Chapter 4:-

Q.1. List and explain the PL features used in implementation of aspects of compilation.

→ There are four PL features -

1. data types
2. data structures
3. scope rules
4. control structure

i] Data types -

- A data is a specification of -
  i. legal values for variables of the type
  ii. legal operations on the legal values of the type.

- The following tasks are involved -
  i. checking legality of an operation for types of operands.
  ii. use type conversion operations.
  iii. use appropriate instr^ sequence of the target machine.

- Ex - consider program segment

```
i : integer;
a,b : real
a := b + 1
```

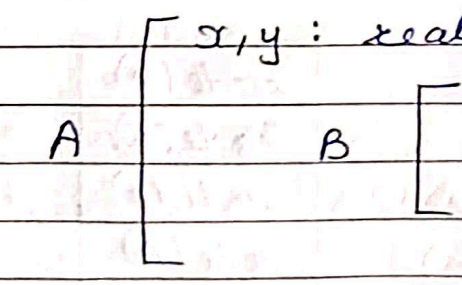
instr^ generated for program segment:

```
CONV_R    AREG, I
ADD_R     AREG, B
MOVEM     AREG, A
```

2] data structure
- A PL permits the declaration and use of data structure like arrays, stacks, records, lists, etc.
- compiler must develop a memory mappings to access memory word(s) allocated to element.
- A user defined type requires mapping of a different kind.
- &...

Ex - program example(input, output);
type
employee = record
name: array[1...10] of character;
age: character;
id: integer;
end
weekday = (mon, tue, wed, thu, fri, sat);
var
info: array[1...500] of employee;
today: weekday;
i, j: integer;
begin
today := mon;
info[i].id := j;
if today = tue then...
end

3] Scope rule -
It determine the accessibility of variables declared in diff block of a program.
Ex -



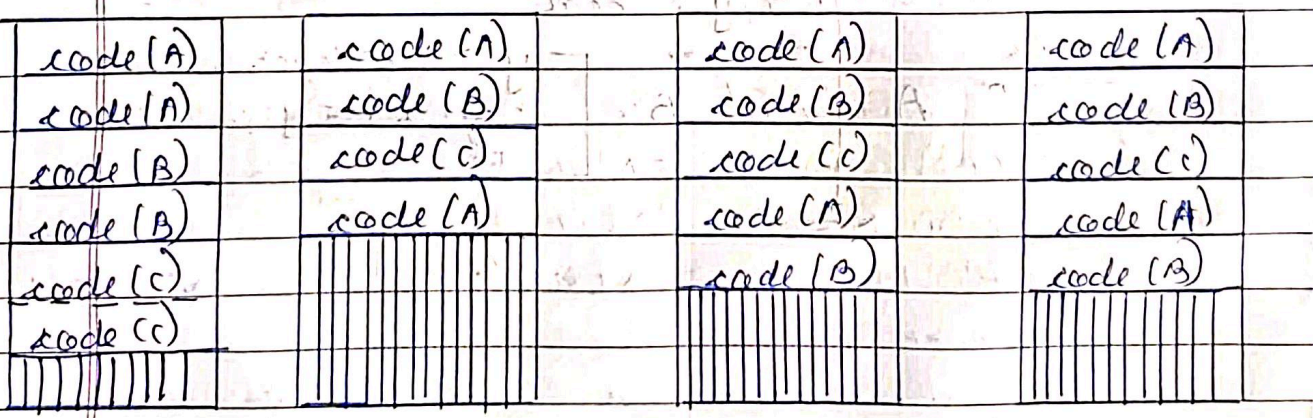
4) control structure-
The control structure of a language is the collection of a language features for altering the flow of control during the execution of a program.

Ex - for i := 1 to 100 do
begin
lab1: if i=10 then...
end;

Q.2. What is memory binding? Explain different types of memory allocation techniques.

→ Memory binding -
Memory binding is an association between the memory address attribute of a data item and the address of a memory area.

Diff types of memory allocation techniques -
① static memory allocation -
In static memory allocation, memory is allocated to a variable before execution of program begins.



static memory allocation
dynamic memory allocation when only program unit A
the situation after A calls B
The situation after B returns to A & A calls C

② Dynamic memory allocation -
- In dynamic memory allocation, memory bindings are established & destroyed during the execution of a program.
- Dynamic memory allocation can be implemented using stack & heaps.
  - Automatic dynamic allocation is implemented using stack.
  - Program controlled allocation is implemented using heap.
- Advantages are -
  - Recursion can be implemented easily.
  - Dynamic allocation can ~~be~~ support data structure whose sizes are determined dynamically.
    Ex - Array

Q.3. How memory allocation done in block structured languages ?

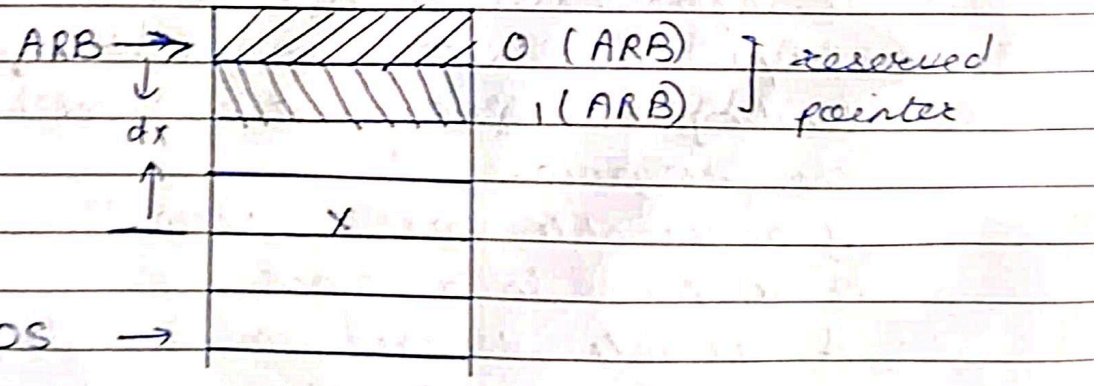
→ i) Scope rules -
- A data declaration using a name namei creates a variable vari. (namei, vari)
- Variable vari is visible at a place in the program if some binding (namei, vari) is effective at that place.
- Scope rules determine which of these binding is effective at a specific place in the program.
- If a variable vari is created with the name namei in a block b
- A variable declared in block b is called a local variable of block b.

- A variable of enclosing block that is accessible within block b is called a non-local variable.
- Example -

```
A [ x, y, z : integer
    B [ g : real;
        C [ h, z : real; ]
    ]
    D [ i, j : integer; ]
]
```

Blocks	accessible variable	
	local	non-local
A	x_A, y_A, z_A	--
B	g_B	x_A, y_A, z_A
C	h_C, z_C	x_A, y_A, g_B
D	i_D, j_D	x_A, y_A, z_A

2) Memory allocation & access -
- Automatic memory allocation can be implemented using the extended stack model.
- Each record in stack has two reserved pointers.
- Each stack record accommodates the variables for one activation of a block, called an activation record.



Q.4. Discuss major issues in code generation for expressions.

→ The major issues in code generation for expression are:

- ① Determination of an evaluation order for the operators in an expression.
  - top & down and bottom up parse.

② selection of instr^ to be used in target code.
- operand descriptor

③ use of registers and handling of partial result.
- registe description.

Q.5. What are triples, quadruples & indirect triples? Explain in detail with ex.

a.6. Write a note on -

a) Expression tree.

→ - It is an abstract syntax tree which depicts the structure of an expression.
- It is used to determine the best evaluation order.
- Example -

```
MOVER   AREG, A
 ADD    AREG, B
MOVEM   AREG, TEMP_1
MOVER   AREG, C
 ADD    AREG, D
MOVEM   AREG, TEMP_2
MOVER   AREG, TEMP_1
 DIV    AREG, TEMP_2
MOVER   AREG, C
 ADD    AREG, D
MOVEM   AREG, TEMP_1
MOVER   AREG, A
 ADD    AREG, B
 DIV    AREG, TEMP_1
```

- The first step associates a register requirement label with each node in the expression.
- The second step analyses the RR labels of the child nodes of a node to determine the order in which they should be evaluated.

b) parameter passing mechanism

c) Pure & impure interpreter

→

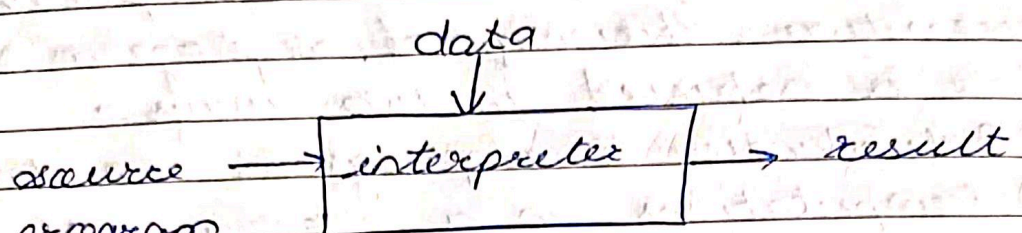


fig:- Pure interpreter.

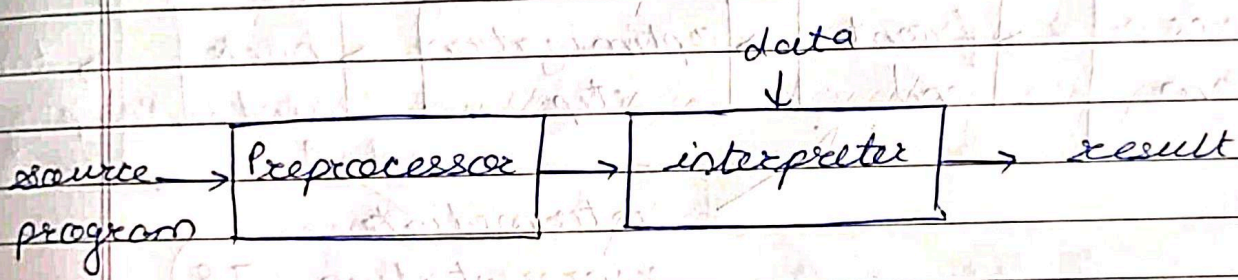


fig:- impure interpreter

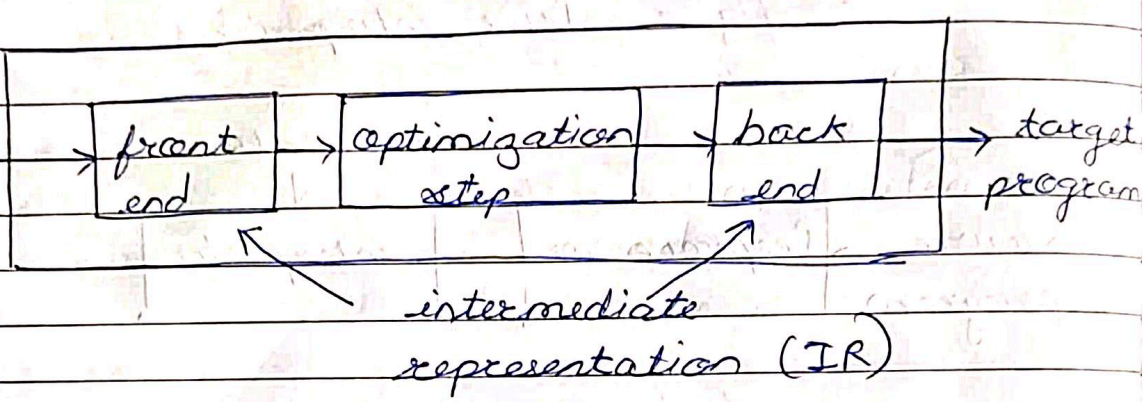
Impure interpreter -

- It performs some preliminary processing of the source program to reduce analysis overhead during interpretation.
- The preprocessor converts the program to an intermediate representation.
- IC can be analyzed more efficiently than the source form of the program.
- Ex - intermediate code for a+b*c using positfix notation look like the following.

S#17	S#4	S#29	*	+

Q.7. What is code optimization? ~~Ed~~ List & explain different optimizing transformations with ex.

→ code optimization -
code optimization aims at improving the execution efficiency of a program.
This is achieved in two ways :-
① Redundancies in a program are eliminated.
② Computations in a program are rearranged to make it executes efficiently.



Optimizing transform used in compiler -
① compile time evaluation
② elimination of common subexpression
③ dead code elimination.
④ frequency red^
⑤ strength red^

1] compile time evaluation -
Execution efficiency can be improved by performing certain actions specified in a program during compilation itself.

eg - an assignment a := 3.14157/2 can be replaced by a := 1.570785

2] elimination of common subexpression:-
Common subexpressions are occurrences of expression yielding the same value.

Eg.

```
a := b * c ;          after eliminating      t := b * c ;
----                  the common             a := t ;
x := b * c + 5.2 ;    subexpression  ⟶       ---
                                             x := t + 5.2 ;
```

3] dead code elimination:-
code which can be omitted from a program without affecting its result is called a dead code.
Ex - an assignment x := <exp> constitute dead code if the value assigned to it is not used in the program, no matter how control flows after executing this assignment.

4] frequency red^ -
Execution time of program can be reduced by moving code from a part of a program which is executed very frequently to another part of a program ~~which~~ which is executed fewer times.
Ex -

```
for I := 1 to 100 do                    x := 25 * a ;
begin                                   for I := 1 to 100 do
    z := i ;              ⟹            begin
    x := 25 * a ;                         z := i ;
    y := x + z ;                          x := 25 * a ;
end ;                                     y := x + z ;
                                        end ;
```

5] Strength red^n

Strength red^n optimisation replaces the occurrences of a time consuming operation by an occurrence of a faster operation.

Ex - Replacement of a multiplication by an addition.

```
for I := 1 to 100 do
begin
    ---
    k := i * 5;
    ----
end;
```

⇒

```
itemp := 5;
for I := 1 to 100 do
begin
    ----
    k := itemp
    ----
    itemp := itemp + 5;
end;
```

3.10. Write the algo. for evaluation order for operators.

→ Algo - Evaluation order for operators

1. Visit all nodes in an expression tree in post order.
   For each node n_i
   (a) if n_i is a leaf node then
       if n_i is the left operand of its parent then
           RR(n_i) := 1;
       else RR(n_i) := 0;

   (b) if n_i is not a leaf node then
       if RR(l_child n_i) NE RR(r_child n_i) then
           RR(n_i) := max(RR(r_child n_i), RR(l_child n_i));
       else RR(n_i) := RR(l_child n_i) + 1;

2. Perform the procedure call evaluation_order (root) which prints a postfix form of the source string in which operators appear in the desired evaluation order.

```
Procedure evaluation-order (node);
if node is not a leaf node then
    if RR(l-child node) ≤ RR(r-child node)
        evaluation-order(r-child node);
        evaluation_order (l-child node);
else
        evaluation_order (l-child node);
        evaluation_order (r-child node);
print node;
end evaluation-order;
```

Q.11. Explain compilation of control structure with suitable ex.

Q.9. Given exp is a*b+c*d*(e+f)+c*d

1) construct operand description for above exp.
2) Write parsing and code generation action.
3) construct exp tree and write the target code.

→ Exp tree for a*b+c*d*(e+f)+c*d

```
MOVER   AREG, A
MULT    AREG, B
MOVEM   AREG, TEMP-1
MOVER   AREG, C
MULT    AREG, D
MOVEM   AREG, TEMP_2
MOVER   AREG, E
ADD     AREG, F
```

```
MULT    AREG , TEMP_2
ADD     AREG , TEMP_1
MOVEM   AREG , TEMP_1
MOVER   AREG , C
MULT    AREG , D
ADD     AREG , TEMP_1
```