NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

AN ALGORITHM DIRECTED COMPUTER AIDED SOFTWARE ENGINEERING (CASE) ENVIRONMENT FOR C

by

Messaouda Ouerd

A thesis submitted to the School of
Graduate Studies and Research in
partial fulfillment of the requirements
for the degree of Master of Computer Science

Department of Computer Science
University of Ottawa

Ottawa-Carleton Institute for Computer Science

ISBN 0-315-62306-3

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis. I authorize University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Messaouda Ouerd

I further authorize University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Messaouda Ouerd

ABSTRACT

The objectives of computer aided software engineering (CASE) systems are to improve productivity during the software development process and the quality of software using software engineering concepts via automation of the software development life cycle. This will result in a reusable software and will decrease the cost and time of software development and maintenance.

The main concern in this thesis is with describing the features of a particular software understanding environment for C. An algorithm directed computer aided software engineering environment for C language has been developed and implemented. The system has been implemented on a Sun Workstation using the Sunview window interface. It provides computer aided software engineering tools which :

1) Assist the user in developing structured algorithms for procedural languages
2) Automatically transform a structured algorithm into a corresponding program
3) Redocument the resulting C program (or any C program developed using any other technique) in an organized representation.

ACKNOWLEDGEMENTS

I am grateful to my supervisor, Dr. Tuncer I. Ören, for all the guidance and advice he has given to me throughout my graduate studies.

I wish to thank Douglas G. King for his useful comments and the stimulating discussions.
I would like, also, to thank with gratitude the government of Algeria for the financial support.

My husband Arab deserves the most thanks for his infinite encouragement, moral support, patience, and understanding.

Finally, I thank my parents for encouraging me throughout the years of my studies.

TABLE OF CONTENTS

DETAILED TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 AIMS AND STRUCTURE OF THE THESIS

Computer technology is developing faster than any technology, with gain in price / performance. " Computer speed and power are now increasing at about thirty percent a year, but software development productivity is increasing only four to seven percent a year." (Perrone, Marietta, 1987, p.104) The software systems of today represent new levels of power and complexity that greatly exceed the capabilities of traditional development process. The task of developing and maintaining new software is difficult to manage, making it the critical task of new systems development, and an important issue in software engineering.

Computer aided software engineering tools are designed to automate most of the tasks in the software engineering life cycle. Better analysis leads to more effective design, easier programming, fewer testing errors, more success during implementation, and reduced maintenance. Automation of the software engineering process improves productivity, reduces costs, and results in higher quality software. Usually, the system's maintainers were not its designers, so the need for clarification, enhanced understanding, and migration of existing software is a real problem. The purpose of software understanding is to establish a basis upon which to carry out software maintenance. The software understanding is a term for a family of related concepts that are concerned with providing an enhanced perception of an existing software product. These concepts are formulated in terms of different representations of the software product at various levels of abstraction. Concepts such as reverse engineering, redocumentation and restructuring have become identified with various types of review carried out on an existing software product. Reverse engineering enables information systems to extract information from old applications and use them as the basis for maintaining those

applications. Reverse engineering does not involve changing the subject system. It is a process of examination, not change or replication.

The algorithm directed computer aided software engineering system, outlined in this thesis, is a forward and reverse engineering environment for C. The functionality provided by the system has its origin in the generic problem of software understanding that has emerged within the broad domain of software engineering. The system provides a subset of the features required to perform software engineering (reverse engineering and forward engineering.) Although the CASE tool described in this thesis has general applicability, its features are particularly useful for the novice programmer ; i.e., the system provides a very effective pedagogical tool.

The thesis consists of four parts. The first part is Chapter 2 which provides a description of computer aided software engineering (CASE) concepts and the need for CASE tools. The second part is Chapter 3. This part gives some existing graphical representations of algorithms. The third part consists of Chapter 4. In this part, a graphical scheme for algorithms is presented and discussed in detail. The last part consists of Chapter 5 to Chapter 7. It provides the full description of the implementation of the algorithm directed computer aided software engineering system.

1.2 ARCHITECTURE OF THE ALGORITHM DIRECTED CASE ENVIRONMENT SYSTEM

The algorithm directed CASE environment is a software environment in which algorithms and C programs may be created, edited and documented in a structured way. Structural elements of the C language are represented in the form of the graphic scheme developed by Ören(1984). The system is implemented in C language, under UNIX SunOS 4.03 operating system, using Sunview functions. A key component of the system is the editor, which enables structural features of C to be manipulated in terms of their graphic scheme " box" representation.

Another key feature of the system is the transformation of the algorithm into its corresponding procedural C program which is displayed in a parallel window to the algorithm window. As depicted in Figure 1.1, the elements of the system environment consist of the following main components.

ALgorithm Editor (ALC) :

The algorithm editor automates the process of editing structured algorithms. Templates can be selected from a menu of templates. They are viewed through the menu window and inserted in the text window. The system (ALC) provides a comprehensive set of services for manipulating templates. Other functions that can be called from the menu window include removing templates (clear the text window), saving a block of templates as a file in the algorithm text and print any algorithm already edited.

Object Code Editor (CALC) :

The object code editor assists a user in transforming an algorithm into corresponding code. The output from CALC is functional and test case source code that can be verified and compiled into machine code. Checking, detecting a number of bugs and obscurities is part of lint (program verifier for C) process.

Organized Representation of C Programs (ORC) :

ORC, organized representation of C programs is a software tool to document correct C programs. The documentation which is automatically generated by ORC is a structured flowchart of the C programs. Text window is used to display documented files. Menu window allows the users to choose the parameters for drawing the outputs for the user taste.

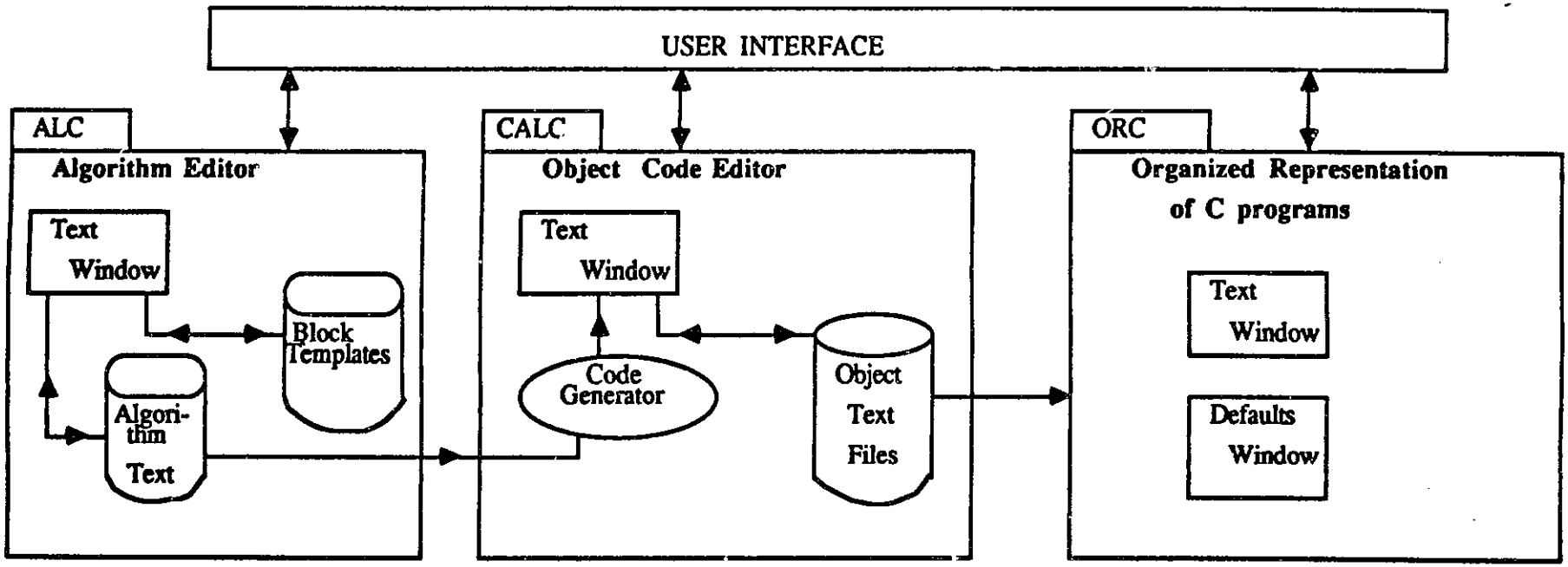


Figure 1.1 Architecture of the Algorithm Directed CASE Environment

Chapter 2

COMPUTER AIDED SOFTWARE ENGINEERING (CASE)

2.1 BACKGROUND

2.1.1 SOFTWARE DEVELOPMENT LIFE CYCLE

Software development follows a planned life cycle which can be generalized with the following six phases : analysis, design, coding, testing, implementation, and maintenance.

In the analysis phase, the requirements (specification of the problem being solved, including objectives, constraints, and business rules) are determined and formally documented. In this phase, alternative solutions satisfying the constraints are tested. A functional specification and a logical model for the best or a feasible solution are generated.

In the design phase, the specified requirements are used to develop the detailed specification for a selected solution, including diagrams relating all programs, subroutines and data flow. The software design is the product of this phase.

The coding or programming phase uses the specification of the solution. Listings and operational manuals will result.

In the testing phase, the software is verified that it satisfies all the requirements already mentioned in the analysis phase.

The implementation phase corresponds to the installing of the software. In this phase, the software, operational manuals, and all required documentation are delivered to the customer. The result of the implementation stage is the final system.

Maintenance phase which includes repair, modification and enhancement of the software for its remaining life is the longest phase of the software development life cycle, since the remaining life of the software is many times the time required to produce the software.

2.1.2 WHAT IS CASE ?

CASE (Computer Aided Software Engineering) includes :

- Software automation
- Combination of tools and methods
- Repackaging of structured concepts
- Redefinition of software environment : tools, methods, hardware, management.

CASE (Computer Aided Software Engineering) refers to the automation of a specific software engineering task or to a complete environment that automates most of the tasks in the software engineering life cycle. The goal of CASE technology is to solve the problems resulting from systems development through automation from analysis of the system to the maintenance.

As expressed by Chikofsky and Rubenstein (1988, p.11) " CASE lets systems analysts document and model an information system from its initial user requirements through design and implementation and lets them apply tests for consistency, compactness, and conformance to standards".

A CASE tool is any software tool which can provide automated assistance in the analysis, design, coding, and maintenance of software systems.

2.1.3 EXAMPLES OF CASE TOOLS

- Specification languages and diagramming techniques are examples of Analysis / Design tools.
- Code generators and testing tools are examples of Implementation tools.
- Reverse engineering tools, reengineering tools, redocumentation tools, and program analyzers are examples of maintenance tools.

2.1.4 THE NEED FOR COMPUTER AIDED TOOLS

" The rapid pace at which hardware innovations are announced, particularly in the area of microprocessor technology, now well exceeds the capabilities of our software development technology ... An entire generation of processor hardware technology has arrived and been superseded without any software to support it reaching the marketplace". (Chikofsky and Rubenstein, 1988, p. 11-12).

The factors which require for computer assistance in the process of software development, as discussed by Shuler (1987, p.7-8) are complexity, consistency, diagnostics and prompting, efficiency, and maintainability of documentation.

Complexity : The development of even a relatively small system requires the consideration of enormous amounts of detail which is often impractical to maintain and evaluate without automated assistance.

Consistency : CASE tools provide a framework for a consistent application of the selected approach across a large number of project participants.

Diagnostics and Prompting : CASE tools provide analysts with diagnostics to improve their work and prompting to help assure their work is consistent and complete.

Efficiency : Computer assistance increases the analyst's efficiency by automating some of the more routine aspects of the structured analysis and design process.

Maintainability of Documentation : Since the system's documentation is prepared by automated techniques, it can be more easily maintained than manually prepared documentation.

2.1.5 THE NEED FOR TOOL INTEGRATION

The tools in a CASE environment should be integrated so that information entered using one tool should become available in all other tools that need it, regardless of their different views or media (e.g., graphic or text). Three potential advantages to this are apparent :

- First, the efficiency and productivity of CASE environment users will be improved. Tool users should not have to re-enter information already captured at an earlier phase with a different tool.
- Second, eliminating redundant user input yields improved consistency between the data stored by different tools with resulting reduction in errors.
- Third, view translation could be provided between tools, without requiring any user input other than selecting the tool with the desired view. This would allow the user to select the tool with the most appropriate view.

The method for integrating tools has long been known to be a problem of importance. Typically, this is thought to be best addressed through a central (common) database for all tools in the environment. A CASE database can be viewed as having a scheme (both logical and physical). So, the various tools needed in a CASE environment may have different views of the object's information system (IS). These views can each be considered to be an IS

model. When two IS model's views are logically the same (i.e., they consider the exact same aspects or qualitities of the IS to be relevant and structure them the same way), we can say that the difference between them is their physical form. For example, an algorithm (a certain semantics) can be represented using pseudo-code in different languages (for example French and English). Representing the algorithm in those languages are two different syntaxes for expressing the same thing (an algorithm). The semantics of the algorithm is the same, only its representation is different.

2.2 OBJECTIVES OF CASE SYSTEMS

CASE technology changes the way we build software systems by automation of the software engineering process which improves productivity, minimizes the total cost of the system, and eliminates many software development and maintenance tasks.

The objectives of CASE systems (McClure, 1988a, p. E1) are :

- Automate software development
- Visual / graphical programming
- Interactive development style
- Minimize the total cost of the system
- Automate generation of documentation
- Automate generation of code
- Automate error checking
- Automate project management

- Improve software quality
- Improve productivity
- Speed up software development
- Formalize software documentation
- Standardize software documentation
- Promote greater control of software development
- Integrate development steps and tools
- Promote software reusability
- Improve software portability

2.3 PARTS OF CASE

The types of CASE can be grouped into :

- Computer Aided Software Forward Engineering
- Computer Aided Software Reverse Engineering
- Computer Aided Software Reengineering

The relationships between these terms is explained in Figure 2.1 (Chikofsky and Cross, 1990, p. 14).

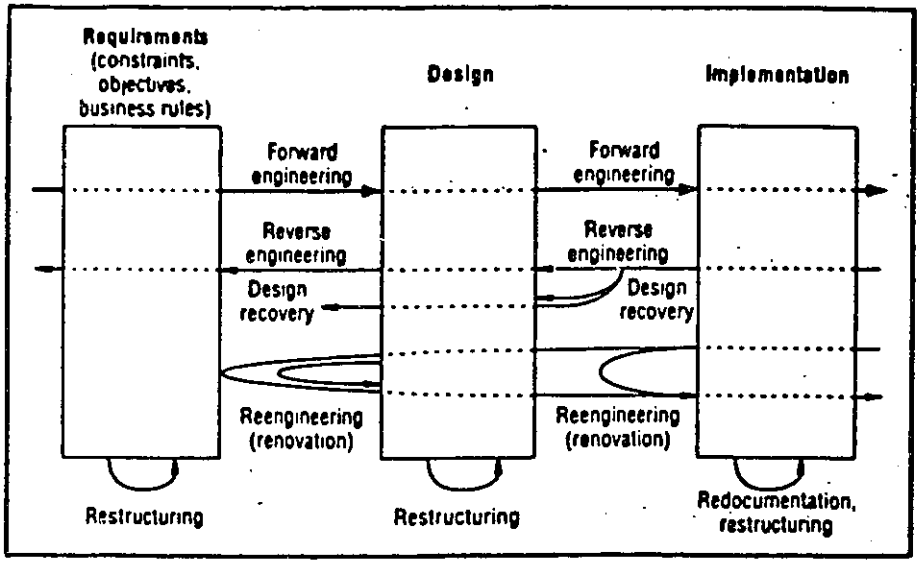


Figure 2.1 The relationships between the CASE terms

2.3.1 COMPUTER AIDED SOFTWARE FORWARD ENGINEERING

" Forward Engineering is the traditional process of moving from high level abstractions and logical, implementation independent designs to the physical implementation of a system." (Chikofsky and Cross, 1990, p. 1). Forward engineering is the process of progressing from requirements (specification of the problem, constraints, and business rules) to the design then to the implementation. Forward engineering translates the "what" specification that defines an application into the "how" of its physical representation.

2.3.2 COMPUTER AIDED SOFTWARE REVERSE ENGINEERING

Reverse engineering " is the process of analyzing a subject system to :

- Identify the system's components and their interrelationships, and
- Create a representation of the system in another form or at a higher level of abstraction." (Chikofsky and Cross, 1990, p. 1).

In software systems, the approaches , or the concepts of reverse engineering apply to gain a basic understanding of a system and its structure. This is very important to obtain a sufficient design level understanding to aid maintenance and support. Reverse engineering tools can help the system's maintainers (who usually are not the designers) to examine and get information about the software product so they can make appropriate changes if needed, or to adapt the product to a different environment.

Beyond increasing comprehensibility of the system the major functions of software reverse engineering include the following functions :

- Generation of alternate representation (graphical and non-graphical) which refers to redocumentation.
- Extracting existing knowledge from a program, and recovering lost knowledge i.e., software elucidation.
- Transformation and recasting of code, data, design / algorithm or requirement ; i.e., software restructuring.

2.3.3 COMPUTER AIDED SOFTWARE REENGINEERING

" Reengineering, also known as both renovation and reclamation, is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form." (Chikofsky and Cross, 1990, p. 1). Reengineering involves or includes forward and reverse engineering. In fact, to achieve an abstract description of the system we need to use reverse engineering and to modify any mechanism, we need also forward engineering.

Figure 2.2 (Bachman, 1988, p. v55) shows the chart of reengineering cycle. It provides an architectural view of CASE with forward and reverse engineering.

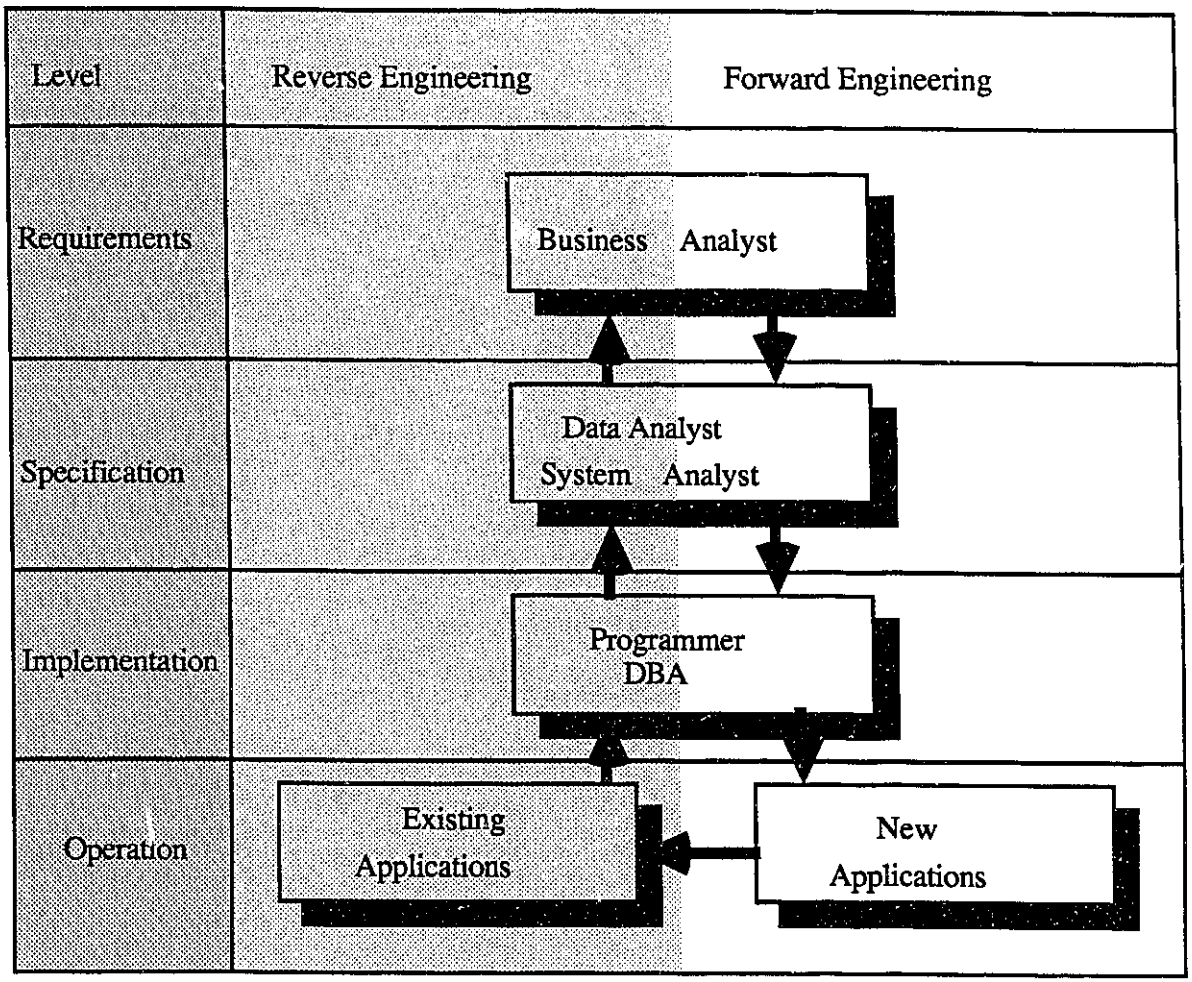


Figure 2.2 Reengineering Cycle (Bachman, 1988, p. v55)

In Figure 2.2 Forward engineering starts at the right top and progress from the most conceptual level (requirements, design) to the most physical level (machine instruction) at the bottom.

Across the horizontal axis, reverse engineering starts at the left bottom with the definition of existing applications and raises the applications to successively higher levels of abstractions.

The reengineering cycle described reflects the continuity of applications systems and their revisions over time. Each time, when the design objects created by the reverse engineering steps are validated and become the revised design objects, they are used in the forward engineering process. As shown in Figure 2.2, new applications become existing applications. This ensure the long life of a system.

" CASE products based on this CASE reengineering life cycle will have tremendous impact on the evolution of IS practices and on the businesses that depend on computer aided application systems for success. Three year projects will become three months projects. Being able to do what is needed today frees IS to make changes as the business environment evolves, rather than trying to predict three years ahead. A half dozen short term changes can offer a business more than one grand leap, which is often misdirected and seldom well executed". (Bachman, 1988, p. V57)

Chapter 3

ALGORITHMS AND THEIR GRAPHICAL REPRESENTATIONS

3.1 ALGORITHMS AND THEIR MAIN FEATURES

A computer program can be viewed as a complex object that contains a large amount of detailed information. This detailed information is necessary even if it obscures the structure of the program. In order to write a program or to understand a program, a guide to its organization and purpose is needed. Thus, the first step is to express our solution in an abstract way at first, omitting the details.

There are several programming paradigms. In the procedural approach, a problem can be solved by applying a special set of instructions. Furthermore, the order of these instructions is known a priori. The method being used to solve the problem is embodied in the algorithm. The algorithm is an abstraction of the actual computer program. As such, it can be studied without referring to any particular computer, programming language, compiler, etc. When we design an algorithm to solve a particular problem, we want to know how much resources, i.e. time (the number of steps required until the algorithm terminates) and space (the amount of memory required to implement the algorithm) an implementation will consume. Mathematical methods are used to predict the time and space needed by an algorithm, and this does not require implementation of the algorithm. This is important for several reasons. The most important is that we can save work by not having to implement algorithms in order to test their suitability.

The meaning for an algorithm is quite similar to that of *recipe* , *process* , *method* , *technique* , *procedure* , or *routine* (Knuth, 1973, p.1.) An algorithm is a finite set of rules which gives a sequence of operations for solving a specific problem. An algorithm has five important features (Knuth, 1973, p.2.)

These features will be highlighted using the Euclid's algorithm for finding the greatest common divisor of two positive integers.

Algorithm E (Euclid's algorithm) (Knuth, 1973, p.2.) Given two positive integers m and n, find their greatest common divisor, i.e., the largest positive integer which evenly divides both m and n.

E1. [Find remainder.] Divide m by n and let r be the remainder.
(We will have $0 \leq r < n$).

E2. [Is it zero ?] If $r = 0$, the algorithm terminates ; n is the answer.

E3. [Interchange.] Set m <-- n, n <-- r, and go back to step E1.

The five features of an algorithm are clarified below :

1) Finiteness. An algorithm must always terminate after a finite number of steps. It is composed of steps. Each step must be well defined ; that we can program a machine to carry it out , if necessary. Algorithm E satisfies this condition, because after step E1 the value of r is less than n, so if $r = 0$, the value of n decreases the next time that step E1 is encountered. A decreasing sequence of positive integers must eventually terminate, so step E1 executed only a finite number of times for any given original value of n.

"A procedure which has all of the characteristics of an algorithm except that it possibly lacks finiteness may be called a computational method. Besides his algorithm for the greatest common divisor of two integers, Euclid also gave a geometrical construction that is essentially equivalent to Algorithm E, except it is a procedure for obtaining the greatest common measure of the lengths of two line segments ; this is a computational method that does not terminate if the given lengths are incommensurate." (Knuth, 1973, p.5)

Another example of computational procedure is to print positive integers. (Grogono and Nelson, 1982, p.83). This procedure is not required to terminate. Its steps are :

1. Set N to zero
2. set N to N+1
3. Print N and go back to step 2 .

2) **Definiteness.** In Algorithm E, the criterion of definiteness as applied to step E1 means that we are supposed to understand exactly what it means to divide m by n and what the remainder is, and make sure that the values of m and n are always positive integers whenever step E1 is to be executed.

3) **Effectiveness.** An algorithm is expected to be effective. This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can be done in a finite length of time. Algorithm E uses only the operations of dividing one positive integer by another, testing if an integer is zero, and setting the value of one variable equal to the value of another.

4) **Input.** An algorithm has zero or more inputs, which are taken from specified sets of objects. In Algorithm E, the values of the input data for which the algorithm is valid are stated: m and n must be positive, nonzero integers.

5) **Output.** An algorithm has one or more outputs, i.e., quantities which have a specified relation to the inputs. The sequence of control is well-defined and it is always clear which is the next step to be executed and when the final step is executed, we have obtained the required result. Algorithm E has one output, namely n in step E2, which is the greatest common divisor of the two inputs.

An algorithm is therefore, a set of rules which gives a sequence of operations for solving a specific problem and has five features which are : Finiteness, definiteness, effectiveness, input, and output.

3.2 SOME GRAPHICAL REPRESENTATION SCHEMES FOR ALGORITHMS

We present several graphical techniques for representing and communicating algorithms. The idea behind the " pictures " we will draw is that the alternative sequence of processing steps will be better distinguished graphically, and the " shape " or structure of the algorithm better displayed. Given an appropriate diagramming technique, it is much easier to describe complex activities and procedures in diagrams than in text. A picture can be much better than a thousand words because it is concise, precise, and clear. There is a new and very important reason for diagramming. The job of systems analysts and builders is evolving from a pencil and paper activity to an activity of computer aided design. This change will improve the productivity of systems builders and increase the quality of the systems they build.

In the sequel the following graphical representation schemes are discussed : Decomposition diagrams, dependency diagrams, hipo diagrams, Warnier Orr diagrams, strucured English, flowcharts, and NS charts.

3.2.1 DECOMPOSITION DIAGRAMS

Decomposition diagrams are used to show organization structures, system structures, program structures, and report structures. High activities are decomposed into lower level activities showing more detail. This top down structuring makes complex organizations or processes easier to comprehend. Decomposition diagrams are a basic tool for structured analysis and design. Most decomposition diagrams are simple tree structures.

The term activity means *function* , *process* , or *procedure* . *Functions* refer to major areas of activity in a corporation, engineering, production, research, and distribution.

The term *process* refers to an activity without describing the mechanisms by which it is accomplished. The process does not indicate the precise method by which the results are accomplished.

Procedure refers to a specific method of accomplishing the process ; It refers to the design carried out by a system analyst. The procedure may refer to document, data flow, screen interaction, and program steps. Figure 3.1 (Martin and McClure, 1985, p. 369) is an example of decomposition diagram.

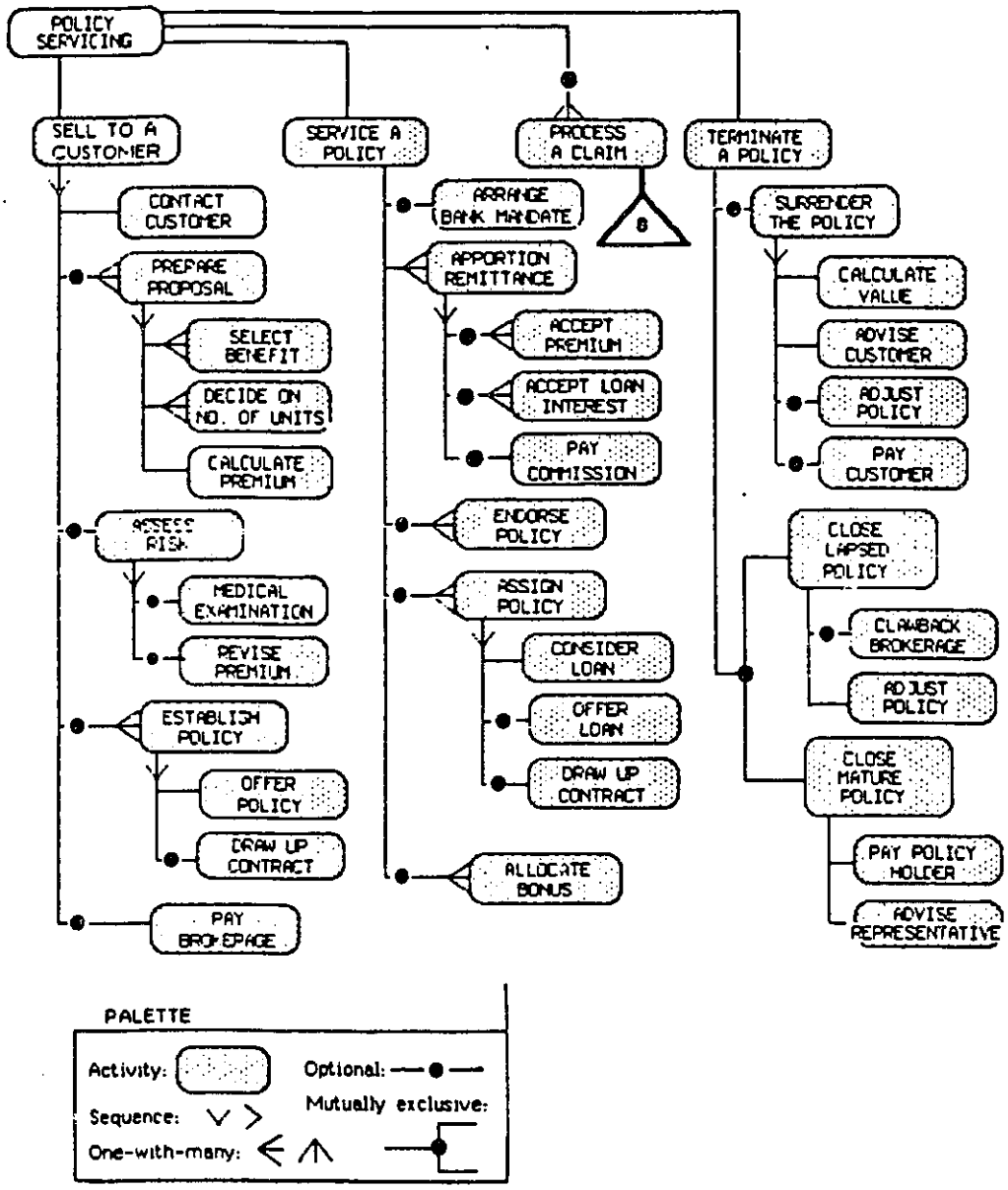


Figure 3.1 Decomposition Diagram (Martin and McClure, 1985, p. 369)

3.2.2 DEPENDENCY DIAGRAMS

In the decomposition diagram, the activities form a hierarchy, but we don't known if certain activities are dependent on others or not. A dependency diagram has blocks showing activities and arrows between blocks showing that one activity is dependent on another. A time dependency exists between two activities if one cannot be carried out until the other has been completed. The arrows in a dependency diagram are often marked with data which are created by one activity and used by another.

There are three types of dependencies which can apply to functions, processes, or procedures (Martin and McClure, 1985, p. 81). These are resource, data, and constraints dependencies.

1) Resource dependency : One activity (A) produces or modifies some resource ; Activity (B) uses this resource, for example DELIVER ORDER cannot occur before PICK GOODS, because there would be nothing to deliver.

2) Data dependency : Activity (A) creates or updates some data and activity (B) uses that data, for example CREATE BACKORDER cannot occur until ACCEPT ORDER has occurred because CREATE BACKORDER needs certain data from the ACCESS ORDER process.

3) Constraint dependency : If an execution of some step in activity (B) depends on a constraint that was set in activity (A) , or the testing of a condition that was set in activity (A).

As shown in Figure 3.2 (Martin and McClure, 1985, p.90), dependency diagrams can be made more generally useful by including additional constructs which are : Optionality, cardinality, branching, mutual exclusivity, loops, parallelism, events, sequence, and flow. Details are shown in Figure 3.2.

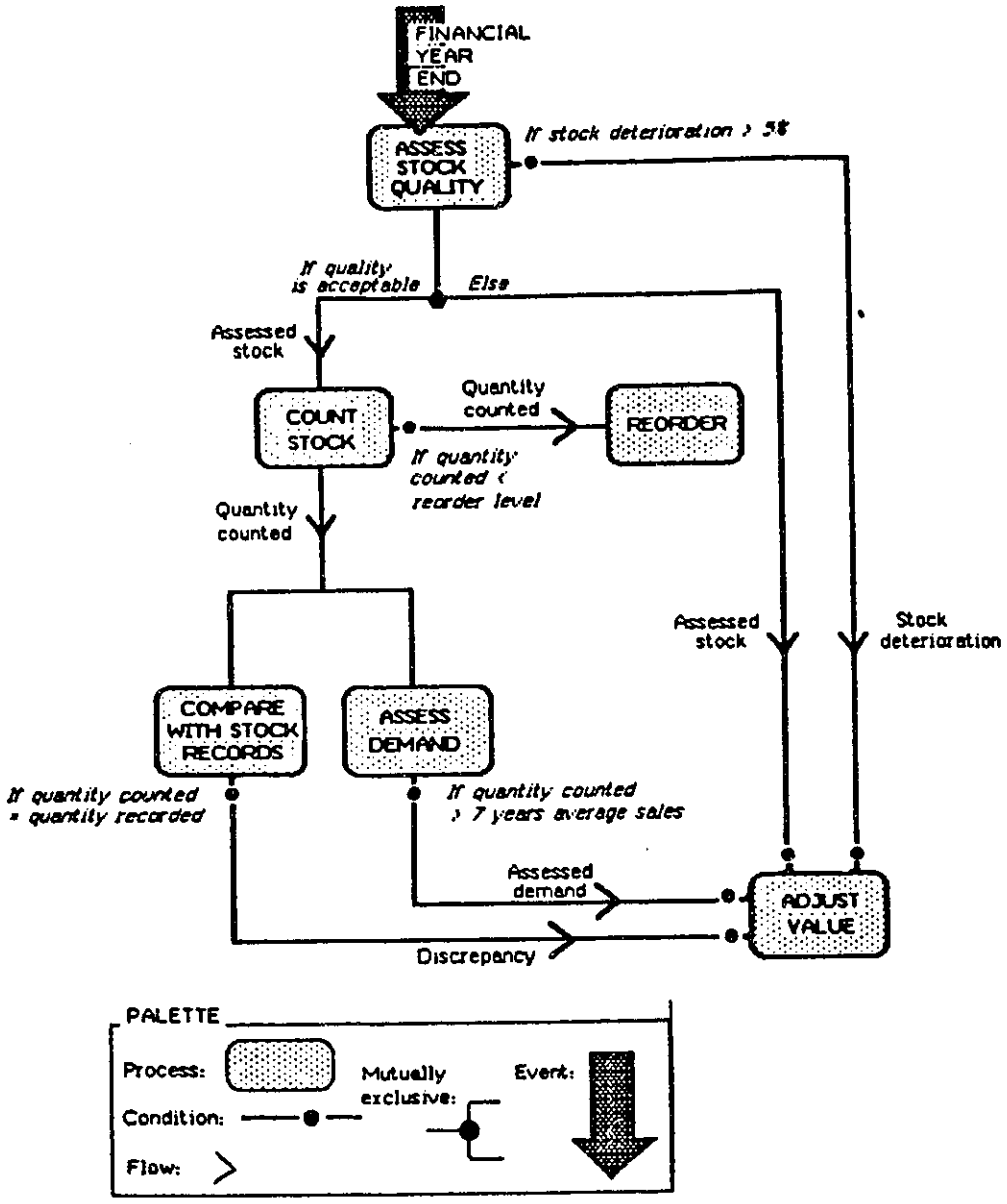


Figure 3.2 Dependency Diagram (Martin and McClure, 1985, p.90)

3.2.3 HIPO DIAGRAMS

A HIPO (Hierarchical Input - Process - Output) diagram is a diagramming technique which can give a general or detailed view of a system or program using three types of diagrams (Martin and McClure, 1985, p.131). These are : Visual table of contents, overview and detail HIPO diagrams.

1) A visual table of contents is a tree-structured decomposition diagram. It shows the overall functional components of a system or program. It does not give any control information, nor does it describe any data components. An example of a visual table of contents for the subscription system (Martin and McClure, 1985, p.132) is shown in Figure 3.3. The purpose of this system is to process three types of subscription transactions: new subscription, renewals, and cancellations.

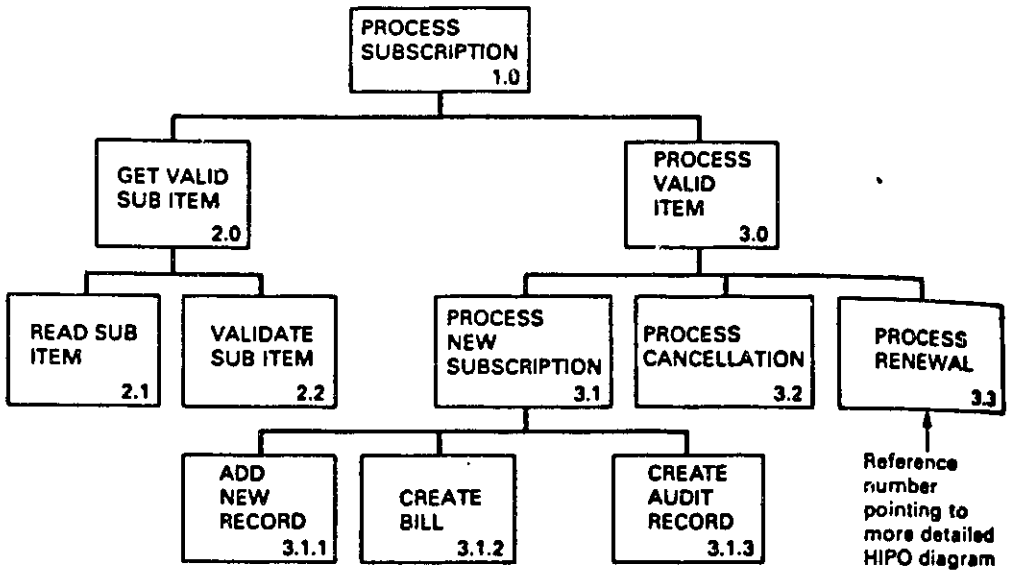


Figure 3.3 The Visual Table of Contents (Martin and McClure, 1985, p.132)

In the visual table of contents, each box can represent a system, subsystem, program, or program module. Its purpose is to show the overall functional components.

2) An overview HIPO diagram gives general information about the inputs, process steps, and outputs of one particular functional components in a system (program). Figure 3.4 (Martin and McClure, 1985, p.132) shows the overview HIPO diagram for the PROCESS SUBSCRIPTION function in the subscription system.

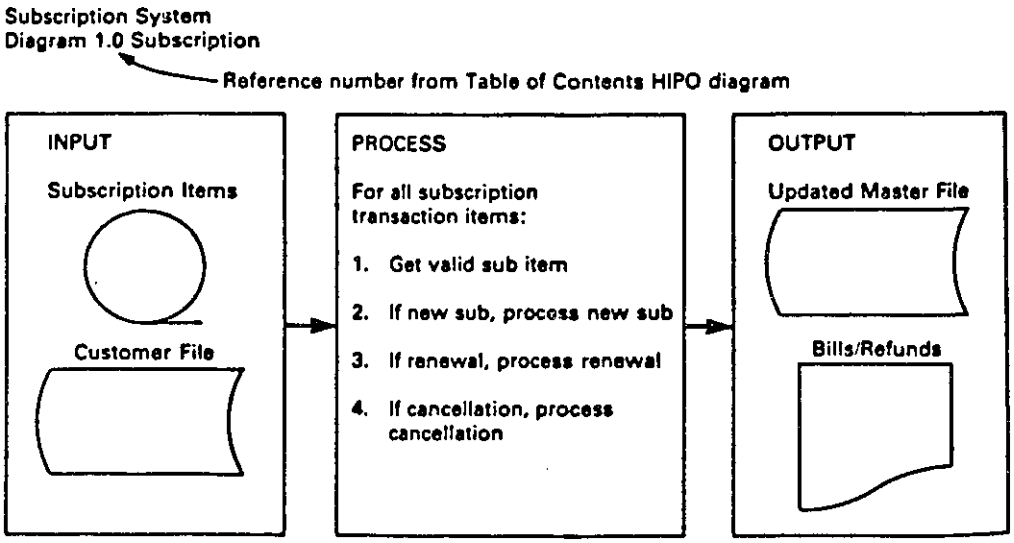


Figure 3.4 An Overview HIPO Diagram (Martin and McClure, 1985, p.132)

3) A detail HIPO diagram provides the information necessary to understand the inputs, processing steps, and outputs for a functional component. It represent the program design and can easily be transformed into program code. Figure 3.5 (Martin and McClure, 1985, p.133) is the detail diagram for the VALIDATE NEW SUB function in the subscription system.

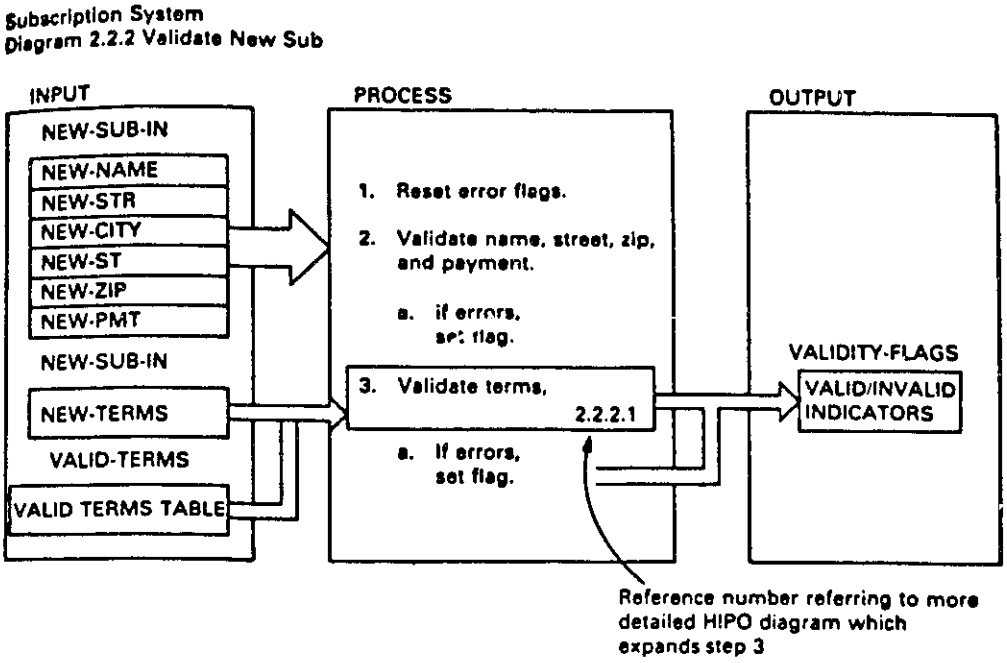


Figure 3.5 A Detail HIPO Diagram (Martin and McClure, 1985, p.133)

HIPO diagramming technique can describe a system or program at any varying degrees of detail during the functional decomposition process. Detail HIPO diagrams relate data to processing steps. HIPO diagrams have no symbols for representing detailed program structures such as conditions, case structures, and loops.

3.2.4 WARNIER ORR DIAGRAMS

Warnier-Orr (Jean Dominique Warnier and Ken Orr) diagrams aid the design of well structured programs. These diagrams use brackets to show the hierarchical decomposition of activities or data. This decomposition can represent a high level overview of a program structure or detailed program logic. It forms the basis of the Warnier-Orr design methodology.

A Warnier-Orr diagram represent graphically the hierarchical structure of a program, a system, or a data structure. It draws the hierarchical structure horizontally across the page with brackets.

1) Representation of data Figure 3.6 (Martin and McClure, 1985, p.139) is an example of a Warnier Orr diagram of an employee file.

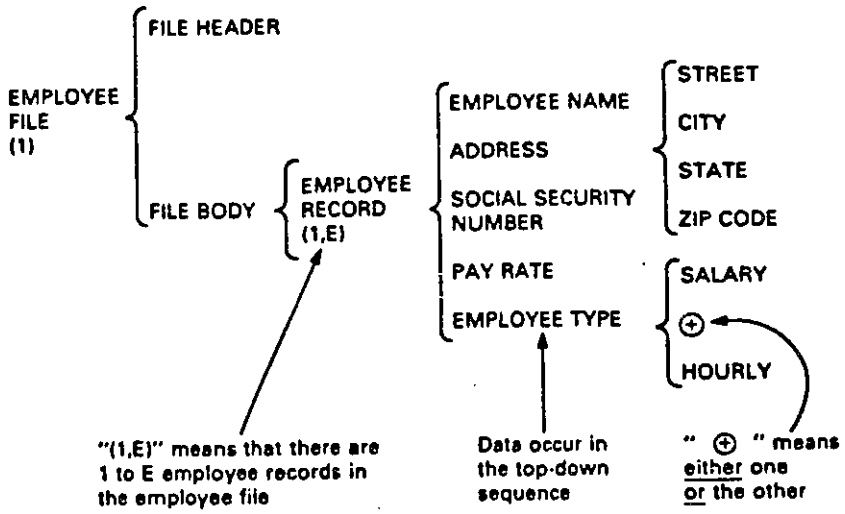


Figure 3.6 Warnier-Orr Diagram of an Employee File (Martin and McClure, 1985, p.139)

- The diagram is read from left to right and from top to bottom within a bracket.
- The brackets enclose logically related items and separate each hierarchical level.
- The items (with meaningful name) are listed vertically.

2) Representation of Program Structure Figure 3.7 (Martin and McClure, 1985, p. 141) shows the Warnier-Orr diagram representing a program structure.

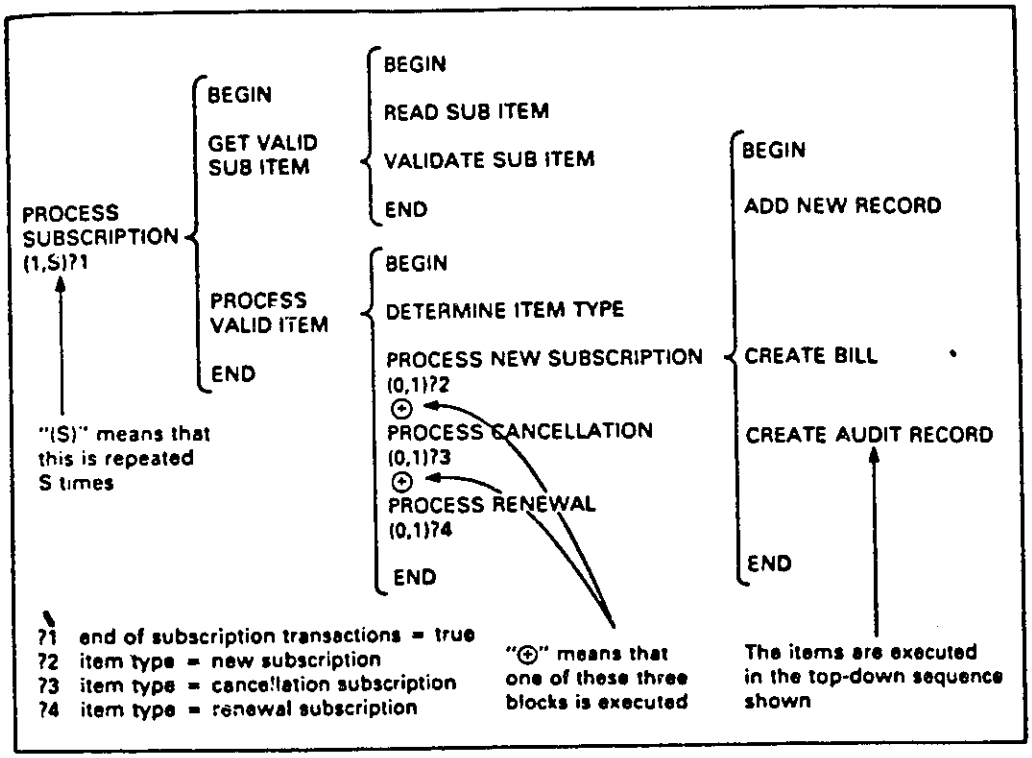


Figure 3.7 Warnier-Orr Diagram for the Subscription System shown in the HIPO Diagram (Martin and McClure, 1985, p.141)

- When representing a program structure, each level in a Warnier -Orr diagram has three components : BEGIN, process step, and END.
- Each level is enclosed in vertical brackets, and the hierarchical structure is read from left to right.
- In a Warnier-Orr diagram, to indicate sequence the processing steps are included at the same hierarchical level and are written in a vertical column one after another.

Warnier-Orr diagrams are easily translated into program code because of BEGIN -END. The Warnier-Orr diagrams provide good documentation for data structures. They do not show conditional logic as well as other details of algorithms.

3.2.5 STRUCTURED ENGLISH

Structured English is a diagramming technique to represent program structures. The figures or the specifications have several important properties :

- They are written in such a way that a user could understand them.
- They are hierarchically structured and use indentation to reveal structures.
- They have a similar code that will be used to implement them.
- Comments that will not be translated into program code are marked with asterisks.
- The detailed program structures (Sequence, condition, repetition, and case) are well defined.
- The sequence structure is a list of items where each item is placed on a separate line. If the item requires more than one line, continuation lines are indented.

- Blocks of instructions are grouped and give a meaningful name which describes their function.
- The structures are indented to show the logical hierarchy.
- Parentheses are used to avoid AND / OR and other ambiguities.
- Keywords are written in bold font.

Figure 3.8 (Dyck, Lawson and Smith, 1979, p.221) gives an example of algorithm using Structured English.

```
module minmax (imports: SET, num; exports: small, large)
* Module to find the smallest and largest entries in a given SET of length num.
* Variables used:
* SET - the given set of numbers
* num - length of the set
* small - smallest entry in the set
* large - largest entry in the set
* Initialize the smallest and largest as the first entry.
small ← set1
large ← set1
* Search the rest of the set for better values.
i ← 2
while i ≤ num do
    [ if seti < small then
          [ small ← seti
      else
          [ if seti > large then
                [ large ← seti
    i ← i + 1
end module
```

Figure 3.8 A Pseudo-code Module to find the Minimum and Maximum Elements in a Set (Dyck, Lawson and Smith, 1979, p. 221)

3.2.6 FLOWCHARTS

Flowcharts were one of the earliest forms of diagramming method. Generally, the flowchart is not considered to be a structured diagramming technique. Its utility is limited to small programs. For larger programs, flowcharts become very cumbersome to use. They can show detailed logic (in an unstructured fashion) but do not give a useful overview of the system functions. The process figure in a flowchart is a rectangular box, the decision figure is a diamond, and the looping figure is formed by drawing a line connecting the figures of the loop into a circle.

Flowcharts do not represent structured design. They encourage GOTO's and nonstructured code which is difficult to maintain. Flowcharts are natural, easy to learn to use, and both easy to draw and to trace. They are too flexible to help us in consistently picturing similar logic process. Unfortunately, flowcharts are not always successful in helping us understand the algorithm being represented. Their major weakness is that they take up too much room. The best way to maintain comprehensibility when moving from page to page is to decompose instead of continuing. However, even if we try to partition a large flowchart, we still may have as many ways of decomposing the logic as we have people trying to represent it.

Figure 3.9 (Mitchell, 1984, p.107) is an example of a flowchart showing the steps to be performed in determining the second largest value in a list.

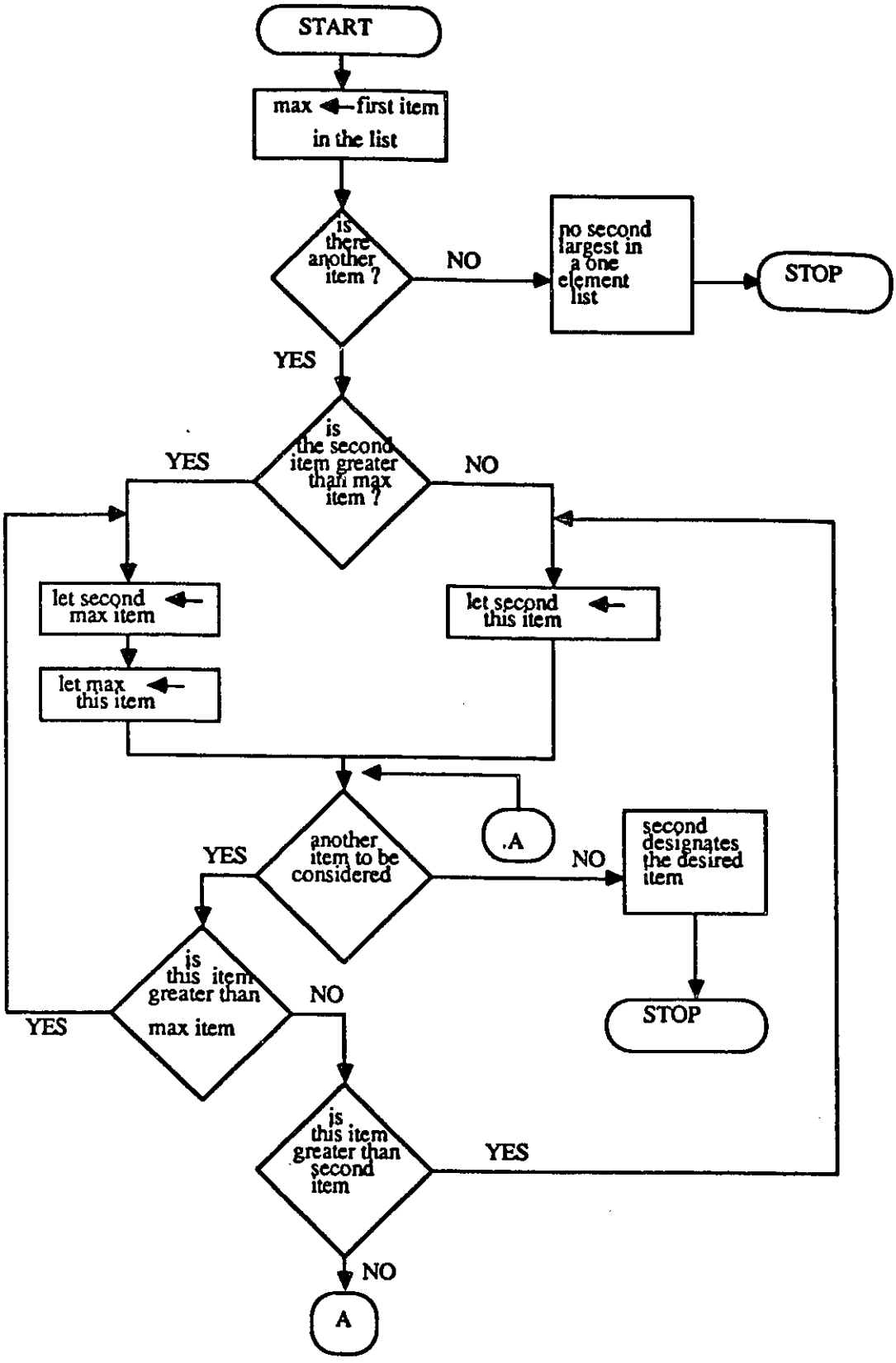


Figure 3.9 Flowchart to find the Largest Value in a List
(Mitchell, 1984, p.107)

3.2.7 NS CHARTS

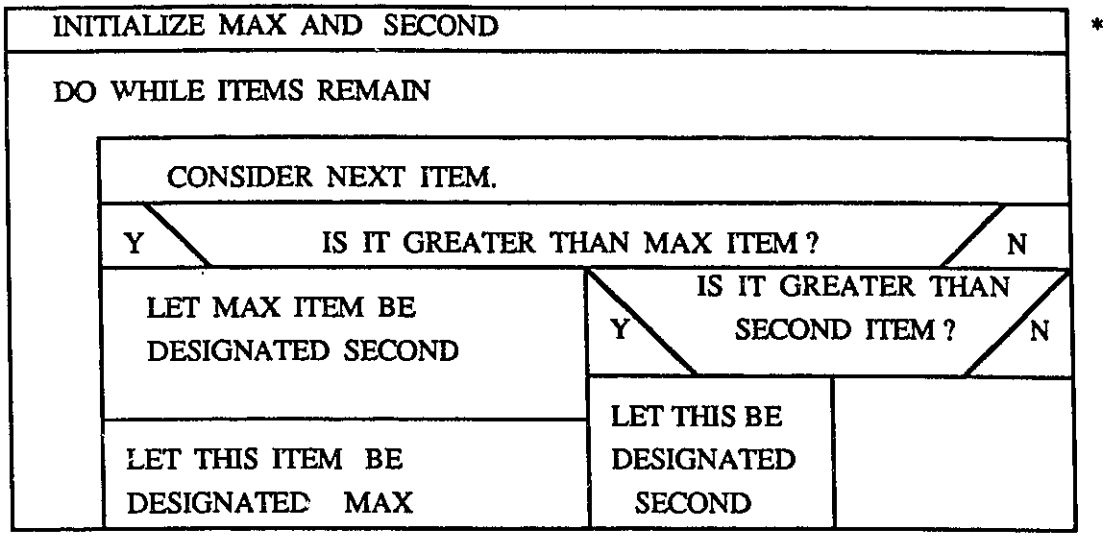
Nassi and Shneiderman set out to replace the traditional flowchart with a chart that offers a structured, hierarchical view of program logic. Nassi-Shneiderman charts are used for detailed program design and documentation. Nassi-Shneiderman charts (NS charts) represent program structures that have one entry point and one exit point and are composed of the control constructs of sequence, selection and repetition. Whereas it is difficult to show nesting and recursion with a traditional flowchart, it is easy with an NS chart. Also, it is easy to convert an NS chart to structured code. However, this conversion is not unique.

The NS chart is a diagramming technique used primarily for detail program design. It is a poor tool for showing the high level hierarchical control structure for a program. The NS diagram technique is only a procedural design tool and cannot be used to design data structures. In addition , although it is easy to read, it is not always easy to draw.

NS charts are motivated by a desire for compactness, a desire for convenient decomposability, and a concern for focusing attention on looping. Flowcharts are not compact due to the space between figures. The construction of flowcharts permits any kind of looping at any time ; hence, the designer is not forced to carefully plan his loops. NS diagrams force the preplanning of all loops because instead of the connecting line of the flowchart, NS diagrams provide a looping figure.

The process figure in NS diagrams is the rectangle, the same as in flowchart. The decision figure is a rectangle also , but is divided into three subrectangles. The looping figure is a rectangle. Figure 3.10 (Mitchell, 1984, p.117) shows the NS diagram for the problem of finding the second largest item in a list.

LEVEL 1



LEVEL 2

INITIALIZE MAX AND SECOND

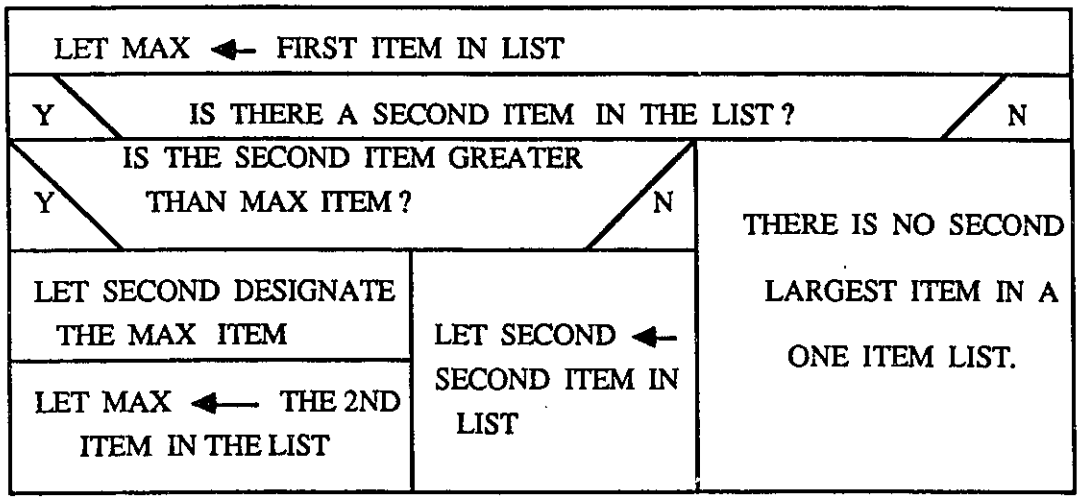


Figure 3.10 NS Diagram for finding the Second Largest Item in a List (Mitchell, 1984, p.117)

(Note : the details of the block identified by a * at level one are given in another block at level two)

Chapter 4

A PROPOSED GRAPHIC SCHEME FOR ALGORITHM-DIRECTED SOFTWARE ENGINEERING

4.1 INTRODUCTION

Clear diagrams play an essential part in designing complex systems and developing programs. When a number of people work on a system or program, the diagrams are an important communication tool. A formal diagramming technique is needed to enable the developers to interchange ideas and to make their separate components fit together with precision. Structured diagramming techniques help developers deal with the large volume of details generated during the program development process. When systems are modified, clear diagrams are an essential aid to maintenance. They make it possible for a new team to understand how the programs work and to design changes.

The introduction of structured techniques into computing was a major step forward. The early structured techniques were pencil and paper methods. Today these techniques need automation. Designs should be created with the aid of a computer. The design should be such a form that it leads to automated code generation. Some diagramming techniques are more appropriate than others for automation. Automation of diagramming should lead to automated checking of specification and automatic generation of program code. Many of the diagramming techniques of the past are not a sound basis for computerized design. They are too casual, unstructured, and cannot represent some of the necessary constructs.

An unified graphic technique to represent algorithms and computer programs was proposed by Ören (1984). The technique facilitates conception and design of algorithms and their translation into computer programs. The technique has the structure preserving capability, i.e., when an algorithm is translated into a program, there exists a one to one correspondence

of the logical structures of the algorithm and the program. Similarly, computerized documentation of a program reveals the identical logical structures of the original algorithm.

- An algorithm specified according to the unified graphic notation can be implemented in one way only. The logical structure of the program is identical to the logical structure of the algorithm.
- The basic building blocks, i.e. , sequential, selection and repetitive blocks, used in the graphic scheme are easily understood.
- The logical structure of the algorithm can be perceived easily by using the graphic technique.

The graphic scheme can be used for several purposes :

- To conceive, graphically edit, and refine structured algorithms.
- To increase the chances of detection and elimination of logical errors in the algorithms and the programs.
- To generate computerized documentation of programs written in structured languages.
- To develop software tools to assist a programmer in the design of structured algorithms and in their translation into structured programs.

The unified technique allows graphic representation and stepwise refinement of algorithms expressed as pseudocodes and computer programs, as well as computerized generation of program documentation. Therefore the technique can enhance the activities involving algorithm design, programming and documentation of computer programs.

The unified graphic scheme has the following important features :

- An aid to clear thinking
- Precise communication between members of the development team.
- System documentation
- Enforcement of good structuring
- An aid to debugging
- An aid to changing systems (maintenance)
- Fast development (with computer aided diagramming)
- Enforcing rigor in specification (when linked to computerized specification)
- Automated checking (with computer-aided tools)
- Enabling end users to review the design
- Encouraging end users to sketch their needs clearly
- linkage to automatic generation of code
- Easy to read
- Quick to draw and to change
- User friendly (because the diagram is obvious in meaning and symbols and mnemonics which the user may not understand are avoided)
- Good for stepwise refinement
- Can be printed out on normal width paper (without excessive divisions into pieces)
- Automatically convertible to program skeleton

The aims of the unified graphic technique is :

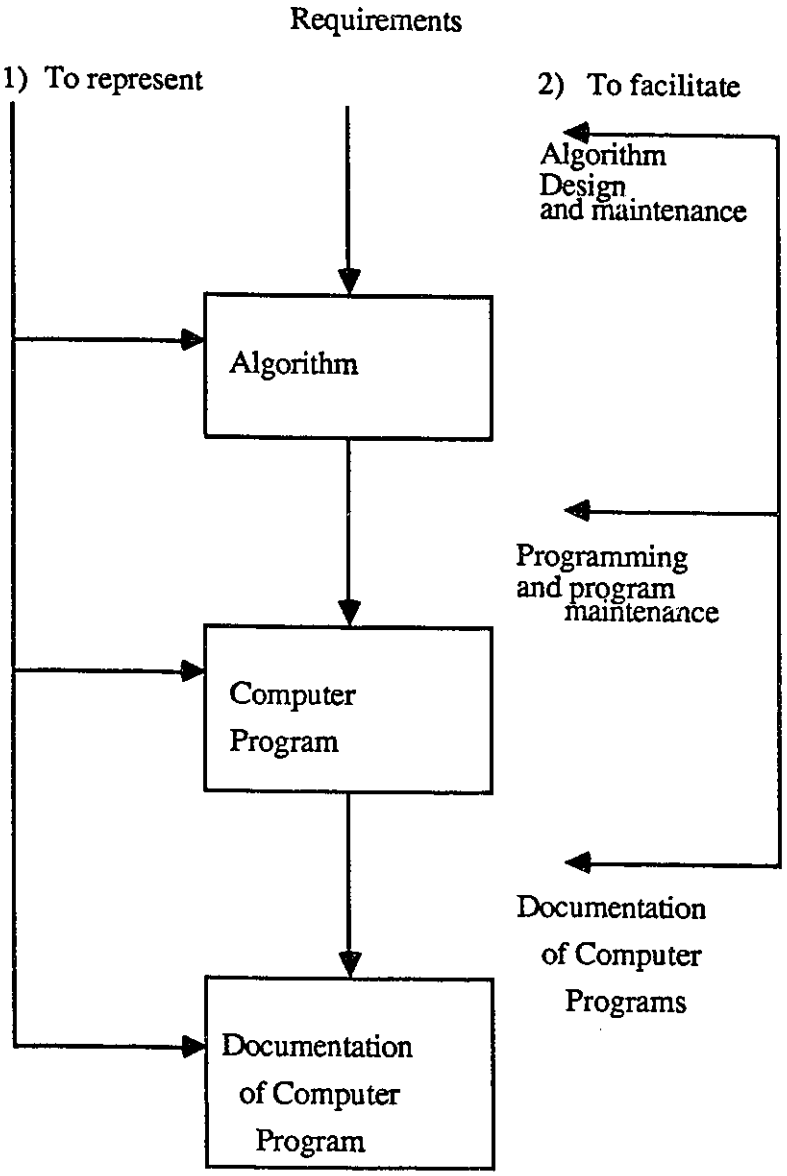


Figure 4.1 Aims of the Unified Graphic Technique

4.2 GRAPHIC STRUCTURES

There are four types of building blocks which are : Program modules, sequential, selection, and repetition blocks. Last three types of blocks can be graphically nested within another block at any desired level. The rightmost line of every block is represented by a common vertical line.

4.2.1 PROGRAM AND PROGRAM MODULES

Program and program modules are used to represent main programs, functions, and procedures. The graphical representations of main program, function, and procedure blocks are given in Figures 4.2 - 4.4 respectively.

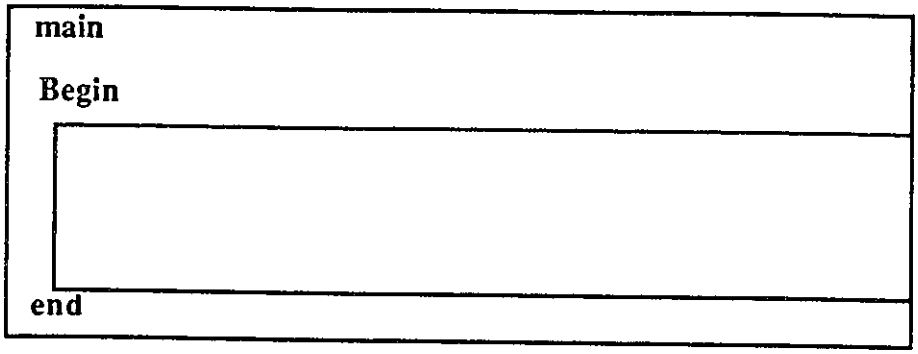


Figure 4.2 Main Program Block

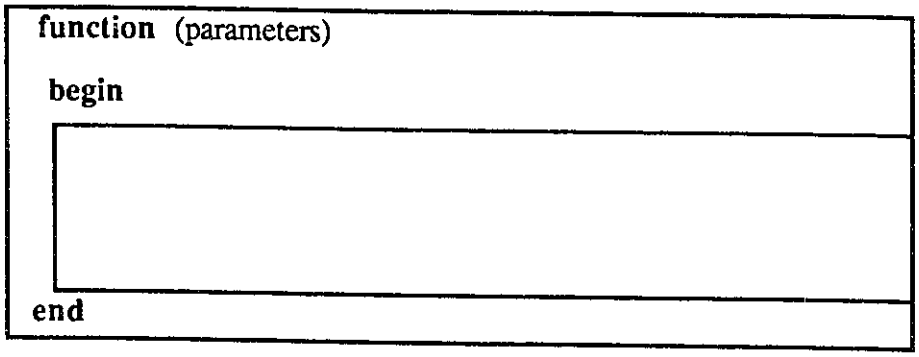


Figure 4.3 Function Block

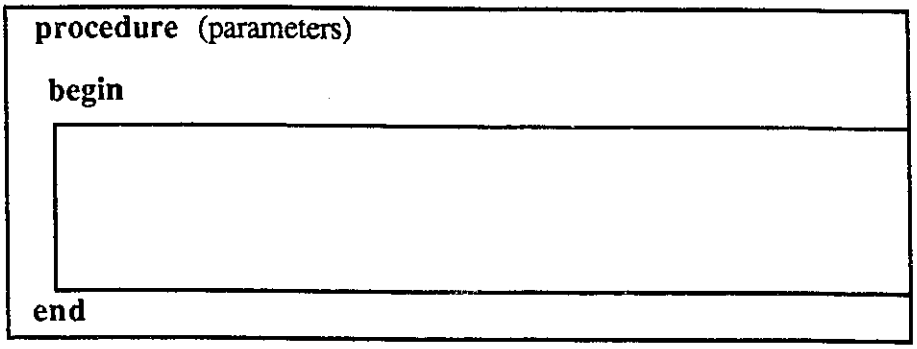


Figure 4.4 Procedure Block

4.2.2 SEQUENTIAL BLOCK

Sequential blocks are represented by rectangles. Instructions to appear in a sequential block are those not involving selection nor repetition. They can be declarations, assignment statements, comments or any simple sequence of instructions.

Figure 4.5 Sequential Block

Initial sequential block has additional information such as :

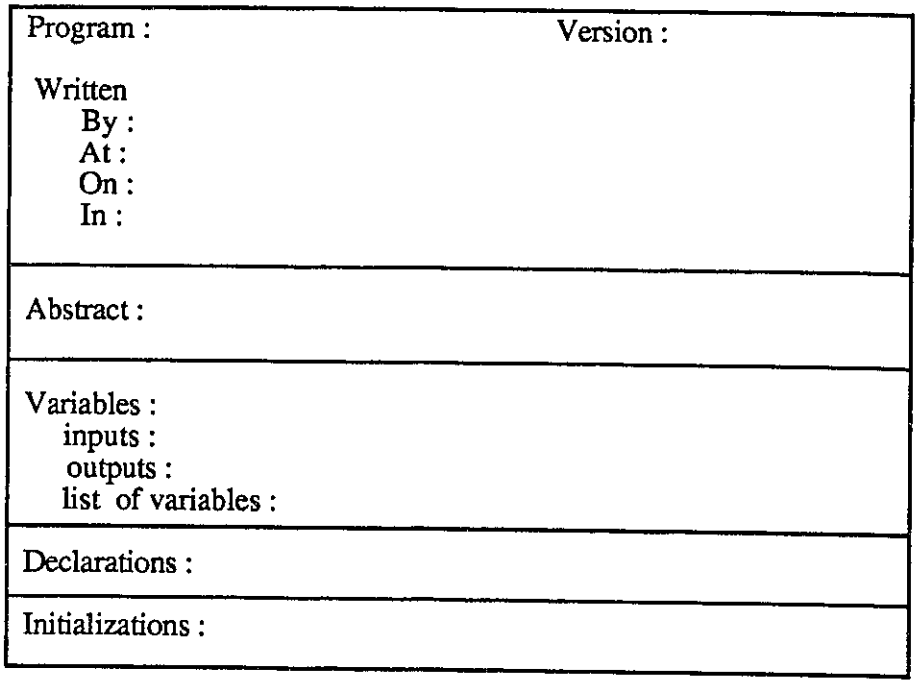


Figure 4.6 A Template of an Initial Sequential Block

4.2.3 SELECTION BLOCKS

The basic selection blocks are if-then block, if-then-else block, if -then -elseif block, and case block. An if-then block consists of a block of code which is executed if the condition specified after "if" condition is satisfied. As seen in Figure 4.7, the scope of the block is clearly indicated by "if endif" pair.

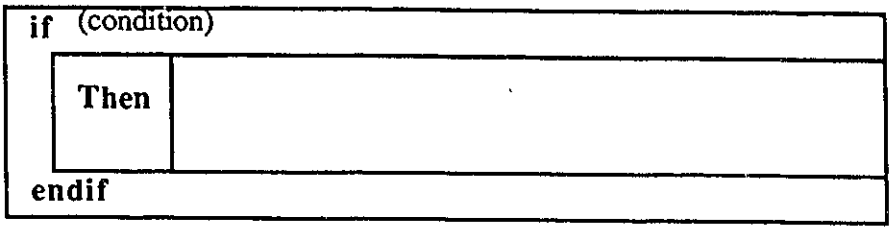


Figure 4.7 If-then Block

An if-then-else block consists of two blocks of code. If the condition specified after if is satisfied, then the then block is executed, otherwise the else block is executed (Figure 4.8).

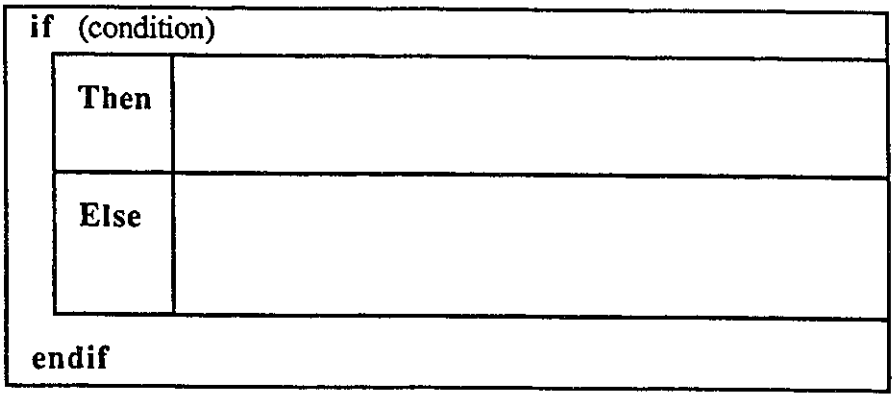


Figure 4.8 If-then-else Block

An if-then-elseif block is represented in Figure 4.9. The final else block is optional.

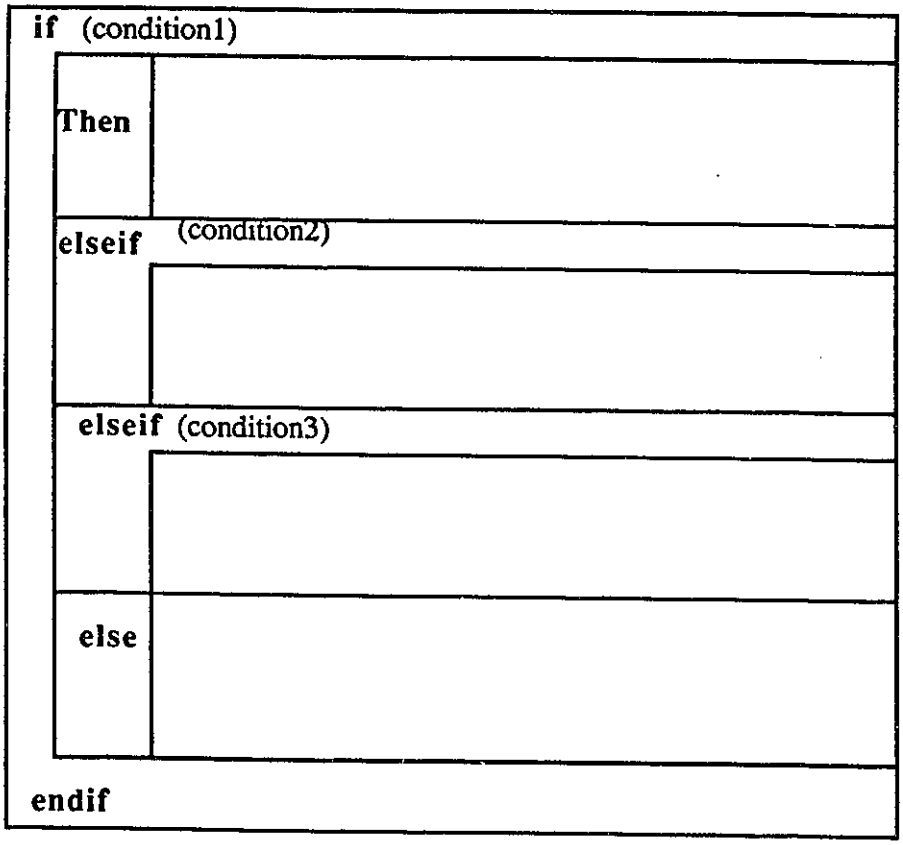


Figure 4.9 If-then-elseif Block

A case block, as seen in Figure 4.10, consists of several subblocks. Only one of the subblocks is executed after entering to the case block. The default subblock is optional and can be used to detect and process unacceptable values of the control variable.

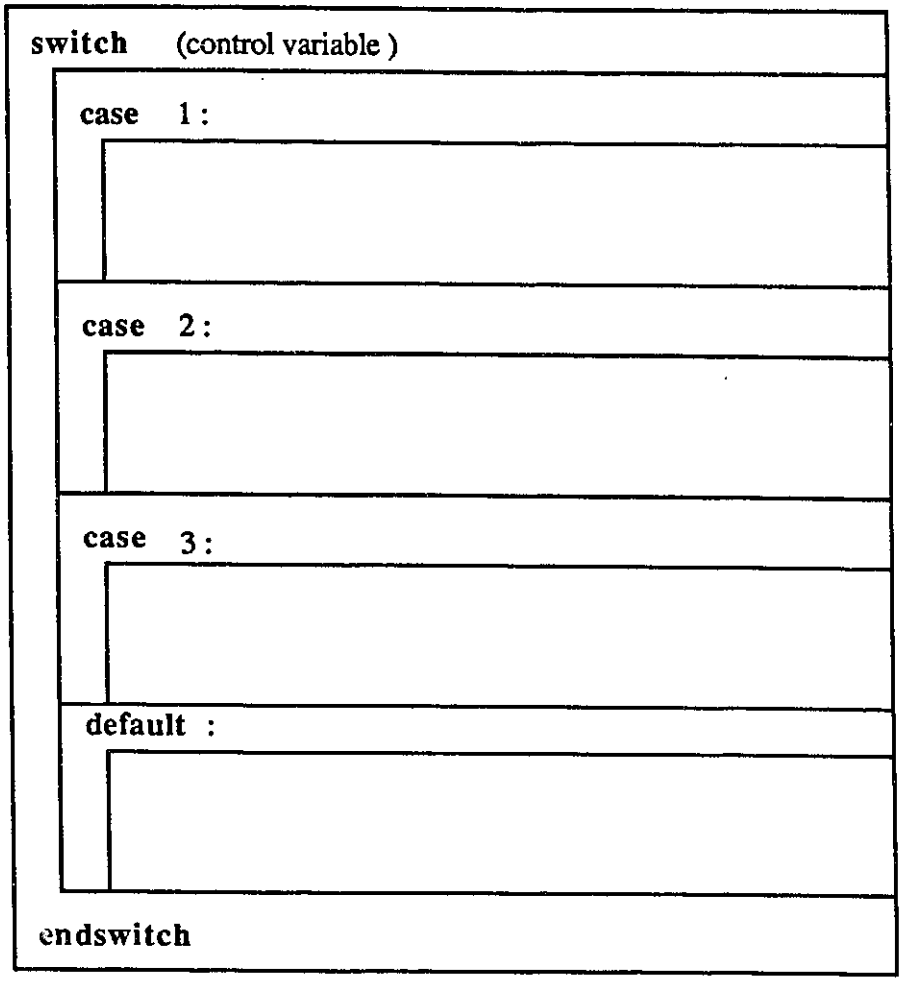


Figure 4.10 Case Block

4.2.4 REPETITION BLOCKS

Basic types of repetition blocks are while block, For block, and Do-while block. A While block is executed so long as the condition expressed after while is satisfied (See Figure 4.11).

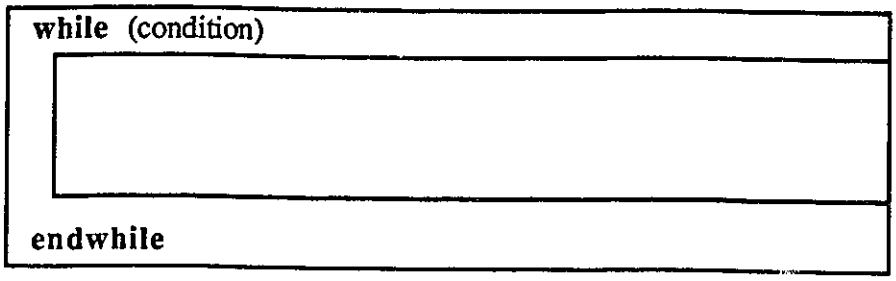


Figure 4.11 While Block

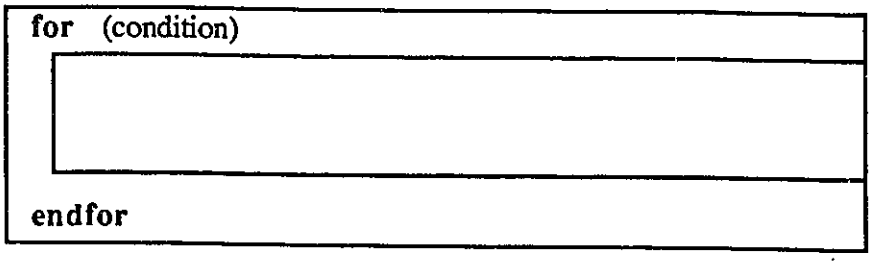


Figure 4.12 For Block

A do while block is represented in Figure 4.13. It is executed so long as the condition expressed after while is satisfied.

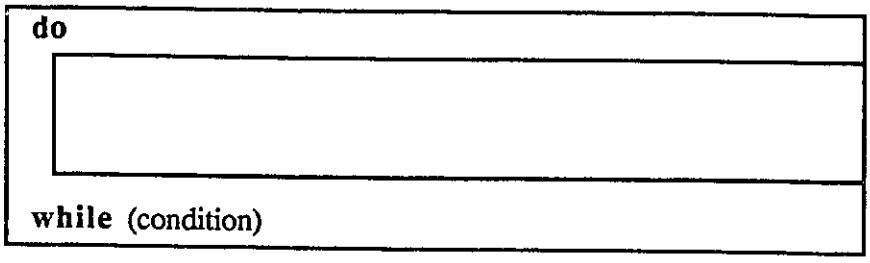


Figure 4.13 Do-while Block

4.3 AN EXAMPLE OF AN ALGORITHM SPECIFICATION USING THE GRAPHIC SCHEME

Figure 4.14 is an algorithm which is taken from (Stinson, 1985, p.26). It merges two sorted arrays A and B, of lengths m and n to obtain a sorted array C of length m+n. It is reexpressed according to the graphical representation scheme. The resulting representation is given in Figure 4.15. The variables smallA and smallB are used to express respectively the minimums of the arrays A and B, and MAXINT is a huge integer.

```
ALgorithm merge (A,B,C,m,n) ;
var i, j, k, samllA, smallB : integer;
    begin
      i := 1; j := 1; k := 1; smallA := A[i]; smallB := B[j];
      While k<= (m+n) do begin
        if smallA <= smallB then begin
        C[k] := smallA; i := i +1;
        if i<= m then smallA := A[i]
         else smallA := MAXINT
        end
        else begin
         C[k] := smallB; j := j+1;
         if j<= n then smallB := B[j]
        else smallB := MAXINT
         end;
         k :=k+1
         end
     end.
```

Figure 4.14 An Algorithm to merge two Sorted Arrays

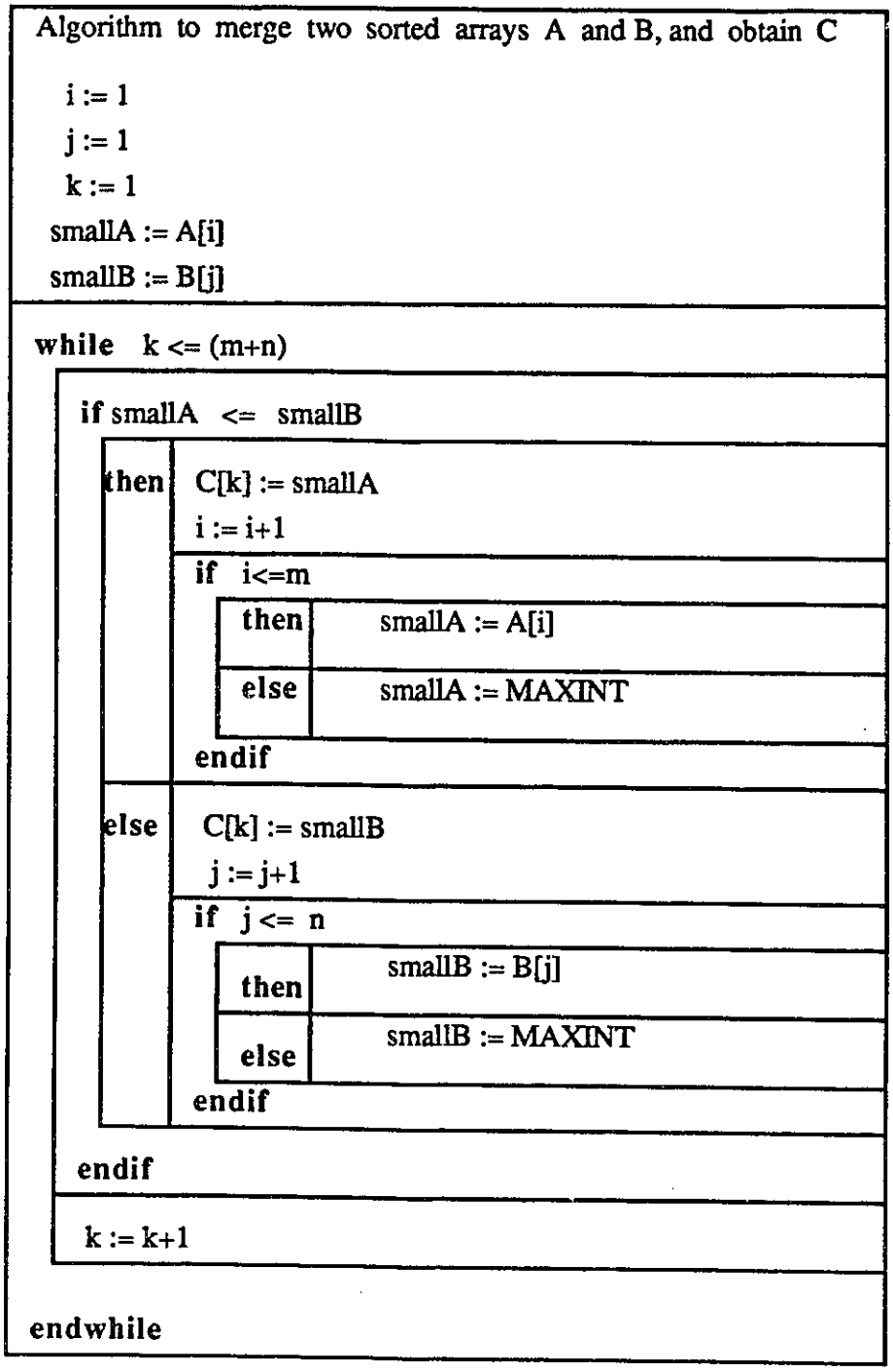


Figure 4.15 An Example of an Algorithm Specification using the Graphical Representation Scheme

Chapter 5

ALGORITHM EDITING ENVIRONMENT (ALC)

5.1 FACILITIES PROVIDED

ALC is a menu driven, Computer Aided Software Engineering (CASE) environment which helps a user to design and edit algorithms in a maner compatible with the C language. It is based upon the notion of the unified graphic scheme and allows the user to choose between a number of given structures which are grouped into program modules, repetition blocks, and selection blocks. Those structures can be placed, nested, copied, moved or deleted. Any sequential block can be added to those structures. Files, can be loaded from the file system, stored to a directory, or printed to a laser or impact printer.

The keyword structures can be produced in French or in English. This option can be selected in the first menu of ALC along with the corresponding user level.

A beginner user level option is also available. This option allows the beginner user to change his mind when selecting a particular template structure. For instance, when the beginner user selects an IF template structure, it is displayed in a pop up window in the ALC main window. At this point ALC prompts the user as to whether or not this is the template that he wants. This is done by displaying the following prompt in error message window : " Insert this template YES NO". When the option YES is selected, the template is inserted at thespecified insertion point. Similarly, when the NO option is selected, the template is not inserted and the pop up window will disappear.

Error messages or any kind of message always appear in a window near the bottom of the screen. A "BEEP" will sound each time an error occurs and the corresponding error message will be displayed in that window.

The structure window is the largest window and the most important in the main screen of ALC. As the structures are selected, they are displayed in that window. When a structure is selected, its relative position within the file is verified. If it is a wrong position, a "BEEP" will sound, an error message will appear in the lower window, and the structure will not be inserted in a wrong place. For the case CASE and IF-ELSEIF-ELSE template structures the user will be prompted for additional information. This information will be the number of additional ELSEIF's etc... that the user wants inserted. The list of the parameters of the procedures and functions is inserted in the text window after the name of the procedure or function between brackets. A set of fonts is also provided.

Finally, a help screen can be invoked at any time. This option displays a file named "ALC.HELP" in a pop-up window over the text window containing the algorithm structures. This file gives a description of the functionality and how to make use of each option in the system. ALC must be invoked from the *suntools* environment. To get into suntools environment, type suntools from the *Sun unix* environment. Then type ALC. The first screen is :

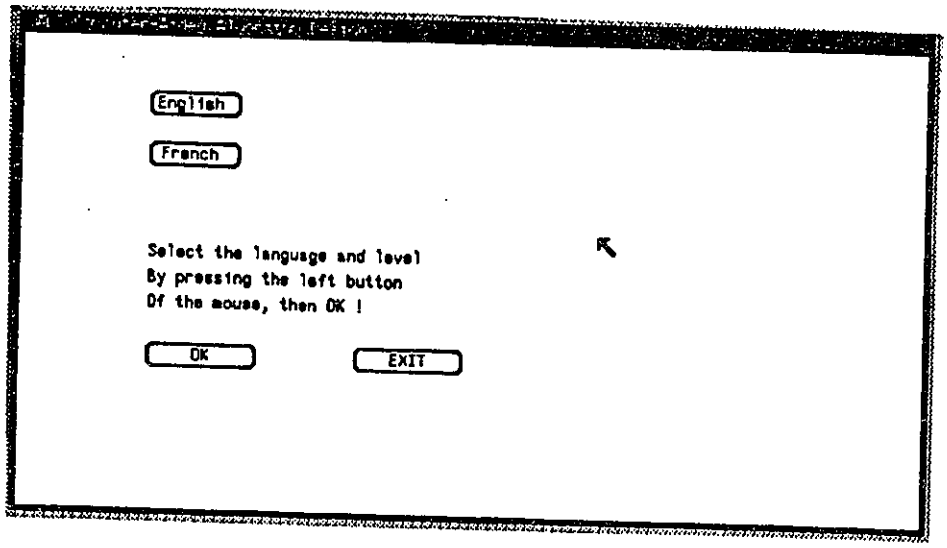


Figure 5.1 First Screen of ALC

The user can select any of the above options by placing the caret on the desired option and pressing the left button of the mouse. Once the specification language and the corresponding user level are chosen the user can select the button "OK" to get into the main menu. The option "EXIT" exits ALC and returns to the suntools environment.

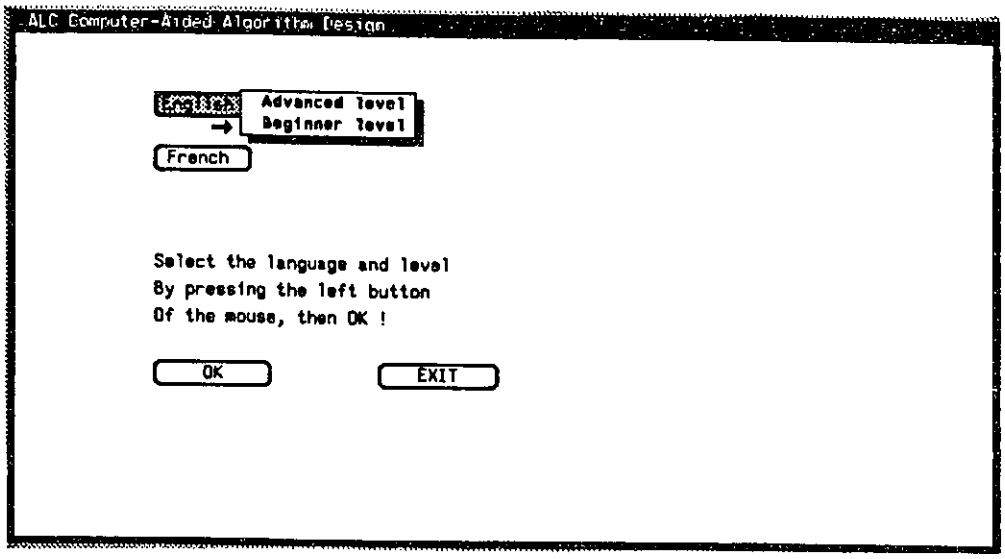


Figure 5.2 English Version will be chosen

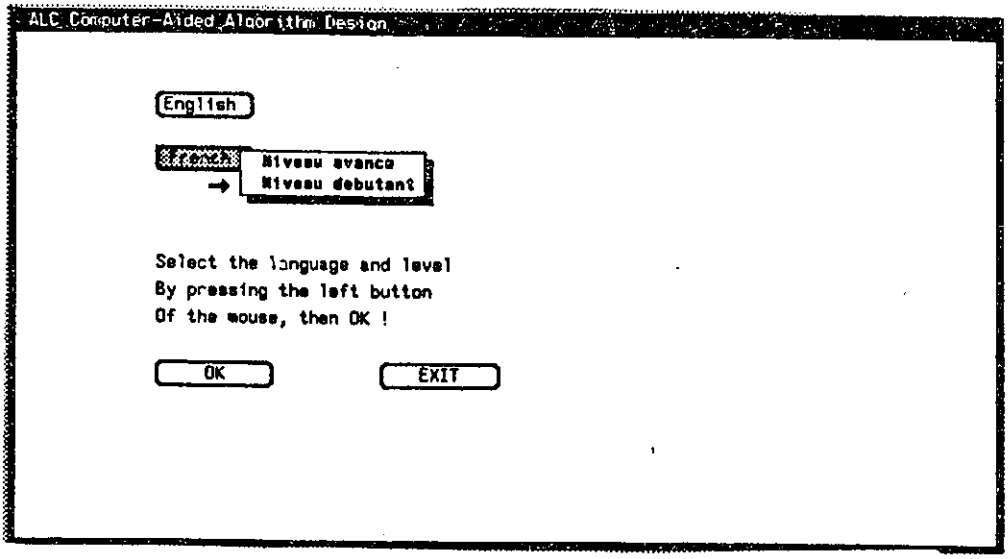


Figure 5.3 French Version will be chosen

The main screen of ALC when the English version is selected is given in Figure 5.4.

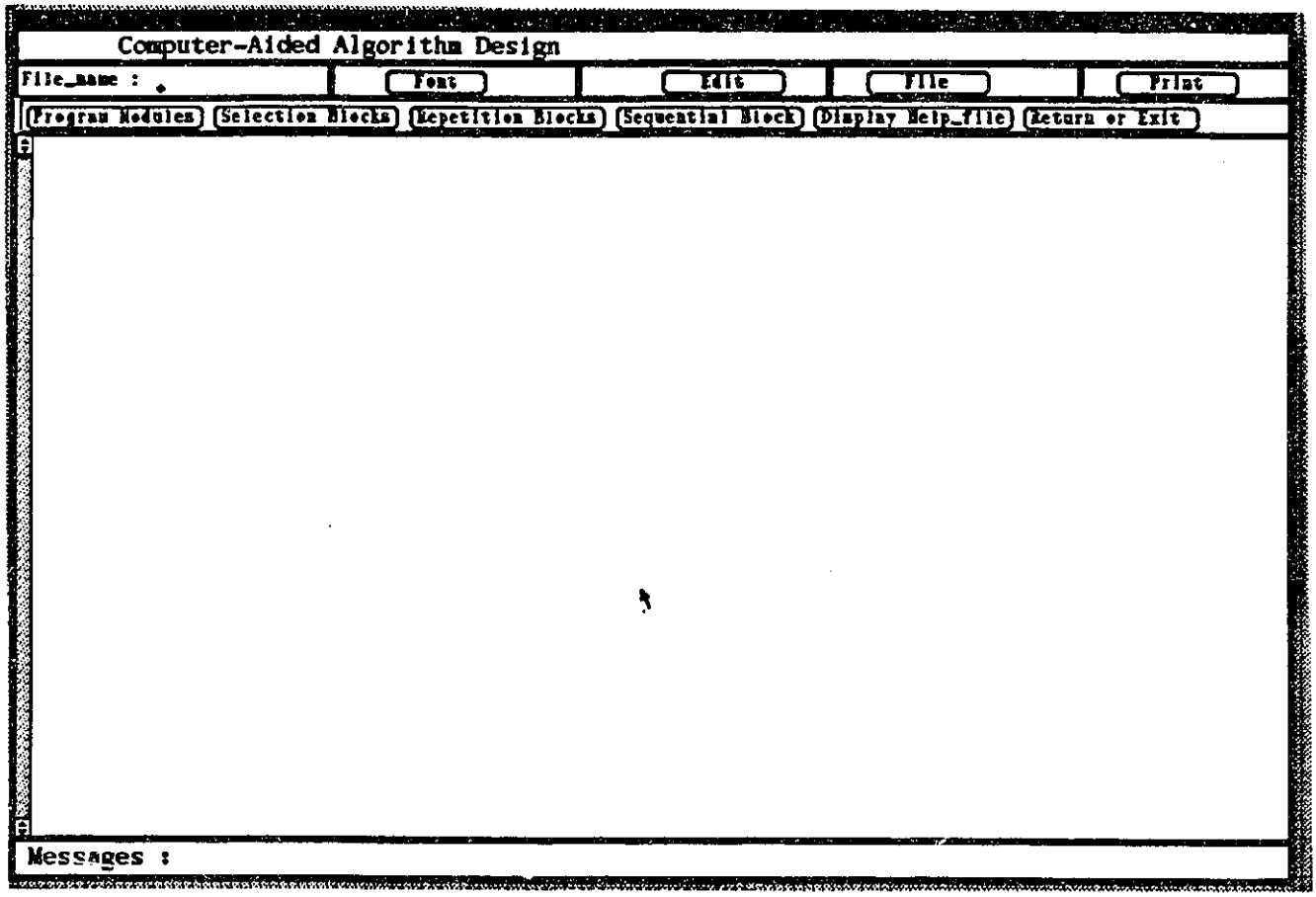


Figure 5.4 Main Screen of ALC

The following figures show the details of editing and updating algorithms. Figure 5.5 shows the screen with specification to insert a program block.

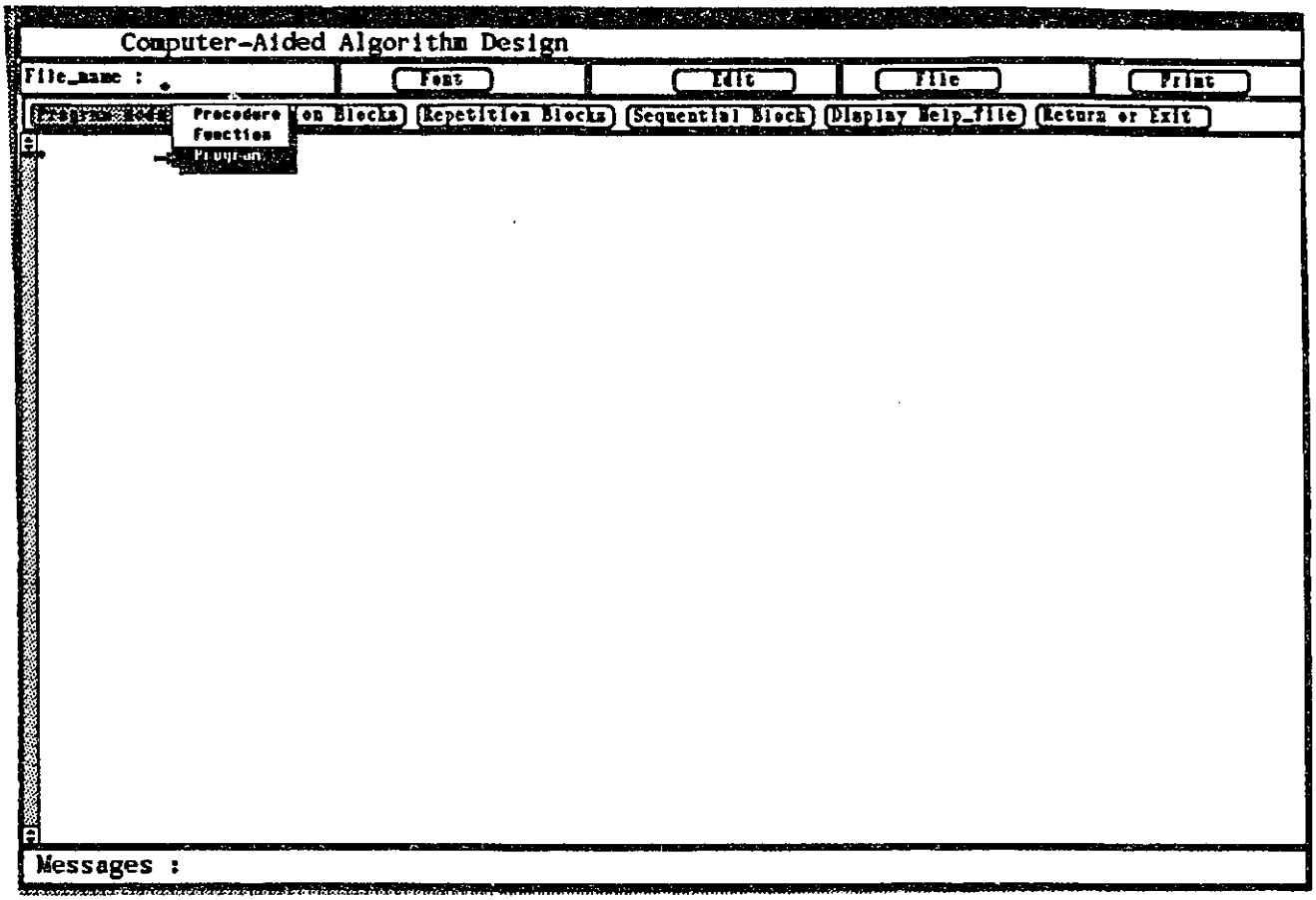


Figure 5.5 Specification to Insert a Program Block

The Figure 5.6 gives the main screen after the insertion of a program (invoked in Figure 5.5) and specification to insert an If-Then-Elseif block.

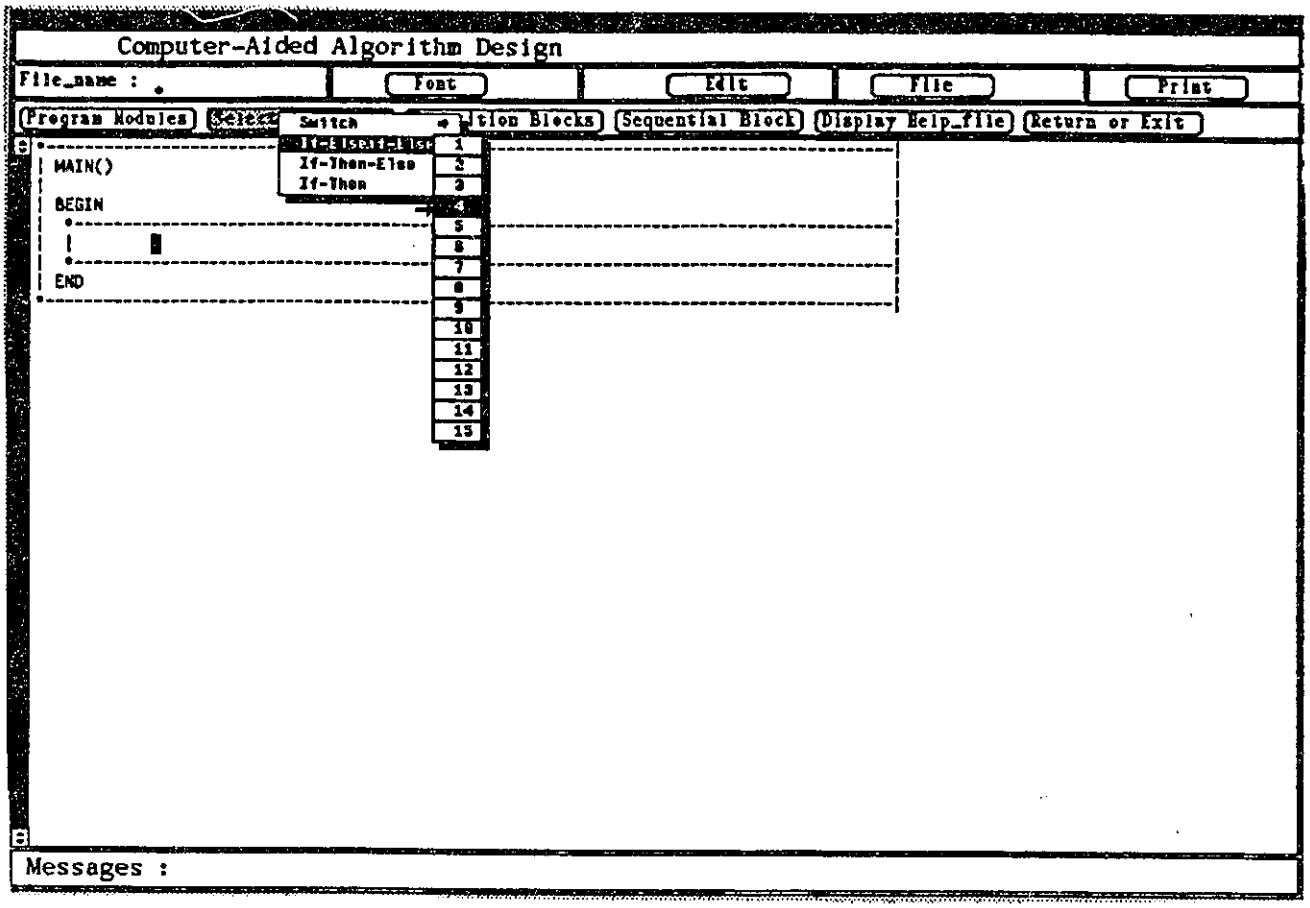


Figure 5.6 Insertion of a Program Block and Specification to insert an If-Then-Elseif Block

Figure 5.7 represents the main screen after the insertion of the If-Then-Elseif block (invoked in Figure 5.6) and specification to insert a case block.

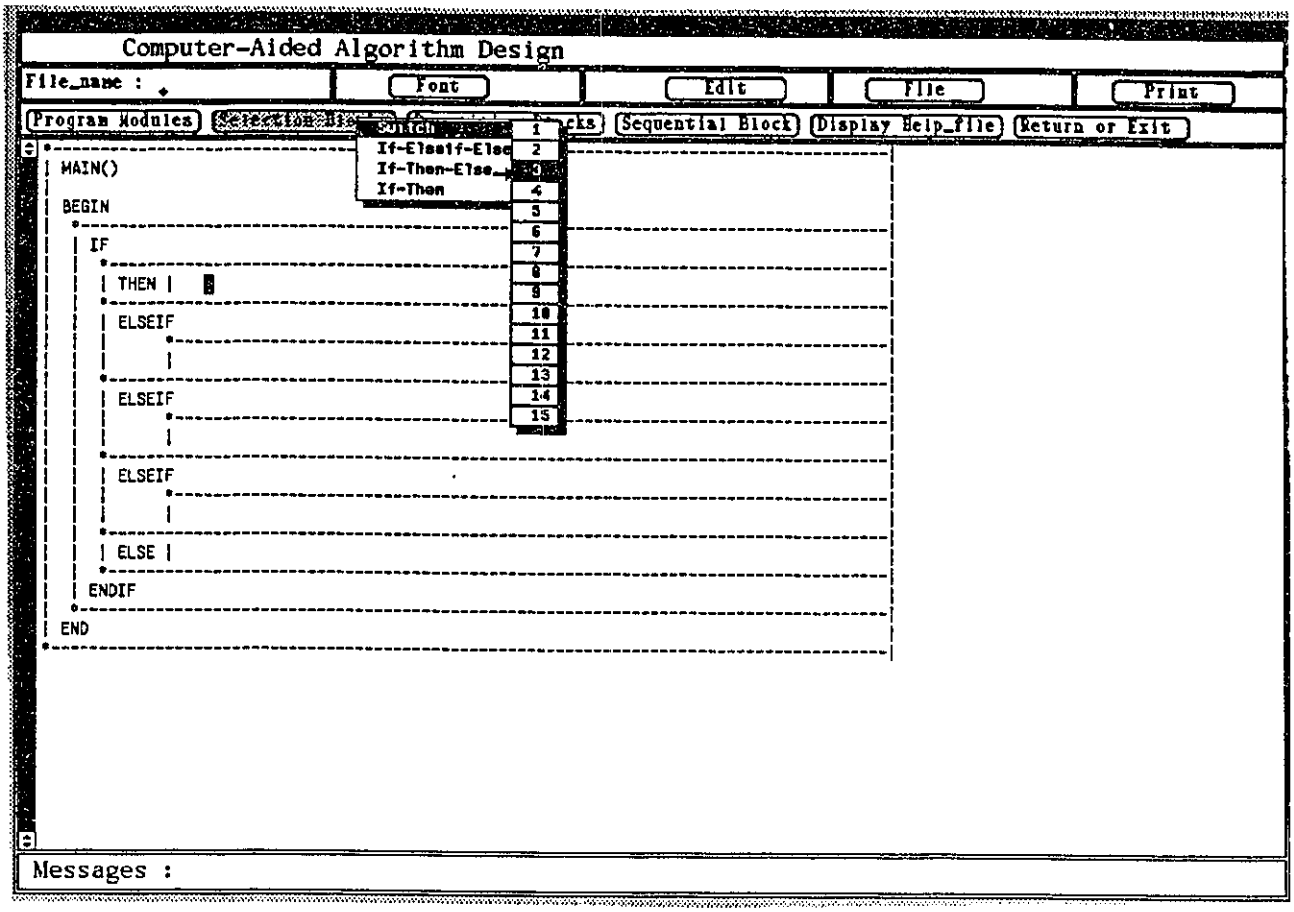


Figure 5.7 Insertion of If-Then-Elseif Block and Specification to insert a Case Block

The insertion of the case block (invoked in Figure 5.7) has been completed. The user specify an insertion of a while block.

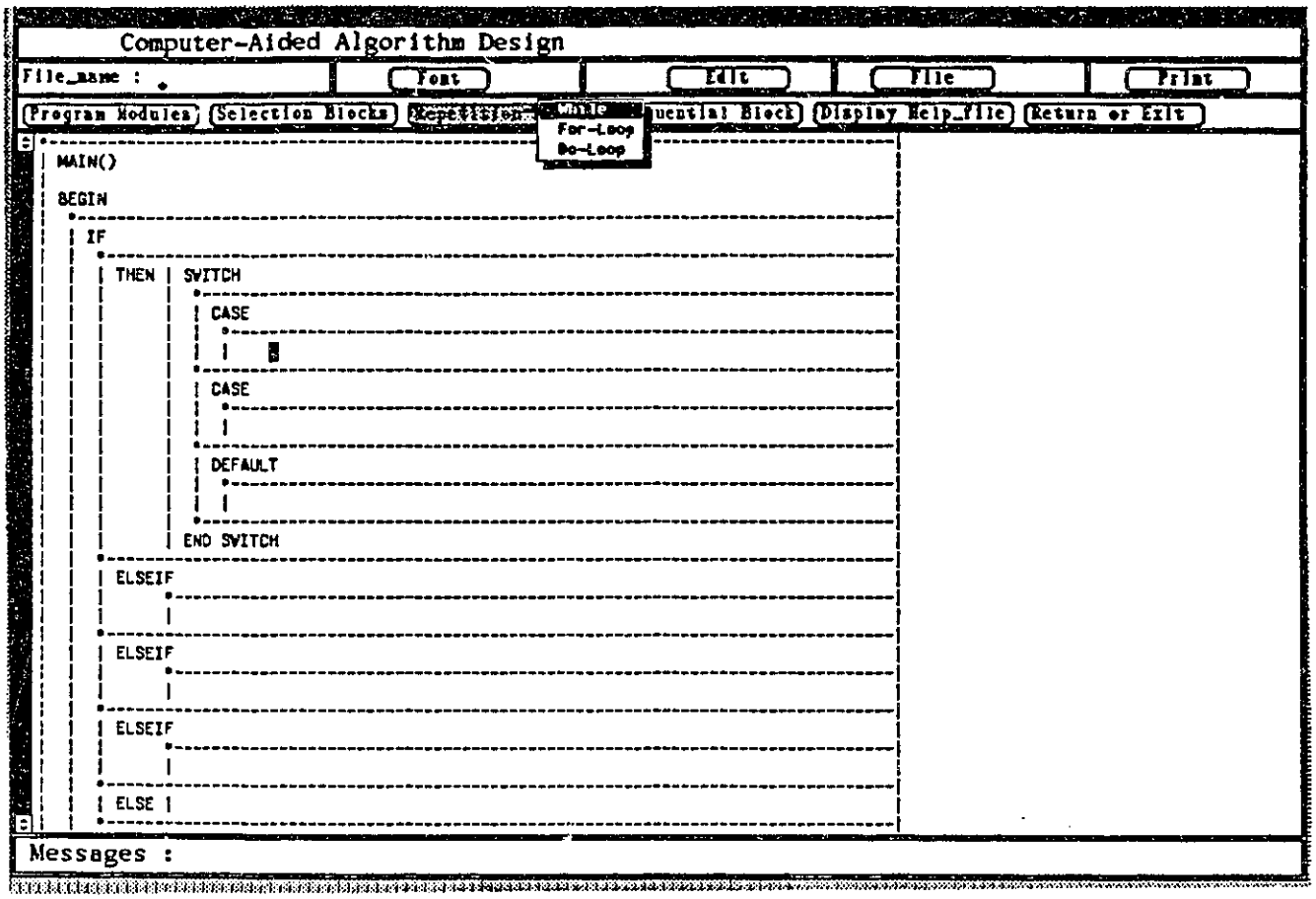


Figure 5.8 Insertion of Case Block and

Specification to Insert a While Block

Figure 5.9 shows the insertion of the while block (invoked in Figure 5.8).

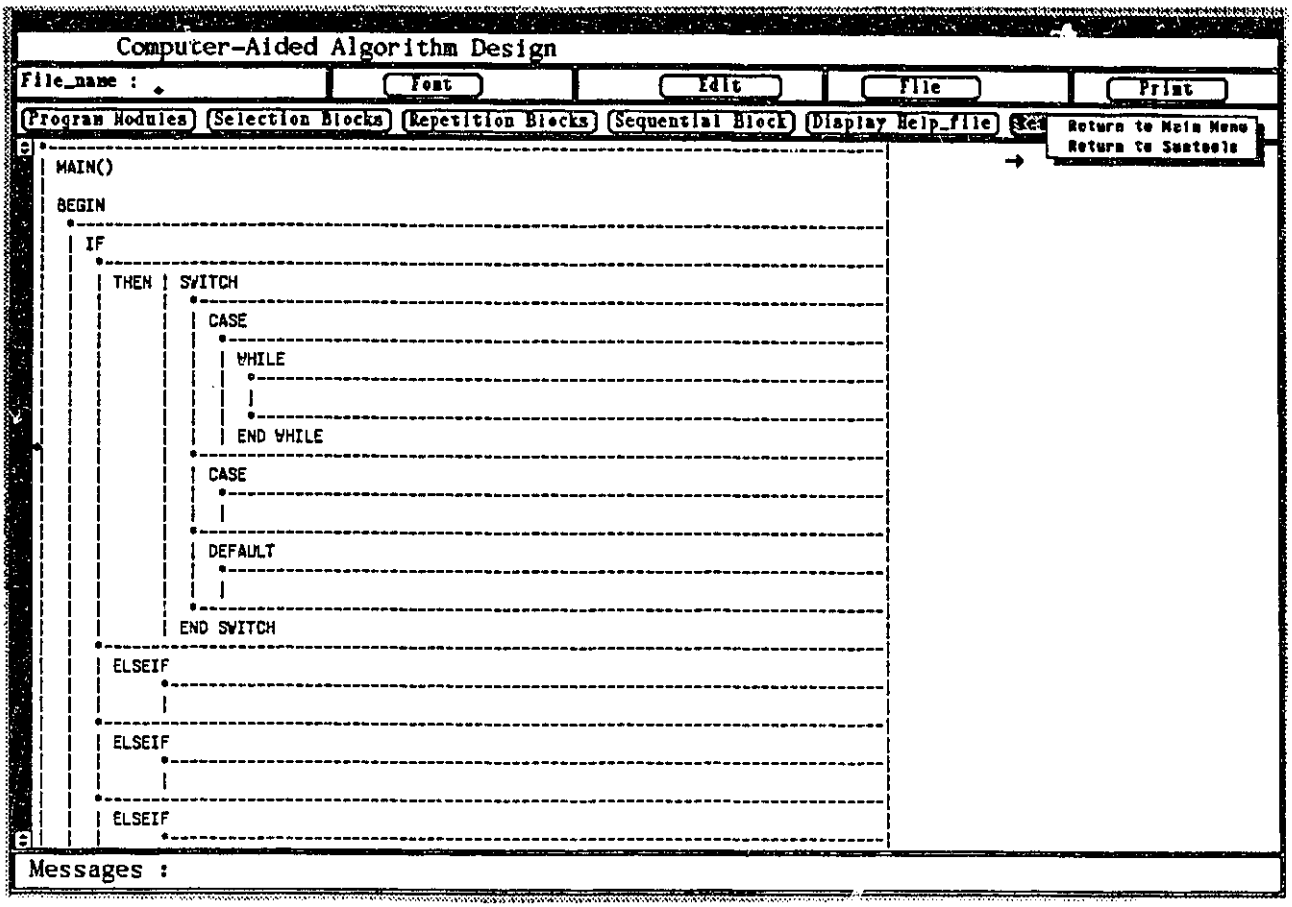


Figure 5.9 Insertion of the While Block Specified in Figure 5.8

The Editing Functions

This menu, represented in Figure below, is one of the most important menus. It allows the user to manipulate the different structures within the algorithm.

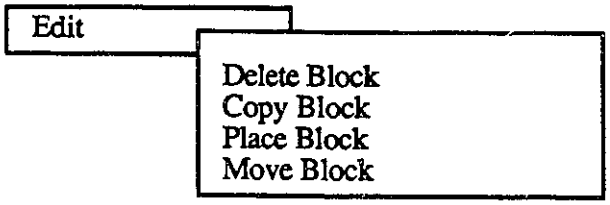


Figure 5.10 Edit Operations Menu

- **Copy Block**

This selection is used when one wants to copy a block, i.e. a structure from one area to another. This is done by placing the cursor inside that block, and selecting this option. A copy of that block will be put in a buffer. To retrieve the copy of the block, use the Place block option within this menu. Once the block is in the buffer, it can be placed as many times as wanted. The structures within the block being moved will also move with the block.

- **Move Block**

To move a block, place the cursor inside the selected block and select the *Move Block* option. This will cause the block selected to disappear. To retrieve the block, use the *Place Block* option. As many copies of the block as wanted may be retrieved.

• **Delete Block**

This option can be selected by positioning the arrow on this label and pressing the left button on the mouse. It allows the user to delete a block of text. The following steps show how to delete a block of text.

- Place the cursor arrow at the desired template structure to be deleted and then click the left button on the mouse.
- Place the cursor arrow inside the edit panel, click the left button on the mouse, and without releasing the button move the cursor arrow up or down in order to select the Delete block option.

• **Place Block**

After selecting the *Copy Block* or *Move Block* option, this will retrieve the block last copied or moved and place it at the cursor location.

File Menu Window

Most of the File menu is directly related to operations done on files. The file menu is given in Figure 5.11.

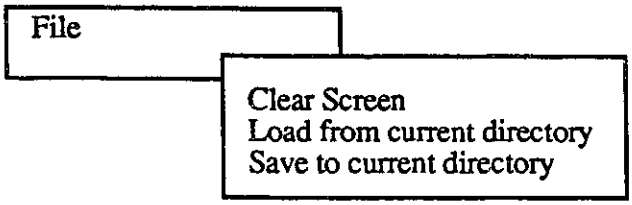


Figure 5.11 File Menu

• **Clear Screen**

The reset option is used when the user is finished with the current algorithm and plans to start another one. This operation will clear the text window of any text and the window

in which the file name is entered. If any of the text has been modified, the user will be prompted with the following message in the errors message area. " Erase the current file in the text window YES NO ". Choosing the YES option, clears the text window of any text, and any edits or changes made to the file in the text window will be lost.

• **Load File**

This option allows the user to load an existing file into the text window. To be able to do so, the user will first have to specify the file name in the window marked : File_name
Enter a file name and try the load option.

• **Save File**

When this option is chosen, the file under the name that is currently specified in the file name window will be saved.

• **Print File**

To be able to print a file, the user has to move the cursor to the print window and enter a file name. He can use either the laser printer or the impact printer.

Error messages window

When an error occurs, a "BEEP" will sound and a message will be displayed in the error message area. These are the most important errors which can occur and the possible solutions that may eliminate the errors. The user can press on the help option each time an error message is displayed to find the solution. The errors concern the rules applied to the indentation of the structures. Some examples of the rules are the following :

- One of the Main module can exist in a program and can occur only at the first level.
- An "if" structure can not be inserted on the line boxe of the graphic scheme.
- A while structure can not be inserted in a struct block.
- The others rules are most of them the C language rules for the structures.

The following error messages are displayed on the window situated at the lower bottom of the screen :

- *File Cannot be saved* : means system error occurred in trying to save the file to the disk. So the user has to try again.

- *File name was not specified* : Tried to either load , save or print a file, but did not enter a file name in the window marked : File_name
So the solution is the to enter a valid file name and retry the option.

- *Invalid file name specified, file cannot be found* : Tried to load or print a file with an invalid file name or the file exists but not on the current directory.
Enter a valid file name and retry the option or copy the file to the ALC directory.

- *Cannot perform block operation* :
The user specified the starting point below the end point in the file.
Then specify the starting point to be above the end point in the file.

- *Invalid insertion point specified* :

The user tried to insert text in an invalid position or tried to insert text within a block that was marked to be moved or copied.

Then move the cursor to a new valid insertion point and retry the insertion option.

- *Block has not been specified to be copied or moved* :

Attempt was made to move or copy a block of text without specifying which block of text had to be moved.

- *Insertion of the template will violate the language syntax rules* :

Attempt was made to insert a template structure in any position which will violate the language syntax rules.

Editing Facilities

In addition to inserting, deleting, moving blocks, ALC allows the user to use other editing facilities which consist of : Insertion point, deleting characters, scrolling buttons, undoing editing operations, selecting single or span of characters and text window facilities.

5.2 ARCHITECTURE OF THE SYSTEM

ALC is implemented in the C language. It requires the Sun view window environment of the Unix operating system. ALC uses many window facilities provided by the Sun view window environment. There are several Sun view window functions and procedures called by the ALC system. Procedures and functions have also been developed for window management purpose. To execute the ALC system, The minimum hardware requirement is a SUN 3 / 50 workstation. Figure 5.12 shows the architecture of ALC.

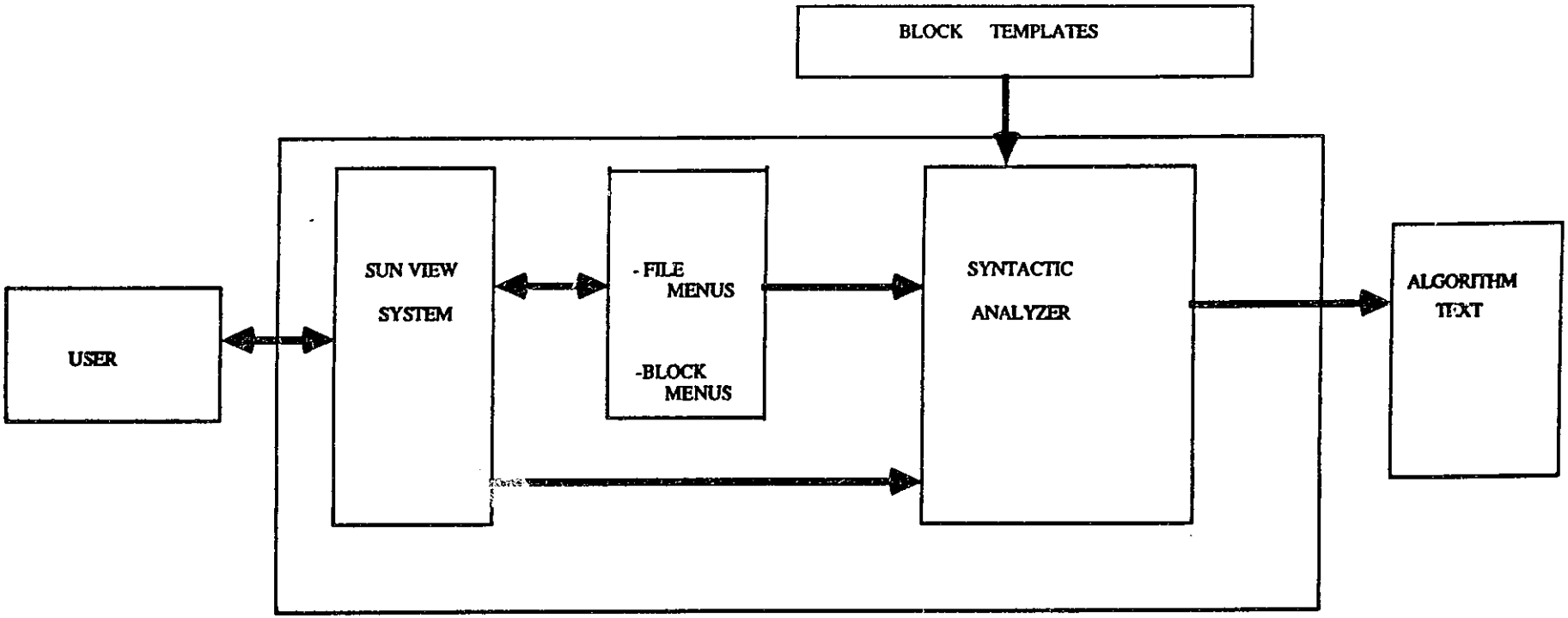


Figure 5.13 Architecture of ALC

Data Flow

Chapter 6

ALGORITHM- DIRECTED PROGRAMMING (CALC)

6.1 INTRODUCTION

CALC (Object Code Editor) is a software tool which translates structured algorithms, produced by ALC using the graphic scheme documented in chapter 4, into C programs. The algorithms edited in ALC are automatically compatible with the C language. When creating program structures using ALC, the syntax of the language C is verified as the blocks are inserted. This guarantees compatibility.

Given a structured algorithm, the major functions of CALC are :

- To transform keywords used in the structured algorithms into their corresponding keywords in C language.
- To remove all the boxes used when editing structured algorithms.
- To properly indent the C program obtained, to show the block structure.

The use of an ASCII format for the saved algorithms of ALC, allows CALC to be implemented as a simple text translator.

The CALC window interacts with the ALC window to allow the user to edit either the code or the corresponding algorithms. Any updates made in the algorithm are reflected in the C program but not the reverse. Figure 6.1 and Figure 6.2 show the screen of ALC with CALC when editing algorithm.

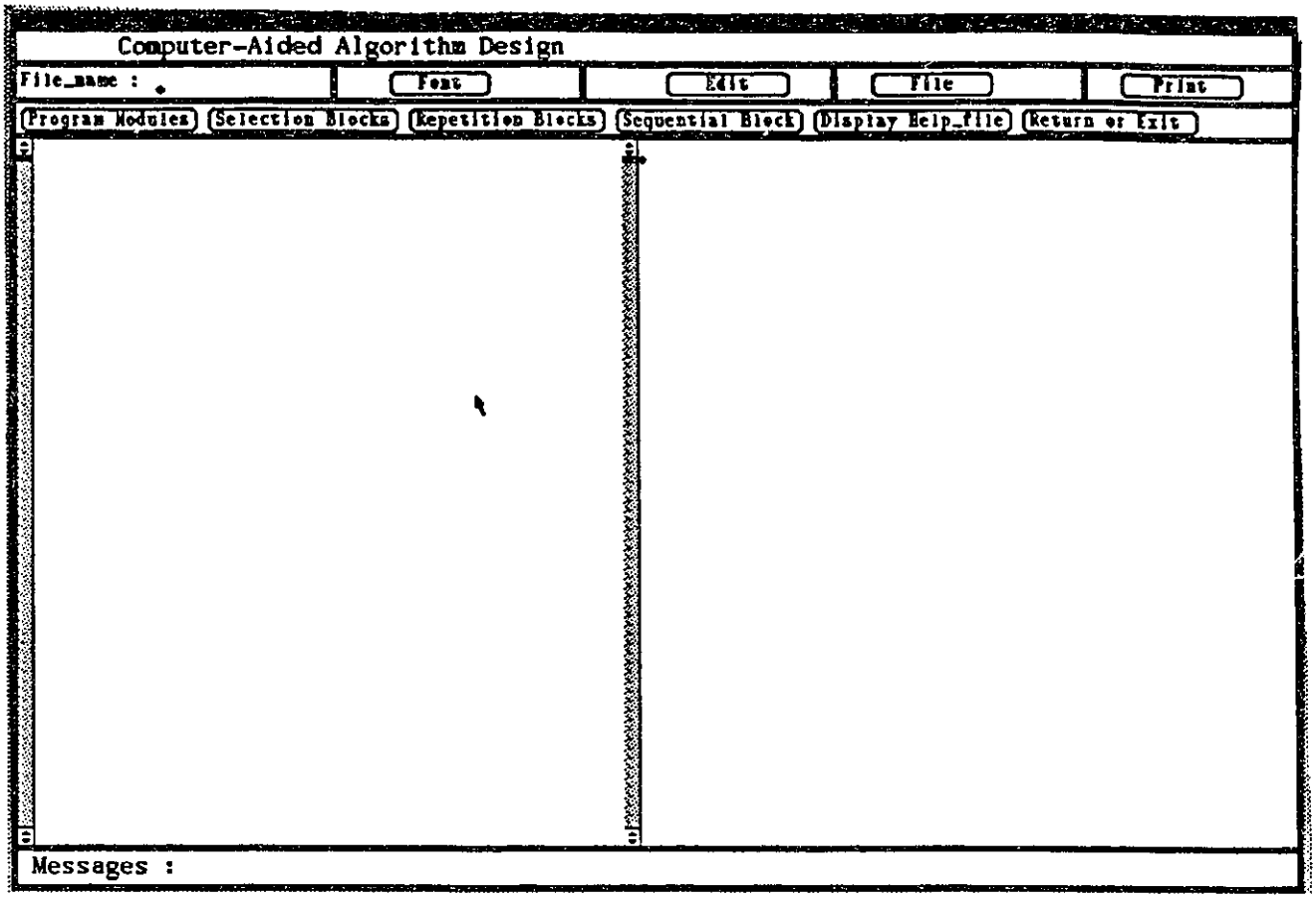


Figure 6.1 The Screen of CALC with ALC

After saving the algorithm edited in the algorithm window under the filename : example, the C version of the file example is displayed on the CALC window.

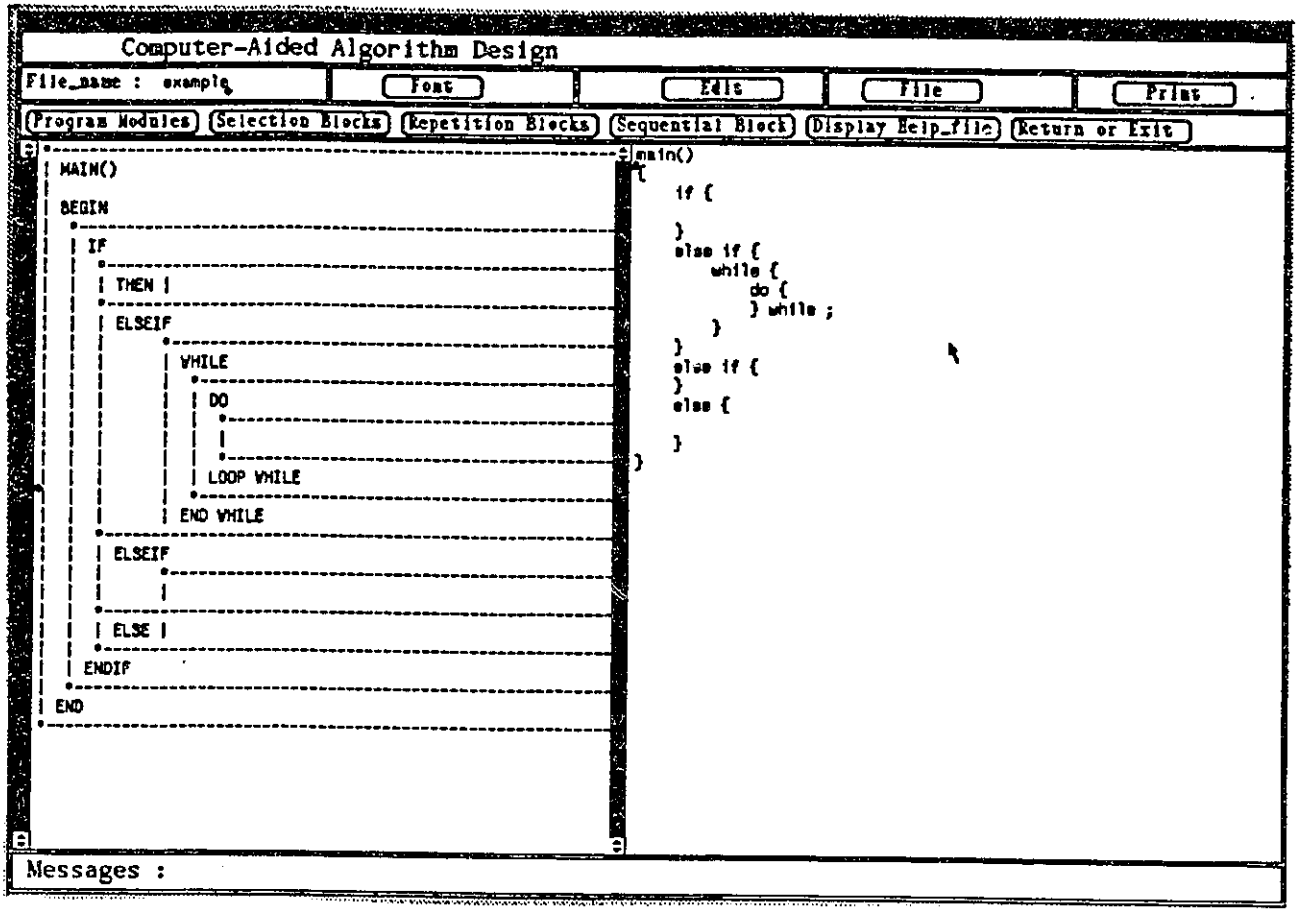


Figure 6.2 The File Example is saved and its C Version is displayed on the CALC window

6.2 USE OF LEX

Lex is a program generator designed for lexical processing of character input streams. Lex accepts a problem oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the programmer in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into tokens matching the expressions. At the boundaries between tokens, program sections provided by the programmer are executed.

The Lex source file is a table of regular expressions and corresponding program fragments. The table is translated by the Lex tool into a program which reads an input stream, and partitions the input into tokens which match the given expressions, as each such token is recognized, the corresponding program fragment is executed. The recognition of the expression is performed by a deterministic finite automaton generated by Lex. The program fragments written by the programmer are executed in the order in which the corresponding regular expressions occur in the input stream.

In general, Lex can be used for simple transformations, or for analysis and statistic gathering on a lexical level.

In the case of CALC as each token is recognized in the ALC input by the lexical analyzer, the Lex written code outputs the appropriate C language construct.

6.3 IMPLEMENTATION OF CALC

CALC program is a Lex specification. It uses boxes and keywords as tokens. Lex will divide the input (structured algorithm) into tokens. Actions (to be performed when each token is recognized) are provided in CALC specification.

The format of the Lex specification is :

{ definitions }

% %

{ rules }

% %

{ programmer subroutines }

In general, the definitions part is optional. For CALC, the definitions are the ALC keywords, boxes, white spaces and line delimiters. They are used to simplify the format of the rules. They provide convenient short forms for use in the regular expressions.

The rules represent the programmer's control decisions; they are a table, in which the left column contains *regular expressions* and the right column contains *actions* , program fragments to be executed when the expressions are recognized. An example of rule is given below :

```
"ELSE"          {

                /*  some C code  */

                }
```

This rule has to look for the string "ELSE" in the input stream and when recognized executes the program fragment code between the two braces. This code fragment can access C

subroutines present in the programmer subroutines section of the specification.

In CALC specification the host procedural language is C and the C library function are also used.

By using the ALC source format, translation to other procedural languages can be provided by writing a translator like CALC for each target language.

To simplify the implementation of back-spacing to cancel indentation, the subroutine rev_indent () inserts ASCII back-space characters in the output. The output is then processed by a program called translator to interpret the back space as deletions of the preceding blanks. The appropriate characters are completely removed from the output.

Chapter 7

ORGANIZED REPRESENTATION OF C PROGRAMS (ORC)

7.1 INTRODUCTION

ORC (Organized Representation of C Programs) is a software package to document syntactically correct C programs. The documentation which is automatically generated by ORC is a structured flowchart of a C program.

ORC accepts as input a C source file which may consist of a complete C program, or one of several subprograms (procedures and / or functions). The graphic scheme used in ORC was documented in Chapter 4. When documenting the programs, the statements which are too long to fit on one line are intelligently wrapped onto several lines. The lines are wrapped at word, or token boundaries, whenever possible.

ORC is implemented on a Sun Workstation. It is written in C using SunView functions.

Examples of ORC documentation

The following two examples show the types of documentation generated by ORC. The first example is provided to show the graphical representation of an if-then-else, a while, and a for structures. Figure 7.1 is a C program to find all lines matching a pattern and Figure 7.2 is the ORC'ed documentation of the C program. The second example depicts the graphical documentation of a case statement. Figure 7.3 shows a C program to count digits, spaces and others characters. Figure 7.4 provides the ORC'ed documentation of the C program which is given in Figure 7.3.

```
#define         MAXLIGNE                1000

main()          /* find all lines matching a pattern */
{

        char ligne[MAXLIGNE];


        while (avoirligne ( ligne , MAXLIGNE )  >  0)
                if (position (ligne,"the") >= 0)
                        printf ("%s",ligne );


}


avoirligne (s,limite)   /* get line into s, return length */
char s[];
int limite;
{

        int c,i;
        i = 0;
        while (--limite > 0 && (c=getchar() ) != EOF && c != '\n' )
                s[i++] = c;
        if ( c == '\n' )
                s[i++] = c;
        s[i] = '\0' ;
        return (i) ;
}


position (s,t)  /* return index of t in s, -1 if none */
char s[], t[];
{
        int i , j, k;

        for (i= 0;s[i] != '0' ; i++)  {
                for (j=i; k=0 ; t[k] !='\0' && s[j] == t[k]; j++, k++)
                        ;
                if (t[k] == '\0' )
                        return(i);
                }
                return(-1) ;
}
```

Figure 7.1 A C Program to Find all Lines Matching a pattern

```
# define MAXLIGNE 1000
 main()
 /* find all lines matching a pattern */

char ligne[MAXLIGNE];

while(avoirligne(ligne,MAXLIGNE)>0)
  if(position(ligne,"the")>=0)
    THEN printf("%s",ligne);
  ENDIF
ENDWHILE


 avoirligne(s,limite)
 /* get line into s, return length */

 char s[];
 int limite;

int c,i;
i=0;
while(--limite>0&&(c=getchar())!=EOF&&c!='\n')
  s[i++]=c;
ENDWHILE
if(c=='\n')
  THEN s[i++]=c;
ENDIF
s[i]='\0';
return (i);


 position(s,t)
 /* return index of t in s, -1 if none */

 char s[],t[];

int i,j,k;
for(i=0;s[i]!='0';i++)
  for(j=i;k=0;t[k]!='\0'&&s[j]==t[k];j++,k++)
    ;
  ENDFOR
  if(t[k]=='\0')
    THEN return (i);
  ENDIF
ENDFOR
return (-1);
```

Figure 7.2 Organized Representation of the C Program given in Figure 7.1.

```
main()    /* count digits,white space,others */
{
        int c,i,blancs,autres,chiffre[10];
        blancs = autres = 0 ;
        for (i=0;i<10;i++)
        chiffre[i] = 0;
        while ((c=getchar()) !=EOF)


          switch (c)  {
          case '0' :
          case '1' :
          case '2' :
          case '3' :
          case '4' :
          case '5' :
          case '6' :
          case '7' :
          case '8' :
          case '9' :
                chiffre[c-'0']++;
                break;
          case '\n' :
          case '\t' :
          case ' ' :
                blancs++;
                break;
        default :
                 autres++;
                 break;
                 }

 printf("chiffres=");
 for (i=0;i<10;i++)  {
   printf("%d",chiffre[i]);
printf("\blancs espace =%d,autres = %d\n",blancs,autres);
        }
}
```

Figure 7.3 A C Program to count digits, white space and others

```
main()
/* count digits,white space,others */
int c,i,blancs,autres,chiffre[10];
blancs=autres=0;
for(i=0;i<10;i++)
  chiffre[i]=0;
ENDFOR
while((c=getchar())!=EOF)
  switch(c)
    case '0':
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
      chiffre[c-'0']++;
    <=|= break;
    case '\n':
    case '\t':
    case '':
      blancs++;
    <=|= break;
    default:
      autres++;
    <=|= break;
  ENDSWITCH
  ENDWHILE
printf("chiffres=");
for(i=0;i<10;i++)
  printf("%d",chiffre[i]);
  printf("\blancs espace =%d,autres = %d\n",blancs,autres);
ENDFOR
```

Figure 7.4 Organized Representation of the C Program given in Figure 7.3.

7.2 USING ORC

Once the user types ORC, the screen given in Figure 7.5 will appear.

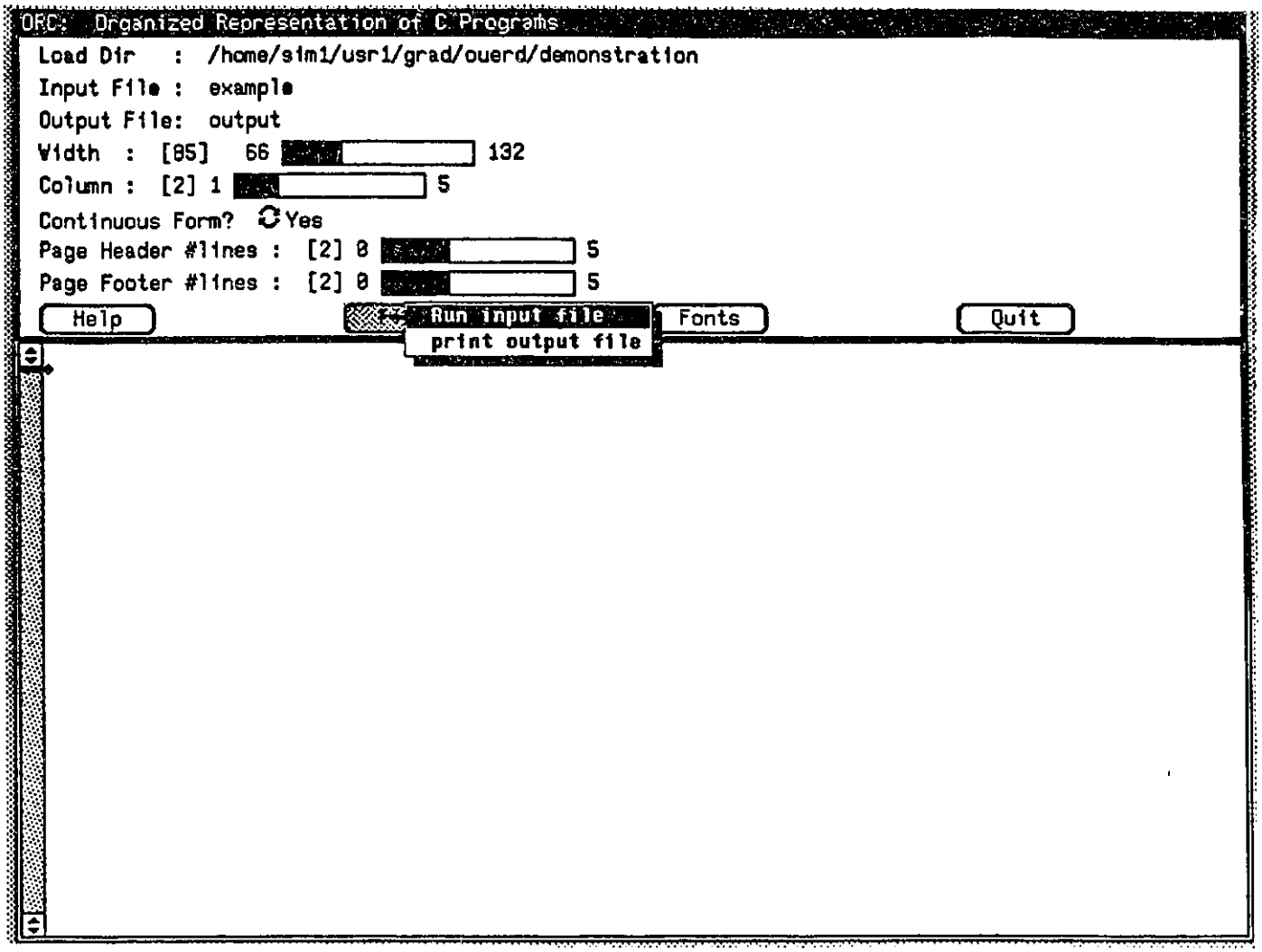


Figure 7.5 The ORC Screen

• A set of pull down menus are provided to facilitate the use of OR

After entering the input file and the output file, the user can select the run file option to document the input file. The output file will be displayed on the screen. (See Figure 7.6. below).

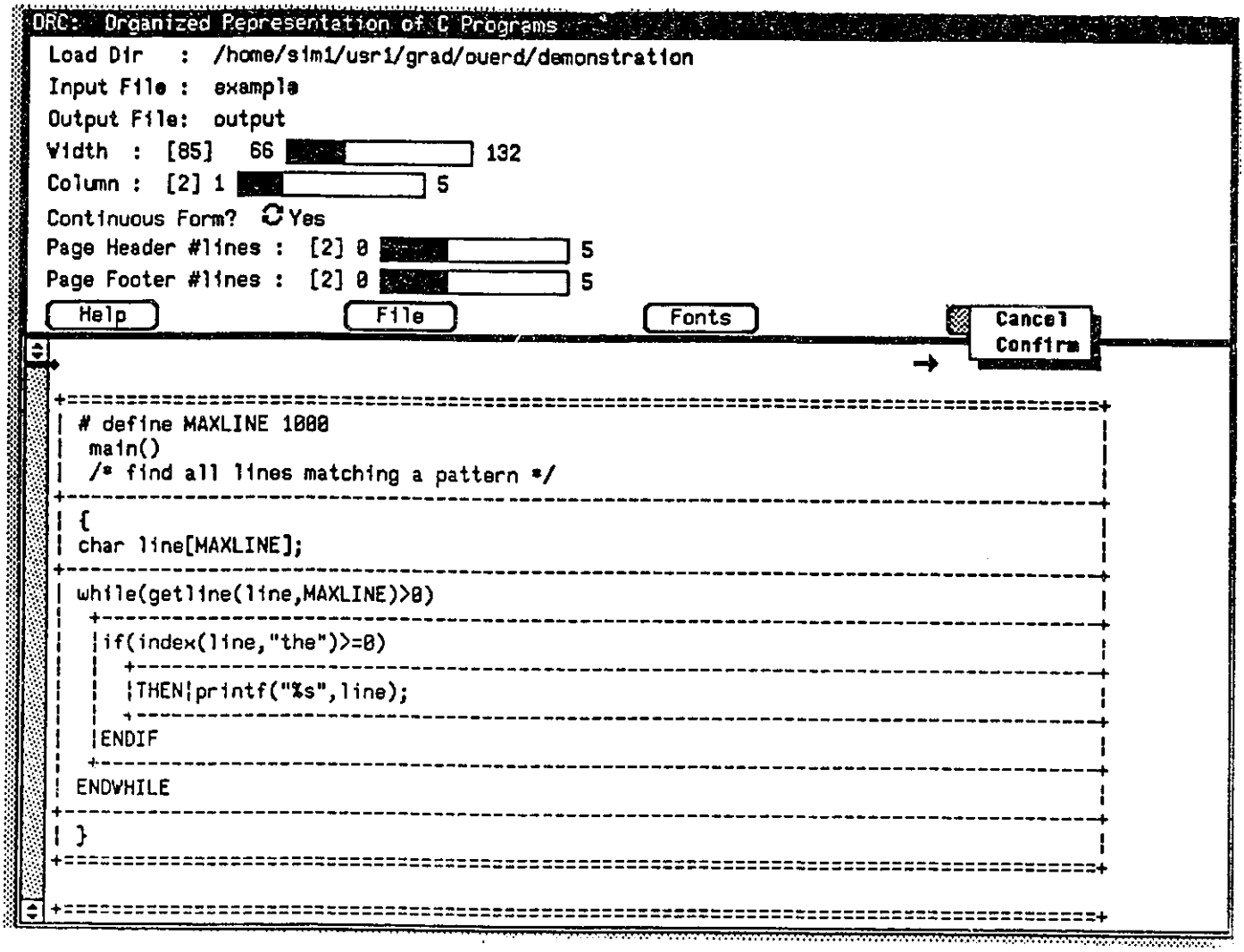


Figure 7.6 The ORC'ed file of the input file specified in Figure 7.5.

A set of parameters are provided for the user of ORC. The meaning of each of the various parameters is given below :

- Input File : This field selects the input filename to be processed. This file should contain valid C source file.

- Output File : This field selects the output filename to be used while processing. The output file will contain the *Organized Representation* corresponding to the input file.

- Width : This field sets the width parameter for the structures blocks. The meaning of the width is shown in Figure 7.7.

- Column : This field sets the column parameter for the structures blocks. The meaning of the column parameter is shown in Figure 7.7.

- Continuous Form : This field selects the option "YES" or "NO". The purpose of this parameter is decide whether or not the user wants the page breaks or not in the output document.

- Page Header #lines : This field sets the header parameter. The meaning of this parameter is shown in Figure 7.7.

- Page Footer #lines : This field sets the footer parameters. The meaning is shown in Figure 7.7.

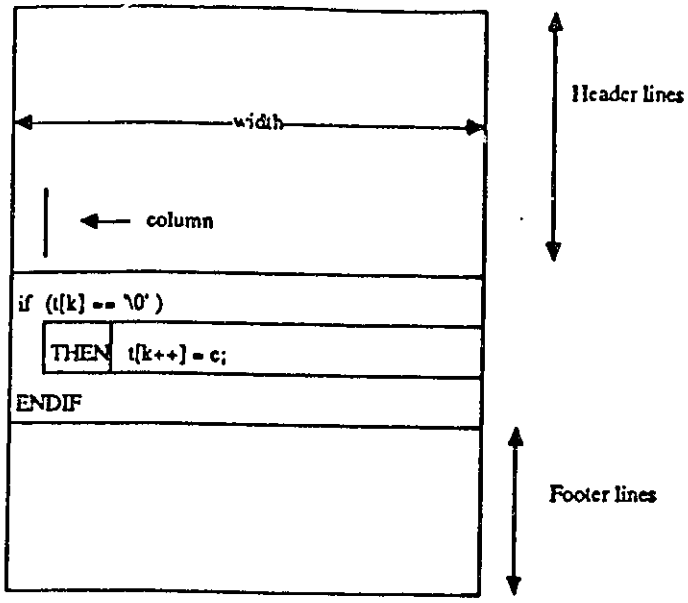


Figure 7.7 The Meanings of the Conversion Parameters are shown for an Example Block in the Output

Default and Valid Values of the Parameters :

In Figure 7.7, we used "[ ]" to represent default values of the parameters. The default values for the ORC parameters have been chosen to provide the user with a quick and easy way to get a nice looking output document from the ORC system. They are shown in Figure 7.8.

Parameter name	Default Values	Valid Values	
		Limit	
		Lower	Upper
Input File	none		
Output File	none		
Width	85	66	132
Column	2	1	5
Page Header lines	2	0	5
Page Footer lines	2	0	5

Figure 7.8 Default and Valid Values used for the Parameters of ORC

7.3 IMPLEMENTATION OF ORC

The purpose of ORC is to generate from a source program a structured output of the same source input. The user specifies the source file to be analyzed. One string of source code will be transmitted at a time to the lexical analyzer or scanner.

The lexical analyzer will transform each string into series of tokens. Once the entire source program has been analyzed, the control is passed to the output generator through two temporary files. One contains the tokens, and the other, the actual value of the non-keyword tokens.

Since the entire program has been transformed in a series of tokens, the output generator will reconstruct the source code by using those tokens. It will produce the same source program in a structured fashion.

Tokens are useful to the output generator for recognizing structures of C.

This approach is used for its simplicity to recognize the structures when all the tokens are clearly identified.

7.3.1 DATA FLOW DIAGRAMS

Figure 7.9 to Figure 7.13 show data flow diagrams of ORC.

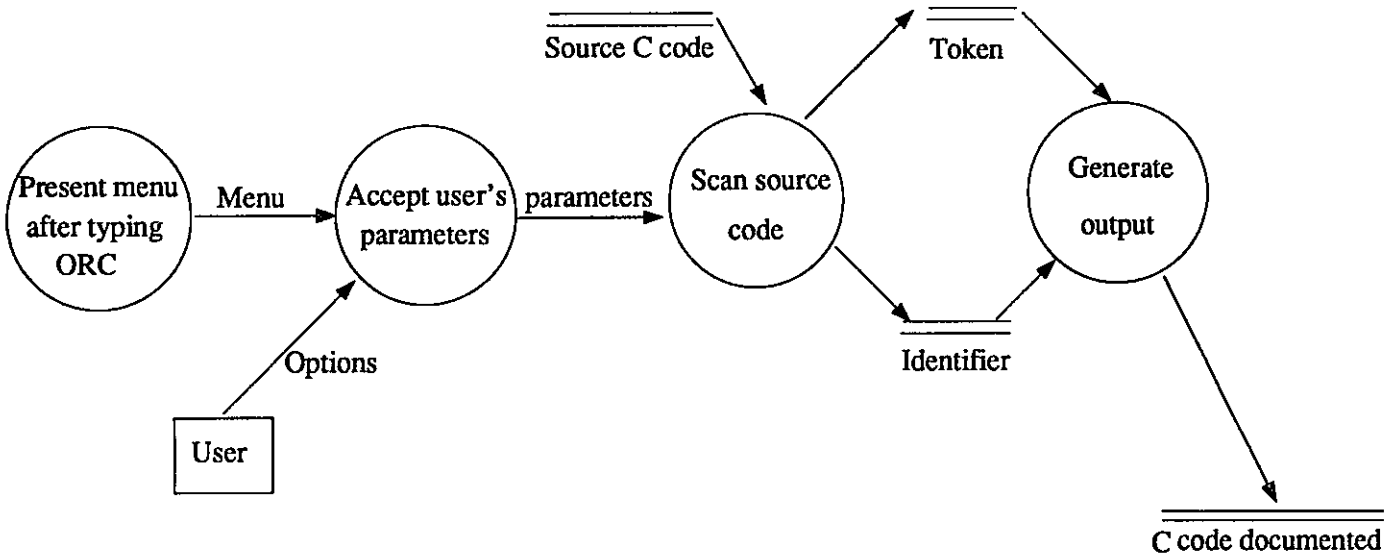


Figure 7.9 ORC Document Generator

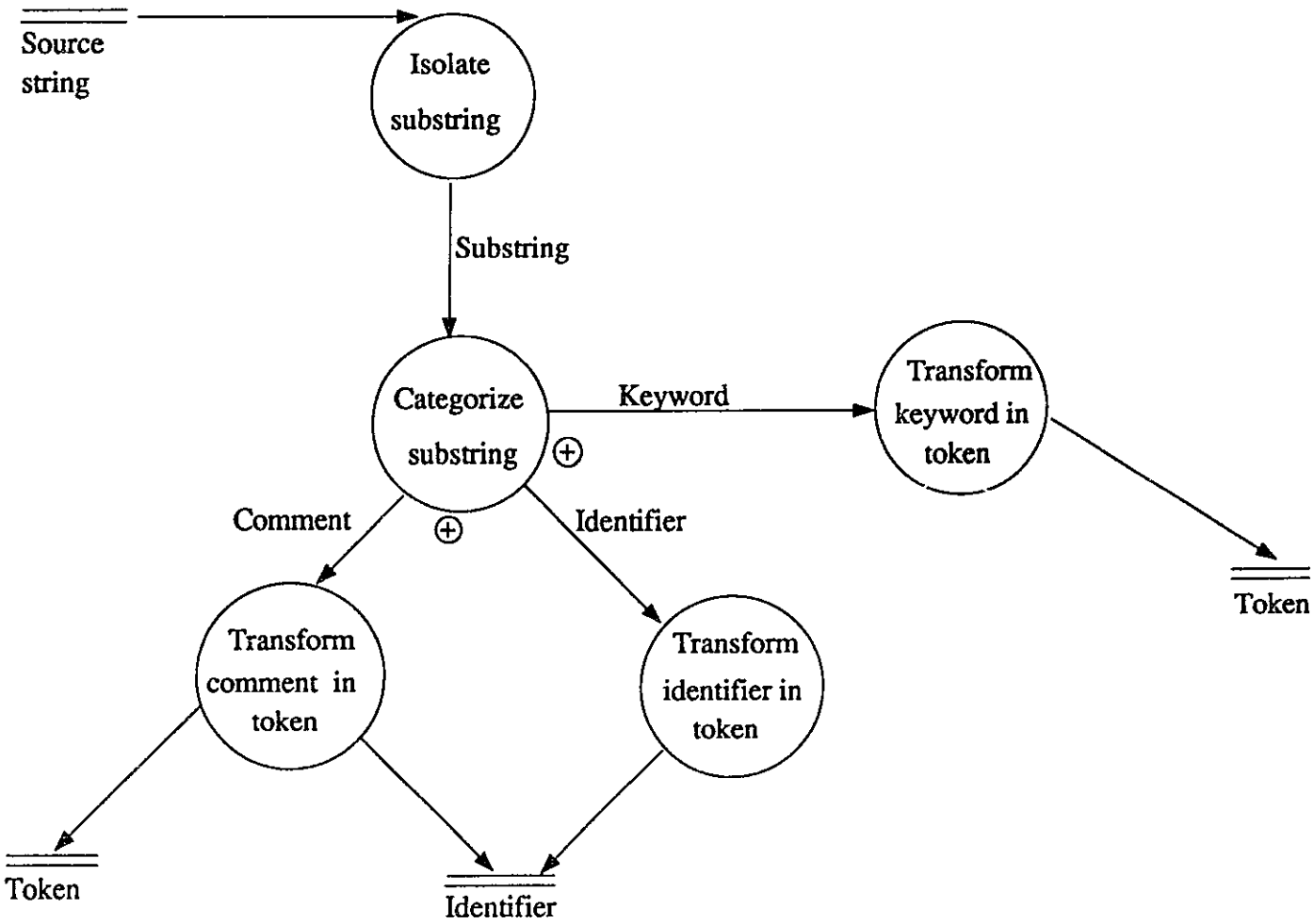


Figure 7.10 Scan Source Code

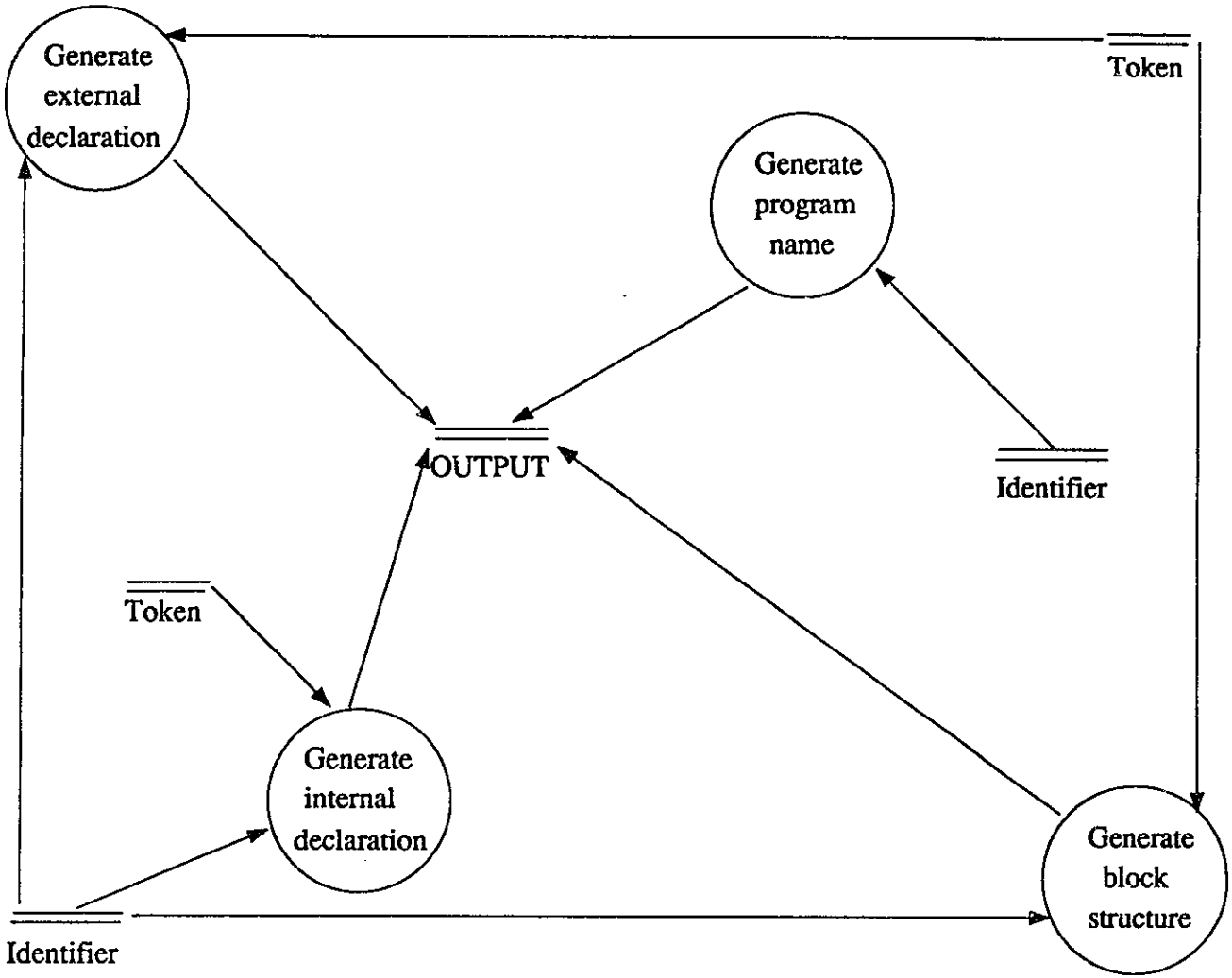


Figure 7.11 Generate Ouput

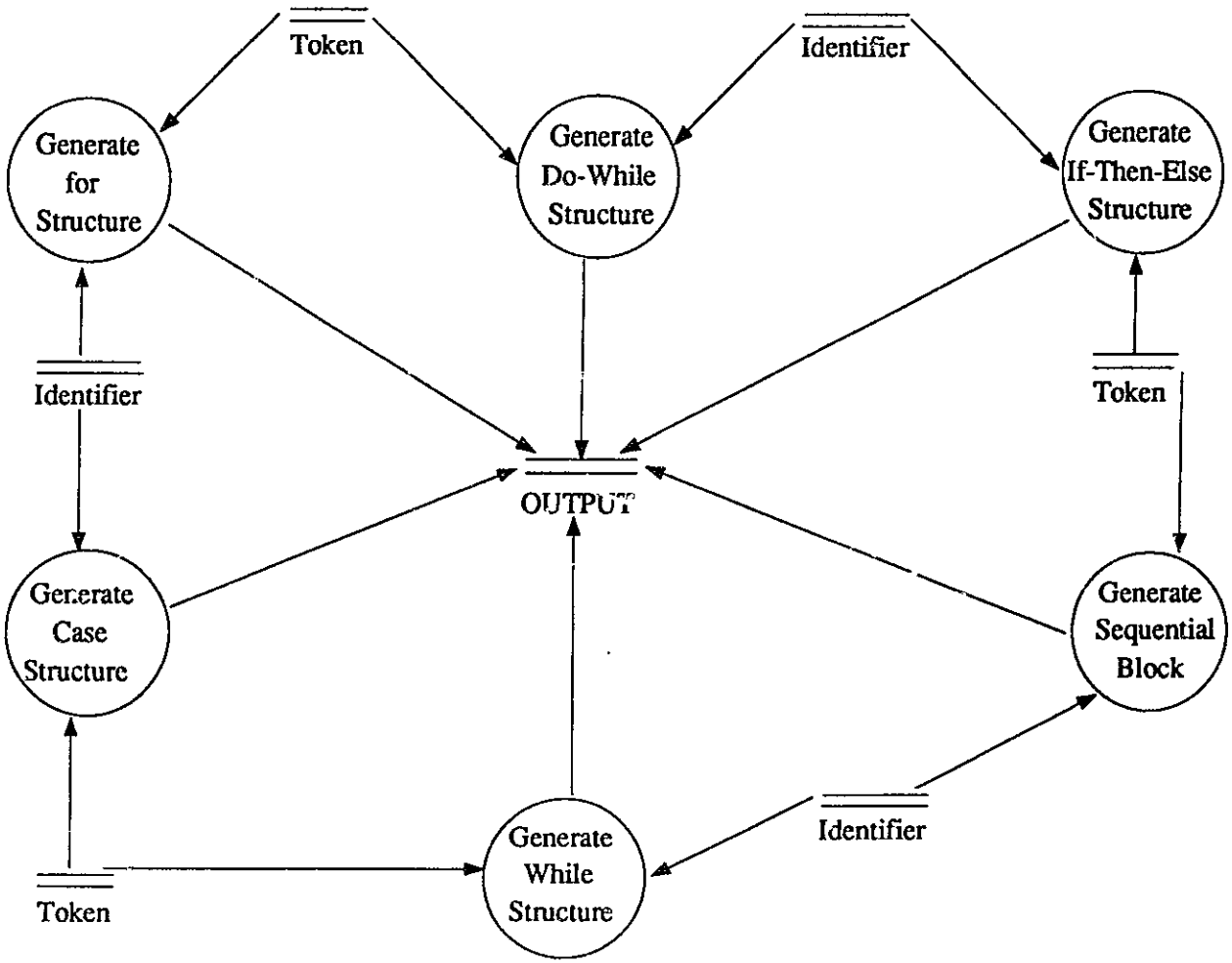


Figure 7.12 Generate Block Structures

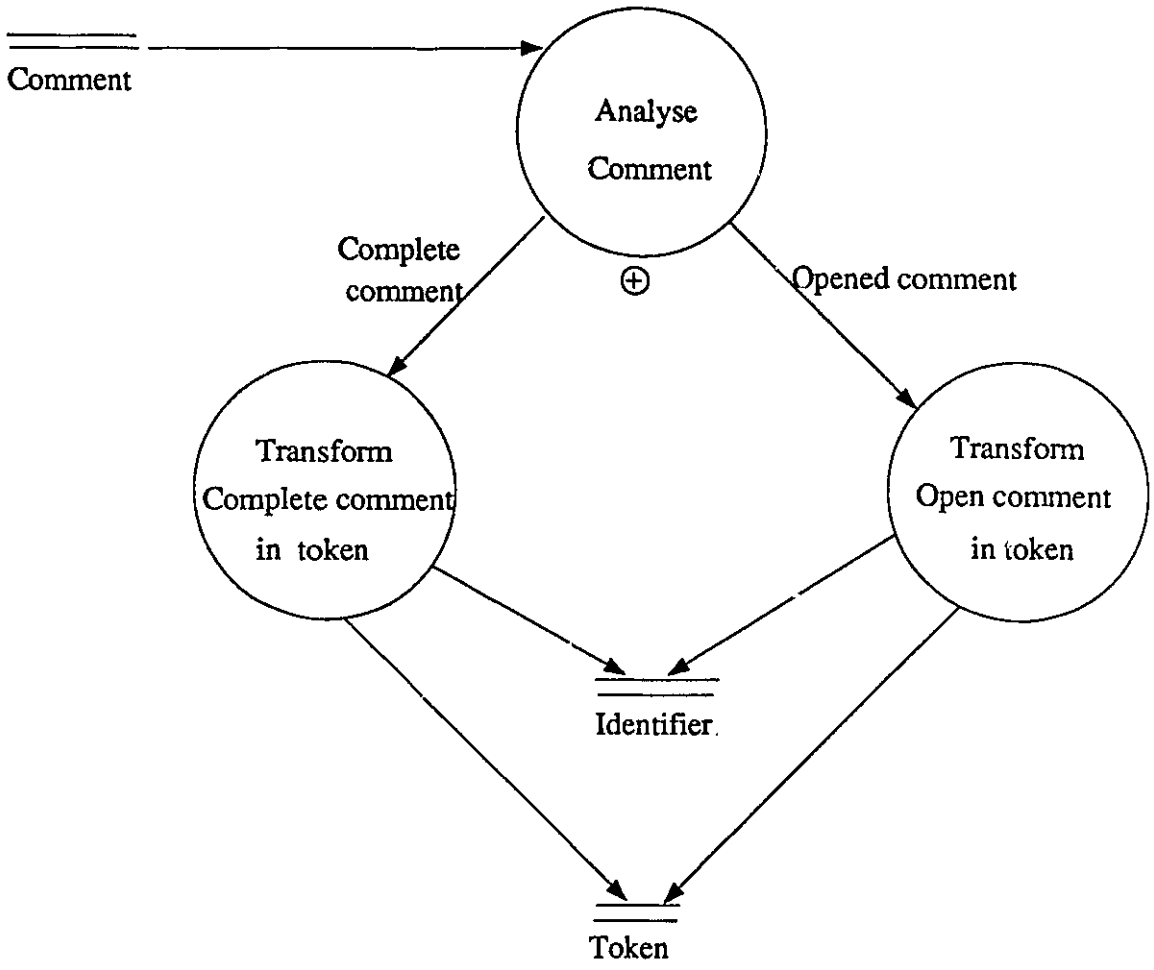


Figure 7.13 Transformation of Comment into Token

Chapter 8

CONCLUSION AND FURTHER RESEARCH

An algorithm directed CASE environment has been implemented and provides important features to enhance comprehension and understanding of algorithms and C programs. Its main purpose is to improve productivity in systems developments and to :

- Facilitate maintenance of existing systems
- Obtain more complete documentation
- Reuse code
- Reduce error and thereby improve quality and user satisfaction
- Built-in quality

The basic building blocks (i.e., sequential, selection, and repetition blocks) used in the graphic scheme do not require detailed understanding of programming. Therefore they can be taught, learned, and assimilated very early in the education process.

The logical structure of the algorithm is identical to the logical structure of the program. It is better seen with a graphic technique.

- The chances of detection and elimination of logical errors are increased
- Comprehension of algorithm is enhanced
- Modification of algorithms can be done with ease and it may be less error prone.

ALC is designed for ease of use by novice programmers in a teaching situation. There was a need for something straightforward to use, particularly for novice programmers, which would help in the construction and teaching of programs. The system provides a very important pedagogical tool.

ALC makes the transition from design to code easier. The correct syntax (i.e, the structures used in the algorithm follow a certain syntax) for algorithm is ensured.

The menus make the system easy to use and friendly.

The C program is displayed in a parallel window to the algorithm window. In this way, the users can obtain immediate feedback on the coverage of program elements, while still inside an editing session.

FUTURE WORK

The statements included in the algorithm can be checked in the C program by adding a Yacc specification to the Lex specification used in CALC.

Furthermore, the diagrams displayed in the ALC text window can be collapsed, i.e ; only the first n levels of structure will be shown.

The research which uses the graphic scheme continues at the Computer Science Department of the University of Ottawa. The activities include extension of the building blocks to other structured languages and other concepts. This includes investigation of building blocks for "concurrent activities " as well as for object-oriented languages such as C++ and Smalltalk.

It is hoped that the system implemented in this thesis will prove its merits to educators as well as to professionals for the conception and design of well-structured algorithms and programs.

REFERENCES

Bachman, C., A CASE for Reverse Engineering, Datamation , July 1, 1988.

Chikofsky, E.J., Cross, H., J.H., (1990.) Reverse Engineering and Design Recovery : A Taxonomy , IEEE Software.

Chikofsky, E.J., Rubenstein, B.L., (1988.) CASE : Reliability Engineering for Information Systems . IEEE Software.

Communications of the ACM (1986.), 29: 11 (Nov.), 1023.

Dyck, V.A., Lauson, J.D., Smith, J.A., (1979.) Introduction to Computing Structured Problem Solving Using WATFIV-S. Reston Publishing Company, Inc. A Prentice Hall Company, Reston, Virginia.

Faroult, S., Simon, D. (1986.) Fortran Structuré et Méthodes Numériques, Dunod, Paris, France.

Grogono, P., Nelson, S.H. (1982.) Problem Solving and Computer Programming, Addison - Wesley.

IBM Journal of Research and Development, Programming Languages and Languages Processors, pp.657-800 • Index, Vol.24, No. 6, Nov. 1980.

Knuth, D.E. (1973.) The Art of Computer Programming, Vol. I : Fundamental Algorithms. Reading, Mass.: Addison- Wesley.

Martin, J. and McClure, C. (1985.) Diagramming Techniques for Analysts and Programmers. Prentice - Hall.

McClure, C., (1988a.) Characteristics of a CASE System. Extended Intelligence, Inc.

McClure, C., (1988b.) Introduction to the CASE Technology. Extended Intelligence, Inc.

Mitchell.W. (1984.) Prelude to Programming, Problem Solving and Algorithms . Reston.

Ören,T.I., L.G.Birta, O.Abou-Rabia, D.G.King, and R. Wendt (1990 - In Press). E/Slam: A Software Understanding Environment for SLAM II Programs. In Proceedings of European Simulation Multiconference (Erlangen-Nuremberg, Germany, June 10-13, 1990), SCS, La Jolla, CA.

Ören, T.I. (1984.) Graphic Representation of Pseudocodes and Computer Programs: A Unifying Technique and a Family of Documentation Programs. In: Proc.of EdComp Conf 83 (First Educational Computing Conf. of IEEE Computer Society), D.C.Rine (Ed.). San Jose, Ca., Oct. 18-20, 1983. IEEE Computer Society, New York, pp. 81-89.

Ören, T.I., King, D.G., (1989.) ORFOR : Organized Representation of Fortran Programs on a Sun Workstation. TR-89-16. Computer Science Department, University of Ottawa, Ontario, Canada.

Perrone, G., Marietta, M., (1987.) Low -cost CASE : Tomorrow's Promise Energing Today, COMPUTER.

Shuller, H.E., (1987.) Requirements for Computer Aided Software Engineering Tools. CASE studies 1987 Conference.

Stinson, D.R. (1985.) An Introduction to the Design and Analysis of Algorithms. Winnipeg, Manitoba, Canada, CBRC.

Sun Microsystems (1986a.) Windows and Window Based Tool : Beginner's Guide. Sun Microsystems, Inc., Mountain View, California, U.S.A.

Sun Microsystems (1988). Programming Utilities and Libraries. Sun Microsystems, Inc., Mountain View , California, U.S.A.

Sun Microsystems (1986b.) Sun View Programmer's Guide. Sun Microsystems, Inc., Mountain View , California, U.S.A.

Venable, J.R., Duane, P.TRUX III., (1988.) An Approach for Tool Integration in a CASE Environment. CASE studies 1988 Conference.