INFORMATION FLOW COMPLEXITY

by

Pakarat Udomphorn
B.A., Ramkhamheang University, Thailand, 1977

-----------------------------

A MASTER'S REPORT

submitted in partial fulfillment of
the requirement for the degree
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1984

APPROVED BY:

[Signature]
MAJOR PROFESSOR
ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my major advisor, professor David Gustafson for his guidance, patience and encouragement. He has made this a pleasant and rewarding learning experience. I also would like to extend a personal thank you to my committee members, Professor William Hankley and Professor Rod Bates for their helpful suggestions and comments.

I am grateful to my mother for her constant support and encouragement which have been of utmost importance to the success of my education.
ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE
CONTENTS

1. INTRODUCTION ........................................ 1
   Organization of the Report ......................... 2
   Complexity Measures .................................. 3
   Types of Flow ......................................... 4
       Intermodule Control flow ....................... 5
       Intermodule Information Flow .................. 5
       Intramodule Control Flow ....................... 7
       Intramodule Information Flow .................. 7
   Relating Information Flow .......................... 8

2. INFORMATION FLOW ....................................... 10
   Explicit Information Flow ............................ 11
   Implicit Information Flow ............................ 15

3. DESIGN ..................................................... 21
   Requirement Specification ........................... 21
   IFC input program ................................... 22
   IFC outputs ........................................... 24
   Running IFC ............................................ 26
   Data Structure ........................................ 27

4. IMPLEMENTATION ......................................... 34
   System diagrams ...................................... 35
   System logic ......................................... 40
   IFC sample program .................................. 46

5. CONCLUSION AND EXTENSION .......................... 49

REFERENCES ................................................ 52

Appendix 
A. GLOBAL VARIABLES AND ROUTINES LISTING ........ 54
B. IFC PROGRAM LISTING ................................. 68
C. USER'S GUIDE ......................................... 100
D. SAMPLE PROGRAM LISTINGS ............................ 106
<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Explicit information flow graph</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Explicit information flow graph of I/O statement</td>
<td>13</td>
</tr>
<tr>
<td>2.3a Explicit information flow graph of routine heading</td>
<td>14</td>
</tr>
<tr>
<td>2.3b Explicit information flow graph of routine heading with returned value</td>
<td>14</td>
</tr>
<tr>
<td>2.4 Implicit information flow graph</td>
<td>15</td>
</tr>
<tr>
<td>2.5 Implicit information flow graph of WHILE statement</td>
<td>17</td>
</tr>
<tr>
<td>2.6 Implicit information flow graph of IF THEN ELSE</td>
<td>18</td>
</tr>
<tr>
<td>2.7 Implicit information flow graph of FOR statement</td>
<td>19</td>
</tr>
<tr>
<td>2.8 Implicit information flow graph of REPEAT statement</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Relating data structure</td>
<td>32</td>
</tr>
<tr>
<td>3.2 A sample of relating data structure</td>
<td>33</td>
</tr>
<tr>
<td>4.1 System diagram</td>
<td>35</td>
</tr>
<tr>
<td>4.2 Hierarchical diagram of Body routine</td>
<td>35</td>
</tr>
<tr>
<td>4.3 Hierarchical diagram of PROCESSVAR_TYPE routine</td>
<td>36</td>
</tr>
<tr>
<td>4.4 Hierarchical diagram of PROCESSPROC routine</td>
<td>36</td>
</tr>
<tr>
<td>4.5 Hierarchical subdiagram of PROCESSVAR_TYPE and PROCESSPROC routine</td>
<td>37</td>
</tr>
<tr>
<td>4.6 Hierarchical diagram of PROCESSBEGIN routine</td>
<td>37</td>
</tr>
<tr>
<td>4.7 Hierarchical diagram of ACCESSBLOCK routine</td>
<td>38</td>
</tr>
<tr>
<td>4.8 Hierarchical diagram of simple statements routine</td>
<td>39</td>
</tr>
<tr>
<td>4.9 Hierarchical diagram of ANALYSIS routine</td>
<td>39</td>
</tr>
<tr>
<td>4.10 Example for explaining INDXTAB</td>
<td>43</td>
</tr>
<tr>
<td>4.11a INDXTAB stack after the analysis of I/O statement</td>
<td>45</td>
</tr>
<tr>
<td>4.11b INDXTAB stack after the analysis of assignment</td>
<td>45</td>
</tr>
<tr>
<td>4.12 IPC sample program and output</td>
<td>48</td>
</tr>
</tbody>
</table>
CHAPTER ONE
INTRODUCTION

This report describes the purpose, design and implementation of a software tool that determines the information flow in Pascal programs. Information flow refers to the dependency of a variable on the value of other variables. The software tool is called the information flow complexity (IFC) program. It is implemented on a PDQ-3 computer. The IFC will be used as a research tool to investigate the complexity of software.

Software is becoming more complex because of an increase in the user's needs and because of the rapid advancement in hardware technology. Rapid advancements in hardware cause new system software to become desirable or necessary. Extensions to existing system software are required in order to take full advantage of the new hardware capabilities. With the increasing complexity of new and existing software, there is an increasing need for powerful and adaptive development tools and techniques. Maintenance costs tend to dramatically increase the costs of the software system, often costing more than the rest of the software over its life cycle.
Complexity metrics attempt to quantify the complexity of software. Complex software may be very hard to understand. Some program modules may be more complex than others and harder to understand. More precisely, the complexity of a program is related to the difficulty people have in understanding the function implemented by the software (Chap 79) (Bak 77). This report is concerned with the use of information flow as a possible complexity metric.

ORGANIZATION OF THE REPORT

Chapter one discusses an overview of the recent work concerning information/control flow. Chapter two introduces a description of the IFC, describing the information flow features associated with the flow structure. Chapter three explains in detail the specification of IFC input programs and the output of the IFC. Included are the major data structures used in the implementation. Chapter four illustrates the system diagram and describes how to build and use the data structure to analyze the information transfer in the input program. Chapter five discusses the conclusions and extensions which can be done to make the IFC program more powerful. There are appendices provided for users who need to use or modify the IFC program. Appendix A contains the listing of global variables and routines. Appendix B contains the listing of IFC program. Appendix C contains user's guide. Appendix D contains the listing of ten sample programs.
COMPLEXITY MEASURES

Evaluating software quality is becoming a very important activity. Much of the recent work in software engineering has been concentrated in various measures concerning software complexity. Thus the definition of complexity has various meanings depending on the object of interest. The metrics may be used in a software development tool as a mechanism for judging the technical quality of software and the adequacy of the design. Several of these complexity measures concentrate on specific structural characteristics of programs. Halstead's software science counts operators and operands (Hal 77). McCabe's cyclometric complexity counts the number of basic paths through the program (McC 76). The study of Zolonowski and Simons (Zol 77) includes the measurement of programming styles and methods and involves the relationship between program characteristics and program error amounts and types. Berlinger (Ber 80) sees the need to derive a measure of mathematical and intuitive correctness.
Types of Flow

In attempting to evaluate software quality, there are many methods frequently mentioned in the area of information (data) and control flow. The information flow involves the flow relation of program variables that have an influence in transferring the value of information upon the flow structure (Denn 76). Data and control flow have been characterized by Withwort as follows:

"Regarding software as a mechanism for output production reduces the problem of understanding a software design to the problem of understanding the flow of control and resulting derivation of the system outputs. Control flow analysis involves tracing paths through the design and understanding the logical condition associated with branching and looping in the structure. Dataflow analysis involves tracing some intermediate data and ultimately to system inputs and constants." (With 80)

The various measures cover either the intramodule or intermodule complexity. The intramodule complexity measures deal with the complexity of algorithms (i.e. the inner working of individual system modules). The intermodule complexity measures deal with the interconnection of the system modules. Based on these observations, complexity measures concerning data/control flow can be classified into four types: (1) intermodule control flow; (2) intermodule information flow; (3) intramodule control flow; (4) intramodule information flow.
(1) Intermodule Control Flow.

Intermodule control flow attempts to examine a basic flow graph of a program, the understanding of how program control is passed from module to module, regardless of program loops and program characteristics. An example is McClure’s complexity metric (McC 76) which involves an examination of the possible execution paths, the control structures and the variables used to direct path selection. In a well-structured program, the execution hierarchy of a program is determined by the values of control variables in a program loop. She defines three sources for module complexity: (1) the control structures used in module invocation, (2) the control variables referenced in module invocation, (3) the commonality of a module which is a part of the execution hierarchy. She suggests that the complexity can be controlled by the following two rules: The complexity of each module should be minimized, and the complexity among modules in a well-structured program should be distributed.

(2) Intermodule Information Flow.

Intermodule information flow metrics are concerned with the interfaces between procedures, modules, and/or subroutines. This can be viewed as the mapping of the inputs to the outputs from a module to module. Previous
work on flow metrics was done by Chapin (Chapin 78). His complexity measure is described by the difficulty in understanding software's function based on the input/output sets in modularized programs and systems. He defines four roles of data (any one item of data may have more than one role): (1) C_control role, (2) P_processing role, (3) T_pass_through role, (4) M_modify role. Data in the C_control role contribute the most to complexity when considering the source of C_data. The sources of data may come from within the module (or are constants), within the subordinate loop body, or from outside of the loop body. He utilizes the techniques of identifying and parsing functional-oriented design complexity, and sets of dependent variables. For large-scale systems, Henry & Kafura (Henry & Kafura 81) proposed that in measuring complexity, an examination of the connection between the system components must be included. The relations generated for each component show the flow of information from input parameters to output parameters, and through the global data structures. The complexity measure is composed of the complexity of the procedure's code which is represented by its length and the complexity of the procedure's connection to its environment.
(3) Intramodule Control Flow.

Intramodule control flow attempts to measure complexity by measuring the control flow structure of the software. An example is McCabe's cyclometric Complexity (McC 76) which describes the decision-making structure of the program. This measure utilizes a graph theoretic technique to assign a complexity measure to a program based on the number of linearly-independent control paths in the program. McCabe sees that the complexity metric must include the measurement of the number of possible execution paths.

(4) Intramodule Information Flow.

Intramodule information flow can be viewed as the interconnection of data within program modules which are designed to be as independent and exclusive of each other as possible (Mac 80). The result is that the interconnection of data within modules should be absolutely clear. To understand how the inner working of the module is performed, an information flow graph is used to show the flow of information between the variables which participate in the processing of data for the creation of the desired output.
The intramodule control flow and the intermodule information flow can help analysts or programmers to understand the control of the system by describing input/output and the control variables used to direct the execution path in the program. However, it does not always measure the use of global variables, the value that may be changed during the information flow from input to output. More precisely, a procedure or function which alters the value of non-local variables during the execution is said to have side-effects. For example, the read statement reads a value from an input file and that value is assigned to a global variable instead of using its own variable declared within the procedure or function. This may do no harm to the system execution, but it makes the system harder to understand and to work with.

To enhance the effectiveness of the computer software, it is essential to develop a better understanding of ways to deal with information flow. It turns out that combining the study of these four types, metric and information flow graph may be used to analyze the complexity of the information flow among the program variables. This may lead to an improved understanding of the design. By tracing the connection among the program variables from the initial node
to every other node that exists, program variables can be identified whether they are dependent or independent. A variable is called dependent if its value is affected by the value of other variables. The dependent variable can be shown by flow graph describing the information flow from the source node to the destination node. An independent variable can be defined as a program variable whose value does not change as the result of processing of the program module. Thus the IFC program is presented to accept the input Pascal program. Every program statement contained within the input program module needs to be examined for the information transfer.
CHAPTER TWO
INFORMATION FLOW

Information flow can be defined as the transfer of information from one object to another. The objects are program variables. Each variable may or may not be involved in an information transfer during the program execution. Operations of the program concerned in this report involve assignment, I/O, call statement, with statement and control statement. During program execution, the information transfer usually has a transitive relation. The flow relation (FR) is transitive if, whenever (A, B) is in FR and (B, C) is in FR, (A, C) is in FR also. This can be described by showing that among the program variables A, B, C there is an information transfer from A to B and from B to C, thus, there is an information transfer from A to C (Denn 76).

The features of information flow can be viewed as: (1) explicit information flow; (2) implicit information flow. The following section gives the definitions and explains how the information flow can be represented by a flow graph. Each node represents a variable declared in the program. An arc represents the information flow that transfers from the source node to the destination node.
EXPLICIT INFORMATION FLOW.

Explicit information flow can be viewed as a straightforward and obvious transfer of information between objects in simple statements (Mac 81). The transfer may occur directly or indirectly. Direct transfer can be described as a transfer of information from source object to destination object which results in a final output. Indirect flow is very much like direct transfer except it involves one or more statements participating in processing a final output.

(a) \[ C := D \]
(b) \[ A := B + C \]

Figure 2.1 Explicit information flow graph.

Figure 2.1 consists of two assignment statements. Statement (b) uses statement (a) to produce a final output A. The flow graph shows a direct information transfer from object D to C in an assignment (a), and also has a transfer from object B to A and from C to A in an assignment (b). However, the result of flow graph shows an indirect information flow, namely, transitive relation, from object D to A in processing these two assignments.
Simple statements used in the explicit information flow in this project are composed of three types: (1) assignment statements, (2) input/output statements, and (3) procedure/function call statements.

(1) ASSIGNMENT statement.

The assignment statement symbol is denoted by ':='.
The object which appears on the left hand side (LHS) on an assignment symbol is defined as the destination. The object on the right hand side (RHS) is defined as the source object. It may contain either an expression or a function call statement. LHS := RHS shows the direct explicit flow which transfers the information from the RHS to LHS.

(2) INPUT/OUTPUT statement.

The input/output statement is used to transfer values to or from the program. Only the I/O statement that performs a value read or write is involved in this analysis. For example, in figure 2.2 the statement READ(A) shows the explicit information flow which is the transfer of an assigned value from the input file to the object A. The statement: WRITE(B, 'sample') shows the direct explicit flow which is the transfer value from the object B to the output file.
(3) **PROCEDURE_FUNCTION statement.**

When a call statement is executed, a mapping is performed depending on the procedure/function declaration. Consider a procedure call with parameter passing. It is obvious that there is at least a direct transfer from the actual parameter to the formal parameter, and a possible direct transfer from the formal parameter back to the actual parameter depending on how the procedure heading is declared. The function call can be used only in an assignment statement while a procedure call can be used by a call statement. These can not be used or contained in any boolean expression of the control statements and in the I/O statement. At the point of call, the exact number of actual parameters and formal parameters are required. Actual parameters must use the variables declared in the prefix declaration, thus they cannot be constants, subprograms or be contained in the expression.

In figure 2.3a, the procedure call statement: `ADD(A,B,C)` causes the explicit flow from actual
parameters $A, B, C$ to formal parameters $x, y, c$ of the routine heading: `PROCEDURE ADD(X,Y,Z;real)`. In figure 2.3b, the routine heading declared as: `PROCEDURE(X,Y;real; VAR Z;real)` causes the explicit flow from actual parameters $A, B, C$ to formal parameters $X, Y, Z$, and also the transfer from formal parameter $Z$ back to actual parameter $C$ in the call statement `ADD(A,B,C)`.

```
PROCEDURE add(X,Y,Z;real);
BEGIN END;
...
...
add(A,B,C);
```

![Figure 2.3a Explicit information flow graph of a call statement.](image)

```
PROCEDURE add(X,Y;real; var Z;real);
BEGIN END;
...
...
add(A,B,C);
```

![Figure 2.3b Explicit information flow graph of a call statement with a returned value.](image)
Implicit information flow can be caused by conditional statements. Conditional statements are composed of two parts: the conditional part and the computational body. The conditional part consists of a boolean expression that needs to be evaluated to specify whether to carry on or to terminate the computational phase. During the evaluating of the conditional phase, if there is any variable contained in the boolean expressions, that variable will have an implicit information flow to the appropriate variables contained in the computational body. The computational part consists of executable statements which can be a simple statement, conditional statement, or compound statement composed of a group of statements introduced by BEGIN and terminate by END.

```plaintext
WHILE A>B DO BEGIN READ(X);
    Y := 1;
    Z := TRUE;
END
```

![Diagram]

Figure 2.4 Implicit information flow
Figure 2.4 shows the variables used in a WHILE statement. There is an implicit information flow from A and B to its computational body: X, Y and Z, and the explicit information flow from IN to X in an I/O statement.

Statements involved in the implicit information flow measure consist of WHILE DO statements, IF THEN ELSE statements, FOR statements, and REPEAT UNTIL statements. The basic concept and rules of each statement are briefly described in the following section.

(1) WHILE statement.

In a loop statement, WHILE <bool> DO <S1>, the conditional part contains a boolean expression that needs to be evaluated to specify whether to continue or terminate the computational body. If there is any variable participating in the boolean expression, <bool>, that variable will have the implicit information flow to the computational body, <S1>.

Figure 2.5 shows an implicit information flow from A and B to the appropriate variables contained in the computational body, a procedure call statement. In fact, the evaluation of A>B causes the changing of actual parameter C after the procedure call ADD is invoked. Thus it is important to recognize that there
is an implicit information flow from object A, and B to object C. There also is an explicit information flow which was previously described.

PROCEDURE ADD (X, Y: real; VAR Z: real);
BEGIN
END;

WHILE A > B DO ADD (A, B, C);

![Diagram of flow graph]

Figure 2.5 WHILE flow graph.

(2) IF THEN ELSE statement.

A conditional statement, IF <bool> THEN <s1> else <s2>, enables the process to select one of the two actions. The selection is made by evaluating the boolean expression, <bool>, in the conditional phase. If there is any variable participating in the evaluation of the boolean expression, there will be an implicit information flow from that control variable to both the THEN and ELSE computational bodies, <s1> and <s2>.
In figure 2.6, the conditional phase, \( d > 0 \), causes the implicit information transfer from object \( D \) to the object \( C \) in an I/O statement, and object \( E \) which appears on the LHS of the assignment statement. The assignment statement, \( E := F \), causes the explicit information flow transfer from object \( F \) to object \( E \). There also has an implicit flow from \( \text{IN} \) to object \( C \) in an I/O statement, \( \text{READ}(C) \).

\[
\text{IF } d > 0 \text{ THEN } E := F
\]

\[
\text{ELSE READ}(C)
\]

![Diagram of flow graph](image)

Figure 2.6 IF THEN ELSE flow graph

(3) \text{FOR} \text{ statement.}

The information flow in the statement: \text{FOR} \ <\text{id}> := <\text{bool1}> \ (\text{TO|DOWNTO}) \ <\text{bool2}> \text{ DO } <\text{s1}> \text{, will work the same way as the WHILE statement except the object "id" will not be involved in an implicit information transfer. When the variable "id" appears in the body, S1, it will be an explicit transfer.}

FOR I := A TO B DO BEGIN
  C := D+1;
  writeln(E);
END;

Figure 2.7 FOR statement flow graph.

Figure 2.7 shows the implicit information flow from A and B to object C in an assignment, and to OUT in the I/O statement. There also is an explicit information flow from D to C in the assignment, and from E to OUT in the I/O statement.

(4) REPEAT statement.

The body of the REPEAT statement looks very much like a compound statement, except that BEGIN and END are missing. UNTIL serves to introduce the boolean expression that needs to be tested to terminate the statement. Any variable used in the evaluation of the boolean expression will cause an implicit information flow to its body.
REPEAT
  READ(A);
  C := D+B;
UNTIL E>10;

Figure 2.8 REPEAT_until flow graph.

Figure 2.8 shows the implicit information flow from object E to C in the assignment statement, and to A in the I/O statement. The assignment statement, C := D+B, causes the explicit flow from the objects D and B to C. The I/O statement, READ(a), causes an explicit information flow from IN to A.
CHAPTER THREE

DESIGN

The information flow complexity (IFC) program is designed to run under the P-system operating system. The P-system supports UCSD Pascal on many microcomputers. This implementation of IFC is done on the PDQ-3 microcomputer system. This chapter explains in detail the requirement specifications, input source program and output, major data structures used in the implementation, system overview and structure diagram that leads to the pseudo code described in the next chapter.

REQUIREMENT SPECIFICATION.

IFC is concerned only with how the information is transferred, thus it is assumed that the input program is 100% free from syntax errors. The following section assumes that the user already knows the concepts and syntax of the Standard Pascal language definition. Only Standard Pascal language is allowed in the input programs.
IFC Input Program.

There are keywords that the input program may not contain unless they are part of the program statement or comments:

ARRAY BEGIN CONST DO
DOWNTO ELSE END FOR
FUNCTION IF OF PROCEDURE
PROGRAM READ READLN RECORD
REPEAT THEN TYPE UNTIL
VAR WHILE WITH WRITE
WRITELN

Besides the keywords described above, there are additional requirements for input program:

(1). comments must start on a new line.
(2). each input line buffer must be terminated by using semi-colon unless it is the end of the program indicated by '.'
(3). The input line buffer can only contain 80 characters. A line of input is read and accumulated to the input buffer until a semi-colon is reached. The following is an example showing a
program statement that is not greater than 80
characters.

IF a > 10 THEN
  WHILE b<> 0 DO
    BEGIN d := 2* e;

The input line buffer would contain:

IF a > 10 THEN WHILE b<>0 DO BEGIN d := 2* e;

Standard Pascal Statements.

Ten operations are acceptable in the IFC input program.
They are in the following:

(1). assignment statement

(2). BEGIN_END

(3). FOR_DO

(4). function call

(5). IF THEN ELSE

(6). I/O statement: WRITE, WRITELN, READ, READLN

(7). procedure call

(8). REPEAT_UNTIL

(9). WHILE_DO

(10). WITH statement
There are three normal outputs: the listing of the input source program, and the arrays KEYTAB and MEXIX which are illustrated in the following. The rest of the tables built during the analysis may be printed on the hard copy, if requested (refer to global data structures in this chapter).

KEYTAB

1  IN
2  OUT
3  main...A
4  main...B
5  main...E
6  main...D

: ........
: ........
METRIX

TO

1  2  3  4  5  6 ... 

FROM

The statement: IF A>10 THEN WHILE B<>0 DO BEGIN D := 2*E; causes an explicit flow from "E" to "D" and the implicit flow from "A" and "B" to "C". This is shown by the array KEYTAB and METRIX above.
Running IFC

To run IFC, the input source program must be created and kept on the floppy disk. Ten IFC input sample programs are provided in this report. The sample program listings and their results can be found in the appendix D. Appendix C briefly describes how to run and operate the existing input programs. To analyze a user's input program, it is important for the user to read and understand the user's guide to the P-system.
DATA STRUCTURE

IFC is used to report the flow of information between objects. These objects, program variables, must be collected in tables. The IFC program uses six tables. The first three tables: TYPTABLE, INFOTABLE, PROCTAB are built during the scanning of the declarations while the last three tables: INDXTAB, KEYTAB and METRIX are built during the scanning of the body routines. The INDXTAB is used to store the temporary variables during the scanning of program statement execution. The KEYTAB and METRIX are the outputs that are printed after the end of an input file is encountered. The following sections describe the six tables.

1) TYPTABLE

During the parsing of a type declaration, names of both record types and arrays of record are stored in the TYPTABLE. This table is used for record type checking of program variables declared in the declaration section. The matching of record name and the suspect type of variable in the TYPTABLE will return the location of first record field kept in the INFOTABLE.

TYPTABLE is an array of records. Each record contains two fields of information: TYPENAME and FIRSTFIELD. TYPENAME represents the name of the record and FIRSTFIELD contains a value pointing to the location of the appropriate record field of INFOTABLE.
(2) INFOTABLE.

INFOTABLE is the most important table in the data structure. It keeps all program variable names and function/procedure names that may or may not have an entry in METRIX.

The INFOTABLE is an array of records. Each record is composed of five fields: INFONAME, NEXTINFO, POINTOINDEX, SCOPE and PARAMTYP. INFONAME represents one of these four kinds: (1) a record field, (2) a program variable, (3) a parameter passed in the procedure heading or (4) function name. NEXTINFO contains a value pointing to the next field of the record structure. POINTOINDEX contains a value pointing to the first field of the record structure if it has a value greater than zero. Zero indicates a simple type. SCOPE represents two kinds of variables: global variables and local variables. A SCOPE equal to zero represents a global variable from the main routine. A number greater than zero indicates the location of its procedure name in the procedure table. PARAMTYP indicates two types of parameter passing: PARAMTYP is one when the parameter passed is a returned parameter while PARAMTYP equal to zero indicates a formal parameter passed without a value returned.
(3) PROCTAB.

This table is created when scanning the routine heading and is used for checking the existing procedure name whenever a procedure name is invoked. The procedure call statement performs the information transfer between modules by the use of PROCTAB. The table has a value indicating the location of the first formal parameter kept in the INFOTABLE. PROCTAB is also used for the termination of the procedure block. The table keeps track of the number of actual parameters passed as kept in the INFOTABLE. Thus INFOTABLE can update its index which points to where the last formal parameter passed is located. Local variable names will be deleted from INFOTABLE because they will not be used again in the program.

PROCTAB is an array of records designed for storing procedure and function names and their details. Each record is composed of four fields: PROCNAME, PARAM_CT, TOINFO and PROCTYPE. PROCNAME contains the procedure or function name of the block in which the name is defined. PARAM_CT represents the number of parameters passed. TOINFO contains a number that points to the first parameter in the INFOTABLE. PROCTYPE represents the kind of routine, a "1" is coded for a function routine and a "0" is coded for a procedure routine.
(4) INDXTAB.

This table performs as a stack machine. A stack counter is incremented when a new program statement is encountered and decremented after finishing scanning the program statement. During the scanning, the current stack is used to keep all the possible program variables that may have an information transfer. The variables that are kept in the INDXTAB must be successfully checked through the INFOTABLE and stored as unique key numbers associated with the KEYTAB. How the INDXTAB is used will be discussed in the system overview in this chapter.

INDXTAB is an array of records. Each record contains four fields: MODE, BLOCK, NEXT and INDXARRAY. MODE has a value range of "0" through "6". "0" is coded for simple statements. "1" is coded for an IF THEN ELSE statement and "2" is coded for an IF THEN statement. "3" is coded for FOR and WHILE DO statements. "4" for REPEAT statement "5" for WITH statement and "6" for entering of the routine body. BLOCK = 1 if the statement following the conditional statement is a compound statement, BLOCK = 0 for simple statement. INDXARRAY is an array type with a maximum size of ten. It is used to store program variables in the unique key number that is successfully searched through the INFOTABLE. NEXT keeps track of how many
unique keys are kept in the INDXARRAY.

(5) KEYTAB.

KEYTAB is an array of character strings. KEYTAB(1) represents information read-in, while KEYTAB(2) is the information written-out. Other elements contain program variables according to INFONAME in INFOTABLE. KEYTAB is used to register program variables before making any information transfer to METRIX.

(6) METRIX.

METRIX is a two dimensional array. Each row and column represent the transfer of information from source(row) to destination(col). The source and destination indices are obtained through KEYTAB (see fig 3.14).
Data structure used during the analysis of the PREFIX declaration.

Data structure used during the analysis of a routine body

Figure 3.1
Relating data structures
PROGRAM sample;

TYPE
    add = RECORD
        name : string;
        phone : integer;
        address : string;
        acctnum : integer;
    end;

VAR
    customer : array(1..50) of add;
    dept : string;

PROCEDURE sale (cacct, cost : integer; VAR bal : integer);
BEGIN
END;

Figure 3.2
A sample of relating data structures.
CHAPTER FOUR

IMPLEMENTATION

IFC as implementated on the PDQ-3 microcomputer consists of approximately 1500 lines of code including imbedded comments. The implementation is stored in the disk library with named files of 'IFC.text' and 'IFC.code'. IFC.text can be found in appendix B. The IFC.code which is the result of the successfully compiled translation of IFC.text is kept in P-code and ready for the user to execute.

The following sections help the user to understand the logic of the program. The listing of routines and global variables used during the implementation appears in appendix A. Included are the hierarchical system diagrams, and an IFC sample program.
Figure 4.1 SYSTEM design

Figure 4.2 Hierarchical diagram of Body routine.
Figure 4.3 Hierarchical diagram of
PROCESSVAR_TYPE routine.

Figure 4.4 Hierarchical diagram of
PROCESSPROC routine.
Figure 4.5 Hierarchical subdiagram of
PROCESSVAR_TYPE and PROCESSPROC routines

Figure 4.6 Hierarchical diagram
of PROCESSBEGIN.
Figure 4.7 Hierarchical diagram of ACCESSBLOCK routine.
Figure 4.8 Hierarchical diagram for simple statements

Figure 4.9 Hierarchical diagram of ANALYSIS routine.
SYSTEM LOGIC.

IFC logically divides the input program into a standard prefix declaration part and a body routine part. The standard prefix declaration consists of constant, type, variable declarations and routine headings. Since constants remain the same during the program execution, the use of constants may be ignored. Variables and types have to be identified and entered into the appropriate tables. A program variable of type array, will be counted as one object during the scanning of the program statements. For example, statements $A[i] := B[j]$ and $A := 5$ cause the information transfer from object $B$ to object $A$.

1) Standard Prefix Declaration.

The IFC scans an input program line by line from top to bottom. The IFC will accumulate an input buffer by making a first call to procedure READALINE and the subsequent call to procedure GETFIRSTWORD, in order to get the first word which is a character string separated by a blank. The word must be in one of the keywords: TYPE, VAR, PROCEDURE, FUNCTION, and BEGIN in order to enter one of these three subroutines: PROCESSVAR_TYPE, PROCESSPROC, or PROCESSBEGIN. These three routines, and READALINE and GETFIRSTWORD are performed in a while_loop
which is terminated when the end of the input file is encountered. To enter the PROCESSVAR_TYPE, the current word must be either VAR or TYPE while the current word of either PROCEDURE or FUNCTION causes an entry into procedure PROCESSPROC. The routine PROCESSVAR_TYPE will collect record names, program variables and keep them in the INFOTABLE and TYPETAB during scanning of the type and variable declaration sections. The next input buffer is read after scanning the current buffer and the first word is detected in order to check whether to continue or to terminate the routine. The routine is terminated only if the current word is in one of the keywords. Procedure PROCESSPROC is used to scan routine headings. Procedure and function names are kept in the PROCTAB and their parameters are kept in the INFOTAB.

(2) Body Routine Analysis.

After the Standard prefix declaration has been analyzed, the body of the routine, which is a sequence of statement introduced by BEGIN and terminated by END, is examined by procedure PROCESSBEGIN. The procedure uses INDXTAB, a stack, to analyze the effect of the program statements which can be classified as: simple statements, control statements and nesting. An entry is added to the stack with a specific MODE and BLOCK
whenever a new statement is encountered. The entry for a simple non-nested statement is deleted immediately after that program statement is analyzed. Then the next input program statement is read and analyzed. This continues until the end of the routine body. Entering a routine body causes a new INDXTAB entry to be added and this entry is deleted when the end of the routine body is encountered. BLOCK is the key point to determine whether to remain or to delete the current stack entry. The stack entries show the number of the control statement and nesting. Conceptually the operation of a simple statement that may involve the explicit flow will be done on the current stack entry. The subsequent implicit flow will be done by traversing down the remaining stack entries. These stack entries will remain on the stack until the computational bodies are completely analyzed.
PROGRAM sample;
VAR a,b,c,d,e:integer;
BEGIN READ (a,b);
  REPEAT
    READ (c);
    IF a>10
      THEN WHILE b<>0
      DO BEGIN d:= 2*e;
        END;
    UNTIL c=0;
  END.

Considering only the body of the routine, the input line buffers would be the following:
Buff....BEGIN READ (a,b);
  ...REPEAT READ (c);
  ...IF a>10 THEN WHILE b<>0 DO BEGIN d:= 2*e;
  ...END;
  ...UNTIL c=0;
  ...END.

Figure 4.10 Example for explaining INDXTAB.
The following relates how INDXTAB stack is used when the routine of Figure 4.10 is input. The entries are shown in Figure 4.11a and 4.11b. The first line buffer; BEGIN READ (a,b); is read. 'BEGIN' of the main body causes the first entry to be added to the stack with MODE=6. The second entry is added with MODE=0 for the I/O statement READ (a,b); and is deleted after the statement is analyzed. The next input buffer; REPEAT READ (c); is read. A stack entry is added with MODE=5 for REPEAT statement. The second entry is added to the stack for the I/O statement and it is deleted after the statement has been analyzed. The statement READ (c); will first make an explicit flow from IN to "c" then traverse down the remaining stack for an implicit flow. In this case, the previous stack is the REPEAT statement of MODE=5; the variable "c" will be kept on this entry until the UNTIL statement is analyzed. Figure 4.11a shows the remaining stack entries after the statement READ (c); is analyzed.
The next line buffer; IF a>10 THEN WHILE b <> 0 DO
BEGIN d:=2*e; is read. Their entries are put on the stack. First, an entry is put on for an IF statement with MODE=2. Second for the WHILE statement with MODE=3 and BLOCK=1 and third for the assignment statement. The current entry is deleted after the assignment is analyzed and traversed down the remaining stack entries. Figure 4.11b shows the remaining stack entries after the
assignment is completely analyzed. The next line buffer
END; causes two entries to be deleted from the stack:
first, for the WHILE statement because of the matching
pair BEGIN_END and second, the entry for the IF
statement with nesting. If BLOCK contains a value equal
to one, this entry will remain on the stack until the
matching pair is encountered. If there is an ELSE
statement, this entry will remain on the stack and the
MODE is changed from 2 to 1 before the ELSE statement
will be analyzed. The next line buffer, UNTIL c=0, is
read. An entry is added on the stack. There is an
implicit flow from the variable names stored in the
current stack entry to the variable names stored in the
previous entry (REPEAT). Then these two entries are
deleted from the stack and the next line buffer is read.

The line buffer, END., which is the end of the
input program causes the last entry to be deleted from
the stack. If it is the END of the subroutine,
INFOTABLE needs to clean up all the local variables, if
any, because they will not be used again in the program.
This is done by procedure MARK_UNUSED.
IPC Sample Program.

This section shows a sample program and output in which users are assumed to be testing one of the provided input programs. Ten input programs have been created and kept in the list directory under prefix name of "IP#", where "#" represents the number 1..10. More than that to update, modify, create and run the input programs, the user needs to read and understand the short manual of instructions and commands. (refer to USER's GUIDE in appendix C.).
PROGRAM SAMPLE;
(* IF1 *)
VAR A, B, C, D, E : INTEGER;
BEGIN
  READ <A, B>;
  REPEAT
    READ <C>;
    IF A > 10 THEN WHILE B <> 0
      DO BEGIN D := 2 * E;
        END;
    UNTIL (C = 0);
  END;

INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFONAME</th>
<th>NEXTINFO</th>
<th>POINTOINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

KEY TABLE

1. IN
2. OUT
3. MAIN...A
4. MAIN...B
5. MAIN...C
6. MAIN...E
7. MAIN...D

MATRIX

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CHAPTER FIVE

CONCLUSION and EXTENSIONS

In summary, the task of the IFC program is to report overall information transfer between objects in the input program. The resulting output, the matrix table, can be used in various ways in the area of software development and software testing techniques. The information flow graph can be derived from the matrix table. The flow graph may be used to analyze the complexity in tracing the connections among the program variables from the initial node to every other node that exists. Path testing of the information flow graph may be used to identify whether the program variable is dependent or independent. Combining both the graph flow and the matrix table can help designers and programmers to understand the logic of the system design that may lead to reduce the number of test cases in path testing.

Path testing may be defined as the process of finding a set of program variables that force the path predicates to the truth value that corresponds to the desired path. Since the information transfer is based on the program variables associated with the function performed, it can be used as an adaptive development tool
for the management of the path-testing. Subsequently, the
matrix table can be summarized as the selected set of the
dependent program variables needed to develop, execute and
evaluate the test case. Thus the independent program
variables whose values do not change as the result of
processing need not be included in the test case.

FUTURE EXTENSIONS

The IFC program covers only the simple data
structures and program statements which were described in
previous chapters. In attempting to make the IFC program
more powerful, there are some future extensions that may be
included in this IFC program.

1. Input specification.

Increase of the input line buffer to greater
than 80 characters per line of program statement.
This would allow the user to construct more
complicated statement or nested statements
especially in the repetition statements and
conditional statements.

The following could be added in the analysis of label declaration (for GOTO statements), enumeration types, dynamic allocation with the use of pointer variables associated with record structures. In the routine heading, the analysis of FORWARD routine may be included in the declaration.

3. More executable program statements.

IFC should involve the use of the case statement and the UCSD Pascal statements such as CONCAT, COPY, INSERT as well as the use of multiple exit routine and goto statements. Included are implicit subroutine and function calls as contained in I/O statements and boolean expressions of the control statements.

4. OUTPUT Specifications.

The IFC output could provide statistics based on the information transfer of either intramodule or intermodule. The examples are the total number of routines used in the input program, total number of the program variables and number of the occurrences of the transfer of information.

If the items discussed above can be included in the IFC program, IFC should apply to most UCSD Pascal programs.
REFERENCES.


APPENDIX A

VARIABLE NAMES AND ROUTINES LISTING

Global variables listing

<table>
<thead>
<tr>
<th>VARIABLES NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>scope of the current program status</td>
</tr>
<tr>
<td>BUFF</td>
<td>contains first word of repetition statements to determine kind of current operation.</td>
</tr>
<tr>
<td>CALL</td>
<td>a boolean flag is set to 'true' while accessing the procedure call</td>
</tr>
<tr>
<td>FLAG</td>
<td>a boolean flag used to indicate end of input source program</td>
</tr>
<tr>
<td>INDX</td>
<td>index to INDXTAB</td>
</tr>
<tr>
<td>INDXTAB</td>
<td>table used for holding program variables that successfully checked through INFOTABLE and may have the information</td>
</tr>
</tbody>
</table>
transfer between objects.

**INFOINDEX** index to INFOTABLE

**INFOTABLE** table used for keeping program variables that may have the information transfer during scanning of program statements.

**IPFILE** temporary file for an input program

**KEYINDX** index to KEYTAB and METRIX

**KEYTAB** output table used for storing program variables names that have an access to information metrix

**LETTER** a..z lowercase only

**LN** the current input line buffer

**METRIX** table used for information transfer from row to column. The number of row and column are referred by KEYINDX

**NAME** name of input source program

**NAMERECC** hold record name which currently in accessed.

**NAMESPACE** contains temporary values that have the same type during the scanning of the variable declaration section and routine
heading.

NUMBERS 0..9

OP, OPERATOR represent the current operation of program statement only use for repetition statements and WITH statement.

OUTFILE temporary file for output

PARAM_ENTRY contains location of INPOTABEL where the first parameter is kept

PROCINDX index to PROCTAB

PROCTAB table used for keeping the procedure names and their information

STATUS scope of the current program status

TYPEINDEX index to TYPETABLE

TYPETABLE table used for keeping type names of record type and links to first record field

WITH_STMT a boolean flag used to indicate whether WITH statement is being used or not

WORD first word of the line buffer
**MAIN Routines listing**

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BODY</td>
<td>to generate all possible information transfer of the input program.</td>
</tr>
<tr>
<td>GETFIRSTWORD</td>
<td>to isolate a first word from current input line buffer.</td>
</tr>
<tr>
<td>INITIALIZE</td>
<td>to define the initial value of static program variables</td>
</tr>
<tr>
<td>PRINTOUT</td>
<td>output tables printouts upon user requested.</td>
</tr>
<tr>
<td>READALINE</td>
<td>to read an input line of code one at a time and accumulate into an input buffer. The routine return to caller when end of statement, ';' or end of input file is encountered.</td>
</tr>
</tbody>
</table>
Routines used to analyze the Prefix Declarations.

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESSLINE</td>
<td>to isolate program variables or actual parameters contained in the current line buffer and save them in the INFOTABLE.</td>
</tr>
<tr>
<td>ACCESSRECORD</td>
<td>to collect record name and record fields of record data structure that may be used both in type declaration and variable declaration. The record name declared in the type declaration is kept in the TYPETAB, otherwise it is kept in the INFOTABLE. The routine scans line per line until the current line is 'end' of record structure.</td>
</tr>
<tr>
<td>ACTUALPARAM</td>
<td>to store each actual parameter into the INFOTABLE and keep the total number of actual parameters declared in the PROCTAB.</td>
</tr>
<tr>
<td>CHECKTYPE</td>
<td>to search a given type through the TYPETAB. If the search is failed, value return is zero. IF not, it is the type of record structure, the</td>
</tr>
</tbody>
</table>
routine returns the value contained
the location of the first record
field kept in the INFOTABLE.

CHOPALINE
to separate program variables
contained in the current line and
temporary kept in the NAMESPACE,
data of an array type.

GETTYPE
to isolate a type string that is
associated with program variables or
record structure declared in the
prefix declaration. If it is an array
type, it will get the string of type
of that array. For example, routine
will return the value <string> of
the following declaration '...array
(1..10) of <STRING>:'

INFOENTRY
to increment a counter of INFOTABLE
and deposit variable names,
parameters and record fields into
the INFOTABLE.

PARAMTEST
to isolate the actual parameter
string declared in the procedure
heading and keep then in the
PROCTAB.

PROCESSPROC
to parse a procedure heading and
correct the information needed to
keep in the PROCTAB.

PROCESSVAR_TYPE to parse variable and type declaration and collect necessary information. In to TYPETABLE and INFOTABLE.

SAMESTATUS to determine whether to terminate or to continue the current operation. The status is changed when the current word contained in the keyword 'type', 'var', 'procedure', 'function' and 'begin'.

TYPENTRY to add a record name and its details to the TYPETAB.
Routines used to analyze the Routine Body.

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS BLOCK</td>
<td>use to generate program statements and is done in a series of operations in side of a big case statement. Each operation perform its function by pushing the program variables that may be contained in the statement onto the stack in order to get ready for information transfer. A new input buffer is read when finished scanning each simple statement and UNTIL statement. For the repetition statement, the current line buffer is updated, the statement body becomes the current line, excluding 'BEGIN' in a compound statement. The routine then returns to the caller.</td>
</tr>
<tr>
<td>ACCESSCALL</td>
<td>to make the implicit information transfer from formal parameters to actual parameters, if any. The passing parameter string must contain program variable names that</td>
</tr>
</tbody>
</table>
have been predeclared in the prefix declaration. Explicit information transfers of the repetition statement are done by searching down the stack.

ACCESSFOR

add a program variable to INDXTAB with mode = 3 and block = 1 if the body statement is a compound statement. To save control variables on the current stack by calling procedure analysis.

ACCESSIF_STMT

similarly to ACCESSFOR routine except enter MODE = 2.

ACCESSUNTIL

keeps a control variable contained in an expression on the stack by calling the ANALYSIS routine. Makes the implicit information transfer from the current top(until) to the previous stack, with MODE=5 for a REPEAT statement, its first word separated and returns to caller.

ACCESSWHILE

similarly to ACCESSFOR routine except enter MODE=3.

ACCESSWITH

add a stack with mode = 4 and block = 1. If the computational statement body is a compound statement, sets a
flag WITH_STMT and calls the ANALYSIS routine with string of record name.

ANALYSIS to isolate a program variable from a given string, searches for the isolated string in INFOTAB. If a match is found, assigned a key number and kept in KEYTAB. The key number will be kept on the current stack in INDXARRAY and will make the information transfer for that program statement, if any.

ASSIGNMENT to make an information transfer from the variables contained on the RHS of the assignment to the variable on the LHS. The RHS and LHS string are examined by the ANALYSIS routine.

CHECKCALL to search for a given string through PROCTAB. If the search is successful, the returned value points to the first parameter.

CKELSE to separate a current line from an IF THEN ELSE statement to be the line which follows the ELSE statement. The previous statement will be returned to caller.
CK_OPERATOR to determine the type of the program statement currently being scanned.

CKRECFIELD to search a given string which may be a record field or program variable while a WITH statement is in effect. According to the global data NAMEREQ, it will link to all validated fields. If the search is successful, the routine will call DEFINEKEY routine to get the unique number.

DEFINEKEY to search for the program variables in the KEYTAB and return is a unique key to the caller.

ELSE_STMT to change the status in the INDXTAB if it is the IF THEN ELSE statement from MODE equals to 2 to 1.

EXPLICITFLOW to examine an explicit information flow after each statement being scanned.

FINDTOKEN to search for a given token proceeds from bottom to top of the INFOTABLE. The routine returns the location where the match is found.

FUNCT_CALL to determine a function call
INDIRECT to increment the stack counter of an 
INDXTAB.

IO-STMT to compute an information transfer, 
and check for explicit information 
flow

KEY_ENTRY to add a new key number and keep the 
given string in the KEYTAB if it has 
not been stored in the table.
Return the key number to caller.

MARK_UNUSED dispose all local variables from 
INFOTABLE

PARAMPASSING to generate the information transfer 
from formal parameters to actual 
parameters.

POP_END decrement index counter of INDXTAB

PROCESSBEGIN to mark an entry into the body 
routine by adding a stack entry in 
INDXTAB with MODE=6, BLOCK= 1 and 
NEXT contains the location where the 
current body routine is being 
accessed. Program statements 
contained in the body routine are 
handled by procedure ACCESSBLOCK. 
After the body routine is 
terminated, INFOTABLE needs to clean 
up all local variables.
RECORDFLOW to scan the WITH statement. The variables contained in a given string may be or may not be a record field. If it is found in the record field, it must be concatenate to the record name before assign the key number in the KEYTAB.

RECENTRY to find the last record name in the given string and return its first field where its kept in INFOTABLE. For example: a given string < rec1.rec2.rec3.field31 >, the routine will return a value that indicates the location in INFOTABLE of the first field of <rec3>.

SKIP to delete array subscription and string contained in the quotation mark,"" from the current input line.

TABLE_LOOKUP to determine whether a given token is a program variable or not. If it is, KEYTAB is checked before an entry is made for a new variable name. The number associated with KEYTAB is entered as a row and column to metrix
TRAVERSE_STACK to sort and examine the variable that kept in the INDXTAB.

METRIX_ENTRY to make an information transfer to the METRIX table.
APPENDIX B

IFC PROGRAM LISTING

PROGRAM IFC;

CONST COMMA = ', ';
  EQUAL = '=';
  ENDSMT = ': ';
  COLON = ': ';
  BLANK = ' ';  
  MAXTABSZIE = 20;

TYPE

TYPEEC        = RECORD
  TYPENAME:STRING;
  FIRSTFIELD:INTEGER;
  END;

INFOTYPE      = RECORD
  INFONAME : STRING;
  NEXTINFO : INTEGER;
  POINTINDEX : INTEGER;
  PARAMTYP : 0..1;
  SCOPE : INTEGER;
  END;

PROCYYP       = RECORD
  PROCNAME:STRING;
  TOINFO : INTEGER;
  PARAMCT : INTEGER;
  PROC_TYPE : 0..1;
  END;

INDEXTP       = RECORD
  BLOCK : 0..1;
  MODE : INTEGER;
  NEXT : INTEGER;
  INDEXARRAY:ARRAY 1..10 OF INTEGER;
  END;

STATUSKIND = (SVAR,STYPE,SPROC,SFUNCT,MAIN);
OPKIND      = (WHILES,WRITES,WRITESLN,READS,READSLN,
                ENDS,ENDD,END_ELSE,REPEATS,UNTILS,FORS,
                WITHS,IFS,NONE);
VAR

IPFILE, OUTFILE: TEXT;
PROC_TAB : ARRAY[1..MAXTABSIZ] OF PROCTYP;
IDX_TAB : ARRAY[1..MAXTABSIZ] OF INDXTP;
TYPE_TABLE : ARRAY[1..MAXTABSIZ] OF TYPREC;
INFOTABLE : ARRAY[1..MAXTABSIZ] OF INFOTYPE;
NAMESPACE : ARRAY[1..10] OF STRING;
MATRIX : ARRAY[1..MAXTABSIZ, 1..MAXTABSIZ] OF INTEGER;
KEYTAB : ARRAY[1..20] OF STRING;
BUFF : ARRAY[WHILES..IFS] OF STRING;
WITH_STMT, CALL, FLAG : BOOLEAN;
RENAME, NAME, LN, WORD : STRING;
OP, OPERATOR : OP_KIND;
ACTION, STATUS : STATUS_KIND;
ALPHA, LETTERS, NUMBERS : SET OF CHAR;
KEY_INDEX, INXX, PROCIDX, TYPINDEX, INFO_INDEX : INTEGER;
FIELDENTRY, PARAM_ENTRY : INTEGER;

PROCEDURE READLINE; FORWARD;
PROCEDURE GETFIRSTWORD; FORWARD;
PROCEDURE ACCESS_CALL(ENTRY: INTEGER; ST: STRING) FORWARD;
PROCEDURE CHOPALINE(VAR CT:INTEGER; CLN:STRING):
(*** GLOBAL : NAMESPACE ***)
(*** CALLED BY : ACCESSLINE, ACCESSREC ***)

VAR P:INTEGER;
   DONE:BOOLEAN;

BEGIN
   DONE := FALSE;
   CT := 0;
   REPEAT
      CT := CT + 1;
      P := POS(COMMA, CLN);
      IF P = 0
         THEN BEGIN NAMESPACE[CT] := CLN;
                   DONE := TRUE;
              END
      ELSE BEGIN NAMESPACE[CT] := COPY(CLN, 1, P - 1);
                  DELETE(CLN, 1, P);
               END;
   UNTIL DONE;
END(*CHOPALINE*);

FUNCTION SAMESTATUS:BOOLEAN:
(*** GLOBAL : WORD ***)
(*** CALLED BY : PROCESSVAR_TYPE ***)

BEGIN IF (WORD = 'TYPE') OR (WORD = 'VAR')
    OR (WORD = 'PROCEDURE') OR (WORD = 'BEGIN')
    OR (WORD = 'FUNCTION')
    THEN SAMESTATUS := FALSE
    ELSE SAMESTATUS := TRUE;
END(*SAMESTATUS*);
PROCEDURE INFOENTRY (I, NEXTFIELD, LOCATIONINFO: INTEGER);
(* * * GLOBAL : INFOTABLE, INFOINDEX, NAMESPACE, ***)
(* * * STATUS, ACTION ***)
(* * * CALLED BY : ACCESSLINE, ACCESSRECORD ***)

BEGIN
  INFOINDEX := INFOINDEX+1;
  IF INFOINDEX > MAXTABSIZE
    THEN WRITELN('INFOTABLE TOO SMALL')
  ELSE WITH INFOTABLE INFOINDEX DO
    BEGIN
      INFONAME := NAMESPACE[I];
      NEXTINFO := NEXTFIELD;
      POINTINDEX := LOCATIONINFO;
      IF ACTION=MAIN THEN SCOPE := 0
      ELSE SCOPE := PROCIDX;
    END;
  END(*INFOENTRY*);  

PROCEDURE CHECKTYPE (VAR MATCH: BOOLEAN; VAR TOTINDEX: INTEGER;
TOKEN: STRING; VAR INDEX: INTEGER);
(* * * GLOBAL : TYPTABLE, TYPINDEX ***)
(* * * CALLED BY : ACCESSLINE, ACCESSRECORD ***)

VAR
  J: INTEGER;
  NAME: STRING;
  DONE: BOOLEAN;

BEGIN MATCH := FALSE;
  J := 0;
  DONE := FALSE;
  TOTINDEX := 0;
  INDEX := 0;
  WHILE NOT DONE DO
    IF (TYPINDEX=0) THEN DONE := TRUE
    ELSE BEGIN J := J + 1;
      IF J>TYPINDEX THEN DONE := TRUE
      ELSE BEGIN NAME := TYPTABLE[J].TYPNAME;
        IF TOKEN=NAME
          THEN BEGIN MATCH := TRUE;
          TOTINDEX := TYPTABLE[J].FIRSTFIELD;
          DONE := TRUE;
          INDEX := J;
        END;
      END;
    END;
END(*CHECKTYPE*);
PROCEDURE TYPENTRY (TOKEN: STRING; NEXTFIELD: INTEGER); 
(* * * GLOBAL : TYPTABLE, TYPINDEX * * *)
(* * * CALLED BY : ACCESSLINE, ACCESSRECORD * * *)
BEGIN
  TYPINDEX := TYPINDEX + 1;
  IF TYPINDEX > MAXTABSIZE
  THEN WRITELN('TYPE_TABLE TOO SMALL')
  ELSE WITH TYPTABLE[TYPINDEX] DO
    BEGIN
      TYPNAME := TOKEN;
      FIRSTFIELD := NEXTFIELD;
    END;
END (*TYPENTRY*);

PROCEDURE GETTYPE (VAR TYTOKEN: STRING; GSIGN, GLN: STRING);
(* * * CALLED BY : ACCESSLINE, ACCESSRECORD, PROCESSPROC * * *)
VAR P, Q: INTEGER;
BEGIN
  IF POS('ARRAY', GLN) > 0
  THEN P := POS(' OF ', GLN) + 4
  ELSE BEGIN
    P := POS(GSIGN, GLN) + 1;
    IF GLN[P] = BLANK
    THEN P := P + 1;
    END;
  Q := POS(';', GLN);
  TYTOKEN := COPY(GLN, P, Q - P);
END (*GETTYPE*);
PROCEDURE ACCESSLINE (ASIGN, ALN: STRING; VAR COUNT: INTEGER);

(*** GLOBAL : TYPTABLE, STATUS. ***)
(*** CALLED BY : PROCESSVAR_TYPE, PROCESSPROC ***)

VAR MATCH: BOOLEAN;
INDEX, I, J, LOCOFINFO: INTEGER;
TOKEN, STR: STRING;

BEGIN
GETYPE (TOKEN, ASIGN, ALN);
CHECKTYPE (MATCH, LOCOFINFO, TOKEN, INDEX);
I := POS (ASIGN, ALN) - 1;
WHILE ALN[I] = BLANK DO I := I - 1;
CASE STATUS OF
STYPE: BEGIN
IF MATCH
THEN BEGIN
TOKEN := COPY (ALN, 1, I);
LOCOFINFO := TYPTABLE [INDEX]. FIRSTFIELD;
TYPOENTRY (TOKEN, LOCOFINFO);
END;
END;
SVAR,
SPROC,
SFUNCT: BEGIN
STR := COPY (ALN, 1, I);
CHOPALINE (COUNT, STR);
IF MATCH
THEN LOCOFINFO := TYPTABLE [INDEX]. FIRSTFIELD;
FOR I := 1 TO COUNT DO
INFOENTRY (I, 0, LOCOFINFO);
END;
END(*CASE*);
END(*ACCESSLINE*);
PROCEDURE ACCESSRECORD (POISITN:INTEGER; PSIGN:STRING);

(* *** GLOBAL : INFOTABLE, INFOINDEX, TYPTABLE, *** *)
(* *** STATUS, LN *** *)
(* *** CALLED BY :  PROCESSVAR_TYPE *** *)

VAR NEXTFIELD, I, J, N : INTEGER;
INDEX, COUNT, LOCOPINFO : INTEGER;
TOKEN, STR : STRING;
FIND : BOOLEAN;

BEGIN

  I := POS (PSIGN, LN) - 1;
  IF LN[I]=BLANK THEN I:=I-1;
  CASE STATUS OF
    STYPE : BEGIN
      TOKEN := COPY (LN, 1, I);
      NEXTFIELD := INFOINDEX + 1;
      TYPENTRY (TOKEN, NEXTFIELD);
      END;
    SVAR : BEGIN
      STR := COPY (LN, 1, I);
      CHOPALINE (COUNT, STR);
      NEXTFIELD := INFOINDEX + COUNT + 1;
      FOR I := 1 TO COUNT DO
        INFOENTRY (I, 0, NEXTFIELD);
      END;
  END (* CASE *);
  N := POSITN+6;
  DELZTE (LN, 1, N);
  WHILE POS ("END;", LN) = 0 DO
    BEGIN
      GETTYPE (TOKEN, COLON, LN);
      CHECKTYPE (FIND, LOCOPINFO, TOKEN, INDEX);
      I := POS (COLON, LN) - 1;
      IF LN[I]=BLANK THEN I:=I-1;
      STR := COPY (LN, 1, I);
      CHOPALINE (COUNT, STR);
      IF FIND
        THEN LOCOPINFO := TYPTABLE[INDEX].FIRSTFIELD;
      FOR I := 1 TO COUNT DO
        BEGIN
          NEXTFIELD := INFOINDEX + 2;
          INFOENTRY (I, NEXTFIELD, LOCOPINFO);
        END;
      READALINE;
      END;
    END (* ACCESSRECORD *);
PROCEDURE PROCESSVAR_TYPE:
(*** GLOBAL : FLAG, WORD, LN, STATUS ***)
(*** CALLED BY : MAIN ***)

VAR DONE:BOOLEAN;
SIGN:STRING;
P,Q,COUNT:INTEGER;
BEGIN
DONE := FALSE;
IF STATUS=STYPE
THEN SIGN := EQUAL
ELSE SIGN := COLON;
DELETE(LN,1,POS(BLANK,LN));
WHILE NOT DONE DO
BEGIN
P := POS('RECORD',LN);
Q := POS('ARRAY',LN);
IF (P>0) AND (Q>0) OR (P>0)
THEN ACCESSRECORD(P,SIGN)
ELSE ACCESSLINE(SIGN,LN,COUNT);
READLINE;
GETFIRSTWORD;
IF NOT SAME STATUS OR FLAG T
THEN DONE := TRUE;
END;
END (* PROCESSVAR_TYPE *)

PROCEDURE PARAMTEST(VAR PARMSTR:STRING);

VAR STR:STRING;
BEGIN
WHILE POS(';',PARMSTR)>0
DO BEGIN
STR := (COPY(PARMSTR,1,POS(';',PARMSTR)));
ACTUALPARAM(STR);
DELETE(PARMSTR,1,LENGTH(STR));
PARAMTEST(PARMSTR);
END;
END(* PARAMTEST *);
PROCEDURE PROCESSPROC;

(* * * GLOBAL : LN, PROCTAB, PROCIDX, ***)
(* * * INFOTABLE, INFOINDEX ***)
(* * * CALLED BY : MAIN ***)

VAR CT, P : INTEGER;

PROCEDURE ACTUALPARAM (S:STRING);
VAR I, PARAM, COUNT : INTEGER;
STR : STRING;
BEGIN STR := S;
  IF POS ('VAR ', STR) > 0
    THEN BEGIN PARAM := 1;
        DELETE (STR, 1, POS ('VAR ', STR) + 3);
    END
  ELSE PARAM := 0;
  I := 1;
  WHILE STR[I] = BLANK DO I := I + 1;
  IF I > 1 THEN DELETE (STR, 1, I - 1);
  ACCESSLINE (COLON, STR, COUNT);
  IF COUNT > 1 THEN FOR I := 1 TO COUNT - 1 DO
    INFOTABLE[INFOINDEX - I].PARAMTY := PARAM;
  INFOTABLE[INFOINDEX].PARAMTY := PARAM;
  CT := CT + COUNT;
END; (*ACTUALPARAM*)

BEGIN (* PROCESSPROC*)
  DELETE (LN, 1, LENGTH (WORD) + 1);
  P := 1;
  WHILE LN[P] IN LETTERS + NUMBERS DO P := P + 1;
  NAMESPACE[I] := COPY (LN, 1, P - 1);
  CT := 0;
  PROCIDX := PROCIDX + 1;
  WITH PROCTAB[PROCIDX] DO
    BEGIN PROCNNAME := NAMESPACE[1];
      IF ACTION = SPUNCT
        THEN BEGIN INFOENTRY (1, 0, 0);
            PROC_TYPE := 0; END
      ELSE PROC_TYPE := 1;
      TOINFO := INFOINDEX + 1;
    END;
  IF POS ('(', LN) > 0
    THEN BEGIN
      LN := 1 COPY (LN, POS ('(', LN) + 1, POS (')', LN) - POS ('(', LN) - 1);
      LN := CONCAT (LN, ' ');
      PARAMTEST (LN);
    END;
  PROCTAB[PROCIDX].PARAMCT := CT;
  READLINE;
  GETFIRSTWORD;
END (*PROCESSPROC*);
**Routines Used to Analyze Routine Body**

**Procedure Matrix_Entry(R,C:Integer)**;

```pascal
(* ** GLOBAL : MATRIX ***)
(* ** CALLED BY : TRAVERSE_STACK, ASSIGNMENT, ***)
(* ** ACCESSUNTIL, ACCESS_CALL, IO_STMT ***)

BEGIN
    MATRIX[R,C] := MATRIX[R,C] + 1;
END;
```

**Procedure Indirect(INBLOCK, INMODE:Integer)**;

```pascal
(* ** GLOBAL : INDXTAB, INDX ***)
(* ** CALLED BY : ASSIGNMENT, ACCESSFOR ***)
(* ** ACCESSUNTIL, ACCESS_IP_STMT ***)
(* ** IO_STMT, ACCESS_CALL, ***)
(* ** ACCESSBLOCK, PROCESSBEGIN ***)

BEGIN
    INDX := INDX + 1;
    WITH INDXTAB[INDX] DO
        BEGIN
            BLOCK := INBLOCK;
            MODE := INMODE;
            NEXT := 0;
        END;
END; (* **INDIRECT***)
PROCEDURE EXPLICITFLOW( I, NUM: INTEGER);
(* GLOBAL : INDXTAB, INDX
  *)
(* CALLED BY : ASSIGNMENT, IO_STMT, ACCESS_CALL *)

VAR J, M, K, ROW : INTEGER;

PROCEDURE TRAVERSE_STACK;
BEGIN
  WITH INDXTAB[I] DO
  BEGIN IF NEXT > 1
    THEN BEGIN M := 1;
      WHILE M < NEXT DO
        BEGIN J := M;
          WHILE J <> NEXT DO
            BEGIN IF INDXARRAY[J] = INDXARRAY[J+1]
              THEN BEGIN K := J+1;
                WHILE K <> NEXT DO
                  BEGIN
                    INDXARRAY[K] := INDXARRAY[K+1];
                    K := K+1;
                    END;
                    NEXT := K-1;
                    END;
                IF J <> NEXT THEN J := J+1;
                END;
            M := M+1;
            END;
        END;
    END;
  IF NEXT > 0 THEN
  FOR J := 1 TO NEXT DO
  BEGIN ROW := INDXARRAY[J];
    MATRIX_ENTRY(ROW, NUM);
  END;
END(*TRAVERSE-STACK*);
BEGIN
  WHILE (INDXTAB[I].MODE < 6) DO
  BEGIN
    CASE INDXTAB[I].MODE OF
    1, 2, 3 : TRAVERSE_STACK;
    5 : WITH INDXTAB[I] DO
      BEGIN
        NEXT := NEXT + 1;
        INDXARRAY[NEXT] := NUM;
      END;
    END(*CASE*);
    I := I-1;
  END;
END(*EXPLICITFLOW*);
PROCEDURE TABLE_LOOKUP (N:INTEGER; LTOKEN:STRING;
   VAR KEY,J:INTEGER);
(*** GLOBAL  : INFOTABLE, PROCTAB, KEYTAB, KEYINDX  ***)
(*** CALLED BY : ANALYSIS  ***)
VAR ST:STRING;

PROCEDURE KEY_ENTRY (KSTR:STRING; VAR KEY_EN : INTEGER);

VAR M : INTEGER;

BEGIN
  M := 1;
  WHILE (KEYTAB[M] <> KSTR) AND (M <= KEYINDX)
  DO M := M+1;
  IF KEYTAB[M] = KSTR THEN KEY_EN := M
  ELSE BEGIN
    KEYINDX := KEYINDX+1;
    KEYTAB[KEYINDX] := KSTR;
    KEY_EN := KEYINDX;
  END;
END (* KEY_ENTRY *);

PROCEDURE FINDTOKEN (OBJECT:STRING; VAR L:INTEGER);

BEGIN
  IF INFOTABLE[L].INFONAME = OBJECT THEN L := L
  ELSE BEGIN
    WHILE (INFOTABLE[L].INFONAME <> OBJECT) AND (L<>1)
    DO L := L-1;
    IF L=1 THEN IF INFOTABLE[L].INFONAME=OBJECT
    THEN L := 1 ELSE L := 0
  END;
END (*FINDTOKEN*)

PROCEDURE DEFINEKEY;

VAR K:INTEGER;

STR:STRING;

BEGIN
  K := INFOTABLE[N].SCOPE;
  IF K=0 THEN STR := 'MAIN'
  ELSE WITH PROCTAB[K] DO
       IF PROC_TYPE=1
       THEN STR := CONCAT('PROC/', FROCNAMEN)
       ELSE BEGIN

IF (PARAMCT>0) AND
   (PROCNAME = LTOKEN)
   AND (ACTION<>SFUNCT)
THEN J := K;
STR := CONCAT('FUNCT/',PROCNAME);
END;
STR := CONCAT(STR,'...');
STR := CONCAT(STR,LTOKEN);
KEY_ENTRY(STR,KEY);
END; (*DEFINEKEY*)

PROCEDURE CKRECFIELD(STR:STRING);
VAR J:INTEGER;
BEGIN
  J := FIELDENTRY;
  WHILE J<>0 DO WITH INFOTABLE[J] DO
    BEGIN IF INFONAME=STR
        THEN BEGIN STR:=CONCAT(RECNAME,'.',LTOKEN);
            KEY_ENTRY(STR,KEY);
            J:=0;
        END
        ELSE BEGIN J:=NEXTINFO;
            IF J=0
            THEN BEGIN FINDTOKEN(STR,N);
                DEFINEKEY;
                END;
            END;
    END;
END; (*CKRECFIELD*)

PROCEDURE RECENTRY(STR:STRING; M:INTEGER; VAR K:INTEGER):
VAR TMP:STRING;
BEGIN
  K:=0;
  DELETE(STR,1,POS(' ',STR));
  IF POS(' ',STR)>0
  THEN TMP:=COPY(STR,1,POS(' ',STR)-1)
  ELSE TMP:=STR;
  WHILE M<>0 DO WITH INFOTABLE[M] DO
    IF INFONAME=TMP
    THEN BEGIN IF TMP=STR THEN K:= POINTOINDEX
        ELSE RECENTRY(STR,POINTOINDEX,K);
        M:=0;
    END
    ELSE M:=NEXTINFO;
END; (*RECENTRY*)
PROCEDURE RECORDFLOW;

VAR P : INTEGER;
  NAMEFIELD : STRING;

BEGIN
  IF RECNAMExx=
  THEN BEGIN
    IF POS(' ', LTOKEN) > 0
    THEN BEGIN
      FINDTOKEN (COPY (LTOKEN, 1, POS (' ', LTOKEN) - 1), N);
      DEFINENKEY;
      RECENTRY (LTOKEN, INFO TABLE [N], POINT INDEX, P);
      FIELDDIR := P;
      INDX:= 1;
      END
    ELSE IF POS (' ', LTOKEN) > 0
    THEN BEGIN
      NAMEFIELD := COPY (LTOKEN, 1, POS (' ', LTOKEN) - 1);
      CHKRECFIELD (NAMEFIELD);
      END
    ELSE CHKRECFIELD (L_TOKEN);
  END; (*RECORDFLOW*)

BEGIN (*TABLE_LOOKUP*)
  J := 0;
  KEY := 0;
  CASE WITH_STMT OF
    FALSE : BEGIN
      IF POS (' ', LTOKEN) > 0
      THEN FINDTOKEN (COPY (LTOKEN, 1, POS (' ', LTOKEN) - 1), N)
      ELSE FINDTOKEN (L_TOKEN, N);
      IF N > 0 THEN DEFINEKEY;
      END;
    TRUE : RECORDFLOW;
  END; (*CASE*)
END (* TABLE_LOOKUP*);
PROCEDURE ANALYSIS(GSTR:STRING; VAR KEY:INTEGER);

VAR PNCALL, TEMP, I, K : INTEGER;
STR, LTOKEN : STRING;

PROCEDURE SKIP;
VAR P:INTEGER;
BEGIN
  IF POS(' ', GSTR) > 0
  THEN BEGIN P := POS(' ', GSTR);
    DELETE(GSTR, P, POS(' ', GSTR) - P + 1);
    SKIP;
    END
  ELSE IF POS('***', GSTR) > 0
  THEN BEGIN P := POS('***', GSTR);
    DELETE(GSTR, P, 1);
    DELETE(GSTR, P, POS('***', GSTR) - P + 1);
    SKIP;
    END;
  GSTR := CONCAT(GSTR, ' ');
END; (* SKIP *)

PROCEDURE PARAMPASSING;

VAR J, ROW, COL: INTEGER;
PARAM_NAME: STRING;

BEGIN
  ROW := KEY;
  PARAM_NAME := INPTABLE[PARAM_ENTRY].INPONAME;
  TABLE_LOOKUP(PARAM_ENTRY, PARAM_NAME, COL, J);
  MATRIX_ENTRY(ROW, COL);
  IF INPTABLE[PARAM_ENTRY].PARAMTYPE = 1
  THEN BEGIN
    MATRIX_ENTRY(COL, ROW);
    EXPLICITFLOW(INDX - 1, ROW);
  END;
END; (*PARAMPASSING*)
BEGIN (* ANALYSIS *)
I := 1;
SKIP;
WHILE I <= LENGTH(GSTR) DO
BEGIN
IF GSTR[I] IN LETTERS
THEN BEGIN K := I;
WHILE GSTR[I+1] IN ALPHA DO I := I + 1;
L_TOKEN := COPY(GSTR,K,I-K+1);
TABLE_LOOKUP(INFOINDEX,L_TOKEN,KEY,FNCALL);
IF FNCALL > 0 THEN IF (GSTR[I+1] = '(') OR (GSTR[I+2] = '(')
THEN BEGIN
STR := COPY(GSTR,I+1,POS('('),GSTR)-I);
TEMP := KEY;
ACCESS_CALL(FNCALL,STR);
KEY := TEMP;
I := I + POS(')',STR);
END
ELSE CALL := FALSE;
END
IF CALL
THEN BEGIN IF KEY > 0 THEN PARAMPASSING;
PARAENTRY := PARAENTRY + 1;
END
ELSE IF KEY > 0 THEN WITH INDXTAB[INDX]
DO BEGIN
NEXT := NEXT + 1;
INDXARRAY[NEXT] := KEY;
END;
I := I + 1;
END;
END; (* ANALYSIS *)

PROCEDURE ACCESSWITH;
VAR N, NUM : INTEGER;
BEGIN
DELETE(LN, 1, LENGTH(WORD) + 1);
IF POS('BEGIN', LN) > 0
THEN INDIRECT(1, 4)
ELSE INDIRECT(0, 4);
WITH_STMT := TRUE;
ANALYSIS(COPY(LN, 1, POS(' DO', LN) - 1), NUM);
IF INDXTAB[INDX].BLOCK = 1
THEN DELETE(LN, 1, POS('BEGIN', LN) + 5)
ELSE DELETE(LN, 1, POS(' DO ', LN) + 3);
GETFIRSTWORD;
END; (*ACCESSWITH*)
PROCEDURE ELSE_STMT;
(* GLOBAL : WORD, INDXTAB *)
(* CALLED BY : ASSIGNMENT, IO_STMT, ACCESSBLOCK *)
BEGIN
GETFIRSTWORD;
WITH INDXTAB[INDX] DO
  IF WORD = 'BEGIN'
  THEN BEGIN BLOCK := 1;
        MODE := 1;
        DELETE(LN, 1, LENGTH(WORD) + 1);
        GETFIRSTWORD;
      END
  ELSE BEGIN BLOCK := 0;
        MODE := 1;
      END;
END(*END_ELSE*)

PROCEDURE CKELSE;
BEGIN
IF POS(' ELSE ', LN) > 0
THEN BEGIN WHILE (INDXTAB[INDX].MODE <> 2) AND
                   (INDXTAB[INDX].BLOCK <> 1) DO
     BEGIN IF INDXTAB[INDX].MODE = 4
       THEN BEGIN RECNME := '';
           WITH_STMT := TRUE;
           END;
       DELETE(LN, 1, POS(' ELSE ', LN) + 5);
     ELSE_STMT END
ELSE BEGIN WHILE (INDXTAB[INDX].BLOCK = 0)
DO BEGIN IF INDXTAB[INDX].MODE = 4
       THEN BEGIN RECNME := '';
           WITH_STMT := TRUE;
           END;
       INDX := INDX - 1;
     END;
READALINE;
GETFIRSTWORD;
END;
END(*CK ELSE*)
PROCEDURE ACCESS_CALL(*ENTRY:INTEGER;*);
(* GLOBAL : CALL, LETTERS, PROCTAB, INFCTABLE, INDEX*)
(* CALLED BY : ACCESSBLOCK *)

VAR M:NUM:INTEGER;

BEGIN
  M := POS('(','ST');
  IF M > 0 THEN BEGIN
    INDIRECT(0,0);
    CALL := TRUE;
    PARAM_ENTRY := PROCTAB[ENTRY].TOINFO;
    WRITELN('PAR ENTRY: ', PARAM_ENTRY);
    ANALYSIS(COPY(ST,M,POS(')','ST') - M),NUM);
    CALL := FALSE;
    INDEX := INDEX - 1;
    IF ST = LN THEN ELSE;
    END
  ELSE IF ST = LN THEN BEGIN READALINE; GETFIRSTWORD; END;
END(*ACCESS_CALL*);

PROCEDURE POP_END(VAR PROC_DONE : BOOLEAN);
(* GLOBAL : INDXTAB, INDEX, ACTION *)
(* CALLED BY : ACCESSBLOCK *)

BEGIN
  PROC_DONE := FALSE;
  CASE INDXTAB[INDEX].MODE OF
    1,2,3,4 : BEGIN
      INDEX := INDEX - 1;
      WHILE (INDXTAB[INDEX].BLOCK = 0) DO BEGIN
        INDEX := INDEX - 1;
        IF INDXTAB[INDEX].MODE = 4 THEN BEGIN RECRYBE:='';
          WITH_STMT := FALSE; END;
        END;
      READALINE;
      GETFIRSTWORD;
    END;
    6 : BEGIN INDEX := INDEX - 1;
      PROC_DONE := TRUE;
      ACTION := MAIN; END;
  END;
END (* POP_END*);
PROCEDURE ASSIGNMENT;
(* GLOBAL : LN, LETTERS, INDXTAB, INDX *)
(* CALLED BY : ACCESSBLOCK *)

VAR P, Q, R, ROW, N, NUM : INTEGER;
STR, TOKEN : STRING;

BEGIN

INDIRECT{0,0};
WHILE LN[1]=BLANK DO DELETE(LN, 1, 1);
P := POS(':=', LN);
R := 0;
IF POS(' ELSE ', LN) > 0 THEN BEGIN
O := POS(' ELSE ', LN);
ROW := INDXTAB[INDX].INDXARRAY.

MATRIX_ENTRY(ROW, NUM);
END;
INDX := INDX - 1;
EXPLICITFLOW(INDX, NUM);
IF R=0 THEN BEGIN
WHILE INDXTAB[INDX].BLOCK = 0
DO WITH INDXTABLE DO BEGIN
INDX := INDX-1;
IF MODE=4 THEN BEGIN
RENAME:='';
WITH_STMT := FALSE;
END;
END;
READALINE;
GETFIRSTWORD;
END;
IF R > 0 THEN BEGIN
WHILE (INDXTAB[INDX].BLOCK=0) AND
(INDXTAB[INDX].MODE<>2)
DO WITH INDXTABLE
DO BEGIN INDEX := INDEX-1;
IF MODE=4 THEN BEGIN
RENAME:='';
WITH_STMT := FALSE;
END;
END;
DELETE(LN, 1, R);
ELSE_STMT;
END;
END (* ASSIGNMENT *);
PROCEDURE ACCESSWHILE;
(* GLOBAL : LN, LETTERS, WORD *)
(* CALLED BY : ACCESSFOR, ACCESSBLOCK *)

VAR P, N, NUM : INTEGER;
BEGIN  P := POS(' DO ', LN);
        ANALYSIS (COPY(LN, 1, P), NUM);
        DELETE(LN, 1, P+3);
        GETFIRSTWORD;
        IF WORD='BEGIN'
          THEN BEGIN
            INDXTAB[INDX].BLOCK := 1;
            DELETE(LN, 1, LENGTH(WORD)+1);
            GETFIRSTWORD;
          END;
        END (* ACCESSWHILE *);

PROCEDURE ACCESSFOR;
(* GLOBAL : LN, LETTERS *)
(* CALLED BY : ACCESSBLOCK *)

VAR P, q, NUM : INTEGER;
BEGIN  INDIRECT(0, 3);
        Q := POS(' DO ', LN);
        ANALYSIS (COPY(LN, POS(' := ', LN), Q-P), NUM);
        P := POS(' :="', LN);
        ANALYSIS (COPY(LN, 1, P), NUM);
        WITH INDXTAB[INDX] DO
          IF NEXT>1 THEN BEGIN
            FOR P:=1 TO NEXT-1 DO
              MATRIX_ENTRY(INDXARRAY[P], NUM);
            NEXT := NEXT-1;
            END;
        DELETE(LN, 1, Q+3);
        GETFIRSTWORD;
        IF WORD='BEGIN'
          THEN BEGIN
            INDXARRAY[INDX].BLOCK := 1;
            DELETE(LN, 1, LENGTH(WORD)+1);
            GETFIRSTWORD;
          END;
        END; (* ACCESSFOR *)
PROCEDURE ACCESSUNTIL;
(* GLOBAL : WORD, LN, LETTERS, *)
(* FLAG, INDXTAB, INDX *)
(* CALLED BY : ACCESSBLOCK *)

VAR M, N, I, J, ROW, COL, NUM : INTEGER;

BEGIN INDIRECT(1, 5);
DELETE(LN, 1, LENGTH(WORD) + 1);
ANALYSIS(COPY(LN, 1, POS(';', LN), NUM);
M := INDXTAB[INDX-1].NEXT;
N := INDXTAB[INDX].NEXT;
IF (N>0) AND (M>0)
THEN BEGIN
  FOR I:=1 TO N DO
    BEGIN ROW := INDXTAB[INDX].INDXARRAY[I];
      FOR J:=1 TO M DO
        BEGIN COL := INDXTAB[INDX-1].INDXARRAY[J];
          MATRIX_ENTRY(ROW, COL);
        END;
    END;
  INDX := INDX-2;
END;
WHILE (INDXTAB[INDX].BLOCK = 0)
DO BEGIN INDX := INDX-1;
  WITH INDXTAB[INDX] DO
    IF MODE=4 THEN BEGIN RCNAME := '';
      WITH_STMT := FALSE;
    END;
  READALINE;
  GETFIRSTWORD;
END(* ACCESS UNTIL *);
PROCEDURE ACCESS_IF_STMT;
(*** GLOBAL : LN, LETTERS, INDXTAB, INDX ***)
(*** CALLED BY: ACCESSBLOCK ***)

VAR N, NUM : INTEGER;

BEGIN INDIRECT(0, 2);
   ANALYSIS(COPY(LN, 4, POS(' THEN ', LN) - 4), NUM);
   DELETE(LN, 1, POS(' THEN ', LN) + 5);
   GETFIRSTWORD;
   IF WORD = 'BEGIN'
      THEN BEGIN
         INDXTAB[INDX].BLOCK := 1;
         DELETE(LN, 1, LENGTH(WORD) + 1);
         GETFIRSTWORD;
      END;
END(* ACCESS IF_STMT *);

PROCEDURE IO_STMT;
(* GLOBAL : OPERATOR, LETTERS, INDXTAB, INDX, *)
(* CALL *)
(* CALLED BY : ACCESSBLOCK *)

VAR N, NUM, KEY, I, J : INTEGER;

BEGIN
   I := POS('(', LN);
   IF I > 0
      THEN BEGIN
         INDIRECT(0, 0);
         IF POS(' ELSE ', LN) > 0 THEN J := POS(' ELSE ', LN)
            ELSE J := POS(';', LN);
         ANALYSIS(COPY(LN, I + 1, J - I), NUM);
         WITH INDXTAB[INDX] DO
            IF NEXT > 0
               THEN IF (OPERATOR = WRITES) OR (OPERATOR = WRITES_LN)
                  THEN BEGIN FOR I := 1 TO NEXT DO
                        MATRIX_ENTRY(INDXARRAY[I], 2);
                        CALL := TRUE;
                        EXPLICITFLOW(INDX-1, 2);
                        CALL := FALSE;
                  END
               ELSE FOR I := 1 TO NEXT DO
                  BEGIN MATRIX_ENTRY(1, INDXARRAY[I]);
                     CALL := TRUE;
                     EXPLICITFLOW(INDX-1, INDXARRAY[I]);
                     CALL := FALSE;
                  END;
         CKElse;
      END
      ELSE BEGIN READALINE; GETFIRSTWORD; END;
END(*IO_STMT*);
PROCEDURE CHECK_CALL(CTOKEN:STRING; VAR PROC_CALL:BOOLEAN;  
  VAR AT_PROC_ENTRY: INTEGER);  
(* GLOBAL : PROCTAB, PROCIDX *)  
(* CALLED BY : ACCESSBLOCK *)  
VAR I : INTEGER;  
BEGIN  
  PROC_CALL := FALSE;  
  AT_PROC_ENTRY := 0;  
  IF PROCIDX>0 THEN FOR I := 1 TO PROCIDX DO  
    IF PROCTAB[I].PROCNAME = CTOKEN  
      THEN BEGIN  
        PROC_CALL := TRUE;  
        AT_PROC_ENTRY := I;  
        I := PROCIDX +1;  
      END;  
END(*CHECK_CALL*);  

PROCEDURE ACCESSBLOCK;  
(* GLOBAL : LN, WORD, OPERATOR, OP, LETTERS *)  
(* CALLED BY : PROCESSBEGIN *)  
VAR DONE,YES := BOOLEAN;  
  I, ENTRY:INTEGER;  
  STR:STRING;  

PROCEDURE CK_OPERATOR(TOKEN:STRING);  
BEGIN FOR OP:=WHILES TO IPS DO  
  IF BUFF[OP]=TOKEN  
    THEN BEGIN OPERATOR:=OP  
      OP := NONE;  
      END  
    ELSE OPERATOR := NONE;  
END (* CK_OPERATOR *);
BEGIN (* ACCESSBLOCK *)
DONE := FALSE;
DELETE (LN, 1, LENGTH (WORD) + 1);
GETFIRSTWORD;
CK_OPERATOR (WORD);
WHILE (NOT DONE) DO
BEGIN
CASE OPERATOR OF
WHILES : BEGIN
INDIRECT (0, 3);
DELETE (LN, 1, LENGTH (WORD) + 1);
ACCESSWHILE;
END;

WRITES, WRITES LN,
READS, READS LN := IO_STMT;
ENDS, ENDD := POP_END (DONE);
END;
END_ELSE : BEGIN
DELETE (LN, 1, POS (' ELSE ', LN) + 5);
ELSE_STMT;
END;
REPEATS : BEGIN
INDIRECT (1, 5);
DELETE (LN, 1, LENGTH (WORD) + 1);
GETFIRSTWORD;
END;
UNTILS : ACCESSUNTIL;
FORS : ACCESSFOR;
WITHS : ACCESSWITH;
IFS : ACCESS_IF_STMT;
NONE : BEGIN
CHECK_CALL (WORD, YES, ENTRY);
IF YES AND ((ACTION=MAIN)
OR (ACTION=SPROC))
THEN ACCESS_CALL (ENTRY, LN)
ELSE IF (POS ('=', LN) > 0
THEN ASSIGNMENT
ELSE
WRITELN ('**ERROR**INVALID STATEMENT');
END;
END (* CASE *);
IF NOT DONE THEN CK_OPERATOR (WORD);
END;
END (* ACCESSBLOCK *);
PROCEDURE MARK_UNUSED;
(* GLOBAL : PROCTAB, PROCIDX *)
(* CALLED BY : PROCESS BEGIN *)

VAR I, J : INTEGER;
BEGIN
  IF PROCIDX > 0
  THEN WITH PROCTAB[PROCIDX] DO
    IF PARAMCT > 0 THEN INFOINDEX := TOINFO + PARAMCT - 1;
  END (* MARK_UNUSED *);

PROCEDURE PROCESSBEGIN;
(* GLOBAL : INDXTAB, INDX *)
(* CALLED BY : MAIN *)
BEGIN
  CASE ACTION OF
    SFUNCT, SPROC : BEGIN
      INDIRECT(1,6);
      INDXTAB[INDX].NEXT := PROCIDX;
      END;
    MAIN : INDIRECT(1,6);
    END;
  ACCESSBLOCK;
  MARK_UNUSED;
  IF NOT FLAG THEN BEGIN
    READALINE;
    GETFIRSTWORD;
    END;
  END (* PROCESSBEGIN *);
PROCEDURE INITIALIZE;

PROCEDURE MATRIXINIT;

VAR I, J : INTEGER;

BEGIN
    FOR I := 1 TO MAXTABLES DO
        FOR J := 1 TO MAXTABLES DO
            MATRIX[I, J] := 0;
    END (*MATRIXINIT *);

PROCEDURE BODYINIT;

BEGIN
    BUFF[WHILES ] := 'WHILE';
    BUFF[WRITES ] := 'WRITE';
    BUFF[WRITE_LN] := 'WRITELN';
    BUFF[READS ] := 'READ';
    BUFF[READS_LN] := 'READLN';
    BUFF[ENDS ] := 'END';
    BUFF[ENDD ] := 'END_;'
    BUFF[END_ELSE] := 'ELSE,'
    BUFF[REPEAT ] := 'REPEAT';
    BUFF[UNTILS ] := 'UNTIL,'
    BUFF[FOR S] := 'FOR';
    BUFF[WITH S] := 'WITH';
    BUFF[IFS ] := 'IF';

END; (* BODYINIT *);
BEGIN (* INITIALIZE ROUTINE *)

WRITELN('ENTER FILE NAME WHICH TO BE MEASURE?');
READLN(NAME);
NAME := CONCAT(NAME,'.TEXT');
RESET(IPFILE,NAME);
REWITE(OUTFILE,'PRINTER:');
INFOINDEX := 0;
TYPINDEX := 0;
PROCIDX := 0;
INDX := 0;
PARAM_ENTRY := 0;
FIELDENTRY := 0;
KEYTAB[1] := 'IN';
KEYINDX := 2;
STATUS := MAIN;
ACTION := MAIN;
CALL := FALSE;
FLAG := FALSE;
WITH_STMT := FALSE;
LETTERS := ['A'..'Z'];
NUMBERS := ['0'..'9'];
ALPHA := LETTERS+NUMBERS+[' '];
RECNAMES := ' ';
BODYINIT;
MATRIXINIT;

END (*INITIALIZE*):

PROCEDURE PRINTOUT;

(*** PROCEDURE P1 IS USED FOR PRINT INFOTABLE ***)
(*** PROCEDURE P2 IS USED FOR PRINT TYPTABLE AND PROCTAB ***)
(*** PROCEDURE P2 IS USED FOR PRINT KEYTAB AND METRIX ***)
(*** GLOBAL : INFOTABLE, TYPTABLE, PROCCTAB, KEYTAB, MATRIX***)
(*** INFOINDEX, TYPIDX, PROVIDX, KEYINDEX. ***)
(*** CALLED BY : MAIN ***)

VAR Q,I,J,K : INTEGER;

PROCEDURE P1;
BEGIN
  READ(Q);
END

PROCEDURE P1;
FOR I := 1 TO 2 DO WRITELN(OUTFILE);
WRITELN(OUTFILE,'INFOTABLE');
WRITELN(OUTFILE);
WRITELN(OUTFILE,'INDEX', ' :5,'INFONAME', ' :5,'NEXTINFO', ' :5,'POINTOINDEX', ' :4,'SCOPE');
WRITELN(OUTFILE);
FOR I := 1 TO INFOINDEX DO WITH INFOTABLE[I] DO BEGIN IF I<10 THEN J := 1 ELSE J := 5;
IF NEXTINFO<10 THEN K := 1 ELSE K := 2;
WRITELN(OUTFILE,' ':2,'I', ':10-J,'INFONAME', ' -- ',
':15-LENGTH(INFONAME)-2,NEXTINFO,' ':14-K,
POINTOINDEX,' ':11,SCOPE);
END;
END;(* P1 *)

PROCEDURE P2;
BEGIN
IF TYPINDEX >0 THEN BEGIN
FOR I := 1 TO 2 DO WRITELN(OUTFILE);
WRITELN(OUTFILE,'TYPTABLE');
WRITELN(OUTFILE,'INDEX', ' :8,'TYPE NAME',
':5,'TO INFOINDEX');
FOR I := 1 TO TYPINDEX DO WITH TYPTABLE[I] DO BEGIN
IF I<10 THEN J := 1 ELSE J := 2;
WRITELN(OUTFILE,' ':2,'I', ':15-J,'TYPNAME',
':15-LENGTH(TYPNAME),FIRSTFIELD);
END;
END;
IF PROCIDX >0 THEN BEGIN
FOR I := 1 TO 2 DO WRITELN(OUTFILE);
WRITELN(OUTFILE,'PROC_TABLE');
WRITELN(OUTFILE,'INDEX', ' :10,'PROCNAME', ' :7,'TOINFO', ' :5,'PARAM COUNT', ' :3,'PROC TYPE');
WRITELN(OUTFILE);
FOR I := 1 TO PROCIDX DO WITH PROCTAB[I] DO BEGIN
IF I<10 THEN J := 1 ELSE J := 2;
WRITELN(OUTFILE,' ':2,'I', ':15-J,
PROCNAME', ':15-LENGTH(PROCNAME),TOINFO,
':12,PARAMCT', ':14,PROC_TYPE);
END;
END;(* P2 *)
PROCEDURE P3;
BEGIN
FOR I := 1 TO 3 DO WRITELN(OUTFILE);
WRITELN(OUTFILE,'KEY TABLE');
WRITELN(OUTFILE);
FOR I := 1 TO KEYINDX DO
BEGIN IF I<10 THEN K := 4 ELSE K := 3;
WRITE(OUTFILE,I,' :K,KEYTAB[I]);
END;
FOR I := 1 TO 3 DO WRITELN(OUTFILE)
WRITELN(OUTFILE,'METRIX');
WRITELN(OUTFILE);
FOR I := 1 TO KEYINDX DO
BEGIN IF I< 9 THEN K := 4 ELSE K := 3;
WRITE(OUTFILE,I,' :K);
END;
WRITELN(OUTFILE);
FOR I := 1 TO KEYINDX DO
BEGIN IF I<10 THEN K := 5 ELSE K := 4;
WRITE(OUTFILE,I,' :K);
FOR J := 1 TO KEYINDX DO
WRITE(OUTFILE,METRIX[I,J],':4);
WRITELN(OUTFILE);
END;
FOR I := 1 TO 3 DO WRITELN(OUTFILE);
END; (* P3 *)

BEGIN (* PRINTOUT *)
FOR I := 1 TO 16 DO WRITELN;
WRITELN('IFC PRINTOUT:');
WRITELN;
WRITELN(' 1. PRINTOUT KEYTAB AND METRIX.');
WRITELN;
WRITELN(' 2. PRINTOUT KEYTAB, METRIX,
INFOTABLE, TYPETAB AND PROCTAB');
FOR I := 1 TO 3 DO WRITELN;
WRITELN('ENTER NUMBER YOU WANT TO BE PRINTED?');
READ (I);
IF I=1 THEN P3 ELSE
IF I=2 THEN BEGIN P1;P2;P3 END
ELSE BEGIN
WRITELN('**ERROR**:ENTER NUMBER EITHER 1 OR 2');
PRINTOUT;
END;
END(*PRINTOUT *);
PROCEDURE READLINE;
(*** GLOBAL : LN, IPFILE, OUTFILE ***)
(*** CALLED BY : ACCESSRECORD, PROCESSVAR_TYPE ***)
(*** PROCESSPROC, ASSIGNMENT, ACCESSUNTIL ***)
(*** IO_STMT, ACCESS_CALL, PROCESSBEGIN, ***)
(*** MAIN ***)

VAR LINE: STRING;
P, Q, R: INTEGER;
MARK, DONE: BOOLEAN;

BEGIN
  IF EOF(IPFILE) THEN FLAG:=TRUE
  ELSE BEGIN
    LN:="";
    MARK:=FALSE;
    DONE:=FALSE;
    WHILE NOT DONE DO BEGIN
      READLN(IPFILE, LINE);
      WRITELN('READ....', LINE);
      WRITELN(OUTFILE, ' ', LINE);
      IF (LENGTH(LINE)>0) OR (LINE <> '') THEN BEGIN
        R := 1;
        WHILE LINE[R] = BLANK DO R := R+1;
        DELETE(LINE, 1, R-1);
        IF (POS('(*', LINE) = 0) THEN BEGIN
          P:=POS('END ', LINE);
          Q:=POS(';', LINE);
          IF P>0 THEN BEGIN
            DONE:=TRUE;
            LN := CONCAT(LINE, BLANK);
          END
          ELSE IF Q>0 THEN BEGIN
            LINE:=CONCAT(LINE, BLANK);
            IF MARK THEN LN:=CONCAT(LN, LINE)
          END;
        END;
      END;
      WHILE POS(' ', LN) > 0 DO DELETE(LN, POS(' ', LN), 1);
    END;
  END;
END (*READLINE*);
PROCEDURE GETFIRSTWORD;

/** GLOBAL : WORD, LN, FLAG
 **/ 
/** CALLED BY : PROCESSPROC, ELSE_STMT, */
/** ASSIGNMENT, ACCESSWHILE, ACCESSUNTIL, */
/** ACCESS_IF_STMT, IO_STMT, ACCESS_CALL, */
/** ACCESSBLOCK, PROCESSBEGIN, MAIN */

VAR P:INTEGER;

BEGIN

P := 1;
WHILE LN[P] = BLANK DO P := P + 1;
DELETE(LN, 1, P - 1);
P := POS(BLANK, LN);
WORD := COPY(LN, 1, P);
IF WORD='END.' THEN BEGIN
  WORD:= 'END.';
  FLAG:= TRUE
END ELSE IF (WORD='END;') OR (WORD='END ') THEN BEGIN
  DELETE(WORD, LENGTH(WORD), 1)
ELSE BEGIN
  P := 1;
  WHILE (WORD[P] IN LETTERS)
  AND (P<LENGTH(WORD)) DO P := P + 1;
  WORD:= COPY(WORD, 1, P - 1);
END;

END; (*GETFIRSTWO RD*)
PROCEDURE BODY;
BEGIN
READALINE;
GETFIRSTWORD;
WHILE NOT FLAG DO
BEGIN
IF WORD='TYPE'
THEN BEGIN STATUS:=STYPE;
PROCESSVAR_TYPE; END
ELSE IF WORD='VAR'
THEN BEGIN STATUS:=SVAR;
PROCESSVAR_TYPE; END
ELSE IF WORD='PROCEDURE'
THEN BEGIN ACTION:=SPROC;
STATUS:=SPROC;
PROCESSPROC; END
ELSE IF WORD='FUNCTION'
THEN BEGIN ACTION:=SFUNCT;
STATUS:=SFUNCT;
PROCESSPROC; END
ELSE IF WORD='BEGIN'
THEN PROCESSBEGIN
ELSE BEGIN READALINE;
GETFIRSTWORD; END
END (* WHILE *)
END(*BODY*);
BEGIN (* MAIN *)
INITIALIZE;
BODY;
PRINTOUT;
END.
APPENDIX C

USER'S GUIDE.

This section serves to instruct the user on how to operate the P-system. The user should read and understand this short manual before he attempts to operate the P-system. The following briefly describes the necessary characteristics and features of the P-system and illustrates how to run the IFC program step by step.

The operation of the P-system is based on the command level modes. The example are F(ile mode and E(dit mode. Each mode has a different set of subcommands that must only be used in that mode. The user may type "?" on the keyboard to have more subcommands displayed on the screen. If a different mode is desired, user must exit the current mode before entering to the new mode. After booting the system, the case for the command mode will display on the screen:

COMMAND: E(dit, R(un, F(ile, C(omp, X(ecute, A(ssen, H(al)?

The following section will briefly explain some subcommands and their use in the file mode and the editor mode. Since the input program to IFC needs to be error free, the user may use the C(ompile command to check for syntax errors that might occur in his input program before executing IFC program.
Pressing "F" for File will enable user to enter the file mode. The file system is used to maintain the storing and retrieving of information or items on the disk storage. The information stored on the disk may be given by the disk directory command L(dir). In this case, a file name with the prefix "IP" refers to an input Pascal program to IFP program.

File: G(et, S(ave, W(hat, N(ew, L(list, R(em, C(hng, Q uit?"

- L(dir) list all the files kept in the disk directory. User may request for IFP input file listing by typing "IP*.text" on the keyboard after the prompt line: disk what vol?
- G(et get the requested file name from disk into the current workfile.
- C(hng to change the name of the file in the directory to a new name.
- N(ew create a new work file, clear the current work file.
- R(em removes files from the disk library.
- Q uit exits from FILER mode.
THE EDITOR

The user presses "E" for the E(dit) command at the command level. The editor allows the user to enter, change and modify the program (Standard Pascal). The editor will ask for the name of the file the user wants to keep on the disk library. In this report, a file with the prefix of "IP##" is recommended, where "##" is the user file name.

E(dit):A(djust, C(py, D(elete, F(ind, I(nsert, J(ump, Q(uit)

I(nsert insert characters into the text buffer as they are typed in.

D(elete deletes characters from the text buffer.

J(ump moves the cursor to the beginning, ending or from marker to marker where a marker is pre-defined using the "set" instruction.

F(ind locate a string of characters in the text buffer

Q(uit exits the EDITOR with options, one of the the four options must be selected by typing U, E, R or W.
INPUT SPECIFICATION

* input a Standard Pascal program.
* use a ';' to separate every program statements, 80 characters per input line.
* comments must start at the new line.
* use only lowercase characters.
* no nested record structure in the type declaration.
* acceptable input program statements.
  - assignment statement
  - BEGIN-END
  - repetition statements:
    FOR, WHILE_DO, REPEAT_UNTIL
  - conditional statement
    IF THEN ELSE
  - call statement:
    call by name only
    use function call in the assignment only
  - WITH statement
  - I/O statement:
    READ, READLN, WRITE,_WRITELN
OPERATION CONCEPTS

In an attempt to demonstrate a sample program and describe the concepts to operate the IFC program, IP$^9$ is selected as an input program to IFC.

(1) Booting the P-system

the following prompt line will appear on the screen

Command: E(dit, R(un, F(ile, C(omp, L(ink, S)ubmit, X{ecute. ?

(2) To create and update a text file

2.1 Enter to FILER mode.................F(ile.

2.2 To create a text file:

Clear work file..................N{ew.
continue to step 2.4

2.3 To update an existing file:

Get existing file..................G(et.

2.4 Exit FILER mode..................Q uit.

2.5 Enter EDIT mode..................E(dit.
(modify your program)

2.6 Exit EDIT mode..................Q uit.

(3) User presses the key "X" on the keyboard.

the following prompt line will appear on the screen:

Execute what file?

3.1 User types "IFC" on the keyboard.
the following prompt line will appear on the screen:

Enter input filename to be run on IPC?

3.2 User types: "IP9" on the keyboard.
The following prompt lines display the echo of the input source program. After the IPC program has finished scanning the input source program, the following prompt lines will appear on the screen:

IPC printout:

1. printout: KEYTAB, METRIX.

2. printout: KEYTAB, METRIX, INFOTABLE, TYPTAB and PROCTAB.

Enter number you wish to have the printout?

3.3 User presses either "1" or "2" on the keyboard, the input source program listing and the tables according to the number pressed will be printed on the printer.

(4) repeat the first step if the user wants to test more input programs.

(5) Press H (alt to leave the system.)
APPENDIX D
Sample Program Listings

PROGRAM SAMPLE;
(* IP1 *)
VAR A, B, C, D, E : INTEGER;
BEGIN
  READ (A, B);
  REPEAT
    READ(C);
    IF A<10 THEN WHILE B <> 0
      DO BEGIN D := 2*E;
      END;
    UNTIL (C=0);
  END.

INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFNAMEx</th>
<th>NEXTINFO</th>
<th>POINTOINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

KEY TABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IN</td>
</tr>
<tr>
<td>2</td>
<td>OUT</td>
</tr>
<tr>
<td>3</td>
<td>MAIN...A</td>
</tr>
<tr>
<td>4</td>
<td>MAIN...B</td>
</tr>
<tr>
<td>5</td>
<td>MAIN...C</td>
</tr>
<tr>
<td>6</td>
<td>MAIN...E</td>
</tr>
<tr>
<td>7</td>
<td>MAIN...D</td>
</tr>
</tbody>
</table>

MATRIX:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROGRAM IP2;

VAR SA : ARRAY[1..10] OF STRING;
   IA : ARRAY[1..10] OF INTEGER;
   K, LN, MAX, KMAX : INTEGER;
   CH : CHAR;

(* MAIN PROGRAM *)

BEGIN
   K := 1;
   MAX := 0;
   REPEAT
      WRITE('SCORE');
      READ(IA[K]);
      IF NOT EOF THEN
      BEGIN
         WRITE('NAME');
         READ(SA[K]);
         LN := IA[K];
         IF IA[K] > MAX THEN
         BEGIN
            MAX := IA[K];
            KMAX := K;
         END;
         K := K+1;
      END;
      UNTIL EOF OR (K > 10);
      WRITELN;
      WRITELN('BEST SCORE: ', IA[KMAX], ',', SA[KMAX]);
   END;

INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFONAME</th>
<th>NEXTINFO</th>
<th>POINTEINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SA</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>IA</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>K</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>LN</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>MAX</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>KMAX</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>CH</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**KEY TABLE**

1  IN
2  OUT
3  MAIN...K
4  MAIN...MAX
5  MAIN...IA
6  MAIN...SA
7  MAIN...LN
8  MAIN...KMAX

**MATRIX**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROGRAM IP3;
VAR X, Y, Z, W : INTEGER;
BEGIN
  READ(X, Y);
  IF X<10 THEN BEGIN
    READ(Z);
    IF Y>5 THEN Writeln('YY := ', Y)
    ELSE W := Z+Y;
  END
  ELSE BEGIN
    READ(W);
    IF Y>0 THEN Writeln('YY := ', Y+W)
    ELSE Z := W + Y;
  END;
  Writeln(X, Y, Z, W);
END.

INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFONAME</th>
<th>NEXTINFO</th>
<th>POINTOINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>W</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

KEY TABLE

- IN
- OUT
- MAIN...X
- MAIN...Y
- MAIN...Z
- MAIN...W

MATRIX

\[
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 2 & 1 \\
1 & 0 & 0 & 0 & 2 & 2 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]
PROGRAM IP4;
CONST
  MAXLENGTH = 1000;
TYPE
  INDEX = 1..MAXLENGTH;
  ROWTYPE = ARRAY[INDEX] OF INTEGER;
VAR
  INROW : ROWTYPE;
  COUNT : 0..MAXLENGTH;
  IX : INDEX;
PROCEDURE SORT (VAR ROW: ROWTYPE; LENGTH : INDEX);
VAR
  JUMP, M, N : INDEX;
  TEMP : INTEGER;
  ALLDONE : BOOLEAN;
BEGIN
  JUMP := LENGTH;
  WHILE JUMP > 1 DO
  BEGIN
    JUMP := JUMP DIV 2;
    REPEAT
      ALLDONE := TRUE;
      FOR M := 1 TO LENGTH - JUMP DO
      BEGIN
        N := M + JUMP;
        IF ROW[M] > ROW[N] THEN BEGIN
          TEMP := ROW[M];
          ROW[M] := ROW[N];
          ROW[N] := TEMP;
          ALLDONE := FALSE;
        END;
      END;
    UNTIL ALLDONE;
  END;
END;
BEGIN
  COUNT := 0;
  READ(INROW,COUNT + 1);
  WHILE NOT EOF DO
  BEGIN
    COUNT := COUNT + 1;
    READ(INROW,COUNT + 1);
  END;
  IF COUNT > 0 THEN BEGIN
    SORT(INROW,COUNT);
    FOR IX := 1 TO COUNT DO
    BEGIN
      WRITE(INROW[IX]);
    END;
  ELSE WRITE('<NO INPUT'>);
END.
### INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFORME</th>
<th>NEXTINFO</th>
<th>POINTEINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INROW.</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>COUNT.</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>LINC.</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>ROW.</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>LENGTH.</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### PROC_TABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>PROCNAME</th>
<th>TOINFO</th>
<th>PARAM</th>
<th>COUNT</th>
<th>PROC TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SORT</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### KEY TABLE

<table>
<thead>
<tr>
<th>1</th>
<th>IN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>OUT</td>
</tr>
<tr>
<td>3</td>
<td>PROC/SORT... LENGTH</td>
</tr>
<tr>
<td>4</td>
<td>PROC/SORT... JUMP</td>
</tr>
<tr>
<td>5</td>
<td>PROC/SORT... ALLDONE</td>
</tr>
<tr>
<td>6</td>
<td>PROC/SORT... M</td>
</tr>
<tr>
<td>7</td>
<td>PROC/SORT... N</td>
</tr>
<tr>
<td>8</td>
<td>PROC/SORT... ROW</td>
</tr>
<tr>
<td>9</td>
<td>PROC/SORT... TEMP</td>
</tr>
<tr>
<td>10</td>
<td>MAIN... COUNT</td>
</tr>
<tr>
<td>11</td>
<td>MAIN... INROW</td>
</tr>
</tbody>
</table>

### MATRIX

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROGRAM IPS:
VAR A, B, C, D, E : INTEGER;
BEGIN
  READ(A, B, C, D);
  REPEAT
    IF A>B THEN
      IF C>0 THEN BEGIN
        WRITE(D, E);
        WHILE (B>0) DO D := A - 1;
      END
      ELSE READ(E);
    UNTIL (E=0);
  END.

INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFONAME</th>
<th>NEXTINFO</th>
<th>POINTERINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

KEY TABLE

1 IN
2 OUT
3 MAIN...A
4 MAIN...B
5 MAIN...C
6 MAIN...D
7 MAIN...E

MATRIX

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
PROGRAM IPS:
VAR A, B, C : INTEGER;

FUNCTION D(X, Y: INTEGER): INTEGER:
VAR M, N: INTEGER;
BEGIN
  READ(M, N);
  IF A THEN D := C * M
  ELSE D := C * N;
END;

BEGIN
  READ <A, C>;
  B := D(A, C) + 1;
END.

INFOTABLE
INDEX  INFONAME  NEXTINFO  POINTOINDEX  SCOPE
 1      A          0         0           0
 2      B          0         0           0
 3      C          0         0           0
 4      D          0         0           1
 5      X          0         0           1
 6      Y          0         0           1

PROC_TABLE
INDEX  PROCNAME  TOINFO  PARAM  COUNT  PROC  TYPE
 1      D         5       2       0
### KEY TABLE

1. IN
2. OUT
3. FUNCT/D... M
4. FUNCT/D... N
5. MAIN... A
6. MAIN... C
7. FUNCT/D... D
8. FUNCT/D... X
9. FUNCT/D... Y
10. MAIN... B

### MATRIX

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROGRAM IP7;

VAR B, C : INTEGER;
    X, Y, Z, W : INTEGER;

FUNCTION D(VAR M:INTEGER; N:INTEGER):INTEGER:
BEGIN
    IF W THEN D := C
    ELSE D := C + 2;
END;
BEGIN
    READ(X, Y, C);
    IF X<10 THEN BEGIN
        Z := 1 + C;
    END
    ELSE BEGIN
        Z := 2;
        IF Y > 5 THEN W := D(X, Y) + Z ELSE W := X+Y;
    END;
    WRITELN(W, X, Y, Z);
END.

INFOTABLE

INDEX  INFONAME  NEXTINFO  POINTINDEX  SCOPE
---  -----  --------  --------  ----
  1    B..     0        0        0
  2    C..     0        0        0
  3    X..     0        0        0
  4    Y..     0        0        0
  5    Z..     0        0        0
  6    W..     0        0        0
  7    D..     0        0        1
  8    N..     0        0        1
  9

PROC_TABLE

INDEX  PROCNAME  TOINFO  PARAM COUNT  PROC TYPE
---  -------  ------  ---------  -------
  1    D       8        2          0
**KEY TABLE**

<table>
<thead>
<tr>
<th></th>
<th>IN</th>
<th>OUT</th>
<th>MAIN...W</th>
<th>MAIN...C</th>
<th>FUNCT/D...D</th>
<th>MAIN...X</th>
<th>MAIN...Y</th>
<th>MAIN...Z</th>
<th>FUNCT/D...M</th>
<th>FUNCT/D...N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MATRIX**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROGRAM IPS:

TYPE
  ATYPE = RECORD
    AA : STRING;
    AB : INTEGER;
    AC : BOOLEAN;
  END;

  BTYPE = RECORD
    BA : STRING;
    BC : INTEGER;
  END;

VAR A : ATYPE;
    B : BTYPE;
    C, D : INTEGER;
    E : BOOLEAN;
BEGIN
  READ (C, D);
  IF C > D THEN
    WITH A DO
    BEGIN
      READ (AA, AB);
      B. BA := AA;
    END;
END;

INFOTABLE

INDEX | INFONAME | NEXTINFO | POINTOINDEX | SCOPE |
------ | -------- | -------- | ------------ |-------|
   1   | AA      |     2    |     0       |   0   |
   2   | AB      |     3    |     0       |   0   |
   3   | AC      |     0    |     0       |   0   |
   4   | BA      |     5    |     0       |   0   |
   5   | BC      |     0    |     0       |   0   |
   6   | A       |     0    |     1       |   0   |
   7   | B       |     0    |     4       |   0   |
   8   | C       |     0    |     0       |   0   |
   9   | D       |     0    |     0       |   0   |
  10   | E       |     0    |     0       |   0   |

TYPTABLE

INDEX | TYPE NAME | TO INFOINDEX |
------ | --------- | ------------|
   1   | ATYPE    |     1       |
   2   | BTYPE    |     4       |
KEY TABLE

1  IN
2  OUT
3  MAIN...C
4  MAIN...D
5  MAIN...A.AA
6  MAIN...A.AB
7  MAIN...B.BA

MATRIX

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROGRAM IP3;

TYPE
  ATYPE = RECORD
    AA : STRING;
    AB : INTEGER;
    AC : BOOLEAN;
  END;

VAR A, B : ATYPE;
    C, D : INTEGER;
    E : BOOLEAN;
BEGIN
  READ <C, D>;
  IF C > D THEN
    WITH A DO
      BEGIN
        READ <AA, AB>;
        E.AA := AA;
        C := D + AB;
      END;
    WRITELN <B, AC>;
    A.AB := C + B.AB;
  END;

INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFONAME</th>
<th>NEXTINFO</th>
<th>POINTINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AA</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>AB</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>AC</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

TYPTABLE

INDEX | TYPE NAME | TO INFOINDEX |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ATYPE</td>
<td>1</td>
</tr>
</tbody>
</table>
# Key Table

<table>
<thead>
<tr>
<th></th>
<th>IN</th>
<th>OUT</th>
<th>MAIN...C</th>
<th>MAIN...D</th>
<th>MAIN...A AA</th>
<th>MAIN...A AB</th>
<th>MAIN...B AA</th>
<th>MAIN...B AC</th>
<th>MAIN...B AB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

# Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROGRAM IP10;
CONST
  FIRST = 2;
VAR
  NUMERA, DENOMINA, LAST, COUNT : INTEGER;
  I : INTEGER;
PROCEDURE LOWTERM(VAR NUM, DEN : INTEGER);
VAR
  NUMCOPY, DENCOPY, REMAINDER : INTEGER;
BEGIN
  NUMCOPY := NUM;
  DENCOPY := DEN;
  WHILE DENCOPY <> 0 DO
  BEGIN
    REMAINDER := NUMCOPY MOD DENCOPY;
    NUMCOPY := DENCOPY;
    DENCOPY := REMAINDER;
  END;
  IF NUMCOPY > 1 THEN BEGIN
    NUM := NUM DIV NUMCOPY;
    DEN := DEN DIV NUMCOPY;
  END;
END;
PROCEDURE ADDRATIONALS(VAR NUM1, DEN1 : INTEGER; NUM2, DEN2 : INTEGER);
BEGIN
  NUM1 := NUM1 * DEN2 + NUM2 * DEN1;
  DEN1 := DEN1 * DEN2;
END;
BEGIN
  NUMERA := 1;
  DENOMINA := 1;
  READ(LAST, COUNT);
  FOR I := FIRST TO LAST DO
  BEGIN
    ADDRATIONALS(NUMERA, DENOMINA, LAST, COUNT);
    LOWTERM(NUMERA, DENOMINA);
    WRITELN(NUMERA:1, '/', DENOMINA:1);
  END;
END.
### INFOTABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>INFONAME</th>
<th>NEXTINFO</th>
<th>POINTERINDEX</th>
<th>SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NUMERA..</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>DENOMINA..</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>LAST..</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>COUNT..</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>I..</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>NUM..</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>DEN..</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>NUM1..</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>DEN1..</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>NUM2..</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>DEN2..</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### PROC_TABLE

<table>
<thead>
<tr>
<th>INDEX</th>
<th>PROCNAME</th>
<th>TOINFO</th>
<th>PARAM</th>
<th>COUNT</th>
<th>PROC TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LOWERTERM</td>
<td>6</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>ADDRATIONALS</td>
<td>8</td>
<td>4</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
**KEY TABLE**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>OUT</td>
<td>PROC/LOWTERM...</td>
<td>NUM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/LOWTERM...</td>
<td>NUMCOPY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/LOWTERM...</td>
<td>DEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/LOWTERM...</td>
<td>DENCOPY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/LOWTERM...</td>
<td>REMAINDER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/ADDITIONAL...</td>
<td>NUM1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/ADDITIONAL...</td>
<td>DEN2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/ADDITIONAL...</td>
<td>NUM2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROC/ADDITIONAL...</td>
<td>DEN1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAIN...</td>
<td>NUMERA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAIN...</td>
<td>DENOMINA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAIN...</td>
<td>LAST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAIN...</td>
<td>COUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MATRIX:**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
INFORMATION FLOW COMPLEXITY

by

Pakarat Udomphorn

B.A., Ramkhamheanu University, Thailand, 1977

----------------------------------

AN ABSTRACT OF MASTER'S REPORT

submitted in partial fulfillment of
the requirement for the degree
MASTER OF SCIENCE

Department of Computer Science
KANSAS STATE UNIVERSITY
Manhattan, Kansas
1984
ABSTRACT

The areas of control flow and information flow are frequently mentioned in techniques for measuring software complexity. Control flow refers to the execution sequence of the statements and information flow refers to the dependency of a variable on the values of other variables. The various measures usually cover either intermodule or intramodule flow. The features of the information flow can be either explicit flow caused by simple statements or implicit flow caused by control statements.

A tool, the information flow complexity (IFC) program was designed to report the overall information transfer between objects (program variables) in a Pascal program. The IFC was written in UCSD Pascal and runs under the P-system on the PDQ-3 microcomputer. A metric which represents the information flow is the output. A flow graph can be derived from this metric and may be used to analyze the complexity of the information flow among the program variables. In addition, this metric can be used to identify the program variables which are dependent or independent. This may lead to an improved understanding of the design.

Presented in this report is an overview of the recent work, a description of IFC, the IFC source code and sample input program listings.