_Blanks (String_Type Statement)
{
  int Index, Index2;
  Index = 0;

  while ((Statement[Index] == ' ') || (Statement[Index] == '\t'))
    Index++;
  for (Index2 = Index; Index2 < 79; Index2++)
    {
      Buffer[Index2 - Index] = Statement[Index2];
    }
}

/*****************************************************************
 * Function : Reads a statement from a file.
 *****************************************************************/
void
Read_From_File ()
{
  int Statement_Read;
  Statement_Read = FALSE;
  File_Index = fgetc (File_Pointer);
  if (File_Index != EOF)
    {
      if (File_Index == '{')
      {
        while (fgetc (File_Pointer) != '}');
      }
      else
      {
        String_Value[0] = File_Index;
        String_Value[1] = '\0';
        strcat (Statement, String_Value);
      }
      while ((File_Index != EOF) && (Statement_Read == FALSE))
      {
        File_Index = fgetc (File_Pointer);
        if (File_Index == '{')
          {
```

```
            while (fgetc (File_Pointer) != '}');
          }
        else
          {
            if (File_Index != '\n')
            {
              if (File_Index != '\t')
                {
                  String_Value[0] = File_Index;
                  String_Value[1] = '\0';
                  strcat (Statement, String_Value);
                }
            }
            else
            {
              if (strcmp (Statement, "") != 0)
                {
                  Remove_Leading_Blanks (Statement);
                  Statement_Read = TRUE;
                  The_Buffer_Is_Filled = TRUE;
                }
              strcpy (Statement, "");
            }
          }
      }
  }
}

/*************************************************************
 * Function : Generates the structure required for the
 * diagram. This routine converts the source code file
 * into the data structure so that it can be given
 * geometric information.
 *************************************************************/
int
Read_Statement_Into_Buffer (int PID, int BufferFilled, NODE *
The_Diagram)
{
  NODE *Temporary_Node, *Dummy_Node;
  int Index, Case_Label;
  String_Type Label_Text, Temporary_String;
  if (File_Index != EOF)
    {
      The_Buffer_Is_Filled = BufferFilled;
      if (BufferFilled == FALSE)
      Read_From_File ();
      if (The_Buffer_Is_Filled == TRUE)
      {
        Temporary_Node = Create_Node (Buffer, KEYWORD_UNKNOWN);
        Add_Child_Node (The_Diagram, PID, Temporary_Node);
        switch (Statement_Type (Buffer))
          {
          case KEYWORD_IF:
            Read_Statement_Into_Buffer (Temporary_Node->ID,
                                        FALSE, The_Diagram);
            Read_From_File ();
            if (Statement_Type (Buffer) == KEYWORD_ELSE)
```

```
            Read_Statement_Into_Buffer (Temporary_Node->ID,
                                        FALSE, The_Diagram);
            break;
          case KEYWORD_PROCEDURE:
            Read_From_File ();
            while (Statement_Type (Buffer) != KEYWORD_END)
            {
              Read_Statement_Into_Buffer (Temporary_Node->ID,
                                          TRUE, The_Diagram);
              Read_From_File ();
            }
            break;
          case KEYWORD_PROGRAM:
            Read_From_File ();
            while (Statement_Type (Buffer) != KEYWORD_END)
            {
              Read_Statement_Into_Buffer (Temporary_Node->ID,
                                          TRUE, The_Diagram);
              Read_From_File ();
            }
            break;
          case KEYWORD_BEGIN:
            Read_From_File ();
            while (Statement_Type (Buffer) != KEYWORD_END)
            {
              Read_Statement_Into_Buffer (Temporary_Node->ID,
                                          TRUE, The_Diagram);
              Read_From_File ();
            }
            break;
          case KEYWORD_FOR:
            Read_Statement_Into_Buffer (Temporary_Node->ID,
                                        FALSE, The_Diagram);
            break;
          case KEYWORD_WHILE:
            Read_Statement_Into_Buffer (Temporary_Node->ID,
                                        FALSE, The_Diagram);
            break;
          case KEYWORD_REPEAT:
            Dummy_Node = Create_Node ("BEGIN", KEYWORD_UNKNOWN);
            Add_Child_Node (The_Diagram, Temporary_Node->ID,
Dummy_Node);
            Read_From_File ();
            while (Statement_Type (Buffer) != KEYWORD_UNTIL)
            {
              Read_Statement_Into_Buffer (Dummy_Node->ID,
                                          TRUE, The_Diagram);
              Read_From_File ();
            }
            if (Statement_Type (Buffer) == KEYWORD_UNTIL)
            strcat (Temporary_Node->Statement, Buffer);
            break;
          case KEYWORD_CASE:
            Read_From_File ();
            while (Statement_Type (Buffer) != KEYWORD_END)
            {
              Case_Label = 0;
```

```
              for (Index = 0; Index < strlen (Buffer); Index++)
                if (Buffer[Index] == ':')
                  {
                  Case_Label = Index;
                  }
              if (Case_Label != 0)
                {
                  sscanf (Buffer, "%s", Label_Text);
                  Dummy_Node = Create_Node (Label_Text,
KEYWORD_BEGIN);
                  Add_Child_Node (The_Diagram,
                              Temporary_Node->ID,
                              Dummy_Node);
                  if (strlen (Buffer) > Case_Label)
                  {
                    strcpy (Temporary_String, &Buffer[Case_Label +
1]);
                    Remove_Leading_Blanks (Temporary_String);
                    Read_Statement_Into_Buffer (Dummy_Node->ID, TRUE,
                                        The_Diagram);
                  }
                  else
                  Read_Statement_Into_Buffer (Dummy_Node->ID, TRUE,
                                      The_Diagram);
                }
              else
                {
                  Dummy_Node = Create_Node (Label_Text,
KEYWORD_BEGIN);
                  Read_Statement_Into_Buffer (Temporary_Node->ID,
TRUE,
                                        The_Diagram);
                }
              Read_From_File ();
            }
          default:
            break;
          }
      }
    }
}

/*************************************************************
 * Function : Reads an entire source code file until the EOF
 * character is encountered.
 *************************************************************/
void
Parse_Source_Code_File (String_Type File_Name, NODE * The_Diagram)
{
  File_Pointer = fopen (File_Name, "r");
  strcpy (Statement, "");
  strcpy (Buffer, "");

  while (File_Index != EOF)
    {
      Read_Statement_Into_Buffer (The_Diagram->ID, FALSE,
The_Diagram);
```

```
    }
  fclose (File_Pointer);
}
```

G. "Geometry_Operations.h"

```
/*************************************************************
 * File : Geometry_Operations.h                              *
 *************************************************************/

/*************************************************************
 * Function : Calculates the maximum value from two variables
 * a and b.
 *************************************************************/
int
Maximum_Value (int a, int b)
{
  if (a > b)
    return a;
  else
    return b;
}

/*************************************************************
 * Function : For one given node (Current_Node), this routine
 * counts the number of child nodes that exist.
 *************************************************************/
int
Number_Of_Children (NODE * Current_Node)
{
  NODE *Cursor;
  int Counter;
  Counter = 0;
  Cursor = Current_Node;
  if (Cursor->Child == NULL)
    return 0;
  else
    {
      Cursor = Cursor->Child;
      while (Cursor != NULL)
      {
        Counter++;
        Cursor = Cursor->Sibling;
      }

    }
  return Counter;
}

/*************************************************************
 * Function : Layout routine which generates coordinates for
 * a BLOCK of statements.
 *************************************************************/
void
Create_BLOCK_Component_Layout (NODE * Current_Node)
{
  NODE *Cursor;
  int X, Y;
  int Index;
  X = Current_Node->X;
```

```
  Y = Current_Node->Y;
  Index = 0;
  Cursor = Current_Node->Child;
  while (Cursor != NULL)
    {
      Cursor->X = X;
      Cursor->Y = Y - Index;
      Index++;
      Cursor = Cursor->Sibling;
    }
  Cursor = Current_Node->Child;
  while (Cursor->Sibling != NULL)
    {
      Add_Edge_To_Node (Current_Node, Cursor, Cursor->Sibling,
                SINGLE_ARROWHEAD1);
      Cursor = Cursor->Sibling;
    }
}

/*************************************************************
 * Function : Layout routine which generates coordinates for
 * an IF statement .
 *************************************************************/
void
Create_IF_Component_Layout (NODE * Current_Node)
{
  NODE *Cursor, *A, *B;
  NODE *a, *b, *c, *d ;
  int X, Y;
  String_Type Temporary_String, If_Conditionition;
  X = Current_Node->X;
  Y = Current_Node->Y;
  A = Current_Node->Child;
  B = A->Sibling;
  if (B == NULL)
    {
      /* This section caters for an if with no else statement */
      B = Create_Node ("B", KEYWORD_DUMMY_NODE);
      A->Sibling = B;
    }

  sscanf (Current_Node->Statement, "%s%s", &Temporary_String,
          &If_Conditionition);

  a = Create_Node (If_Conditionition, IF_SYMBOL);
  b = Create_Node ("FALSE", LABEL_SYMBOL);
  c = Create_Node ("c", KEYWORD_DUMMY_NODE);
  d = Create_Node ("d", KEYWORD_DUMMY_NODE);

  B->Sibling = a;
  a->Sibling = b;
  b->Sibling = c;
  c->Sibling = d;
  d->Sibling = NULL;

  B->Parent = a->Parent = b->Parent = c->Parent =
    d->Parent = Current_Node;
```

```
  A->X = a->X = c->X = X;
  B->X = b->X = d->X = X + 1;

  a->Y = b->Y = Y;
  A->Y = B->Y = Y - 1;
  c->Y = d->Y = Y - 2;

  Cursor = Current_Node;
  if (B->Type == KEYWORD_DUMMY_NODE)
    {
      Add_Edge_To_Node (Cursor, b, B, NO_ARROWHEAD);
    }
  else
    Add_Edge_To_Node (Cursor, b, B, SINGLE_ARROWHEAD1);

  Add_Edge_To_Node (Cursor, a, b, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, c, d, NO_ARROWHEAD);

  Add_Edge_To_Node (Cursor, a, A, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, A, c, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, B, d, SINGLE_ARROWHEAD1);

}

/*************************************************************
 * Function : Layout routine which generates coordinates for
 * an FOR statement .
 *************************************************************/
void
Create_FOR_Component_Layout (NODE * Current_Node)
{

  NODE *Cursor, *A;
  NODE *a, *b, *c, *d, *e, *f, *g, *h, *i, *j, *k, *l, *m;
  int X, Y;
  String_Type For, Variable, Assignment, Initializer_Value, To,
Final_Value,
    Initializer, Condition, Increment;
  X = Current_Node->X;
  Y = Current_Node->Y;
  strcpy (Initializer, "");
  strcpy (Condition, "");
  strcpy (Increment, "");

  sscanf (Current_Node->Statement, "%s %s %s %s %s %s", &For,
&Variable,
       &Assignment, &Initializer_Value, &To, &Final_Value);

  strcat (Initializer, Variable);
  strcat (Initializer, " := ");
  strcat (Initializer, Initializer_Value);

  strcat (Condition, Variable);
  strcat (Condition, " < ");
  strcat (Condition, Final_Value);
```

```
strcat (Increment, Variable);
strcat (Increment, " = ");
strcat (Increment, Variable);
strcat (Increment, " + 1");
A = Current_Node->Child;


a = Create_Node (Initializer, KEYWORD_GENERAL);
b = Create_Node ("b", KEYWORD_DUMMY_NODE);
c = Create_Node ("c", KEYWORD_DUMMY_NODE);
d = Create_Node (Condition, IF_SYMBOL);
e = Create_Node ("TRUE", LABEL_SYMBOL);
f = Create_Node ("f", KEYWORD_DUMMY_NODE);
g = Create_Node ("g", KEYWORD_DUMMY_NODE);
h = Create_Node ("h", KEYWORD_DUMMY_NODE);
i = Create_Node (Increment, KEYWORD_GENERAL);
j = Create_Node ("j", KEYWORD_DUMMY_NODE);
k = Create_Node ("k", KEYWORD_DUMMY_NODE);
l = Create_Node ("l", KEYWORD_DUMMY_NODE);
m = Create_Node ("m", KEYWORD_DUMMY_NODE);

A->Sibling = a;
a->Sibling = b;
b->Sibling = c;
c->Sibling = d;
d->Sibling = e;
e->Sibling = f;
f->Sibling = g;
g->Sibling = h;
h->Sibling = i;
i->Sibling = j;
j->Sibling = k;
k->Sibling = l;
l->Sibling = m;
m->Sibling = NULL;

A->Parent = a->Parent =
b->Parent = c->Parent = d->Parent = e->Parent =
f->Parent = g->Parent = h->Parent = i->Parent =
j->Parent = k->Parent = l->Parent = m->Parent =
Current_Node;

a->X = b->X = d->X = f->X = h->X = k->X = X;
e->X = A->X = i->X = l->X = X + 1;
c->X = g->X = j->X = m->X = X + 2;

a->Y = Y;
b->Y = c->Y = Y-1;
d->Y = e->Y = Y-2;
f->Y = A->Y = g->Y = Y-3;
h->Y = i->Y = j->Y = Y-4;
k->Y = l->Y = m->Y = Y-5;

Cursor = Current_Node;


Add_Edge_To_Node (Cursor, a, b, NO_ARROWHEAD);
```

```
  Add_Edge_To_Node (Cursor, d, f, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, f, h, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, h, k, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, d, e, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, l, m, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, m, j, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, j, g, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, g, c, NO_ARROWHEAD);

  Add_Edge_To_Node (Cursor, b, d, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, e, A, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, A, i, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, i, l, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, c, b, SINGLE_ARROWHEAD1);
}

/*************************************************************
 * Function : Layout routine which generates coordinates for
 * an WHILE statement .
 *************************************************************/
void
Create_WHILE_Component_Layout (NODE * Current_Node)
{
  NODE *Cursor, *A;
  NODE *a, *b, *c, *d, *e, *f, *g, *h, *i;
  String_Type Temporary_String, While;
  int X, Y;

  X = Current_Node->X;
  Y = Current_Node->Y;
  sscanf (Current_Node->Statement, "%s %s", &While,
&Temporary_String);

  A = Current_Node->Child;
  a = Create_Node ("a", KEYWORD_DUMMY_NODE);
  b = Create_Node ("b", KEYWORD_DUMMY_NODE);
  c = Create_Node (Temporary_String, IF_SYMBOL);
  d = Create_Node ("TRUE", LABEL_SYMBOL);
  e = Create_Node ("e", KEYWORD_DUMMY_NODE);
  f = Create_Node ("f", KEYWORD_DUMMY_NODE);
  g = Create_Node ("g", KEYWORD_DUMMY_NODE);
  h = Create_Node ("h", KEYWORD_DUMMY_NODE);
  i = Create_Node ("i", KEYWORD_DUMMY_NODE);

  A->Sibling = a;
  a->Sibling = b;
  b->Sibling = c;
  c->Sibling = d;
  d->Sibling = e;
  e->Sibling = f;
  f->Sibling = g;
  g->Sibling = h;
  h->Sibling = i;
  i->Sibling = NULL;

  a->Parent = b->Parent = c->Parent =
  d->Parent = e->Parent = f->Parent =
```

```
  g->Parent = h->Parent = i->Parent = Current_Node;

  a->X = c->X = e->X = g->X = X;
  d->X = A->X = h->X = X + 1;
  b->X = f->X = i->X = X + 2;

  a->Y = b->Y = Y;
  c->Y = d->Y = Y - 1;
  e->Y = A->Y = f->Y = Y - 2;
  g->Y = h->Y = i->Y = Y - 3;

  Cursor = Current_Node;

  Add_Edge_To_Node (Cursor, c, e, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, e, g, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, h, i, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, i, f, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, f, b, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, c, d, NO_ARROWHEAD);

  Add_Edge_To_Node (Cursor, a, c, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, d, A, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, A, h, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, b, a, SINGLE_ARROWHEAD1);
}

/*****************************************************************
 * Function : Layout routine which generates coordinates for
 * an REPEAT statement .
 *****************************************************************/
void
Create_REPEAT_Component_Layout (NODE * Current_Node)
{
  NODE *Cursor, *A;
  NODE *a, *b, *c, *d, *e;
  String_Type Temporary_String;
  int X, Y;
  int Index;
  X = Current_Node->X;
  Y = Current_Node->Y;

  for (Index = 5; Index < MAXIMUM_STRING_LENGTH - 1; Index++)
    Temporary_String[Index - 5] = Current_Node->Statement[Index];
  A = Current_Node->Child;
  a = Create_Node ("a", KEYWORD_DUMMY_NODE);
  b = Create_Node ("b", KEYWORD_DUMMY_NODE);
  c = Create_Node ("c", KEYWORD_DUMMY_NODE);
  d = Create_Node (&Temporary_String[7], IF_SYMBOL);  /*IF SYMBOL */
  e = Create_Node ("FALSE", LABEL_SYMBOL);

  A->Sibling = a;
  a->Sibling = b;
  b->Sibling = c;
  c->Sibling = d;
  d->Sibling = e;
  e->Sibling = NULL;
```

```
  a->Parent = b->Parent =
  c->Parent = d->Parent = e->Parent = Current_Node;

  a->X = A->X = d->X = X;
  b->X = c->X = e->X = X + 1;

  a->Y = b->Y = Y;
  A->Y = c->Y = Y - 1;
  d->Y = e->Y = Y - 2;

  Cursor = Current_Node;

  Add_Edge_To_Node (Cursor, e, c, NO_ARROWHEAD);
  Add_Edge_To_Node (Cursor, c, b, NO_ARROWHEAD);

  Add_Edge_To_Node (Cursor, a, A, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, A, d, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, b, a, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Cursor, d, e, SINGLE_ARROWHEAD1);
}

/*****************************************************************
 * Function : Layout routine which generates coordinates for
 * a CASE  statement .
 *****************************************************************/
void
Create_CASE_Component_Layout (NODE * Current_Node)
{

  NODE *Cursor, *A;
  NODE *a, *b, *c, *b_temp, *A_temp, *c_temp, *Last_A, *Last_b,
*Last_c;

  int X, Y;
  int Index, Number_Of_Children_Count;
  X = Current_Node->X;
  Y = Current_Node->Y;

  Number_Of_Children_Count = Number_Of_Children (Current_Node);
  Cursor = Current_Node->Child;
  A = Cursor;
  b = Create_Node (Cursor->Statement, IF_SYMBOL);
  a = Create_Node (Current_Node->Statement, CASE_SYMBOL);
  c = Create_Node ("c", KEYWORD_DUMMY_NODE);

  A->X = b->X = a->X = c->X = X;
  A->Y = Y - 2;
  b->Y = Y - 1;
  a->Y = Y;
  c->Y = Y - 3;
  Last_b = b;
  Last_c = c;
  Last_A = A;

  Cursor = Cursor->Sibling;
  for (Index = 1; Index != Number_Of_Children_Count; Index++)
    {
```

```
      b_temp = Create_Node (Cursor->Statement, IF_SYMBOL);
      c_temp = Create_Node ("c", KEYWORD_DUMMY_NODE);
      A_temp = Cursor;
      b_temp->X = c_temp->X = A_temp->X = X + Index;
      b_temp->Y = Y - 1;
      c_temp->Y = Y - 3;
      A_temp->Y = Y - 2;

      Last_b->Sibling = b_temp;
      Last_c->Sibling = c_temp;
      Last_A = A_temp;
      Last_b = b_temp;
      Last_c = c_temp;
      Cursor = Cursor->Sibling;
    }

  Last_A->Sibling = b;
  Last_b->Sibling = a;
  a->Sibling = c;
  Last_c->Sibling = NULL;

  Add_Edge_To_Node (Current_Node, a, b, SINGLE_ARROWHEAD1);
  b_temp = b;
  A_temp = A;
  c_temp = c;
  for (Index = 0; Index != Number_Of_Children_Count - 1; Index++)
    {
      if (b_temp->Sibling != NULL)
      Add_Edge_To_Node (Current_Node, b_temp,
                        b_temp->Sibling, SINGLE_ARROWHEAD1);
      Add_Edge_To_Node (Current_Node, b_temp, A_temp,
SINGLE_ARROWHEAD1);
      Add_Edge_To_Node (Current_Node, A_temp, c_temp,
SINGLE_ARROWHEAD1);
      if (c_temp->Sibling != NULL)
      Add_Edge_To_Node (Current_Node, c_temp->Sibling,
                        c_temp, NO_ARROWHEAD);
      b_temp = b_temp->Sibling;
      A_temp = A_temp->Sibling;
      c_temp = c_temp->Sibling;
    }
  Add_Edge_To_Node (Current_Node, Last_b, Last_A, SINGLE_ARROWHEAD1);
  Add_Edge_To_Node (Current_Node, Last_A, Last_c, SINGLE_ARROWHEAD1);

}

/*************************************************************
 * Creates a Hierarchical diagrams layout
 *************************************************************/

void
Create_Hierarchy_Diagram_Layout (NODE * Current_Node)
{
  int Number_Of_Children_Count;
  int Index;
  NODE *Temporary_Node, *Cursor, *Cursor2, *Cursor3, *Last_Node;
  int X, Y;
```

```
if (Current_Node != NULL)
{
  X = Current_Node->X;
  Y = Current_Node->Y;

  if (Current_Node->Child != NULL)
    {
      Number_Of_Children_Count = Number_Of_Children (Current_Node);
      Temporary_Node = Create_Node (Current_Node->Statement,
                                    CASE_SYMBOL);
      Cursor = Current_Node->Child;
      Index = 0;
      while (Cursor != NULL)
        {
          Cursor->Y = Y - Index;
          Cursor->X = X + 2;
          Index++;
          Cursor = Cursor->Sibling;
        }
      Temporary_Node->X = X;
      Temporary_Node->Y = Y;
      Cursor3 = Temporary_Node;
      Add_Child_Node (Current_Node, Current_Node->ID,
Temporary_Node);
      Cursor2 = Current_Node->Child;
      Last_Node = NULL;
      if (Number_Of_Children_Count != 1)
      {
      for (Index = 0; Index != Number_Of_Children_Count; Index++)
        {
          Temporary_Node = Create_Node ("", KEYWORD_DUMMY_NODE);
          Temporary_Node->Y = Y - Index;
          Temporary_Node->X = X + 1;
          if (Index == 0)
            Add_Edge_To_Node (Current_Node, Cursor3,
                              Temporary_Node, NO_ARROWHEAD);
          Add_Child_Node (Current_Node, Current_Node->ID,
Temporary_Node);
          if (Last_Node != NULL)
            Add_Edge_To_Node (Current_Node,
                              Temporary_Node, Last_Node,
NO_ARROWHEAD);
          Last_Node = Temporary_Node;
          Add_Edge_To_Node (Current_Node,
                            Temporary_Node, Cursor2,
SINGLE_ARROWHEAD1);
          Cursor2 = Cursor2->Sibling;
        }
      }
      else
            Add_Edge_To_Node(Current_Node,
                  Temporary_Node, Cursor2, SINGLE_ARROWHEAD1);
    }
  else
    {
      Current_Node->X = X;
```

```
      Current_Node->Y = Y;
    }
  Create_Hierarchy_Diagram_Layout (Current_Node->Child);
  Create_Hierarchy_Diagram_Layout (Current_Node->Sibling);
}
}

/*************************************************************
 * Function : Layout routine which determines the node type
 * and then assigns node positions based on the type of node.
 *************************************************************/
void
Create_Flowchart_Diagram_Layout (NODE * Current_Node)
{
  if (Current_Node != NULL)
    {
      switch (Current_Node->Type)
      {
      case (KEYWORD_BEGIN):
        Create_BLOCK_Component_Layout (Current_Node);
        break;
      case (KEYWORD_PROGRAM):
        Current_Node->X = Current_Node->Y = 0;
        Create_BLOCK_Component_Layout (Current_Node);
        break;
      case (KEYWORD_IF):
        Create_IF_Component_Layout (Current_Node);
        break;
      case (KEYWORD_WHILE):
        Create_WHILE_Component_Layout (Current_Node);
        break;
      case (KEYWORD_REPEAT):
        Create_REPEAT_Component_Layout (Current_Node);
        break;
      case (KEYWORD_FOR):
        Create_FOR_Component_Layout (Current_Node);
        break;
      case (KEYWORD_CASE):
        Create_CASE_Component_Layout (Current_Node);
        break;
      case (KEYWORD_DUMMY_NODE):
        break;
      default:
        break;
      }
      Create_Flowchart_Diagram_Layout (Current_Node->Child);
      Create_Flowchart_Diagram_Layout (Current_Node->Sibling);
    }

}

/*************************************************************
 * Function : Shifts the Nodes X, Y values by DX and DY.
 *************************************************************/
void
Shift_Nodes_By_DX_and_DY (NODE * Current_Node, int DX, int DY)
{
```

```
  NODE *Cursor;
  Cursor = Current_Node->Child;
  while (Cursor != NULL)
    {
      Shift_Nodes_By_DX_and_DY (Cursor, DX, DY);
      Cursor = Cursor->Sibling;
    }
  Cursor = Current_Node;
  if (Cursor != NULL)
    {
      Cursor->X += DX;
      Cursor->Y += DY;
    }
}

/*****************************************************************
 * Function : Determines the nodes size based on type of node
 *****************************************************************/
void
Set_Node_Size (NODE * Current_Node)
{
  NODE *Cursor;
  if (Current_Node->Expanded == FALSE)
    {
      Current_Node->Width1 = Current_Node->Width2 =
      Font_Width (Statement_Table[Current_Node->Type]) / 2 +
      NODE_BOUNDARY_WIDTH + NODE_SEPERATION_X;
      Current_Node->Height1 = Current_Node->Height2 =
      Font_Height () / 2 +
      NODE_BOUNDARY_HEIGHT + NODE_SEPERATION_Y;
    }
  else
    {
      switch (Current_Node->Type)
      {
      case KEYWORD_PROGRAM:
      case KEYWORD_BEGIN:
      case KEYWORD_STATEMENT_BLOCK:
      case KEYWORD_IF:
      case KEYWORD_WHILE:
      case KEYWORD_REPEAT:
      case KEYWORD_FOR:
      case KEYWORD_CASE:
        MAXX = MINX = Current_Node->X;
        MAXY = MINY = Current_Node->Y;
        Cursor = Current_Node->Child;
        while (Cursor != NULL)
          {
            if ((Cursor->X - Cursor->Width1) < MINX)
            MINX = Cursor->X - Cursor->Width1;
            if ((Cursor->X + Cursor->Width2) > MAXX)
            MAXX = Cursor->X + Cursor->Width2;

            if ((Cursor->Y - Cursor->Height2) < MINY)
            MINY = Cursor->Y - Cursor->Height2;

            if ((Cursor->Y + Cursor->Height1) > MAXY)
```

```
            MAXY = Cursor->Y + Cursor->Height1;
            Cursor = Cursor->Sibling;
          }
        Current_Node->Width1 = Current_Node->X - MINX;
        Current_Node->Width2 = MAXX - Current_Node->X;
        Current_Node->Height2 = Current_Node->Y - MINY;
        Current_Node->Height1 = MAXY - Current_Node->Y;
        break;
      case LABEL_SYMBOL:
        Current_Node->Width2 = Font_Width (Current_Node->Statement) +
4;
        Current_Node->Height1 = Font_Height () + 4;
        Current_Node->Width1 = 1;
        Current_Node->Height2 = 1;
      case KEYWORD_DUMMY_NODE:
        Current_Node->Width1 = Current_Node->Width2 =
NODE_SEPERATION_X;
        Current_Node->Height1 = Current_Node->Height2 =
NODE_SEPERATION_Y;
        break;
      default:
        Current_Node->Width1 = Current_Node->Width2 =
          Font_Width (Current_Node->Statement) / 2 +
          NODE_BOUNDARY_WIDTH + NODE_SEPERATION_X;
        Current_Node->Height1 = Current_Node->Height2 =
          Font_Height () / 2 +
          NODE_BOUNDARY_HEIGHT + NODE_SEPERATION_Y;
        break;
      }
    }
}

/*************************************************************
 * Function : This routine applies an extended version of the
 * force scan algorithm. This 'tidies' the nodes by removing
 * node overlaps and eliminates wasted space. This produces
 * a clearer and compact diagram.
 *************************************************************/
void
Adjust_Node_Layout (NODE * Current_Node)
{
  NODE_LIST *Temporary_Node_List;
  NODE *Reference_Node;
  if (Current_Node != NULL)
    {
      if (Current_Node->Expanded == TRUE)
      Adjust_Node_Layout (Current_Node->Child);
      Adjust_Node_Layout (Current_Node->Sibling);
      if (Current_Node->Child != NULL)
      {
        Reference_Node = Create_Node ("Reference Point",
KEYWORD_DUMMY_NODE);
        Reference_Node->X = Current_Node->X;
        Reference_Node->Y = Current_Node->Y;
        Temporary_Node_List = Create_Node_List (Current_Node,
Reference_Node);
```

```
        Sort_Node_List_By_X_Value (Temporary_Node_List);
        Set_Accumulative_Force_X (Temporary_Node_List);

        Sort_Node_List_By_Y_Value (Temporary_Node_List);
        Set_Accumulative_Force_Y (Temporary_Node_List);

        Shift_Nodes_In_List (Temporary_Node_List,
                             Reference_Node->DX, Reference_Node->DY);
        Destroy_Node_List (Temporary_Node_List);
      }
      Set_Node_Size (Current_Node);
    }
}

/*************************************************************
 * Function : This routine maps coordinates to the viewing
 * screen. In this case it involves flipping the Y coordinate
 * information.
 *************************************************************/
void
Map_Coordinates_To_Display (NODE * Current_Node)
{
  int TempValue;
  if (Current_Node != NULL)
    {
      Map_Coordinates_To_Display (Current_Node->Child);
      Map_Coordinates_To_Display (Current_Node->Sibling);
      Current_Node->Y *= -1;
      TempValue = Current_Node->Height1;
      Current_Node->Height1 = Current_Node->Height2;
      Current_Node->Height2 = TempValue;
    }
}

/*************************************************************
 * Function : Combined routine which tidies node positioning,
 * an then shifts the entire diagram so that it fits on the
 * drawing area. This shifting effect creates a LEFT ALIGNED
 * diagram. That is, the diagram's top left corner is aligned
 * with the drawing areas top left corner. (plus a border)
 *************************************************************/
void
Adjust_Diagram_Layout (NODE * Current_Node)
{
  int DX, DY;
  The_Diagram->Expanded = TRUE;
  Adjust_Node_Layout (The_Diagram);
  DX = -(The_Diagram->X - The_Diagram->Width1) + BORDER_DISTANCE_X;
  DY = -(The_Diagram->Y - The_Diagram->Height2) + BORDER_DISTANCE_Y;
  Shift_Nodes_By_DX_and_DY (The_Diagram, DX, DY);
}
```

H. "Display_Routines.h"

```
/*************************************************************
 * File : Display_Routines.h                                 *
 *************************************************************/

/*************************************************************
 * Function : To return the Font_Width of a text string
 * in pixels. Current routine returns an approximate value for
 * the default font. (System specific).
 *************************************************************/
int
Font_Width (String_Type TheText)
{
  return (strlen (TheText) * FONT_WIDTH);
}

/*************************************************************
 * Function : To return the height of the currently selected
 * font. This routine currently returns an approximate value
 * for the default font. (System specific).
 *************************************************************/
int
Font_Height ()
{
  return FONT_HEIGHT;          /* Approximation */
}

/*************************************************************
 * Function : Sets Background drawing colour to white.
 *************************************************************/
void
Set_Background_White (void)
{
  XSetBackground (XtDisplay (Drawing_Area), The_Graphics_Context,
              WhitePixelOfScreen (XtScreen (Drawing_Area)));
}

/*************************************************************
 * Function : Sets Foreground drawing colour to black.
 *************************************************************/
void
Set_Foreground_Black (void)
{
  XSetForeground (XtDisplay (Drawing_Area), The_Graphics_Context,
              BlackPixelOfScreen (XtScreen (Drawing_Area)));
}

/*************************************************************
 * Function : Sets Background drawing colour to black.
 *************************************************************/
void
Set_Background_Black (void)
{
  XSetBackground (XtDisplay (Drawing_Area), The_Graphics_Context,
              BlackPixelOfScreen (XtScreen (Drawing_Area)));
```

```
}

/*************************************************************
 * Function : Sets Foreground drawing colour to white.
 *************************************************************/
void
Set_Foreground_White (void)
{
  XSetForeground (XtDisplay (Drawing_Area), The_Graphics_Context,
              WhitePixelOfScreen (XtScreen (Drawing_Area)));
}

/*************************************************************
 * Function : Sets the current drawing colour to a black
 * foreground, with a white background colour.
 *************************************************************/
void
Set_White_Background_Black_Foreground (void)
{
  Set_Background_White ();
  Set_Foreground_Black ();
}

/*************************************************************
 * Function : Sets the current drawing colour to a white
 * foreground, with a black background.
 *************************************************************/
void
Set_Black_Background_White_Foreground (void)
{
  Set_Background_Black ();
  Set_Foreground_White ();
}

/*************************************************************
 * Function : Creates a two dimensional arrowhead, rotates to
 * the correct angle then displays it.
 *************************************************************/
void
Draw_Arrow_Symbol (int x1, int y1, int x2, int y2, int Arrow_Style)
{
  int Y1, Y2;
  XPoint List_Of_Points[4];
  Y1 = y1;
  Y2 = y2;
  List_Of_Points[0].x = x2 - ARROW_SIZE / 2;
  List_Of_Points[0].y = y2 - ARROW_SIZE;
  List_Of_Points[1].x = x2 + ARROW_SIZE / 2;
  List_Of_Points[1].y = y2 - ARROW_SIZE;
  List_Of_Points[2].x = x2;
  List_Of_Points[2].y = y2;
  List_Of_Points[3].x = x2 - ARROW_SIZE / 2;
  List_Of_Points[3].y = y2 - ARROW_SIZE;
  switch (Arrow_Style)
    {
    case UP_ARROW:
      Rotate_Points (List_Of_Points, 3, x2, y2, 180);
```

```
      break;
    case LEFT_ARROW:
      Rotate_Points (List_Of_Points, 3, x2, y2, 270);
      break;
    case RIGHT_ARROW:
      Rotate_Points (List_Of_Points, 3, x2, y2, 90);
      break;
    };
  Draw_Polygon (List_Of_Points, 3, TRUE);
}

/*************************************************************
 * Function : Displays edges between nodes. Various edges are
 * drawn with different attributes. For instance a arrowhead
 * may be included. This routine also determines the direction
 * of the line and sets the appropriate arrowhead direction.
 *************************************************************/
void
Draw_Edges (NODE * Current_Node)
{
  EDGE *Cursor;
  int X1, X2, Y1, Y2, Arrow_Style;
  if (Current_Node != NULL)
    {
      Cursor = Current_Node->Edge;

      while (Cursor != NULL)
      {
        if (Cursor->Node1->X == Cursor->Node2->X)
          {
            X1 = X2 = Cursor->Node1->X;
            if (Cursor->Node1->Y > Cursor->Node2->Y)
            {
              Arrow_Style = UP_ARROW;
              Y1 = Cursor->Node1->Y - Cursor->Node1->Height2 +
                NODE_SEPERATION_Y;
              Y2 = Cursor->Node2->Y + Cursor->Node2->Height1 -
                NODE_SEPERATION_Y;
            }
            else
            {
              Arrow_Style = DOWN_ARROW;
              Y1 = Cursor->Node1->Y + Cursor->Node1->Height1 -
                NODE_SEPERATION_Y;
              Y2 = Cursor->Node2->Y - Cursor->Node2->Height2 +
                NODE_SEPERATION_Y;
            }
          }

        if (Cursor->Node1->Y == Cursor->Node2->Y)
          {
            Y1 = Y2 = Cursor->Node1->Y;
            if (Cursor->Node1->X > Cursor->Node2->X)
            {
              Arrow_Style = RIGHT_ARROW;
              X1 = Cursor->Node1->X - Cursor->Node1->Width2 +
                NODE_SEPERATION_X;
```

```
            X2 = Cursor->Node2->X + Cursor->Node2->Width1 -
              NODE_SEPERATION_X;
          }
          else
          {
            Arrow_Style = LEFT_ARROW;
            X1 = Cursor->Node1->X + Cursor->Node1->Width2 -
              NODE_SEPERATION_X;
            X2 = Cursor->Node2->X - Cursor->Node2->Width1 +
              NODE_SEPERATION_X;
          }
        }
        Draw_Line (X1, Y1, X2, Y2);
        if (Cursor->Arrow_Style == SINGLE_ARROWHEAD1)
          Draw_Arrow_Symbol (X1, Y1, X2, Y2, Arrow_Style);
        Cursor = Cursor->Next;
      }
      if (Current_Node->Expanded == TRUE)
      Draw_Edges (Current_Node->Child);
      Draw_Edges (Current_Node->Sibling);
    }
}

/*****************************************************************
 * Function : Draws a diamond shape.
 * This routine resizes the diamond shape so that it fits
 * around the nodes text. This code could be changed so that
 * the text could be trimmed, or the if symbol is modified so
 * that all of the displayed text is contained within the
 * IF symbol.
 *****************************************************************/
void
Draw_IF_Symbol (NODE * Current_Node)
{
  int Width, Height, A, B, C, D;
  int FW, FH;
  XPoint List_Of_Points[9];
  Width = Current_Node->Width1 - NODE_SEPERATION_X;
  Height = Current_Node->Height1 - NODE_SEPERATION_Y;
  A = Current_Node->Y - Height;
  B = Current_Node->X + Width;
  C = Current_Node->X - Width;
  D = Current_Node->Y + Height;
  FW = Font_Width (Current_Node->Statement) / 2;
  FH = Height;                  /*Font_Height()/4; */

  List_Of_Points[0].x = Current_Node->X;
  List_Of_Points[0].y = Current_Node->Y - Height;
  List_Of_Points[1].x = Current_Node->X + FW;
  List_Of_Points[1].y = Current_Node->Y - FH;
  List_Of_Points[2].x = Current_Node->X + Width;
  List_Of_Points[2].y = Current_Node->Y;

  List_Of_Points[3].x = Current_Node->X + FW;
  List_Of_Points[3].y = Current_Node->Y + FH;
  List_Of_Points[4].x = Current_Node->X;
  List_Of_Points[4].y = Current_Node->Y + Height;
```

```
  List_Of_Points[5].x = Current_Node->X - FW;
  List_Of_Points[5].y = Current_Node->Y + FH;
  List_Of_Points[6].x = Current_Node->X - Width;
  List_Of_Points[6].y = Current_Node->Y;
  List_Of_Points[7].x = Current_Node->X - FW;
  List_Of_Points[7].y = Current_Node->Y - FH;

  List_Of_Points[8].x = Current_Node->X;
  List_Of_Points[8].y = Current_Node->Y - Height;



/*  List_Of_Points[0].x = Current_Node->X;
   List_Of_Points[0].y = A;
   List_Of_Points[1].x = B;
   List_Of_Points[1].y = Current_Node->Y;
   List_Of_Points[2].x = Current_Node->X;
   List_Of_Points[2].y = D;
   List_Of_Points[3].x = C;
   List_Of_Points[3].y = Current_Node->Y;
   List_Of_Points[4].x = Current_Node->X;
   List_Of_Points[4].y = A;
 */


  Set_Black_Background_White_Foreground ();
  Draw_Polygon (List_Of_Points, 8, TRUE);
  Set_White_Background_Black_Foreground ();
  Set_Foreground_Colour (Current_Node->Colour);
  Draw_Text_Centered (Current_Node->X, Current_Node->Y,
                      Current_Node->Statement);
  Draw_Lines (List_Of_Points, 9);
}

/*************************************************************
 * Function: Generates a grid with size
 * (Screen_Width * Screen_Height).
 * With grid markers GRID_SIZE apart
 *************************************************************/
void
Draw_Grid (void)
{
  int Index1, Index2;
  for (Index1 = 0; Index1 <= Screen_Width; Index1 = Index1 +
GRID_SIZE)
    for (Index2 = 0; Index2 <= Screen_Height; Index2 = Index2 +
GRID_SIZE)
      Draw_Line (Index1, Index2, Index1, Index2);
}

/*************************************************************
 * Function : Draws an X shape marker at a specified location.
 * Routine for marking out points on the diagram.
 *************************************************************/
```

```
void
Draw_Marker_Symbol (int X1, int Y1)
{
  int Size = 5;
  Draw_Line (X1 - Size, Y1 + Size, X1 + Size, Y1 - Size);
  Draw_Line (X1 - Size, Y1 - Size, X1 + Size, Y1 + Size);
}

/*************************************************************
 * Function : Draws a rectangle around the an entire diagram
 * component. This helps the user to select a diagram
 * component by understanding the area the component
 * encompasses.
 *************************************************************/
void
Draw_Boundaries (NODE * Component)
{
  if (Component != NULL)
    {
      if (Component->Expanded == TRUE)
      Draw_Boundaries (Component->Child);
      Draw_Boundaries (Component->Sibling);

      Draw_Rectangle (Component->X - Component->Width1,
                  Component->Y - Component->Height2,
                  Component->X + Component->Width2,
                  Component->Y + Component->Height1, FALSE);
    }
}

/*************************************************************
 * Function : This routines draws each individual flowchart
 * component with specific node attributes.
 * e.g Colour, Width, Height, etc.*
 *************************************************************/
void
Draw_Flowchart_Component (NODE * Component)
{
  int X1, Y1;

  if (Component != NULL)
    {
      X1 = Component->X;
      Y1 = Component->Y;
      Set_Foreground_Colour (Component->Colour);
      if (Component->Expanded == FALSE)
      {
        Set_Black_Background_White_Foreground ();
        Draw_Rectangle (Component->X - Component->Width1 +
NODE_SEPERATION_X,
                  Component->Y - Component->Height1 +
NODE_SEPERATION_Y,
                   Component->X + Component->Width2 -
NODE_SEPERATION_X,
                  Component->Y + Component->Height2 -
NODE_SEPERATION_Y,
                   TRUE);
```

```
        Set_White_Background_Black_Foreground ();
        Set_Foreground_Colour ("BLUE");
        Draw_Rectangle (Component->X - Component->Width1 +
NODE_SEPERATION_X,
                        Component->Y - Component->Height1 +
NODE_SEPERATION_Y,
                         Component->X + Component->Width2 -
NODE_SEPERATION_X,
                        Component->Y + Component->Height2 -
NODE_SEPERATION_Y,
                         FALSE);
        Draw_Text_Centered (X1, Y1, Statement_Table[Component-
>Type]);
      }
      else
      {
        switch (Component->Type)
          {
          case KEYWORD_BEGIN:
          case KEYWORD_STATEMENT_BLOCK:
          case KEYWORD_IF:
          case KEYWORD_WHILE:
          case KEYWORD_REPEAT:
          case KEYWORD_PROGRAM:
          case KEYWORD_PROCEDURE:
          case KEYWORD_FOR:
          case KEYWORD_CASE:
            break;
          case LABEL_SYMBOL:
            Set_Foreground_Colour ("BLUE");
            Draw_Text (X1 + 2, Y1 - 2, Component->Statement);
            break;
          case IF_SYMBOL:
            Draw_IF_Symbol (Component);
            break;
          case CASE_SYMBOL:
            Set_Black_Background_White_Foreground ();
            Draw_Ellipse (Component->X - Component->Width1 +
                          NODE_SEPERATION_X,
                          Component->Y - Component->Height1 +
                          NODE_SEPERATION_Y,
                          Component->X + Component->Width2 -
                          NODE_SEPERATION_X,
                          Component->Y + Component->Height2 -
                          NODE_SEPERATION_Y, TRUE);
            Set_White_Background_Black_Foreground ();
            Draw_Ellipse (Component->X - Component->Width1 +
                          NODE_SEPERATION_X,
                          Component->Y - Component->Height1 +
                          NODE_SEPERATION_Y,
                          Component->X + Component->Width2 -
                          NODE_SEPERATION_X,
                          Component->Y + Component->Height2 -
                          NODE_SEPERATION_Y, FALSE);
            Set_Foreground_Colour (Component->Colour);
            Draw_Text_Centered (X1, Y1, Component->Statement);
            break;
```

```
            case KEYWORD_DUMMY_NODE:
              break;
            default:
              {
              Set_Black_Background_White_Foreground ();
              Draw_Rectangle (Component->X - Component->Width1 +
                              NODE_SEPERATION_X,
                              Component->Y - Component->Height1 +
                              NODE_SEPERATION_Y,
                              Component->X + Component->Width2 -
                              NODE_SEPERATION_X,
                              Component->Y + Component->Height2 -
                              NODE_SEPERATION_Y, TRUE);
              Set_White_Background_Black_Foreground ();
              Draw_Rectangle (Component->X - Component->Width1 +
                              NODE_SEPERATION_X,
                              Component->Y - Component->Height1 +
                              NODE_SEPERATION_Y,
                              Component->X + Component->Width2 -
                              NODE_SEPERATION_X,
                              Component->Y + Component->Height2 -
                              NODE_SEPERATION_Y, FALSE);
              Set_Foreground_Colour (Component->Colour);
              Draw_Text_Centered (X1, Y1, Component->Statement);
              break;
              }
            }
        }
        if (Component->Expanded == TRUE)
        Draw_Flowchart_Component (Component->Child);
        Draw_Flowchart_Component (Component->Sibling);
    }
}
```

I. "Node_List_Sorting_Routines.h"

```
/*************************************************************
 * File : Node_List_Sorting_Routines.h                       *
 *************************************************************/

/*************************************************************
 * Function : Sorts a node list by X values.
 *************************************************************/
Sort_Node_List_By_X_Value (NODE_LIST * The_List)
{
  NODE_LIST *Cursor, *Next, *Previous;
  NODE *Temporary_Node;
  int SORTED;
  SORTED = FALSE;
  if (The_List != NULL)

    while (SORTED == FALSE)
      {
      SORTED = TRUE;
      Cursor = The_List;
      Previous = Cursor;
      while (Cursor != NULL)
        {
          Next = Cursor->Next;
          if (Next != NULL)
            {
            if (Cursor->Node_Pointer->X > Next->Node_Pointer->X)
              {
                Temporary_Node = Cursor->Node_Pointer;
                Cursor->Node_Pointer = Next->Node_Pointer;
                Next->Node_Pointer = Temporary_Node;
              }
            }

          if (Previous->Node_Pointer->X > Cursor->Node_Pointer->X)
            SORTED = FALSE;
          Previous = Cursor;
          Cursor = Cursor->Next;
        }

      }
  Cursor = The_List;
}

/*************************************************************
 * Function : Sorts a node list by Y values.
 *************************************************************/
Sort_Node_List_By_Y_Value (NODE_LIST * The_List)
{
  NODE_LIST *Cursor, *Next, *Previous;
  NODE *Temporary_Node;
  int SORTED;
  SORTED = FALSE;
  if (The_List != NULL)
```

```
    while (SORTED == FALSE)
      {
      SORTED = TRUE;
      Cursor = The_List;
      Previous = Cursor;
      while (Cursor != NULL)
        {
          Next = Cursor->Next;
          if (Next != NULL)
            {
            if (Cursor->Node_Pointer->Y > Next->Node_Pointer->Y)
              {
                Temporary_Node = Cursor->Node_Pointer;
                Cursor->Node_Pointer = Next->Node_Pointer;
                Next->Node_Pointer = Temporary_Node;
              }
            }
          if (Previous->Node_Pointer->Y > Cursor->Node_Pointer->Y)
            SORTED = FALSE;
          Previous = Cursor;
          Cursor = Cursor->Next;
        }

      }
  Cursor = The_List;
}

/*************************************************************
 * Function : Prints node list details.
 *************************************************************/
Print_Node_List_Details (NODE_LIST * The_List)
{
  NODE_LIST *Cursor;
  Cursor = The_List;
  while (Cursor != NULL)
    {
      printf ("Node %d (%d,%d) (%d+%d, %d+%d) [%s] Type=[%d]\n",
            Cursor->Node_Pointer->ID,
            Cursor->Node_Pointer->X,
            Cursor->Node_Pointer->Y,
            Cursor->Node_Pointer->Height1,
            Cursor->Node_Pointer->Height2,
            Cursor->Node_Pointer->Width1,
            Cursor->Node_Pointer->Width2,
            Cursor->Node_Pointer->Statement,
            Cursor->Node_Pointer->Type);
      Cursor = Cursor->Next;
    }
}
```

J. "Force_Scan_Routines.h"

```
/*************************************************************
 * File : Force_Scan_Routines.h                              *
 *************************************************************/

/*************************************************************
 * Function : Routine which calculates the distance(Y) from
 * the outside of one node(A) to the outside of another node(B)
 *************************************************************/
int
Calculate_Distance_Y (Ah, Aw, Bh, Bw, A, B)
     int Ah, Aw, Bh, Bw;
     NODE *A, *B;
{
  double M_AB, M_Aa, b, X, Y;

  M_AB = fabs ((B->Y - A->Y) / (B->X - A->X));
  M_Aa = (Ah + Bh) / (Aw + Bw);
  if (M_AB > M_Aa)
    Y = A->Y + Ah + Bh;
  else
    {
      b = A->Y - (M_AB * A->X);
      X = A->X + Aw + Bh;
      Y = (M_AB * X) + b;
    }
  return Y - A->Y;
}

/*************************************************************
 * Function : Routine which calculates the distance(X) from
 * the outside of one node(A) to the outside of another node(B)
 *************************************************************/
int
Calculate_Distance_X (Ah, Aw, Bh, Bw, A, B)
     int Ah, Aw, Bh, Bw;
     NODE *A, *B;
{
  double M_AB, M_Aa, b, X, Y;

  M_AB = fabs ((B->Y - A->Y) / (B->X - A->X));
  M_Aa = (Ah + Bh) / (Aw + Bw);
  if (M_AB > M_Aa)
    {
      Y = A->Y + Ah + Bh;
      b = A->Y - M_AB * A->X;
      X = (Y - b) / M_AB;
    }
  else
    X = A->X + Aw + Bw;
  return X - A->X;
}

/*************************************************************
 * Function : This routine works out the force(X) required to
```

```
 * push the node B closer to A so that B rests on A. This is
 * determined by examining the slope of the line then calling
 * the appropriate routine (Calculate_Distance_X).
 ************************************************************/
int
Calculate_Force_In_X_Direction (NODE * A, NODE * B)
{
  double DX, DY;
  DX = DY = 0;

  if ((A->X < B->X) && (B->Y == A->Y))
    DX = A->Width2 + B->Width1;
  else if ((A->X > B->X) && (B->Y == A->Y))
    DX = A->Width1 + B->Width2;
  else if ((A->X < B->X) && (A->Y < B->Y))
    DX = Calculate_Distance_X (A->Height1, A->Width2,
                               B->Height2, B->Width1, A, B);
  else if ((A->X < B->X) && (B->Y < A->Y))
    DX = Calculate_Distance_X (A->Height1, A->Width2,
                               B->Height1, B->Width1, A, B);
  else if ((A->X > B->X) && (B->Y < A->Y))
    DX = -Calculate_Distance_X (A->Height2, A->Width1,
                              B->Height1, B->Width2, A, B);
  else if ((A->X > B->X) && (B->Y > A->Y))
    DX = -Calculate_Distance_X (A->Height1, A->Width1,
                              B->Height2, B->Width2, A, B);

  return -(B->X - A->X - DX);
}

/************************************************************
 * Function : This routine works out the force(Y) required to
 * push the node B closer to A so that B rests on A. This is
 * determined by examining the slope of the line then calling
 * the Appropriate routine (Calculate_Distance_Y).
 ************************************************************/
int
Calculate_Force_In_Y_Direction (NODE * A, NODE * B)
{
  double DX, DY;
  DX = DY = 0;
  if ((A->X == B->X) && (B->Y > A->Y))
    DY = A->Height1 + B->Height2;
  else if ((A->X == B->X) && (B->Y < A->Y))
    DY = -A->Height2 - B->Height1;
  else if ((A->X < B->X) && (B->Y > A->Y))
    DY = Calculate_Distance_Y (A->Height1, A->Width2,
                               B->Height2, B->Width1, A, B);
  else if ((A->X < B->X) && (B->Y < A->Y))
    DY = -Calculate_Distance_Y (A->Height1, A->Width2,
                              B->Height1, B->Width1, A, B);
  else if ((A->X > B->X) && (B->Y < A->Y))
    DY = -Calculate_Distance_Y (A->Height2, A->Width1,
                              B->Height1, B->Width2, A, B);
  else if ((A->X > B->X) && (B->Y > A->Y))
    DY = Calculate_Distance_Y (A->Height1, A->Width1,
                              B->Height2, B->Width2, A, B);
```

```
  return -(B->Y - (A->Y + DY));
}

/*************************************************************
 * Function : Each Node is given a DX value of initially 0.
 * This routine sets DX values for a list of nodes. Once
 * these values have been set then the nodes can be shifted
 * according to the determined DX value.
 * Shifting nodes (using the force scan algorithm )
 * reduces node seperation and node overlap.
 *************************************************************/
Set_Accumulative_Force_X (NODE_LIST * Node_List)
{
  NODE_LIST *Cursor, *Cursor2, *Cursor3, *Cursor4;
  NODE_LIST *Current_Block_Begin, *Current_Block_End;
  int FINISHED, Max_Force, Current_Force;
  Cursor = Node_List;
  while (Cursor != NULL)
    {
      Cursor2 = Cursor;
      FINISHED = FALSE;
      while (FINISHED == FALSE)
      {
        if (Cursor2 != NULL)
          {
            if (Cursor->Node_Pointer->X == Cursor2->Node_Pointer->X)
            Cursor2 = Cursor2->Next;
            else
            FINISHED = TRUE;
          }
        else
          FINISHED = TRUE;
      }
      Current_Block_Begin = Cursor;
      Current_Block_End = Cursor2;
      Cursor3 = Current_Block_Begin;
      Cursor4 = Current_Block_End;
      if (Cursor4 != NULL)
      {
        Max_Force = Calculate_Force_In_X_Direction (Cursor3-
>Node_Pointer,
                                            Cursor4->Node_Pointer);
        while (Cursor3->Node_Pointer->ID != Current_Block_End-
>Node_Pointer->ID)
          {
            Cursor4 = Current_Block_End;
            while (Cursor4 != NULL)
            {
              Current_Force = Calculate_Force_In_X_Direction (
                                            Cursor3->Node_Pointer,
                                           Cursor4->Node_Pointer);
              if (Current_Force > Max_Force)
                Max_Force = Current_Force;
              Cursor4 = Cursor4->Next;
            }
```

```
            Cursor3 = Cursor3->Next;
          }
        Cursor = Current_Block_End;
        while (Cursor != NULL)
          {
            Cursor->Node_Pointer->DX += Max_Force;
            Cursor = Cursor->Next;

          }
      }
      Cursor = Cursor2;
    }
}

/*************************************************************
 * Function : Each Node is given a DY value of initially 0.
 * This routine sets DY values for a list of nodes. Once
 * these values have been set then the nodes can be shifted
 * according to the determined DY value.
 * Shifting nodes (using the force scan algorithm )
 * reduces node seperation and node overlap.
 *************************************************************/
Set_Accumulative_Force_Y (NODE_LIST * Node_List)
{
  NODE_LIST *Cursor, *Cursor2, *Cursor3, *Cursor4;
  NODE_LIST *Current_Block_Begin, *Current_Block_End;
  int FINISHED, Max_Force, Current_Force;
  Cursor = Node_List;
  while (Cursor != NULL)
    {
      Cursor2 = Cursor;
      FINISHED = FALSE;
      while (FINISHED == FALSE)
      {
        if (Cursor2 != NULL)
          {
            if (Cursor->Node_Pointer->Y == Cursor2->Node_Pointer->Y)
            Cursor2 = Cursor2->Next;
            else
            FINISHED = TRUE;
          }
        else
          FINISHED = TRUE;
      }
      Current_Block_Begin = Cursor;
      Current_Block_End = Cursor2;
      Cursor3 = Current_Block_Begin;
      Cursor4 = Current_Block_End;
      if (Cursor4 != NULL)
      {
        Max_Force = Calculate_Force_In_Y_Direction
          (Cursor3->Node_Pointer, Cursor4->Node_Pointer);
        while (Cursor3->Node_Pointer->ID != Current_Block_End-
>Node_Pointer->ID)
          {
            Cursor4 = Current_Block_End;
            while (Cursor4 != NULL)
```

```
            {
              Current_Force = Calculate_Force_In_Y_Direction
                (Cursor3->Node_Pointer, Cursor4->Node_Pointer);
              if (Current_Force > Max_Force)
                {
                  Max_Force = Current_Force;
                }
              Cursor4 = Cursor4->Next;
            }
            Cursor3 = Cursor3->Next;
          }
        Cursor = Current_Block_End;
        while (Cursor != NULL)
          {
            Cursor->Node_Pointer->DY += Max_Force;
            Cursor = Cursor->Next;

          }
      }
      Cursor = Cursor2;
    }
}

/*************************************************************
 * Function : Two previous functions set the DX and DY values
 * This routine shift all nodes in the Node_List to their new
 * location. This routine also shift child nodes.
 *************************************************************/
Shift_Nodes_In_List (NODE_LIST * Node_List, int DX, int DY)
{
  NODE_LIST *Cursor;
  Cursor = Node_List;
  while (Cursor != NULL)
    {
      Shift_Nodes_By_DX_and_DY (Cursor->Node_Pointer,
                        Cursor->Node_Pointer->DX - DX,
                        Cursor->Node_Pointer->DY - DY);
      Cursor->Node_Pointer->DX = Cursor->Node_Pointer->DY = 0;
      Cursor = Cursor->Next;
    }
}
```

K. "Menu_Selection_Routines.h"

```
/*************************************************************
 * File : Menu_Selection_Routines.h                          *
 *************************************************************/

/*************************************************************
 * Function : Sets drawing Colours.
 *************************************************************/
void
Set_Drawing_Colours ()
{
  XSetForeground (XtDisplay (Drawing_Area), The_Graphics_Context,
              WhitePixelOfScreen (XtScreen (Drawing_Area)));
  Set_Foreground_Colour (FOREGROUND_COLOUR);
  Set_Background_Colour (BACKGROUND_COLOUR);
}

/*************************************************************
 * Function : Refreshes the drawing area.
 *************************************************************/
void
Redraw_Diagram_To_Bitmap ()
{
  Read_Configuration_File (CONFIGURATION_FILE_NAME);
  Adjust_Diagram_Layout (The_Diagram);
  Set_Drawing_Colours ();
  if (DISPLAY_GRID == TRUE)
    {
      Set_Foreground_Colour (GRID_COLOUR);
      Draw_Grid ();
    }
  if (DISPLAY_BOUNDARIES == TRUE)
    {
      Set_Foreground_Colour (BOUNDARY_COLOUR);
      Draw_Boundaries (The_Diagram);
    }
  if (DISPLAY_EDGES == TRUE)
    {
      Set_Foreground_Colour (EDGE_COLOUR);
      Draw_Edges (The_Diagram);
    }
  if (DISPLAY_COMPONENTS == TRUE)
    Draw_Flowchart_Component (The_Diagram);
  Redraw_Bitmap ();
}

/*************************************************************
 * Function : Exits the application and frees up memory.
 * Also saves any changes made to the configuration settings.
 *************************************************************/
void
Quit_The_Application ()
{
  Destroy_Tree (The_Diagram);
  Save_Configuration_File (CONFIGURATION_FILE_NAME);
```

```
  XFreePixmap (XtDisplay (Drawing_Area), Users_Bitmap);
  XtCloseDisplay (XtDisplay (Top_Level));
  exit (0);
}

/*************************************************************
 * Function : Resizes the drawing area based on the diagram
 * size.
 *************************************************************/
void
Resize_Bitmap ()
{
  Screen_Width = The_Diagram->Width1 +
    The_Diagram->Width2 +
    (2 * BORDER_DISTANCE_X);
  Screen_Height = The_Diagram->Height1 +
    The_Diagram->Height2 +
    (2 * BORDER_DISTANCE_Y);
  XtDestroyWidget (Drawing_Area);
  Drawing_Area = XtVaCreateManagedWidget ("Drawing_Area",
                                xmDrawingAreaWidgetClass,
                                Main_Window,
                                XmNtranslations,
                             XtParseTranslationTable (translations),
                                XmNunitType, XmPIXELS,
                                XmNwidth, Screen_Width,
                                XmNheight, Screen_Height,
                                XmNresizePolicy, XmNONE, NULL);
  XtAddCallback (Drawing_Area, XmNexposeCallback,
              Redraw_Diagram_To_Bitmap, NULL);
  XFreePixmap (XtDisplay (Drawing_Area), Users_Bitmap);
  Users_Bitmap = XCreatePixmap (XtDisplay (Drawing_Area),
                        RootWindowOfScreen (XtScreen (Drawing_Area)),
                        Screen_Width, Screen_Height,
                      DefaultDepthOfScreen (XtScreen
(Drawing_Area)));

}

Widget menubar;
/*************************************************************
 * Function : Routine which generates a dialog box, and
 * allows the user to select a source code file from the
 * file system.
 *************************************************************/
void
Open_File_Dialog_Box ()
{
  printf ("Opening file... [To be implemented]\n");
}

/*************************************************************
 * Function : Creates a dialog box which allows the user
 * to type in the file name that will be saved.
 * This dialog box allows the user to save the file as a
 * diagram file or a bitmap file.
 * This routine currently dumps the users bitmap to a file.
```

```
 ************************************************************/
void
Save_File_Dialog_Box ()
{
  printf ("Saving file\n");
  XWriteBitmapFile (XtDisplay (Drawing_Area), "Temp.bmp",
Users_Bitmap,
                Screen_Width, Screen_Height, -1, -1);
}
void Display_The_Diagram (void);
/************************************************************
 * Function : Saves an XFIGURES File and executes XFIG;
 * assumes XFIG is in the path.
 ************************************************************/
void
Save_XFIG_File ()
{
  XFIG_File_Pointer = fopen(XFIG_FILE_NAME,"w");
  Output_To_FIG = TRUE;
  Display_The_Diagram ();
  Output_To_FIG = FALSE;
  fclose(XFIG_File_Pointer);
  execlp("xfig", "xfig", XFIG_FILE_NAME, NULL);
}

/************************************************************
 * Function : Collapses the entire diagram so that each
 * diagram component can be individually expanded as required
 * by the user.
 ************************************************************/
void
Collapse_All_Of_The_Diagram ()
{
  Collapse_All_Nodes (The_Diagram);
  The_Diagram->Expanded = TRUE;
  Redraw_Diagram_To_Bitmap ();
  Resize_Bitmap ();
}

/************************************************************
 * Function : Expands all of the diagram so that all of the
 * diagram components are displayed.
 ************************************************************/
void
Expand_All_Of_The_Diagram ()
{
  Expand_All_Nodes (The_Diagram);
  The_Diagram->Expanded = TRUE;
  Redraw_Diagram_To_Bitmap ();
  Resize_Bitmap ();
}

/************************************************************
 * Function : Routine used for prototyping.
 ************************************************************/
void
Do_Nothing ()
```

```
{
}

/*************************************************************
 * Function : Toggles the visibility of the grid. This
 * routine also writes this change to the configuration file.
 *************************************************************/
void
Toggle_Grids_Visibility ()
{
  if (DISPLAY_GRID == TRUE)
    DISPLAY_GRID = FALSE;
  else
    DISPLAY_GRID = TRUE;
  Save_Configuration_File (CONFIGURATION_FILE_NAME);
  Redraw_Diagram_To_Bitmap ();
}

/*************************************************************
 * Function : Toggles the visibility of the boundaries.
 * The boundaries are rectagles drawn around diagram
 * components. This routines saves the visibility state
 * of the boundaries to the configuration file.
 *************************************************************/
void
Toggle_Boundaries_Visibility ()
{
  if (DISPLAY_BOUNDARIES == TRUE)
    DISPLAY_BOUNDARIES = FALSE;
  else
    DISPLAY_BOUNDARIES = TRUE;
  Save_Configuration_File (CONFIGURATION_FILE_NAME);
  Redraw_Diagram_To_Bitmap ();
}

/*************************************************************
 * Function : Toggles the visiblity of the diagram edges.
 * The edges are lines which join diagram components.
 * The change in the settings are stored into the configuration
 * file.
 *************************************************************/
void
Toggle_Edges_Visibility ()
{
  if (DISPLAY_EDGES == TRUE)
    DISPLAY_EDGES = FALSE;
  else
    DISPLAY_EDGES = TRUE;
  Save_Configuration_File (CONFIGURATION_FILE_NAME);
  Redraw_Diagram_To_Bitmap ();
}

/*************************************************************
 * Function :Toggles the visibility of the diagram components.
 * Changes made to this setting are stored into the
 * configuration file.
 *************************************************************/
```

```
void
Toggle_Components_Visibility ()
{
  if (DISPLAY_COMPONENTS == TRUE)
    DISPLAY_COMPONENTS = FALSE;
  else
    DISPLAY_COMPONENTS = TRUE;
  Save_Configuration_File (CONFIGURATION_FILE_NAME);
  Redraw_Diagram_To_Bitmap ();
}

/*************************************************************
 * Function : Sets the background drawing colour to a
 * specified "Colour"
 *************************************************************/
Set_Background_Colour (String_Type Colour)
{
  Display *dpy = XtDisplay (Drawing_Area);
  Colormap cmap = DefaultColormapOfScreen (XtScreen (Drawing_Area));
  XColor col, unused;
  if (!XAllocNamedColor (dpy, cmap, Colour, &col, &unused))
    {
      char buf[32];
      sprintf (buf, "Can't alloc %s", Colour);
      XtWarning (buf);
    }
  else
    {
      XSetForeground (dpy, The_Graphics_Context, col.pixel);
      XFillRectangle (XtDisplay (Drawing_Area),
                  Users_Bitmap, The_Graphics_Context,
                  0, 0, Screen_Width, Screen_Height);
    }
}

/*************************************************************
 * Function : Sets the foreground drawing colour to a
 * specified "Colour".
 *************************************************************/
Set_Foreground_Colour (String_Type Colour)
{
  Display *dpy = XtDisplay (Drawing_Area);
  Colormap cmap = DefaultColormapOfScreen (XtScreen (Drawing_Area));
  XColor col, unused;
  if (!XAllocNamedColor (dpy, cmap, Colour, &col, &unused))
    {
      char buf[32];
      sprintf (buf, "Can't alloc %s", Colour);
      XtWarning (buf);
    }
  else
    XSetForeground (dpy, The_Graphics_Context, col.pixel);
}


/*************************************************************
 * Function : Displays an about box, with information
```

```
 * that accredits the author and displays copyright
 * information.
 ************************************************************/
void
Do_About_Dialog ()
{
  printf ("Displays an about box. [To be implemented]\n");
}

/************************************************************
 * Function : Displays general information about the
 * programs implementation.
 ************************************************************/
void
Do_Program_Details_Dialog ()
{
  printf ("Displays a dialog box about the program
implementation.\n");
  printf ("[To be implemented]\n");
}

/************************************************************
 * Function : Displays a dialog box that contains a
 * set of diagram attributes that can be set. A list of
 * these diagram attributes are in the configuration file.
 ************************************************************/
void
Settings_Dialog_Box ()
{
  printf ("Displays Diagram settings dialog box... [To be
implemented]\n");
}

/************************************************************
 * Function : Scales the diagram by a factor of 2.
 ************************************************************/
void
Zoom_In_By_2 ()
{
  Font font;
  char Font_Name[80];
  Zoom_Factor += 1;

  sprintf (Font_Name, "-b&h-*-*-r-*-*-%d-*-*-*-*-*-*-*",
Zoom_Factor);
  font = XLoadFont (XtDisplay (Drawing_Area), Font_Name);
  if (font != 0)
    XSetFont (XtDisplay (Drawing_Area), The_Graphics_Context, font);
  Font_Structure = XLoadQueryFont (XtDisplay (Drawing_Area),
Font_Name);
  Adjust_Diagram_Layout (The_Diagram);
  Screen_Width = The_Diagram->Width1 +
    The_Diagram->Width2 +
    (2 * BORDER_DISTANCE_X);
  Screen_Height = The_Diagram->Height1 +
    The_Diagram->Height2 +
    (2 * BORDER_DISTANCE_Y);
```

```
  Resize_Bitmap ();
  XtVaSetValues (Main_Window, XmNwidth, Screen_Width + 4,
            XmNheight, Screen_Height + 35, NULL);
}

/*************************************************************
 * Function : Scales the diagram by a factor of 0.5
 *************************************************************/
void
Zoom_Out_By_2 ()
{
  Font font;
  char Font_Name[80];
  if (Zoom_Factor >= 2)
    Zoom_Factor -= 1;

  sprintf (Font_Name, "-b&h-*-*-r-*-*-%d-*-*-*-*-*-*-*",
Zoom_Factor);
  font = XLoadFont (XtDisplay (Drawing_Area), Font_Name);
  if (font != 0)
    XSetFont (XtDisplay (Drawing_Area), The_Graphics_Context, font);
  Font_Structure = XLoadQueryFont (XtDisplay (Drawing_Area),
Font_Name);
  Adjust_Diagram_Layout (The_Diagram);
  Screen_Width = The_Diagram->Width1 +
    The_Diagram->Width2 +
    (2 * BORDER_DISTANCE_X);
  Screen_Height = The_Diagram->Height1 +
    The_Diagram->Height2 +
    (2 * BORDER_DISTANCE_Y);
  Resize_Bitmap ();
  XtVaSetValues (Main_Window, XmNwidth, Screen_Width + 4,
            XmNheight, Screen_Height + 35, NULL);
}

/*************************************************************
 * Function : Creates a dialog box which the user can
 * select the zoom quantity.
 * Currently sets zoom factor to 10 and uses the default.
 *************************************************************/
void
Zoom_In_By_User_Defined_Percent ()
{
  Font font;
  Zoom_Factor = 10;

  font = XLoadFont (XtDisplay (Drawing_Area), FONT_NAME);
  if (font != 0)
    XSetFont (XtDisplay (Drawing_Area), The_Graphics_Context, font);
  Font_Structure = XLoadQueryFont (XtDisplay (Drawing_Area),
FONT_NAME);
  Adjust_Diagram_Layout (The_Diagram);
  Screen_Width = The_Diagram->Width1 +
    The_Diagram->Width2 +
    (2 * BORDER_DISTANCE_X);
  Screen_Height = The_Diagram->Height1 +
    The_Diagram->Height2 +
```

```
    (2 * BORDER_DISTANCE_Y);
  Resize_Bitmap ();
  XtVaSetValues (Main_Window, XmNwidth, Screen_Width + 4,
            XmNheight, Screen_Height + 35, NULL);
}
```

L. "Menu_Generation_Routines.h"

```
/*************************************************************
 * File : Menu_Generation_Routines.h                         *
 *************************************************************/

typedef struct
  {
    int Reason;
    XEvent *Event;
    int Click_Count;
  }
XmPushButtonCallbackStructure;

typedef struct MENU_ITEM
  {
    char *Label;
    WidgetClass *Class;
    char Mnemonic;
    char *Accelerator;
    char *Accelerator_Text;
    void (*Callback) ();
    XtPointer Callback_Data;
    struct MENU_ITEM *Sub_Items;
  }
Menu_Item;


/*************************************************************
 * Defines the FILE menu options
 *************************************************************/
Menu_Item File_Menu_Option[] =
{
  {"Open", &xmPushButtonGadgetClass, 'O', "Alt<Key>O", "Alt+O",
   Open_File_Dialog_Box, (Menu_Item *) NULL},
  {"Save", &xmPushButtonGadgetClass, 'S', "Alt<Key>S", "Alt+S",
   Save_File_Dialog_Box, (Menu_Item *) NULL},
  {"View As XFIGURE", &xmPushButtonGadgetClass, 'V', "Alt<Key>V",
"Alt+V",
   Save_XFIG_File, (Menu_Item *) NULL},
  {"Exit", &xmPushButtonGadgetClass, 'x', "Alt<Key>x", "Alt+x",
   Quit_The_Application, (Menu_Item *) NULL},
  NULL,
};


/*************************************************************
 * Defines the DIAGRAM menu options
 *************************************************************/
Menu_Item Diagram_Menu_Option[] =
{
  {"Settings", &xmPushButtonGadgetClass, 'S', "Ctr<Key>S", "Ctrl+S",
   Settings_Dialog_Box, (Menu_Item *) NULL},
  {"Collapse All", &xmPushButtonGadgetClass, 'C', "Ctrl<Key>C",
"Ctrl+C",
   Collapse_All_Of_The_Diagram, (Menu_Item *) NULL},
```

```
  {"Expand All", &xmPushButtonGadgetClass, 'E', "Ctrl<Key>E",
"Ctrl+E",
   Expand_All_Of_The_Diagram, (Menu_Item *) NULL},
  {"Display Grid", &xmPushButtonGadgetClass, 'G', "Ctrl<Key>G",
"Ctrl+G",
   Toggle_Grids_Visibility, (Menu_Item *) NULL},
{"Display Boundaries", &xmPushButtonGadgetClass, 'B', "Ctrl<Key>B",
"Ctrl+B",
 Toggle_Boundaries_Visibility, (Menu_Item *) NULL},
  {"Display Edges", &xmPushButtonGadgetClass, 'x', "Ctrl<Key>x",
"Ctrl+x",
   Toggle_Edges_Visibility, (Menu_Item *) NULL},
{"Display Components", &xmPushButtonGadgetClass, 'o', "Ctrl<Key>o",
"Ctrl+o",
 Toggle_Components_Visibility, (Menu_Item *) NULL},
  NULL,
};


/*************************************************************
 * Defines the ZOOM menu options.
 *************************************************************/
Menu_Item Zoom_Menu_Option[] =
{
  {"In (*2)", &xmPushButtonGadgetClass, 'I', "Alt<Key>I", "Alt+I",
   Zoom_In_By_2, (Menu_Item *) NULL},
  {"Out (*2)", &xmPushButtonGadgetClass, 'O', "Alt<Key>O", "Alt+O",
   Zoom_Out_By_2, (Menu_Item *) NULL},
  {"Default Zoom", &xmPushButtonGadgetClass, 'D', "Alt<Key>D",
"Alt+D",
   Zoom_In_By_User_Defined_Percent, (Menu_Item *) NULL},
  NULL,
};


/*************************************************************
 * Defines the HELP menu items;
 *************************************************************/
Menu_Item Help_Menu_Option[] =
{
  {"About", &xmPushButtonGadgetClass, 'A', "Alt<Key>A", "Alt+A",
   Do_About_Dialog, (Menu_Item *) NULL},
  {"Program", &xmPushButtonGadgetClass, 'P', "Alt<Key>P", "Alt+P",
   Do_Program_Details_Dialog, (Menu_Item *) NULL},
  NULL,
};


/*************************************************************
 * Function : Creates a pull down menu, with a selected
 * Parent 'Widget'.
 *************************************************************/
Widget
Create_Pulldown_Menu (Parent, Menu_Title, Menu_Mnemonic, Items)
     Widget Parent;
     char *Menu_Title, Menu_Mnemonic;
     Menu_Item *Items;
```

```
{
  Widget Pull_Down, Cascade, Menu_Widget;
  int Index;
  XmString Text_String;

  Pull_Down = XmCreatePulldownMenu (Parent, "_pulldown", NULL, 0);

  Text_String = XmStringCreateSimple (Menu_Title);
  Cascade = XtVaCreateManagedWidget (Menu_Title,
                                     xmCascadeButtonGadgetClass, Parent,
                                     XmNsubMenuId, Pull_Down,
                                     XmNlabelString, Text_String,
                                     XmNmnemonic, Menu_Mnemonic, NULL);
  XmStringFree (Text_String);
  for (Index = 0; Items[Index].Label != NULL; Index++)
    {
      if (Items[Index].Sub_Items)
      Menu_Widget = Create_Pulldown_Menu (Pull_Down,
                                          Items[Index].Label,
                                          Items[Index].Mnemonic,
                                          Items[Index].Sub_Items);
      else
      Menu_Widget = XtVaCreateManagedWidget (Items[Index].Label,
                                             *Items[Index].Class,
                                             Pull_Down, NULL);
      if (Items[Index].Mnemonic)
      XtVaSetValues (Menu_Widget, XmNmnemonic, Items[Index].Mnemonic,
NULL);
      if (Items[Index].Accelerator)
      {
        Text_String = XmStringCreateSimple
(Items[Index].Accelerator_Text);
        XtVaSetValues (Menu_Widget,
                       XmNaccelerator, Items[Index].Accelerator,
                       XmNacceleratorText, Text_String, NULL);
        XmStringFree (Text_String);
      }
      if (Items[Index].Callback)
      XtAddCallback (Menu_Widget,
                     (Items[Index].Class == &xmToggleButtonWidgetClass
||
                    Items[Index].Class == &xmToggleButtonGadgetClass) ?
                     XmNvalueChangedCallback :
                     XmNactivateCallback,
                     Items[Index].Callback,
Items[Index].Callback_Data);
    }
  return Cascade;
}


/*************************************************************
 * Function : Creates the menu bar along with the pull down
 * menus.
 *************************************************************/
Widget
Create_Menu_Bar (Main_Window)
```

```
    Widget Main_Window;
{
  Widget Menu_Bar, Create_Pulldown_Menu ();
  Menu_Bar = XmCreateMenuBar (Main_Window, "menubar", NULL, 0);
  Create_Pulldown_Menu (Menu_Bar, "File", 'F', File_Menu_Option);
  Create_Pulldown_Menu (Menu_Bar, "Diagram", 'D',
Diagram_Menu_Option);
  Create_Pulldown_Menu (Menu_Bar, "Zoom", 'Z', Zoom_Menu_Option);
  Create_Pulldown_Menu (Menu_Bar, "Help", 'H', Help_Menu_Option);
  XtManageChild (Menu_Bar);
  return Menu_Bar;
}
```

M. "Mouse_Events.h"

```
/*************************************************************
 * File : Mouse_Events.h                                     *
 *************************************************************/
void
Display_The_Diagram (void)
{
  Set_Foreground_Colour (FOREGROUND_COLOUR);
  Set_Background_Colour (BACKGROUND_COLOUR);
  if (Output_To_FIG == TRUE)
    {
      fprintf (XFIG_File_Pointer,"#FIG 3.1\n");
      fprintf (XFIG_File_Pointer,"Portrait\n");
      fprintf (XFIG_File_Pointer,"Center\n");
      fprintf (XFIG_File_Pointer,"Inches\n");
      fprintf (XFIG_File_Pointer,"1200 2\n");
    }

  if (DISPLAY_GRID == TRUE)
    {
      Set_Foreground_Colour (GRID_COLOUR);
      Draw_Grid ();
    }
  if (DISPLAY_BOUNDARIES == TRUE)
    {
      Set_Foreground_Colour (BOUNDARY_COLOUR);
      Draw_Boundaries (The_Diagram);
    }
  else
    {
      Draw_Rectangle(The_Diagram->X - The_Diagram->Width1,
                     The_Diagram->Y - The_Diagram->Height2,
                     The_Diagram->X + The_Diagram->Width2,
                     The_Diagram->Y + The_Diagram->Height1, FALSE);
    }

  if (DISPLAY_EDGES == TRUE)
    {
      Set_Foreground_Colour (EDGE_COLOUR);
      Draw_Edges (The_Diagram);
    }
  if (DISPLAY_COMPONENTS == TRUE)
    Draw_Flowchart_Component (The_Diagram);

  Resize_Bitmap ();
}

/*************************************************************
 * Function : allows the user to click the mouse on the drawing
 * area. This will either expand or collapse a node.
 *************************************************************/
void
draw (widget, event, args, num_args)
     Widget widget;
     XButtonEvent *event;
```

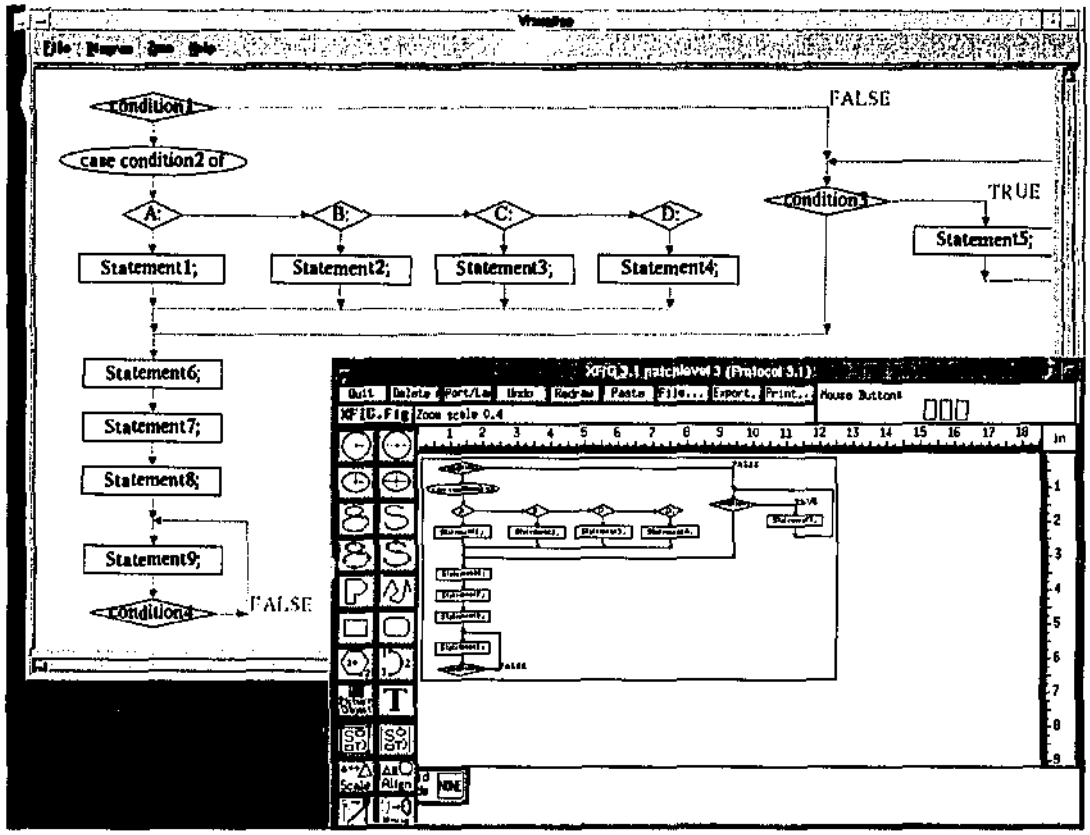
```
    String *args;
    int *num_args;

{
  SearchNode = NULL;
  FOUND = FALSE;
  X = event->x;
  Y = event->y;

  Node_Selected (X, Y, The_Diagram);
  if (FOUND == TRUE)
    {
      if (SearchNode->Expanded == TRUE)
      SearchNode->Expanded = FALSE;
      else
      SearchNode->Expanded = TRUE;
    }
  Adjust_Diagram_Layout (The_Diagram);
  Display_The_Diagram ();

}

/*************************************************************
 * Function : refreshes the drawing area.
 *************************************************************/
void
Draw_Bitmap (Drawing_Area, client_data, cbs)
    Widget Drawing_Area;
    XPointer client_data;
    XmDrawingAreaCallbackStruct *cbs;
{
  XCopyArea (
            XtDisplay (Drawing_Area), Users_Bitmap,
            cbs->window,
            The_Graphics_Context,
            0, 0, Screen_Width, Screen_Height, 0, 0);
}
```

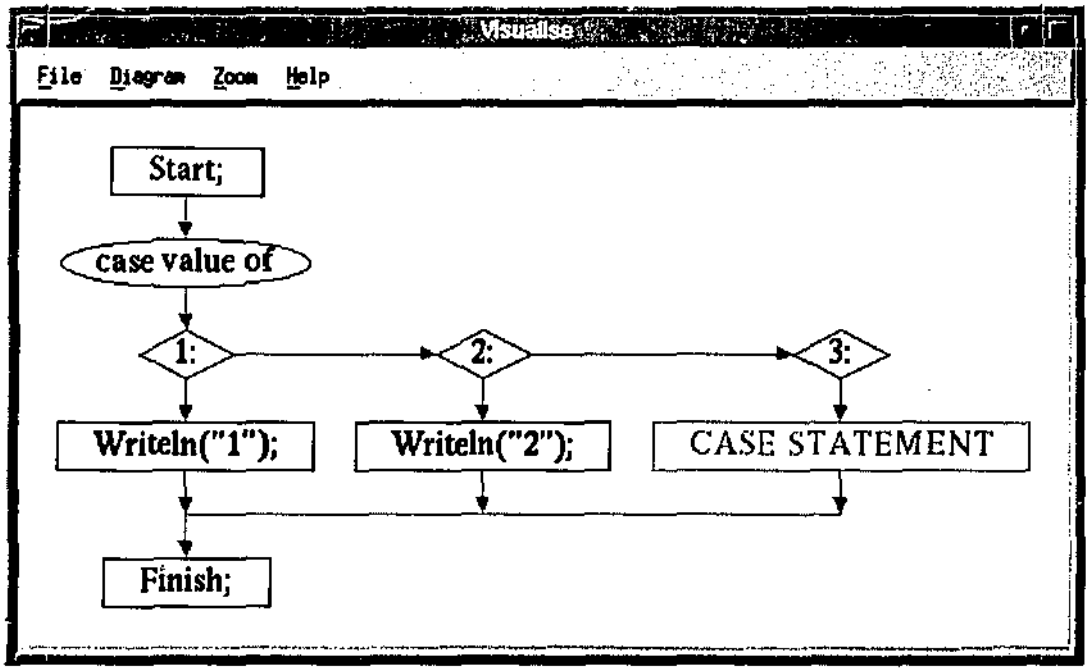
N. "User_Interface.h"

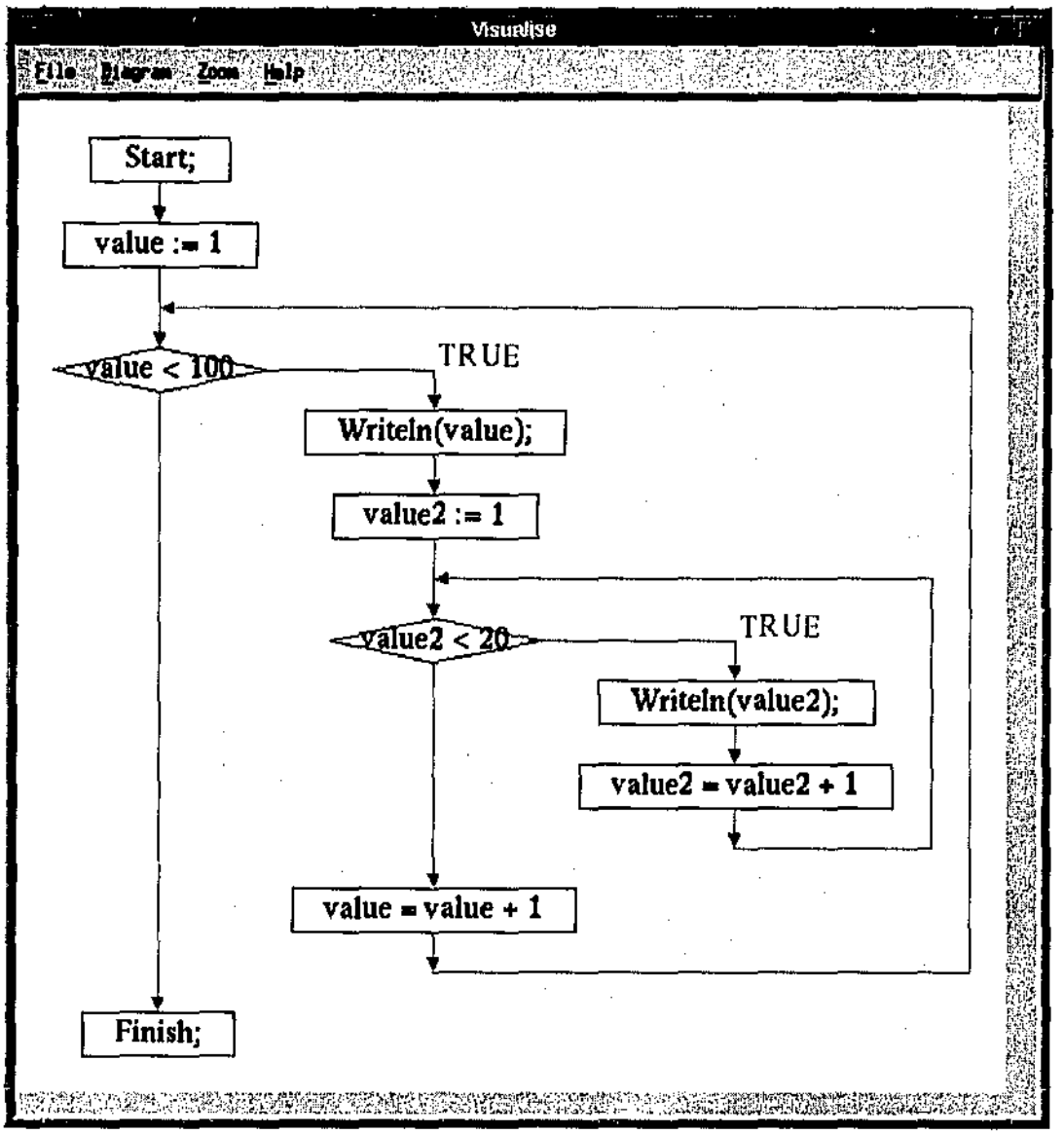
```
/*************************************************************
 * File : User_Interface.h                                   *
 *************************************************************/

Create_User_Interface (Argument_List_Count, Argument_List)
     int Argument_List_Count;
     char *Argument_List[];
{
  XtActionsRec actions;
  Font font;

  void draw (), Draw_Bitmap (), clear_it ();

  Screen_Width = Screen_Height = 0;
  Screen_Width = The_Diagram->Width1 +
    The_Diagram->Width2 +
    (2 * BORDER_DISTANCE_X);
  Screen_Height = The_Diagram->Height1 +
    The_Diagram->Height2 +
    (2 * BORDER_DISTANCE_Y);

  Top_Level = XtVaAppInitialize (&The_Application, "Software
Visualisation",
              NULL, 0, &Argument_List_Count, Argument_List, NULL,
NULL);
  Main_Window = XtVaCreateManagedWidget ("Main_Window",
                                xmMainWindowWidgetClass, Top_Level,
                                XmNscrollingPolicy, XmAUTOMATIC,
                                XmNwidth, Screen_Width + 4,
                                XmNheight, Screen_Height + 35,
                                NULL);
  menubar = Create_Menu_Bar (Main_Window);

  actions.string = "draw";
  actions.proc = draw;
  XtAppAddActions (The_Application, &actions, 1);
  Drawing_Area = XtVaCreateManagedWidget ("Drawing_Area",
                                xmDrawingAreaWidgetClass,
                                Main_Window,
                                XmNtranslations,
                             XtParseTranslationTable (translations),
                                XmNunitType, XmPIXELS,
                                XmNwidth, Screen_Width,
                                XmNheight, Screen_Height,
                                XmNresizePolicy, XmNONE, NULL);
  XtAddCallback (Drawing_Area, XmNexposeCallback,
            Redraw_Diagram_To_Bitmap, NULL);
  Users_Bitmap = XCreatePixmap (XtDisplay (Drawing_Area),
                        RootWindowOfScreen (XtScreen (Drawing_Area)),
                        Screen_Width, Screen_Height,
                        DefaultDepthOfScreen (
                                      XtScreen (Drawing_Area)));
  The_Graphics_Context = XCreateGC (XtDisplay (Drawing_Area),
```

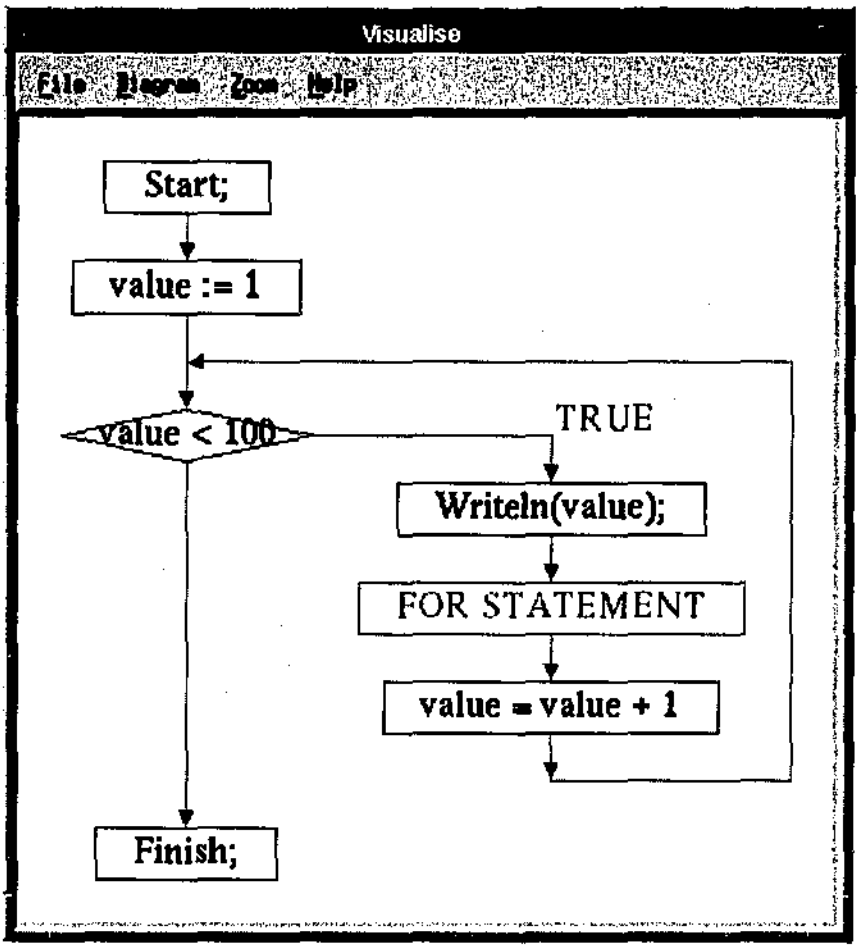
```
                        RootWindowOfScreen (XtScreen
(Drawing_Area)),
                                GCForeground, &Graphics_Context_Values);
  font = XLoadFont (XtDisplay (Drawing_Area), FONT_NAME);
  if (font != 0)
    XSetFont (XtDisplay (Drawing_Area), The_Graphics_Context, font);
  Font_Structure = XLoadQueryFont (XtDisplay (Drawing_Area),
FONT_NAME);
  Adjust_Diagram_Layout (The_Diagram);
  Screen_Width = The_Diagram->Width1 +
    The_Diagram->Width2 +
    (2 * BORDER_DISTANCE_X);
  Screen_Height = The_Diagram->Height1 +
    The_Diagram->Height2 +
    (2 * BORDER_DISTANCE_Y);
  Resize_Bitmap ();
  XtVaSetValues (Main_Window, XmNwidth, Screen_Width + 4,
            XmNheight, Screen_Height + 35, NULL);

  XtRealizeWidget (Top_Level);
  XtAppMainLoop (The_Application);
}
```

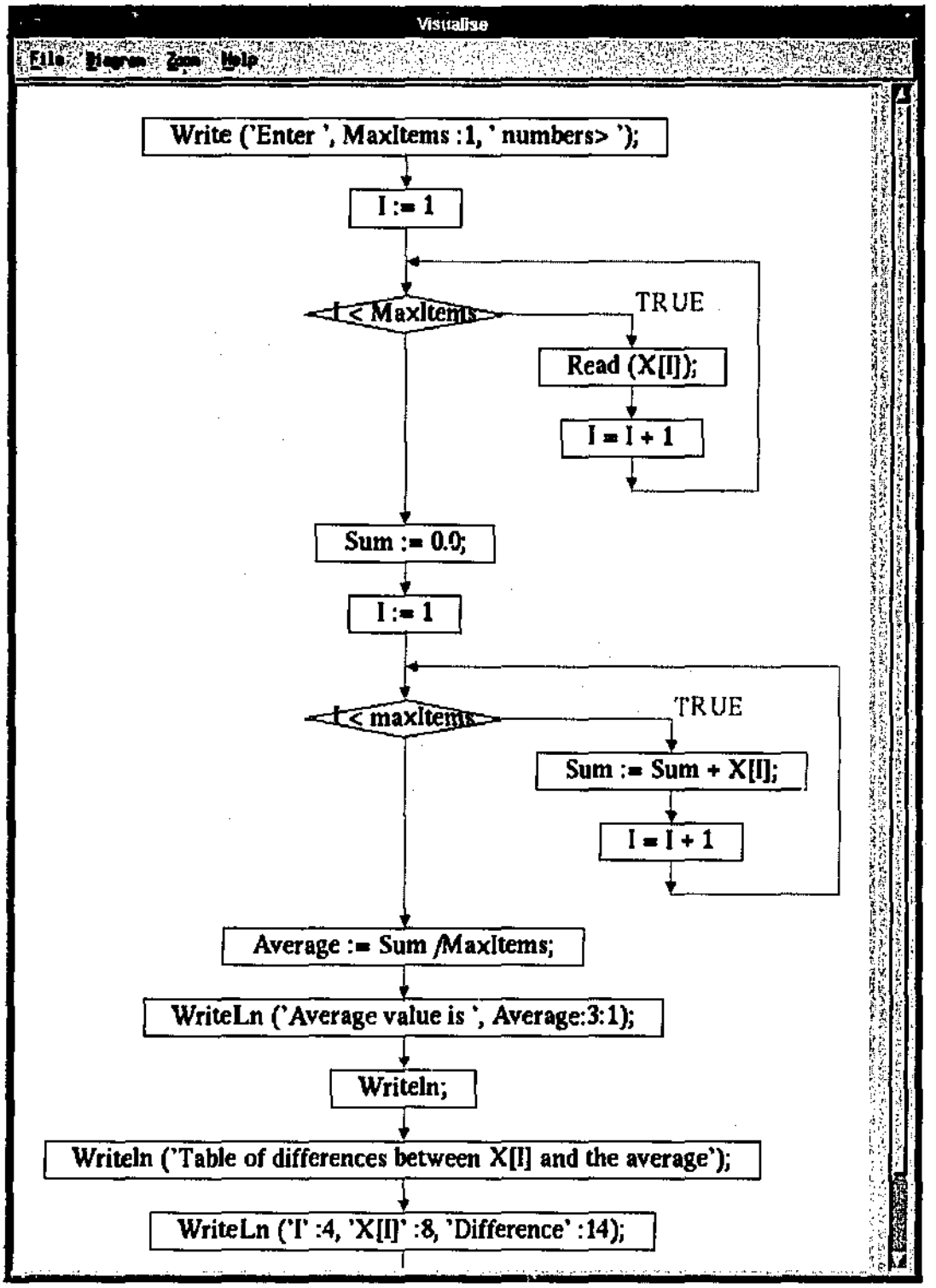
O. Sample Program Screen Dumps






Visualise
File Diagram Zoom Help
Start;
value := 1
value < 100
TRUE
Writeln(value);
value2 := 1
value2 < 20
TRUE
Writeln(value2);
value2 = value2 + 1
value = value + 1
Finish;


Visualise
File Diagram Zoom Help
Start;
value := 1
value < 100
TRUE
Writeln(value);
FOR STATEMENT
value = value + 1
Finish;


Visualise
File Diagram Zoom Help
Write ('Enter ', MaxItems :1, ' numbers> ');
I := 1
I < MaxItems
TRUE
Read (X[I]);
I = I + 1
Sum := 0.0;
I := 1
I < maxItems
TRUE
Sum := Sum + X[I];
I = I + 1
Average := Sum /MaxItems;
WriteLn ('Average value is ', Average:3:1);
Writeln;
Writeln ('Table of differences between X[I] and the average');
WriteLn ('I' :4, 'X[I]' :8, 'Difference' :14);

P. Pascal Program Cryptogram

```
{p384 Koffman}
program Cryptogram (Input, Output);


{Generates cryptograms corresponding to input messages.}
type
    Letter = 'A'..'Z';
    CodeArray = array [Letter] of Char;
var
    Code : CodeArray;
procedure ReadCode (var Code {output} : CodeArray);
{
    Reads in the code symbol for each letter.
    Pre : none
    Post : 26 data values are read into array Code.
}
var
    NextLetter : Letter;
begin {ReadCode}
    WriteLn ('First specify the code.');
    Writeln ('Enter a code symbol under each letter.');
    Writeln ('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
    {Read each code symbol into array Code.}
    for NextLetter := 'A' to 'Z' do
        Read (Code[NextLetter]);
    ReadLn;
    Writeln
end; {ReadCode}
procedure Encrypt (Code {input} : CodeArray);
```

```
{
    Reads each character and prints it or its code symbol.
    Pre : Array Code is defined.
    Post : Each character read was printed or its code
           symbol was printed and the sentinel was just read.
    Uses: Cap (see Fig. 9.19)
}
const
    Sentinel = '#';
var
    NextChar : Char;
begin {Encrypt}
    WriteLn ('Enter each character of your message:');
    WriteLn ('terminate it with the symbol', Sentinel);
    repeat
        Read(NextChar);
        NextChar := Cap(NextChar);
        if NextChar in ['A'..'Z'] then
            WriteLn (Code[NextChar]:5)
        else
            Writeln (NextChar:5)
    until NextChar = Sentinel
end; {Encrypt}
begin {Cryptogram}
    {Read in the code symbol for each letter.}
    ReadCode (Code);
    {Read each character and print it or its code symbol.}
    Encrypt (Code);
end. {Cryptogram}
```

Q. Pascal Program - Figures

```
{p 433 Koffman}

procedure GetFigure (var OneFig {output} : Figure);
{
    Defines tag field of OneFig.
    Pre : None
    Post : The tag field value corresponds to the next data character.
}
var
    FigChar : Char;
begin
    WriteLn ('Enter the kind of object');
    Write ('Enter C (Circle), R (Rectangle), or S (Squre)> ');
    Read (FigChar);
    if FigChar in ['c', 'C', 'r', 'R', 's', 'S'] then
        case FigChar of
        'c','C': OneFig.Shape := Circle;
        'r','R': OneFig.Shape := Rectangle;
        's','S': OneFig.Shape := Square
        end
    else
        OneFig.Shape := Other
end;
procedure ReadFigure (var OneFig {input/output} : Figure);
{
    Enters data into OneFig.
    Pre : The tag field of OneFig is defined.
    Post: The characteristics of OneFig are defined.
}
```

```
begin
    with OneFig do
    case Shape of
    Circle: begin
            Write ('Enter radius>');
            ReadLn (Radius)
            end; {Circle}
    Rectangle: begin
            Write ('Enter width >');
            ReadLn (Width);
            Write ('Enter height>');
            ReadLn (Height);
            end; {Rectangle}
    Square: begin
            Write ('Enter length of side>');
            ReadLn (Side)
            end; {sqare}
    Other : WriteLn (Characteristics are unknown')
    end
end;
procedure ComputePerim (var OneFig {input/output} : Figure);
{
    Defines Perimeter field of OneFig.
    Pre: The tag field and characteristics of OneFig are defined.
    Post: Assigns value to Perimeter field.
}
begin {ComputePerim}
    with OneFig do
    case Shape of
        Circle : Perimeter := 2.0 * Pi * Radius;
```

```
            Rectangle : Perimeter := 2.0 * (Width + Height);
            Square : Perimeter := 4.0 * Side;
            Other : Perimeter := 0.0
        end {case}
    end;
    procedure ComputeArea (var OneFig {input/output} : Figure);
    {
        Defines Area field of OneFig.
        Pre : The tag field and characteristics of OneFig are defined.
        Post : Assigns value to Area field.
    }
    begin {ComputeArea}
        with OneFig do
        case Shape of
        Circle: Area := Pi * Radius * Radius;
        Rectangle: Area := Width * Height;
        Square: Area := Side * Side;
        Other: Area := 0.0
        end
    end
    procedure DisplayFig (OneFig {input} : Figure);
    {
        Display the characteristics of OneFig.
        Pre : All fields of OneFig are defined.
        Post : Displays each field of OneFig.
    }
    begin {DisplayFig}
        with OneFig do
        {Display shape and characteristics}
        begin
```

```
        Write ('Figure shape is ');
        case FigShape of
        Circle : begin
                WriteLn ('Circle');
                Writeln ('Radius is '), Radius :4:2)
                end;
        Rectangle : begin
                WriteLn ('Rectangle');
                WriteLn ('Height is ', Height :4:2);
                Writeln ('Width is ', Widht :4:2);
                end; {Rectangle}
        Square: begin
                Writeln ('Square');
                Writeln ('Side is ', Side :4:2)
                end;
        Other: WriteLn ('No characteristics for figure')
    end; {case}
    {Display area and perimeter}
    WriteLn ('Area is ', Area:4:2);
    WriteLn ('Perimeter is ', Perimeter :4:2)
    end {with}
end {DisplayFig}
```

R. Pascal Program ShowDiff

```
{Koffman, p 362}
program ShowDiff (Input, Output);
{
 Computes the average value of an array of data and
 prints the difference between each value and the average.
}
const
    MaxItems = 8;
type
    IndexRange = 1..MaxItems;
    RealArray = array [IndexRange] of Real;
var
    X: RealArray;
    I: IndexRange;
    Average,
    Sum : Real;
begin
    {Enter the data}
    Write ('Enter ', MaxItems :1, ' numbers> ');
    for I := 1 to MaxItems do
        Read (X[I]);
    {Compute the average value.}
    Sum := 0.0;
    for I := 1 to maxItems do
        Sum := Sum + X[I];
    Average := Sum / MaxItems;
    WriteLn ('Average value is ', Average:3:1);
    Writeln;
```

```
    {Display the difference between each item and the average.}
    Writeln ('Table of differences between X[I] and the average');
    WriteLn ('I' :4, 'X[I]' :8, 'Difference' :14);
    for I := 1 to MaxItems do
        WriteLn (I :4, X[I] :8:1, X[I]-Average :14:1)
end. {ShowDiff}
```

S. Worked Example – Nested IF Component converted to a flowchart.

This example is a detailed description of the process followed by the prototype in order to generate a flowchart. Below is a sample piece of code that will be converted to a flowchart by the prototoype:

Nested If Component

```
Start;
If c1 then
    Begin
        Writeln ("c = true")
        If C2 then
            Writeln ("c2 = true");
        Else
            Writeln ("c2 = false");
    End
Else
    Writeln ("c = false");
Finish;
```

To create a flowchart from the source code requires the following procedure to be applied.

Step 1 – Parse source code

Step 2 – Add extra diagram information

Step 3 – Apply abstract layout information.

Step 4 – Assign node sizes.

Step 5 – Apply force-scan algorithm.

Step 6 – Display Routines.

A detailed explanation for each step will be given to help understand the conversion process.

Step 1 – Parse source code

Parsing the source code will produce a structure as shown below:

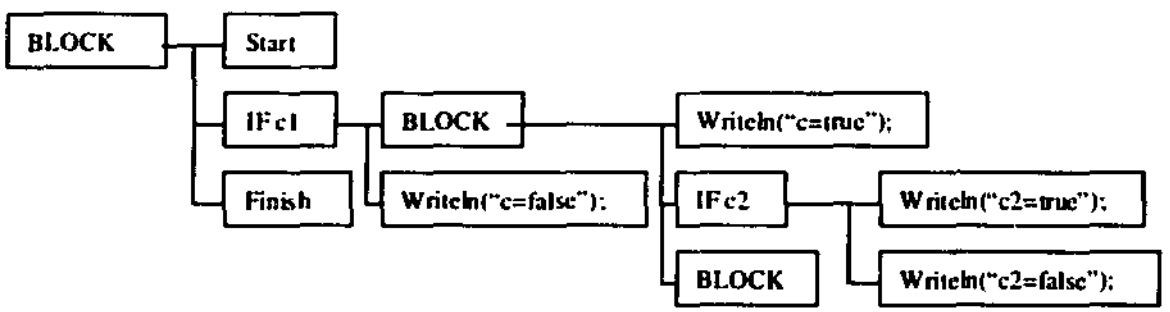


Figure 105 Sample Structure after parsing source code.

NOTE: The first node shown is a BLOCK component. This is the root node for the data structure.

Step 2 – Add extra diagram information

Now that we have stored the information about the source code we can add extra diagram information such as symbols and diagram edges. The addition of diagram information is demonstrated below:

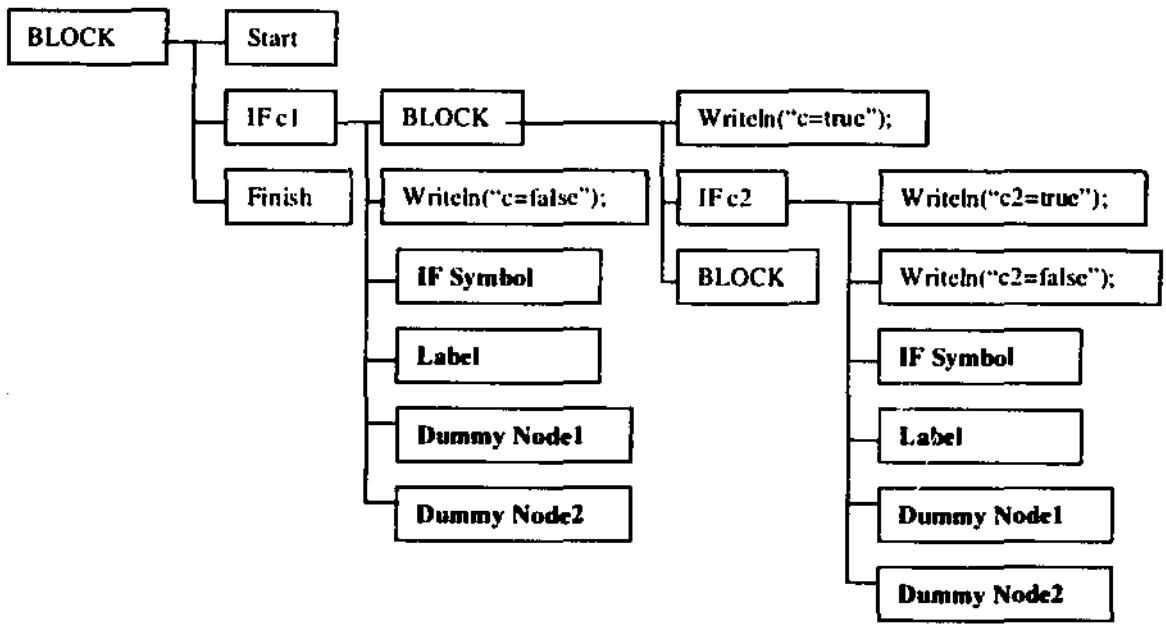


Figure 106 Sample Structure with extra diagram information added. NOTE: Extra nodes have text bolded.

To add extra diagram information the program would process each individual node starting at the root node and then work its way to the farthest descent of the root node. The first node encountered would be the *BLOCK* node. As a block node does not require any extra nodes only edge information would be added by joining the nodes *"Start" to "If c1" to "Finish"*.

Next the program would process the "IF c1" node. Here it is required to add 4 extra nodes which include an IF Symbol, a Label, and 2 Dummy Nodes. The edge information is more complicated for this node but follows the same principle as the *BLOCK* node. The edges would be created as follows:

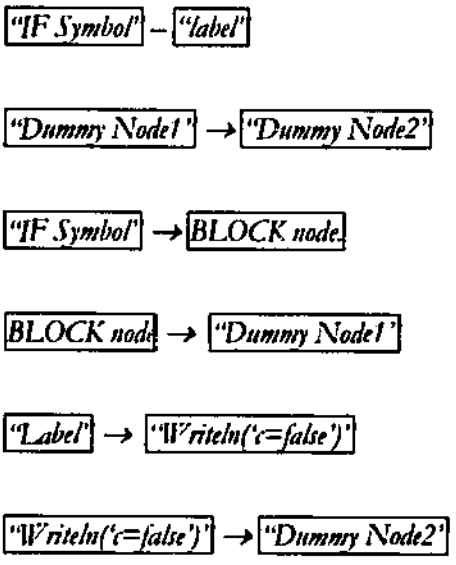


Figure 107 Sample edge table information for "IF C1" node.

This process is repeated recursively until every node within the tree structure has been processed with node and edge information.

Step 3 – Apply abstract layout information.

The application of abstract layout information involves describing each node in relation to other sibling nodes. For example the root node being a block component will have its child node positions described as:

Column("Start", "IF c1", "Finish")

Row("Start")

Row("IF c1")

Row("Finish")

Figure 108 Sample abstract layout information for root node.

NOTE: This example has been updated to use the functions ROW and COLUMN to generate the layout. The use of these functions simplifies the definition of the layout process as well as making the layout easier to understand.

The first IF component located from the root node will have a abstract layout as follows:

Row("IF Symbol", "Label")

Row(A, B)

Row("Dummy Node1", "Dummy Node2")

Column("IF Symbol", A, "Dummy Node1")

Column("Label", B, "Dummy Node2")

Figure 109 Sample abstract layout description for first IF Statement.

Step 4 – Assign node sizes.

For our example the set of nodes that have a definable size include all general statements, *IF*-symbols and labels. In our example the general statements include all Writeln statement. To determine the size of these nodes the functions TextWidth and TextHeight are used. The output device will determine amount for these values. Also the style of font will determine the overall size for these nodes. The same process to determine a general statement is applied to all symbols within the diagram.

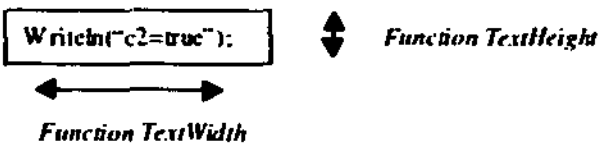


Figure 110 Sample general statement with the text width and the text height indicated.

Step 5 – Apply force-scan algorithm.

The Force-Scan algorithm is the most important step as it generates all geometric information such as size and location for each individual node. In our example the force scan algorithm would be applied to the farthest descendant of the root node and then applied in a direction toward the root node.

The child nodes of the statement "IF c2" would be positioned according to where the force scan algorithm places them. Once this set of nodes has been processed with the Force-Scan algorithm the node size for the "IF c2" statement can be set and its sibling nodes can now have the Force-Scan algorithm applied. This process continues for all nodes until the location and size of every node is determined.

The order in which the Force-Scan algorithm is applied can be seen by the following diagram:

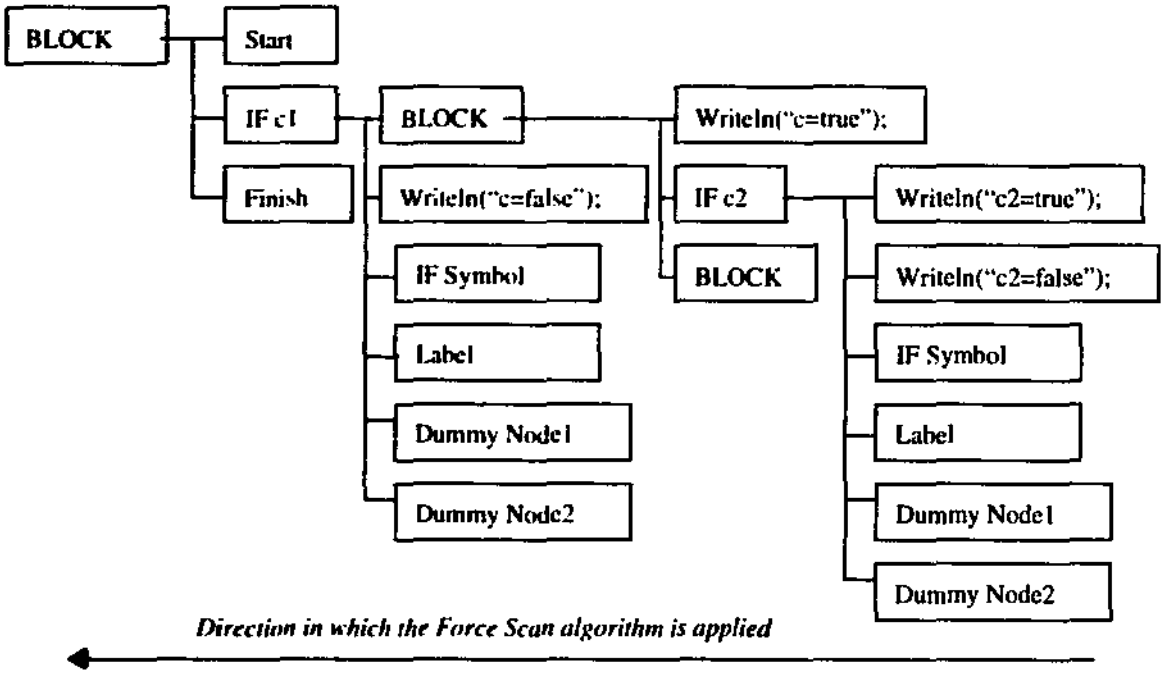


Figure 111 Direction in which force scan is applied to the data structure

Step 6 – Display Routines

With a set of primitives we can draw lines and text to the output device. This set of primitive display routines may include functions such as DrawRectangle, DrawLine, DrawArrow, and DrawText.

Further definitions of display routines would include the ability to draw IF and CASE symbols as well as the ability to draw individual components and diagram edges.

The final flowchart as viewed onscreen:

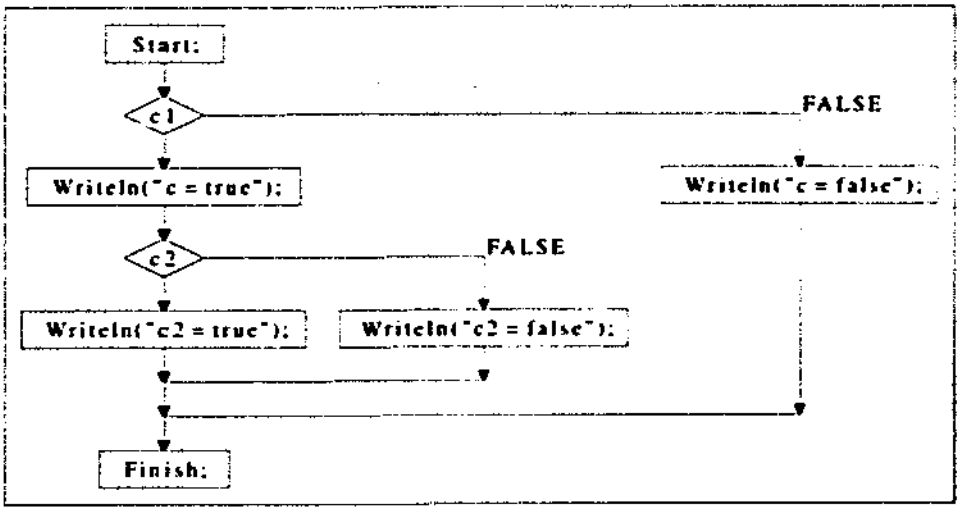


Figure 112 Final flowchart as display by the prototype.