INTRODUCTION TO PL/1

PL/I is a structured language to develop systems and applications programs (both business and scientific).

Significant features :

- Allows Free format
- Regards a program as a continuous stream of data
- Supports subprogram and functions
- Uses defaults

Building blocks of PL/I :

- Made up of a series of subprograms and called Procedure
- Program is structured into a MAIN program and subprograms.
- Subprograms include subroutine and functions.

Every PL/I program consists of :

- At least one Procedure
- Blocks
- Group

- There must be one and only one MAIN procedure to every program, the MAIN procedure statement consists of :
  - Label
  - The statement ‘PROCEDURE OPTIONS (MAIN)’
  - A semicolon to mark the end of the statement.

Coding a Program :

1. Comment line(s) begins with /* and ends with */. Although comments may be embedded within a PL/I statements , but it is recommended to keep the embedded comments minimum.

2. The first PL/I statement in the program is the PROCEDURE statement :

AVERAGE : PROC[EDURE] OPTIONS(MAIN);

AVERAGE -- it is the name of the program(label) and compulsory and marks the beginning of a program.

OPTIONS(MAIN) -- compulsory for main programs and if not specified , then the program is a subroutine.

A PL/I program is compiled by PL/I compiler and converted into the binary , Object program file for link editing .

Advantages of PL/I are :

1. Better integration of sets of programs covering several applications.
2. Easier interchangeability of files
3. Fewer programs to be written in a machine oriented language.
4. Better machine utilization, greater use of channels , more flexible storage assignment, better intercept handling.

The process of compiling a PL/I program and executing it in the computer's memory takes place through JCL.

Some restrictions :

Col # 1 for O.S.

Col # 2 - 72 for PL/I statements

Col # 73 - 80 ignored by complier

PL/I Language Components :

1. **Character sets** : 60 characters in total, A - Z , extended characters e.g # , @ , $, 0 -9 and 21 special characters.

2. **Identifiers** : Used to name data, procedures , files , labels of PL/I statements and keywords.

   An identifiers for data names , statement labels and internal procedure names may be from 1 to 30 alphabetic characters (A – Z,@, !, #, $) , numeric digits (0-9) and underscore character.

3. **Statement format** :

   LABEL : KEYWORD STATEMENT OPTIONS ;

   the source program . A PL/I statement may be continued across several lines.One line may contain several statements terminated by semi-colon.

4. **PL/I constants** : A constant is a data item that does not have a name and whose value cannot change in a program. There are six types of constants , but only 2 or 3 types are used.

i) Decimal Fixed-point constants

ii) Decimal Floating-point constants(using E-notation).

iii) Character string constants : up to 256 characters , must be enclosed by ‘ , if an apostrophe is a part of data , then it should be preceded by another apostrophe with no intervening blanks.The repetition factors can be specified.

iv) Bit-string constants : It’s a series of binary digits enclosed in single quotes and followed by the letter B. Used as flag , setting 1 or 0 , repetition factor can be used.

v) Binary Fixed-Point and Binary Floating-Point constants .

Data types :

FIXED DECIMAL	Two digits/byte plus sign	Comm. Appl
FIXED BINARY	Half word or full word	Comm & scientific appl
FLOAT DECIMAL	Full word, double word or two double words	Scientific appl Arithmetic
PICTURE	One digit / byte	
CHARACTER or PICTURE	One character/byte	
BIT	One byte	Boolean oper

DECLARE statement :

The purposes :

i) For implicit declaration

ii) For data aggregation

iii) For input and output files

Base and Scale and precision attributes

DECLARE PRICE DECIMAL FIXED(5,2)

DECLARE PI FLOAT DECIMAL(6)

DECLARE COUNT FIXED BINARY(15)

DECLARE PI FLOAT BINARY(31)

DECLARE BETA FIXED BINARY(15,2)

Storage classes :

Any data name occupies some space in the main memory to hold the data , when this storage space will be allocated to a data is decided depending on the *storage class* of the data .

The types are :

1. Automatic (Default)-- can be abbreviated as AUTO
2. Static
3. Based
4. Controlled --- can be abbreviated as CTL

Based : The address is delimited by a pointer variable.

E.g. DCL P POINTER;

DCL A(100) FIXED BASED(P);

Address can be assigned in any of the following ways:

1. By assigning the value returned by the ADDR built-in function.
2. By assigning the value of another pointer.
3. With the SET option of a READ statement.
4. With the SET option of a LOCATE statement.

Controlled : Similar to based , in this programmer has a greater degree of control in the allocation of storage than he does for STATIC or AUTOMATIC. DCL A(100) INIT((100) 0) CONTROLLED ;

Partially declared identifiers :

1. The base : DCL A DECIMAL ; DCL B BINARY;
2. The scale : DCL C FIXED; DCL D FLOAT;
3. The base and precision : DCL AA DECIMAL(16);

   DCL BB BINARY(53);
4. The scale and precision : DCL CC FIXED(9,2) ;

   DCL DD FLOAT(16);

List-directed Input:

GET LIST (A,B,C,D);

GET LIST (A,B,C,D) COPY;

List-directed Output: The default line size for a PUT LIST is 120 positions.

Different tab positions on the screen are :

column no 1,25,49,73,97,121(only be used to print a record having length more than 120 characters). Different data items are printed beginning at predetermined tab positions.

PUT LIST (A,B,C,D);

PUT LIST (50,’abc’,123.445);

To print at a specified tab position ,i.e. without preceding the previous position , specify a character-string of a blank to the preceding tab positions.

E.g. PUT LIST (50,’ ‘,’abc’,’ ‘,123.445);---

First value will be printed at col no 1, second value at 49th col. , third value at 97th col , a blank will be printed at both 25th and 73rd col positions.

Whenever PUT LIST statement is executed for the first time , it automatically skips to a new page ,if you want to print different values in different lines or pages , SKIP and PAGE options can be used.

PAGE is used to advance to the top of a new page.

SKIP is used to advance by a number of lines specified . If the no. of line is not specified , then assumed as SKIP(1).

E.g. PUT PAGE LIST (‘ABC’);

PUT SKIP LIST (1212);

PUT SKIP(2) LIST(343);

A skip(0) causes a suppression of the line feed.

E.g. PUT PAGE LIST (‘ SALES REPORT’);

PUT SKIP(0) LIST ((12)’_’) ;

To specify the line number :

e.g. PUT PAGE LINE(10) LIST(A,B,C);

PUT PAGE;

PUT SKIP(2);

PUT LINE(10) ;

The Assignment Statement :

e.g.TOTAL_COST = COST * QUANTITY;

A,B,C = 0;

The Arithmetic Operations :

There are five basic arithmetic operators :

	Symbol	Operations
1.	**	Exponentiation
2.	*	Multiplication
3.	/	Division
4.	+	Addition
5.	-	Subtraction

Concatenation :

The || symbol is used is to join string or Bit data.

e.g. NAME1 = ‘IIS’;

NAME2 = ‘INFOTECH’

COMPANY_NAME = NAME1 || NAME2;

DATA1 = ‘1100’B;

DATA2 = ‘0011’B;

DATA3 = DATA1 || DATA2 ;

----This will store ‘11000011’B in DATA3.

Built-in Functions :

The following are the classes of Built-in functions :

1. Arithmetic function
2. Mathematical function
3. Array-handling function
4. String handling
5. Condition handling
6. Storage control
7. Miscellaneous

Arithmetic function :

ABS(-44) --- 44

CEIL(3.333) --- 4

FLOOR(3.333) ---- 3

MIN(2,6,4) ---- 2

MAX(2,6,4) ---- 6

TRUNC(3.33) --- 3.00

SIGN(-4) --- -1

SIGN(0) ---- 0

SIGN(4) --- 1

ROUND(123.456,1) --- 123.500

ROUND(123.456,2) --- 123.460

ROUND(123.456,0) ---- 123.000

ROUND(123.456,-1) ---- 120.000

ROUND(123.456,-2) ---- 100.000

ROUND(123.456,-3) ---- 0

MOD (25,6) --- 1

Mathematical function

SIN , COS , TAN , LOG , LOG10 , SQRT

String handling

SUBSTR('IIS INFOTECH',5,4) --- INFO

A CHAR(10) --- A = ' 10'

LENGTH(A) -- 3

REPEAT('A',2) --- 'AA'

TRANSLATE ('CAT','C','B') --- 'BAT'

Miscellaneous

DATE returns date in the form of YYMMDD.

TIME returns current time in the form of HHMMSSTTT.

When these functions are used they should be explicitly declared as BUILTIN .

UNSPEC (identifier) - Converts any identifier to a BIT string.

DCL A CHAR(2) INIT(‘AB’);

PUT LIST (UNSPEC(A)); --- prints the binary equivalent of ‘AB’

UNSPEC(A) = ‘0100000101000011’B; -- The value ‘AC’ will be assigned to the variable A

An example for Pseudo-variable:

```
BLANK : PROC OPTIONS(MAIN);
   DCL NAME CHAR(15) INIT('NOW IS THE TIME');
   DCL NAME1  CHAR(12) VARYING;
   DCL I FIXED BINARY(15);
   DCL J FIXED BIN(15) INIT(1);
   DO I =1 TO LENGTH(NAME);
        IF SUBSTR(NAME,I,1)=' '
         THEN;
        ELSE
        DO;
```

```
            SUBSTR(NAME1,J,1)=SUBSTR(NAME,I,1);
             J=J+1;
            END;
END;
PUT LIST(NAME1);

 END BLANK;
```

Data Conversion: When mixed data types appear in an arithmetic expression , data items are converted to the data format of the other.

DECIMAL is converted to BINARY.

FIXED is converted to FLOAT.

BIT is converted to CHARACTER

A zero bit becomes a character 0.

Arithmetic to CHARACTER(length should be > 2 bytes)

CHARACTER to arithmetic

Arithmetic(without symbol) to BIT

BIT(always unsigned) to arithmetic

CHARACTER to BIT

Selection Construct (Logical testing) :

IF statement : The list of comparison operators ..

Symbols used are GE or >= , GT or > , NE , ~= ,LT or < , LE or <= , NL , NG .

The list of Logical operators ….

Symbols used are ~ or ^ , & , | .

```
IF condition1  THEN DO ……;
    ……;
    ……;
    END;

IF A=B THEN
    IF A=C THEN  X=1;
    ELSE;
ELSE
  X=3;
```

The SELECT statement :

----------Format- 1----------

```
SELECT ( identifier) ;
    WHEN( value1) statement;
    WHEN( value2) statement;
OTHERWISE statement ;
END;
```

----------Format - 2 ----------

```
SELECT ;
    WHEN( cond1) statement;
    WHEN( cond2) statement;
    OTHERWISE statement ;
END;
```

The Comparison and Logical operators in the Assignment statement

e.g. A=B=C;

A=B>C;

A= B>C & D>E;

A=B>C | D>E;

Conditions and On-units

During the execution of a PL/I program , a number of conditions could arise causing a program to interrupt.It may be the detection of an unexpected error or of an occurrence that is expected but at an unpredictable time.

Some conditions may be raised during file handling :

- ENDFILE (filename) - ENDPAGE(filename)
- RECORD(filename) - TRANSMIT(filename)

They will be discussed during file handling only.

Some conditions may be raised during arithmetic operations :

- CONVERSION(default is enabled)

e.g. DCL X BIT(4);

X= ‘10A1’;

- SIZE (default is disabled)

e.g. DCL A FIXED DEC(4);

DCL B FIXED DEC(5) INIT(12345) ;

A=B;

- FIXEDOVERFLOW(for fixed-point variables, default is enabled)

e.g. DCL A FIXED DEC(5);
DCL B FIXED DEC(5);
DCL C FIXED DEC(5);
A=99999; B=88888;
C=A * B;

- OVERFLOW (for floating-point variables, default is enabled)

e.g. A = 55E71; B = 23E11;
C=A*B;

- UNDERFLOW (for floating-point variables, default is enabled)

e.g. A = 55E-71;

B = 23E-11;

C=A*B;

- ZERODIVIDE(default is enabled)

e.g. A=10;

B=0;

C=A/B;

- CHECK (used for debugging, default is disabled)

```
(CHECK):
AVERAGE:PROC OPTIONS(MAIN);
:
GET LIST(A,B,C);
AVG=(A+B)/2;
PUT LIST('THE AVERAGE IS', AVG);
END AVERAGE;
---- (CHECK(A,B)):
--- (CHECK):AVG=(A+B)/2;
```

The status of Conditions :

Conditions are enabled or disabled through a condition prefix , which is the name of one or more conditions separated by commas, enclosed in parentheses, and prefixed to a statement by a colon. The word NO preceding the condition name indicates that the condition is to be disabled.

E.g. (NOFIXEDOVERFLOW):SUM=A+B+C;

(SIZE,NOFIXEDOVERFLOW):PROG1:PROC OPTIONS(MAIN);

(NOSIZE):Y=A*B/C;

/* An example of On - unit (SIZE) and pseudo-variable (SIZE):

```
 THIRD : PROCEDURE OPTIONS(MAIN);
  DCL NAME CHAR(10) VARYING ,
   I FIXED BIN(15),
   A  CHAR(1),
   BITVAR BIT(8);
  GET LIST(NAME);
    DO I = 1 TO LENGTH(NAME);
           A = SUBSTR(NAME,I,1);
          UNSPEC(A) = UNSPEC(A) | '00100000'B;
```

```
        PUT SKIP LIST (UNSPEC(A));
        SUBSTR(NAME,I,1) = A;
    END ;
PUT LIST (NAME);
END THIRD;
```

Changing the action taken :

ON statement is used to specify what action will be taken if the condition arises.In the absence of a program-supplied ON statement for an enabled condition , standard system action will be taken. If , after having supplied your own on-unit for a condition , you wish to return to the standard system action for that on-unit, simply specify another ON statement with the keyword SYSTEM.For multiple lines of actions enclose the commands within BEGIN; …. END;

```
ON ERROR BEGIN;
  ON ERROR
        SYSTEM;
    :
    :
    SNAP ERROR
END;
```

Null action:

```
ON ENDPAGE ;
```

First time the error is encountered the first ON ERROR will be executed and after entering the BEGIN block , if it encounters another error then it will not execute the first one , but the second one i.e. system will handle the condition in the default way.

SNAP ERROR is used to execute the first ON ERROR even after the first time execution instead of second one.

Simulating conditions :

Can simulate execution of any specified ON condition statement by :

SIGNAL condition_name ;

If the specified condition is not enabled , then it'll be taken as null statement;

One of the uses of this statement is in program checkout to test the action of an on-unit and to determine that the correct program action is associated with the condition.

BUILT-IN FUNCTIONS FOR ON-UNITS

1. ONCODE : e.g. ON ERROR
   PUT LIST(ONCODE);
2. ONLOC : e.g. MAIN: PROC OPTIONS(MAIN);
   :
   SUBRT:PROC;
   :
   :
   END SUBRT;
   END MAIN;
3. ONCHAR : e.g. CHAR=ONCHAR;
4. ONSOURCE : e.g. SOURCE=ONSOURCE;

Iteration Constructs :

1. Label_name : DO WHILE(condition);
   :
   :
   END label_name;
2. DO UNTIL(condition);
   :
   :
   END;
3. DO I=n to m BY [+/-] x ;
4. DO M=1 TO 10,21 TO 30, 41 TO 50;

5. DO K= 1 TO 5, 8 TO 18 BY 2 , 50 TO 55 BY 5, 40 TO 44;

6. DO J=3,12,6,5,45

7. DO I=n BY x;

   :

   :

   END; /* some other condition must be mentioned to come out of the loop , otherwise it'll be an infinite loop or FIXEDOVERFLOW will occur.*/

8. DO K=1 TO 10 WHILE (A>10);

9. DO K=1 TO 10,11 WHILE (A>10);

10. DO K=1 TO 10,11 BY 0 WHILE (A>10);

11. DO K=1 TO 10 UNTIL (A>10);

12. DO NAME = ‘AAA’,’BBB’,’CCC’;

Nested DO-groups :

```
DO I=n TO M;
   DO J= x TO y;
   :
   :
   END;
END;
```

Leaving a loop :

1. By assigning a greater value than the highest value to the index variable
2. By GOTO statement which is not advised by structured programming.
3. By LEAVE statement .

E.g. LEAVE block_name ; -- comes out from the specified block

LEAVE ; -- comes out from the current block

Array handling :

e.g. DCL DAY_NAMES(7) CHAR(10);

```
DCL LIST(-2:3) FIXED BIN(15,0)
                    INIT(10,20,15,25,30,40);
DCL TABLE(6,2) FIXED DEC (5);
DCL TABLE(-2:3,4:5) FIXED DEC(5) INIT (0);
    -- only possible in Mainframe
DCL B(5,3) FIXED INIT ( (15)0);
DCL B(9,9) FIXED INIT ((81)-1);
DCL A(2,2) FIXED INIT(1,2,3,4);
DCL TABLE(10) CHAR(2) INIT((2)'A');
    -- only possible in Mainframe
DCL TABLE(10) CHAR(2) INIT((2) (2)'A');
```

DCL TABLE(10) CHAR(5) INIT((10) (1)'ABC');

DCL A(3) INIT(10,*,30);

Array assignments:

Scalar to Array :

DCL MONTHS(12) FIXED(4,1);

MONTHS=0;

MONHS(5)=10.5;

Array to Array :

DCL A(5,5) ,B(5,5);

A=B;

Array Expressions:

Prefix operators and Arrays:

1) A=-A;

Infix operators and Arrays:

1) B=A*5 ; --Every elements of A is multiplied by 5 and stored in B.

2) A=A * A(1,2); -- Multiplies every elements by the value of the element at (1,2) and stores the data into A.

3) C=A +B;

On conditions using String :

1. STRINGRANGE condition(default is disabled)
2. STRINGSIZE condition (default is enabled)

On conditions using Array :

1.SUBSCRIPTRANGE condition(default is disabled)

Array cross-sections:

B(*,3) ; B(5,*); B(*,*) i.e. B ;

e.g. S=SUM(B(*,3))

An example on Matrix multiplication :

```
C=0;
DO I=1 TO L;
   DO J=1 TO N;
        DO K=1 TO M;
              C(I,J)= A(I,K) * B(K,J) +C(I,J);
        END;
   END;
END;
```

An example on Matrix multiplication using cross sections :

```
C=0;
DO I=1 TO L;
   DO J=1 TO N;
        C(I,J)= SUM(A(I,*) * B(*,J));
   END;
END;
```

```
/* An example using array */
REVERS:PROC OPTIONS(MAIN);
        DCL STR1(10) FIXED (10)
   INIT(1,2,3,4,5,6,7,8,9,10);
        DCL STR2(10) FIXED (10) ;
        DCL I        FIXED (15);
        DCL J        FIXED (15) INIT(1);
         DO I =10 TO 1 BY -1 ;
                STR2(J)=STR1(I);
                J=J+1;
       END;
       PUT LIST(STR2);
END REVERS;
```

```
/* An example to sort data stored in an array */
SORT1 : PROC OPTIONS (MAIN);
   DCL ARRAY1(10) FIXED (10);
   DCL I  FIXED(2), J  FIXED(2), TEMP FIXED(2);
  PUT LIST ('ENTER 10 NUMBERS');
  DO I = 1 TO 10;
       PUT SKIP  LIST ( 'ENTER NO. ',I);
       GET (ARRAY1(I));
  END;
   DO I = 1 TO 9;
       DO J = I+1 TO 10;
```

```
            IF (ARRAY1(I) > ARRAY1(J)  THEN
                  DO;
                     TEMP = ARRAY1(I);
                     ARRAY1(I) = ARRAY1(J);
                     ARRAY1(J) = TEMP;
                  END;
      END;
  END;
  DO I = 1 TO 10;
      PUT  (ARRAY1(I));
   END;
END SORT1;
```

```
/* An example to handle two arrays */
STARR:PROC OPTIONS(MAIN);
  DCL     1 HISTORY (2),
    2 NAME CHAR(20) ,
    2 GRADE FIXED(4,1);
 DCL  1 HONOR_STUDENT(2),
    2 NAME CHAR(20) ,
    2 GRADE FIXED(4,1);
 DCL I FIXED DEC(1) INIT(1);
 PUT LIST('ENTER DATA OF STUDENTS:');
```

```
DO I=1 TO 2;
        PUT SKIP LIST('ENTER NAME');
        GET LIST(HISTORY(I).NAME);
        PUT SKIP LIST('ENTER GRADE');
         GET LIST(HISTORY(I).GRADE);
END;
DO I=1 TO 2;
  IF HISTORY(I).GRADE>=92.5 THEN
  DO;
        HONOR_STUDENT(I).NAME
  =HISTORY(I).NAME;
        HONOR_STUDENT(I).GRADE =
                                  HISTORY(I).GRADE;
```

```
        END;
END;
DO I=1 TO 2;
        PUT SKIP LIST('NAME IS');
        PUT LIST(HONOR_STUDENT(I).NAME);
        PUT SKIP LIST('GRADE IS');
        PUT LIST(HONOR_STUDENT(I).GRADE);
END;

END STARR;
```

Controlled storage with array bounds determined dynamically :

```
CONTR : PROC OPTIONS(MAIN);
        DCL ( A(*,*),B(*,*),C(*,*)) CONTROLLED;
        GET LIST(I,J); ALLOCATE A(I,J),B(I,J);
        GET LIST(A,B); CALL ADDAR(A,B,C);
        PUT LIST(C);
  ADDAR : PROC(R,S,T);
        DCL (R(*,*),S(*,*) ,T(*)) CONTROLLED;
        ALLOCATE T(LBOUND(R,1) :
  HBOUND(R,1));
L1:     DO K=LBOUND(R,1) TO HBOUND(R,1);
        T(K)=0;
```

```
L2 :            DO I=LBOUND(R,2) TO HBOUND(R,2);
                T(K)=R(K,J) + S(K,J) + T(K);
                END L2;
        END L1;
                FREE R,S;
    END ADDAR;
  END CONTR;

/* Note : C can be passed as an argument although it
   has not yet been allocated. */
```

Array manipulation and built-in functions :

The DIM built-in

The LBOUND built-in

The HBOUND built-in

The SUM built-in

The PROD built-in

I/O operations and Arrays :

e.g. DCL AMOUNT(5) FIXED DEC(3);

GET LIST(AMOUNT);

e.g. DCL TABLE(6,2) FIXED DEC(3);

GET LIST (TABLE);

The Repetitive specification of a data item :

e.g.DCL DAY_NAMES(7) CHAR(10);

GET LIST((DAY_NAMES(I) DO I=1 TO 7));

e.g.DCL TABLE(6,2) FIXED DEC (5);

GET LIST (((TABLE(I,J) DO I=1 TO 6) DO J=1 TO 2));

e.g. GET LIST((A(I) DO I=1TO 5),(B(J),C(J) DO J=1 TO 3));

Structures :

```
DCL 1 EMP_DET,
            2 EMP_NO  FIXED DEC(5) ,
            2 EMP_NAME  CHAR(20),
            2 EMP_SALARY,
                  3 BASIC     FIXED (9,2),
                  3 INDIA_ALLOW FIXED(9,2),
                  3 MEDICAL(12),
                        4 MED_MONTH  FIXED(9,2);
```

The statement could be written in a continuous string also.

Array of Structures :

```
DCL 1 EMP_DET(100),
            2 EMP_NO  FIXED DEC(5) ,
            2 EMP_NAME  CHAR(20),
            2 EMP_SALARY,
                  3 BASIC     FIXED (9,2),
                  3 INDIA_ALLOW  FIXED(9,2),
                  3 MEDICAL(12),
                        4 MED_MONTH  FIXED(9,2);
```

If attribute are to be explicitly declared in a structure, they may only be specified for elementary items.

e.g. DCL EMP_DET1 CHAR(100);

EMP_DET1=EMP_DET; ---- ERROR

e.g. DCL EMP_DET1 LIKE EMP_DET;

--it'll copy the structure of EMP_DET to EMP_DET1.

e.g. EMP_DET1=EMP_DET , BY NAME ;

-- it'll only copy the elements having the same names.

e.g. EMP_DET1=' ';

-- it'll assign the same space to all character elements.

Overlay defining:

e.g. DCL EMP_DET2(100) DEFINED EMP_DET;

Built-in Functions for structures :

STRING function : It concatenates all the elements in an array or a structure into a single character or bit-string element .Thus, if it is desired to concatenate a number of elementary items found in a structure or array , it would be easier to code the STRING function than to code the concatenation operation a number of times. STRING may also be used as a pseudo-variable.

E.g.

```
DCL 1 EMP_DET,
        2 EMP_NO CHAR(2)  INIT('11'),
        2 EMP_NAME  CHAR(20)  INIT(' JOAN K.
                                HUGHES'),
                2 EMP_ADD CHAR(30) INIT('JOHN WILEY
                                & SONS ,ENGLAND');
DCL ITEM  CHAR(45);
ITEM=EMP_DET;                   --illegal
ITEM=STRING(EMP_DET);
EMP_DET=ITEM;
STRING(EMP_DET)=ITEM;
```

Pictures :

The pictures are used for the following purposes :

1. To treat character-strings as arithmetic quantities.
2. To treat arithmetic quantities as character-strings.
3. To edit data
4. To validate data

The general form of pictures for editing and validating is : PICTURE ‘picture specification characters’

There are two types of pictures :

1. Decimal pictures
2. Characters-string pictures

Decimal pictures : The base and scale are implicitly DECIMAL and FIXED respectively.

e.g DCL A PICTURE ‘9999V99’;

DCL B PICTURE ‘(4)9V9(2);

DCL C PICTURE ‘S9999V99;

DCL D PIC ‘999S’;

Editing data for printed output :

Using Z , . , B, /, $, S, +, -, *,CR, DR AND , .

E.g. DCL A PIC’****$S9,999V.99CR’;

A=1234.54;

PUT LIST(A); ------ The output is : *1234.54

Character-string pictures :

The symbols used are : A, X, 9 .

E.g. DCL NAME PIC ’(20)A’;

DCL ADDRESS PIC ‘(20)X;

DCL PINCODE PIC ‘999999’;

Pictures in Stream I/O :

Some editing of data in stream I/O is automatic.In *list-directed* and *data-directed* output of arithmetic coded data , leading zeros will be suppressed and minus sign and decimal point inserted if necessary.

Full editing capability may be achieved in *list-directed* output by simply assigning data items to identifiers that have the PICTURE attribute and then issuing a PUT LIST on those identifiers.In *edit-directed* I/O, full editing of data may be implemented by using a format item called P-format , containing any character allowed in PICTURE and used to describe the characteristics of external data.

E.g. DCL VALUE FIXED DEC(8,2);

DCL EDIT_DATA PIC'$$$$.$$$V.99CR';

VALUE=A*B;

P’picture specification’ :

e.g. GET FILE(SYSIN) EDIT(A,B,C,D) (COL(1) ,
P’ZZZ9’, P’99V99’,P’AA999’,P’(5)9);

e.g. DCL ASSETS FIXED DEC(11,2);
ASSETS = 45326985.76;
PUT EDIT (ASSETS) (P’$$$$,$$$,$$$V.99’);

The output is : b$45,326,985.76

e.g. ASSETS=2500.00;

PUT EDIT(ASSETS) (P'$ZZ,ZZZ,ZZZV.99');

The output is : $bbbbb2,500.00

e.g. DCL TODAY CHAR(6);

TODAY='080880';

PUT EDIT(TODAY)(P'99/99/99');

The output is 08/08/80

Procedures:

Sub-routine procedures :

1. Invoked by CALL statement
2. Separately compiled ,from the invoking procedures, called as ***external procedures*** .
3. The length of these procedures' names is limited to 7 or 8 characters.
4. A STOP or EXIT statement in both separately compiled procedure and nested sub-routine procedure , abnormally terminates execution of that sub-routine and of the entire program associated with the procedure that invoked it.

5. Ideally an external sub-routine should not terminate the entire program , should return an error indicator to the calling program indicating whether or not an error has been detected during the execution of the sub-routine. It’s the function of MAIN program to decide what action is taken.

6. Arguments passed to a called procedure must be accepted by parameters of the calling procedure.

7. A built-in functions may be specified as arguments to other subprogram , that function is

may be executed before the subroutine is called and the value is being passed when there is no argument of that built-in or the name of the built-in is enclosed by parentheses or the built-in is passed to another built-in as an argument. Otherwise the name is passed to the sub-routine.

8. If arguments and parameters are of same type, then the parameters share the same memory area of that of parameters , called as ***call by reference***.

9. If arguments and parameters are of different types or constants , then the different memory area is allocated to the parameters , called as ***call by value***.

Dummy arguments :There are created in the following cases:

1) If an argument is a constant

2) If an argument is an expression involving operators.

3) If an argument is an expression in parentheses.

4) If an argument is a variable whose attributes are different from the attributes declared for the parameter in an entry name attribute specification appearing in the invoking block.

5) If an argument is itself a function reference containing arguments.

The ENTRY attribute :It directs the compiler(in Mainframe only) to generate the coding to convert one or more arguments to conform to the attribute of the corresponding parameters .Declare the sub-routine within the main procedure as:

DCL SUBRT ENTRY(types of parameters);

After conversion , the values are stored in dummy arguments in different locations and names are automatically assigned by compiler, which can not be accessed by programmers.Any manipulation done by the called procedure only effects those dummy arguments and hence original arguments remain unchanged.

```
/* An example of Sub-routine using ENTRY( CALL BY
   VALUE)*/
SUBPR : PROC OPTIONS(MAIN);
 DCL A FIXED BINARY(15) INIT(10),
    B FIXED BINARY (15) INIT(15),
    C FIXED BINARY(15);
    DCL ADDP ENTRY ( FIXED DEC(15), FIXED DEC(15), );
     CALL ADDP(A,B,C);
   PUT LIST(C);
  ADDP : PROC(A,B,C);
    DCL A FIXED DEC(15),
    B FIXED DEC(15),
    C FIXED BIN(15);
    C=A+B;
 END ADDP;
 END SUBPR;
```

The External attribute:

Specifies a name to be known to other procedures containing an EXTERNAL declaration of the same name(must be limited to 6 characters).

```
PROG: PROC OPTIONS(MAIN);
   DCL ARRAY(200) FIXED EXTERNAL;
   DCL SUM FIXED(7) EXTERNAL;
   GET LIST(ARRAY);
   CALL ADDSUM;
   PUT LIST('RESULT IS',SUM);
```

```
    ADDSUM : PROC;
    DCL ARRAY(200) FIXED EXTERNAL;
    DCL SUM FIXED(7) EXTERNAL;
    SUM=0;
    DO K=1 TO 200
        SUM=SUM+ARRAY(K);
    END;
END ADDSUM;

END PROG;
```

```
/* An example of Sub-routine using builtin */
TIMED: PROC OPTIONS(MAIN);
   DCL TIME BUILTIN;
   DCL TSTR              CHAR(8) ;
  CALL CTIME ((TIME),TSTR);
   PUT LIST (TSTR);

CTIME:PROC(T,X) ;
        DCL T CHAR(9);
        DCL X CHAR(8);
         DCL A CHAR(2);
```

```
    DCL B CHAR(2);
     DCL C CHAR(2);
    A=SUBSTR(T,1,2);
     B=SUBSTR(T,3,2);
     C=SUBSTR(T,5,2);
     X=A || ':' || B || ':' ||C;
  END CTIME;
END TIMED;
```

Function Procedures:

1. A function is a procedure that returns a single value to the invoking procedure.
2. Invoked in the same manner as PL/I built-in functions are referenced.
3. Should not be invoked by CALL statement.
4. The RETURN statement is used to terminate a function and a single value to the point of invocation.

```
CALC:PROCEDURE(A,B,C);
      RETURN(A+B+C);
END CALC;
```

5. By default , if the function name begins with the letters A to H or O to Z , then the result will be DECIMAL FLOAT(6).

   Names starting with I to N return a result with the attributes FIXED BINARY (15).

   D=CALC(A,B,C);

   To return data of some other type , use RETURNS keyword , in both invoking and invoked procedures.

```
CALC:PROCEDURE(A,B,C) RETURNS(FIXED
   DECIMAL(7));
        RETURN(A+B+C);
   END CALC;
```

6. The RETURNS keyword , when specified in a PROCEDURE or ENTRY statement , is referred to as the RETURNS option.In the previous example , the invoking procedure must also specify that CALC is returning a FIXED DECIMAL value of the same precision because these attributes differ from the default.

The RETURNS attribute , specified in a DECLARE statement for an entry name , indicates the attributes of the value returned by that function.

```
MAIN: PROCEDURE OPTION(MAIN);
   DCL CALC ENTRY RETURNS (FIXED DEC(7));
   DCL SUM FIXED DEC(7);
   GET LIST (A,B,C);
   SUM=CALC(A,B,C);
   :
   :
END MAIN;
```

```
/* An example of Function */
SUBPR : PROC OPTIONS(MAIN);
 DCL A FIXED BINARY(15) INIT(10),
    B FIXED BINARY (15) INIT(15),
    C FIXED BINARY(15);
   C= ADDP(A,B);
   PUT LIST(C);
  ADDP : PROC(A,B) RETURNS(FIXED BIN(15));
    C=A+B;
    RETURN (C);
 END ADDP;
 END SUBPR;
```

```
/* An example of function using builtin */
TIMED: PROC OPTIONS(MAIN);
   DCL TIME BUILTIN;
   DCL TRES              CHAR(8) ;
  TRES= CTIME (TIME);
   PUT LIST (TRES);

CTIME:PROC(T,X) RETURNS(CHAR(8));
        DCL T CHAR(9);
        DCL X CHAR(8);
         DCL A CHAR(2);
          DCL B CHAR(2);
```

```
        DCL C CHAR(2);
        A=SUBSTR(T,1,2);
         B=SUBSTR(T,3,2);
         C=SUBSTR(T,5,2);
         X=A || ':' || B || ':' ||C;
         RETURN(X);
   END CTIME;
END TIMED;
```

Recursive procedures :

When a procedure invokes *itself* , its said to be *recursive*.

e.g. the computation of factorial where

n! = 1*2*3…….*n

```
----code in the calling procedure--------
N=4;
RESULT=CALC_FACT(N);

-----code in the called procedure ---------
CALC_FACT:PROCEDURE(M);
   K=1;
```

```
DO I=1 TO M;
        K=K*I;
    END;
    RETURN(K);
END CALC_FACT;
-----code using RECURSIVE option----------
CALC_FACT:PROCEDURE(M) RECURSIVE;
    K=M - 1;
    IF K=1 THEN
        I=M;
    ELSE
        I=M * CALC_FACT(K);
    RETURN(I);
END CALC_FACT;
```

File declarations :

1. Define the file via a DECLARE statement.

   -- DECLARE EMPLOYEE FILE(other attributes);

   Attributes associated with a file include the type of transmission (STREAM or RECORD) , the direction of transmission (INPUT ,OUTPUT , UPDATE in the case of disk files) and the physical environment of the file e.g. ENV / ENVIRONMENT to specify BLKSIZE or RECSIZE or record type(F,V,VB,FB) or device type.

---- DECLARE EMPLOYEE FILE INPUT STREAM
ENV (options);

2. File must be opened before communicating via OPEN statement and can also provide additional file attributes at open time.

   File attributes :

   1. FILE attribute
   2. INPUT/OUTPUT attribute
   3. STREAM attribute
   4. ENVIRONMENT attribute
   5. PRINT attribute

The OPEN statement :

Files are opened automatically , hence not required to give the command explicitly . But still used to specify the additional file attributes and/or options may be specified at open times.

```
OPEN FILE(FILE1), FILE(FILE2);
 OPEN FILE(FILE1) PAGESIZE(50);
OPEN FILE(FILE1) LINESIZE(121);
```

3. Process information in the file via READ or WRITE or REWRITE or GET or PUT statements.

Default of pre-defined files :

When not mentioned in GET or PUT statement , SYSIN is for the standard input or SYSPRINT for the output file.

E.g. GET LIST (A,B,C); --- is equivalent to

GET FILE(SYSIN) LIST (A,B,C):

4. Close the file via CLOSE statement.

CLOSE FILE (EMPLOYEE);

In the initial design of PL/I , stream I/O was intended for scientific applications. Now-a-days stream I/O is used in both scientific and commercial programming applications , although the majority of commercial applications use record I/O . Stream I/O is most often used for a terminal or printer files , although it could also be used for tape or disk unblocked records in consecutive files.

Stream I/O :

All input and output data items are in the form of stream of characters. In input stream , characters are converted to the internal attributes of the identifiers specified in the GET statement. On output , coded arithmetic data items are automatically converted back to character form before the output takes place.

There are three types :

1. List-directed I/O
2. Edit-directed I/O
3. Data-directed I/O

Edit-directed I/O:

1. The format of the data items must be mentioned.
2. General format :

GET EDIT (data list)(format list);

PUT EDIT (data list)(format list);

e.g. GET EDIT (EMP_NO , EMP_NAME, BASIC, INDIA_ALLOW, MED_MONTH) (COLUMN(1), F(5) , A(20),F (9,2), F(9,2), F(9,2));

e.g. GET EDIT (EMP_NO , EMP_NAME, BASIC, INDIA_ALLOW, MED_MONTH) (R(RFMT));

:

RFMT: FORMAT(COLUMN(1), F(5) , A(20),F (9,2), F(9,2), F(9,2));

Rules :

a) All data list items have corresponding format items

b) If there are more format than data items , there is a return to the beginning of the format list.

E.g. GET EDIT(A,B,C) (F(4),F(5));

c) The data list item need not have the same width specification as the corresponding format items.

E.g. DCL NAME CHAR(15);

GET EDIT (NAME) (A(20));

d) I/O continues until the data list is exhausted.

e) Input data items may be pseudo-variables

e.g. DCL NAME CHAR(20);

GET EDIT (SUBSTR(NAME,5,10)) (A(20));

f) Data list items may be names of data aggregates.

E.g. DCL TABLE(10) FLOAT DEC(6);

GET EDIT(TABLE) (COLUMN(1),F(6,2));

Equivalent to :

E.g. GET EDIT ((TABLE (I) DO I=1 TO 10))
(COLUMN(1),F(6,2));

E.g.DCL TABLE(2,5) FLOAT DEC(6);

PUT EDIT (TABLE) (F(10));

Equivalent to:

e.g. GET EDIT (((TABLE(I,J) DO J=1 TO 5)
DO I=1 TO 2)) (F10));

Equivalent to :

```
DO I=1 TO 2;
        DO J=1 TO 5;
                GET EDIT (TABLE(I,J)) (F10));
        END;
END;
```

g) Output data items may be built-in functions

h) Output data items may be PL/I constant

i) Data items may consists of Element Expression

j) More than one data list and corresponding format list may be specified

e.g. GET EDIT (data list1) (format list1) (data list2) (format list2) …;

3. A file name could also be specified :

GET FILE(filename) EDIT (data list)(format list);

PUT FILE(filename) EDIT (data list)(format list);

Data-directed I/O:

Data-directed Input:

Each item in the input is in the form of an assignment that specifies both the value and the variable to which it is to be assigned.

E.g. GET DATA (list of items); -- its optional and maxm 320 items can be mentioned.

A=12.3, B=23, C='ABCDEF',D='10101'B;

GET DATA (C,B,A,D);

GET DATA (A,B,C,D,E); --- E is not altered by the input operation.

GET DATA (A,B); --- error as C,D are in the input stream but not in the data list. And raises the NAME condition.

```
ON NAME (SYSIN)
BEGIN;
:
:
END;
```

GET DATA; --the names in the stream may be any names known at the pint of the GET statement. A data list in the GET statement is optional ,because the semi-colon determines the number if items to be obtained from the stream.

If the data includes the name of an array , subscripted references to the array may appear in the stream, although subscripted names cannot appear in the data list. The entire array need not appear in the stream , only those elements that actually appear in the stream will be assigned .

E.g. DCL TABLE(10) FIXED (5,2);

GET DATA (TABLE);

the input stream consists of the following assignment statements:

TABLE(3)=34.34; TABLE(10)=12.12;

Data-directed output :

Each data item is placed in the stream in the form of assignment statements separated by blanks. The last item output by each PUT statement followed by a semicolon.Fixed point binary and floating point binary data appear in fixed-point decimal format.

```
DCL A FIXED DEC(5) INIT(0);
DCL B FIXED DEC(5) INIT(0);
DCL C FIXED BIN(5) INIT(170);
PUT DATA(A,B,C);  PUT PAGE DATA(A,B,C);
PUT SKIP(3) DATA(A,B,C);   PUT LINE(5)
                    DATA (A,B,C);
```

The output to the default file SYSPRINT in the format :

A=0 B=0 C=170;

If the output is to other than a printer , one blank is placed between each assignment statement.

The data list may be an element , array , a subscripted name, structure or a repetitive specification involving any of these elements or further repetitive specification.

PUT DATA;

PUT PAGE DATA (‘SALES REPORT’); --error

GET FILE (filename) DATA(A,B,C);

Data to be entered as:

: A=……., B=…… , C=…….

PUT FILE(filename) DATA(A,B,C);

The COUNT Built-in function:

DCL INFILE FILE INPUT STREAM;

GET FILE (INFILE) DATA;

I=COUNT(INFILE); /* no of elements is stored */

Output data item may be Built-in function:

PUT EDIT (DATE)(P'99/99/99');

Example:

```
ON ENDFILE (DATA)
  MORE_RECORDS=NO;
 MORE_RECORDS=YES;
 READ FILE(DATA) INTO(DATA_AREA);
 DO WHILE (MORE_RECORDS);
       WRITE FILE(PRINT_FILE) FROM (DATA_AREA);
       READ FILE(DATA) INTO(DATA_AREA);
  END;
```

Conditions and On-Units :

The ERROR condition is raised as a result of the standard system action and terminate the program and return control to the O.S.

The ON statements are used to specify action to be taken when a specified condition causes a program to interrupt.

E.g. ON condition on-unit;

In absence of any of these On-Units specified in the program, system will raise ERROR condition.The following conditions are by default enabled , can not be disabled.

1.ENDFILE(filename) 2. ENDPAGE(filename)

3. RECORD(filename) 4.TRANSMIT(filename)

5. CONVERSION 6. SIZE

ON ENDFILE : This condition is raised during a GET or READ operation.

ON ENDPAGE :This condition is raised when a PUT statement results in an attempt to start a new line beyond the limit specified for PAGESIZE(specified in OPEN statement).

ON KEY condition : Raised during operations on keyed records in any of the following cases:

1. The keyed record can not be found for a READ or REWRITE statement.

2. An attempt is made to add a duplicate key by a WRITE statement.

3. The key has not been correctly specified.

4. No space is available to add the keyed record.

E.g. ON KEY(FILE1)

BEGIN;

FLAG=YES;

```
    CODE=ONCODE;
        PUT SKIP EDIT('KEY ERROR FOR
        FILE1 , ONCODE IS',CODE) (A);
        END;
  :
  :
  READ FILE(FILE1) INTO(DATA_AREA)
  KEY(PARTNO);
  IF FLAG THEN
      CALL ERR-ROUTINE;
  ELSE
    CALL UPDATE_ROUTINE;
```

ON UNDEFINEDFILE condition : Raised whenever an attempt to open a file is unsuccessful.

1. A conflict in attributes exists.
2. Attributes are incomplete.

ON RECORD condition : Raised during a READ,WRITE , REWRITE operation. The input operations it is raised by either of the following :

1. The size of the record on the data set is greater than the size of the variable into which the record is to be read (for F,V,U format).

2. The size of the record is less than the size of the variable (for F format).

For a WRITE or REWRITE operations:

1. When the size of the variable from which the record is to be written is greater than the maximum size specified for the record (F,V,U formats).

2. When the size of the variable is less than the size specified for the record (F format).

ON TRANSMIT condition :This condition is raised due to any hardware malfunction . It indicates that the data transmitted is incorrect.

Built-In Functions :

ONKEY : It extracts the value of the key for the record that caused an I/O condition to be raised.

E.g.
```
DCL KEY_ERRCHAR(9) VARYING;
    DCL ONKEY BUILTIN;
    ON KEY(FILE1)
        BEGIN;
            KEY_ERR=ONKEY;
            :
            :
    END;
```

ONFILE : It determines the name of the file for which an I/O or CONVERSION condition was raised and returns that name to point of invocation.

E.g.
```
DCL NAM CHAR(3) VARYING;
DCL ONFILE BUILTIN;
ON KEY(FILE2)
        BEGIN;
                NAME=ONFILE;
                :
                :
        END;
```

Record I/O :

1. Declare the file by DECLARE statement :

```
e.g. DCL DATA FILE INPUT RECORD ENV
                (F RECSIZE (80) );
     DCL PRINT_FILE OUTPUT RECORD  ENV
                ( F RECSIZE (80) );
     DCL DATA_AREA       CHAR(80);
```

2. The keyword RECORD specifies the type of I/O .
3. The record I/O statements that are used to communicate with files include READ, WRITE, REWRITE and DELETE.

Carriage control in Record I/O :

1. The program counter is automatically incremented as and when a WRITE command is used.
2. When program counter reaches the maximum line per page , the program gives the command to skip the page to the printer.
3. To accomplish the carriage control for record output , append an extra character to the beginning of each record.
4. A keyword must be added to the ENVIRONMET section of the file declaration to notify PL/I that

these carriage control characters are being used in the program and that the I/O routines are to interpret the first character of each record accordingly.

5. Two different sets are CTLASA or CTL360 .
6. If CTLASA is used , the carriage control operation is performed before the printing , in case of CTL360 , after printing.
7. Generally CTL360 characters are faster than CTLASA.

Example :

```
LINECNT: PROC OPTIONS(MAIN);
   DCL DATAIN FILE INPUT RECORD ENV(F
                                    RECSIZE(80));
   DCL  FILE INPUT RECORD ENV
                          (F RECSIZE(80));
   DCL PRINT_FILE FILE OUTPUT RECORD ENV
                     (F RECSIZE(81) CTLASA );
   DCL DATA_AREA    CHAR(80);
   DCL LCTR         FIXED(3);
   DCL MORE_RECORDS    BIT(1);
```

```
DCL NO          BIT(1);
DCL PRINT_AREA  CHAR(81);
ON ENDFILE(DATAIN)
     MORE_RECORDS=NO;
OPEN FILE(DATAIN), FILE(PRINT_FILE);
LCTR=55;
READ FILE(DATAIN) INTO(DATA_AREA);
DO WHILE (MORE_RECORDS);
     LCTR=LCTR+1;
     IF LCTR>55 THEN
          DO;
```

```
            PRINT_AREA='1' || DATA_AREA;
                    LCTR=0;
            END;
    ELSE
            PRINT_AREA=' ' ||DATA_ARE;
    WRITE FILE(PRINT_FILE)  FROM
(PRINT_AREA);
    READ FILE(DATAIN) INTO(DATA_AREA);
END;
```

File Organization :

1. Types of file access are ,SEQUENTIAL and DIRECT.
2. Types of file organization are , CONSECUTIVE (sequential) , INDEXED(indexed sequential) , REGIONAL(direct or random) , VSAM(ESDS, KSDS, RRDS).
3. Different file attributes :
    a) FILE b) EXTERNAL/INTERNAL
    c) STREAM/RECORD
    d) INPUT/ OUTPUT/ UPDATE

```
/*AN EXAMPLE TO CREATE AN INDEXED FILE FOR
  STUDENT DATA */

WRprogram : PROC OPTIONS(MAIN);
DCL STUD FILE OUTPUT INDEXED RECORD;
DCL  1 STUD_DATA,
       2 STUD_ROLL            FIXED BINARY(15),
       2 STUD_NAME            CHAR(20),
       2 STUD_MARKS           FIXED BIN(15);

DCL CONT CHAR(1) INIT('Y');
```

```
DO WHILE (CONT='Y');
   PUT SKIP LIST ('ENTER DATA FOR STUDENT');
   PUT SKIP LIST ('ENTER ROLL FOR STUDENT');
   GET LIST (STUD_ROLL);
   PUT SKIP LIST ('ENTER NAME FOR STUDENT');
      GET LIST(STUD_NAME);
   PUT SKIP LIST ('ENTER MARKS FOR STUDENT');
   GET LIST (STUD_MARKS);
   WRITE FILE(STUD) FROM (STUD_DATA)
   KEYFROM(STUD_ROLL) ;
```

```
   PUT SKIP LIST ('WANT TO CONT(Y/N)?');
   GET SKIP LIST (CONT);
END;
CLOSE FILE(STUD);
END WRprogram;
```

```
/* MENU PROGRAM TO CALL DIFF. PROGRAMS */
 MENUprogram: PROC OPTIONS(MAIN);
 DCL RDprogram EXT ENTRY;
  DCL WRprogram EXT ENTRY;
  DCL MODprogram EXT ENTRY;
 DCL CH CHAR(1);
 PUT SKIP LIST ('          MENU          ');
 PUT SKIP LIST ('      1. TO ADD A RECORD');
 PUT SKIP LIST ('      2. TO READ A RECORD');
 PUT SKIP LIST ('      3. TO MODIFY A RECORD');
 PUT SKIP LIST ('      4. EXIT                ');
```

```
PUT SKIP(2) LIST ('ENTER YOUR CHOICE(1-4):');
 GET LIST(CH);
 IF CH='1'   THEN
          CALL WRprogram;
 ELSE
    IF CH='2'  THEN
         CALL RDprogram;
 ELSE
     IF CH='3'   THEN
            CALL MODprogram;
```

```
        ELSE
              IF CH='4'     THEN
                        RETURN;
              ELSE
                        PUT LIST ('WRONG CHOICE');
END MENUprogram;
```

```
/* TO CREATE A FILE FOR STUDENT DATA */
WRprogram : PROC ;
        DCL STUD FILE OUTPUT RECORD;
         DCL  1 STUD_DATA,
                2 STUD_ROLL FIXED BINARY(15),
                2 STUD_NAME CHAR(20),
                2 STUD_MARKS FIXED BIN(15);
/*OPEN FILE STUD OUTPUT;*/
DCL CONT         CHAR(1) INIT('Y');
DO WHILE (CONT='Y');
  PUT SKIP LIST ('ENTER DATA FOR STUDENT');
  PUT SKIP LIST ('ENTER ROLL FOR STUDENT');
```

```
   GET LIST (STUD_ROLL);
   PUT SKIP LIST ('ENTER NAME FOR STUDENT');
    GET LIST(STUD_NAME);
   PUT SKIP LIST ('ENTER MARKS FOR STUDENT');
   GET LIST (STUD_MARKS);
   WRITE FILE(STUD) FROM (STUD_DATA)  ;

   PUT SKIP LIST ('WANT TO CONT(Y/N)?');
   GET SKIP LIST (CONT);
END;
CLOSE FILE(STUD);
END WRprogram;
```

```
/* TO MODIFY FILE FOR STUDENT DATA */
MODprogram : PROC ;
      DCL STUD FILE UPDATE RECORD;
      DCL  1 STUD_DATA,
             2 STUD_ROLL FIXED BINARY(15),
              2 STUD_NAME CHAR(20),
              2 STUD_MARKS FIXED BIN(15);
  /*OPEN FILE STUD INPUT;*/
      DCL RNO FIXED BINARY(15);
      DCL CONT CHAR(1) INIT('Y');
```

```
ON ENDFILE(STUD)
     BEGIN;
          PUT SKIP LIST('RECORD NOT FOUND');
          GO TO OUT;
    END;
DO WHILE (CONT='Y');
     PUT SKIP LIST('ENTER ROLL NO TO BE
                                        MODIFIED');
      GET LIST(RNO);
      READ FILE(STUD) INTO (STUD_DATA)  ;
     SEARCH : IF STUD_ROLL=RNO  THEN
          BEGIN;
```

```
PUT LIST ('ENTER NEW DATA');
PUT SKIP LIST ('ENTER NEW ROLL FOR
                                    STUDENT');
GET LIST (STUD_ROLL);
PUT SKIP LIST ('ENTER NEW NAME FOR
                                    STUDENT');
GET LIST(STUD_NAME);
PUT SKIP LIST ('ENTER NEW MARKS FOR
                                    STUDENT');
GET LIST (STUD_MARKS);
REWRITE FILE(STUD) FROM (STUD_DATA)  ;
GO TO OUT;
```

```
            END;
        ELSE
            BEGIN;
            READ FILE(STUD) INTO (STUD_DATA)  ;
            GOTO SEARCH;
        END;
      OUT:    PUT SKIP LIST('WANT TO CONT(Y/N):');
        GET LIST(CONT);
    END;
   CLOSE FILE(STUD);
END MODprogram;
```

```
/* TO READ FILE FOR STUDENT DATA */
RDprogram : PROC ;
DCL STUD  FILE INPUT RECORD;
DCL  1 STUD_DATA,
       2 STUD_ROLL FIXED BINARY(15),
       2 STUD_NAME CHAR(20),
       2 STUD_MARKS FIXED BIN(15);
  /*OPEN FILE STUD INPUT;*/
  DCL FLAG CHAR(1) INIT('Y');
  ON ENDFILE(STUD)
        FLAG='N';
        PUT SKIP LIST('END OF FILE');
```

```
READ FILE(STUD) INTO (STUD_DATA)  ;
   DO WHILE (FLAG='Y');
        PUT SKIP LIST(STUD_ROLL);
        PUT SKIP LIST(STUD_NAME);
        PUT SKIP LIST(STUD_MARKS);
        READ FILE(STDATA) INTO (STUD_DATA)  ;
    END;
   CLOSE FILE(STUD);
END RDprogram;
```