PMCD Solve QP

BE-C
Roll No-423012

DCOER
Page No.: 1

PMCD (Insem 2016 Question Paper)

Q1

a) What are different storage allocation strategies ? Explain.

→ Based on the subdivision of runtime following different storage allocation strategis are used in each of the three data aree

1. Static Allocation
2. Stack Allocation
3. Heap Allocation

1. Static Allocation -
   - The storage for all data objects is allocated at compile time.
   - It is done at Compile time
   - Variables are fixed in size
   - If we choose more static data space than require, we may waste remaining space.
   - Compiler allocates space only for global variables at compile time

2. Stack Allocation -
   - The run time storage is managed as stack.
   - Data object can be created dynamically
   - Used to allocate local variable
   - Register allocation works best for stack

allocation objects.
- Memory allocation & freeing are partially predictable.

3. Heap allocation -
- Storage is allocated & deallocated at run time as per need.
- Used to allocate dynamic objects.
- Never allocate to register
- Allocation & release are unpredictable

b. Define
1. lexme
2. Tokens

1. lexme -
It is smallest logical units of a program such as A, B, 10, if, + etc.

2. Tokens -
classes or category of similar lexemes such as identifier number, Constant, Operator etc.

for eg.

```
Mult_three (float num1 ,float num2 ,float num
{
    float ans;
    ans = num1 * num2 * num3;
    return (ans);
}
```

Token	lexeme
Keywords	return , float
Identifier	num1, num2, num3, ans
Delimeter	(, ;,) {, }
Operator	= , +

- Lexical analyzer must also pass additional information along with the tokens. These items of information are called attributes for tokens.
- Generally more than one lexeme matches with the pattern

c. What are symbol table ? Explain in breif different ways of organizing the symbol table.

→ - To keep track of scope & binding information symbol table is maintained.
- Symbol table is searched for every occurences of name. Name can be variable, Constants, procedures, tempories, label statement.

- If new name occure or new information is found symbol table is updated.

- Size of symbol table is selected large enough to handle the source program.

- Symbol Table mechanism / DS for organizing Symbol Table
  - linear list
  - Search Trees
  - Hash tables

• Linear List -

It is simple data structure & easy to implement. In this either one array or several arrays are used to store the name & related information. A

Available index is used to store name into the table & to search particular name from the table.

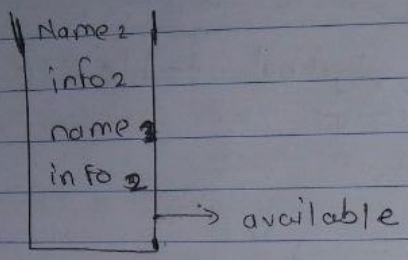


fig. linear DS

- Search Trees -

The property of binary search tree is used to add the records into this data structure. It is efficient approach for organization of symbol table.

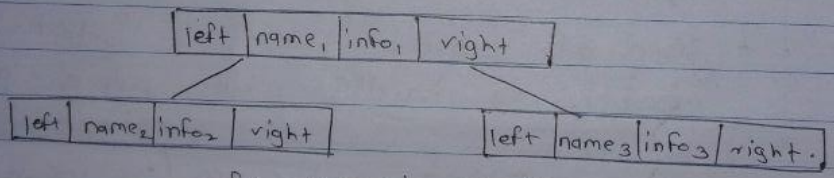


fig. Search Tree

- Hash Tables -

Hash table is a table of p pointer numbered from 0 to p-1 that point to the symbol table of a record within the symbol table

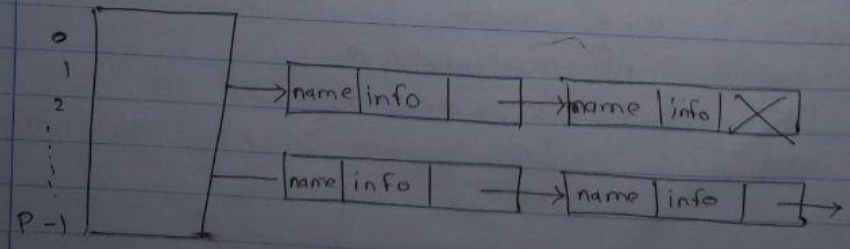


fig. Hash table of size p.

Q2

a) Explain briefly about input buffering in reading source program for finding tokens.

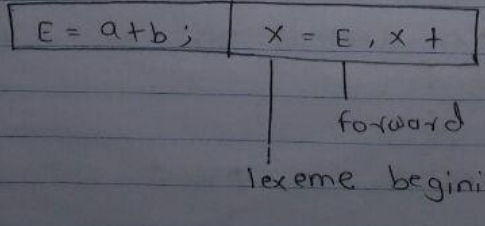
→ 1. lexical analyzer reads the source program character by character from the secondary storage but it is costly.

2. It also reduces the amount of time in the lexical analyzer phase.

3. While processing input character a large amount of time is consumed in moving characters. So for that specialized buffering technique have been developed.

4. To maintain input buffer, lexical analyzer uses two pointers: a lexeme begining pointer & a forward pointer to keep track of the pointer of the input string scanned.

5) Input buffer operation :-



```
if (fp = = eof (buffer)
{
    reload buffer 2;
    fp : = fp + 1;
}
else if (fp = eof (buffer))
{
    reload buffer;
    fp = adress of buffer/;
}
  Else
  { fp = fp + 1;
  }
    fp = forward pointer.
```

b. Explain Garbage Collection techniques.

→ 1. The internal data is deallocated explicitely by the Compiler & user allocated data is deallocated either by user or the garbage Collector.

2. The technique need to implement dynamic allocation depends on how storage is deallocated.

3. If deallocation is implicit, then the run time

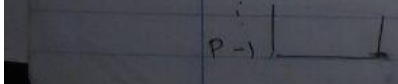


fig. Hash table of size p.

Support package is responsible for determining when a storage block is no longer needed.

Explicite Deallocation
- Explicite Allocation of fixed blocks
- Explicite Allocation of variable sized Blocks

Implicite Deallocation -
- Referance Counts.
- Marketing Techniques

Q3 6

a) Compare Top down & bottom up parser.

Top Down	Bottom up
1) Parse is starting with the root node	Parse is starting with leaf node
2) Tow-down parsing a general form of parser is called as recursive descent parsed	A special for of recursive descent parser that needs to backtracking is called predictive / Bottom up parser.
3. Easy to Construct manuall	Difficult to Construct

b. Explain type checking & type conversion.

→ Type checking -

- Any identifier must be declared before it is used.
- Type checker translation scheme synthesize the type of each expression from the types of its sub-expression

- The grammer given below generate prog. which are represented by a non-terminal P. A program consist of a sequence of declaration followed by a sequence of statement

P → D ; S
D → D ; D | TL
T → int | float | char
L → L [num] | * L | id
S → id := E | if (E) S ; | while (E) S | S1 ; S2
E → literal | num | id | E + E

Production Rule	Expression
D → TL	L.Type := T.type
T → int	T.Type = int
T → char	T.type = char
T → float	T.type = float
L → id	add type : (id.entry, L.type)

fig. Hash table

Type Conversion -

- If we consider expression like x + i where x is of type float & i is integer Since the representation of int & float is diff. with in comp & diff machine instr. are used for operation int & float

eg. ~~x + i~~

E → num	E.type = integer
E → num.num	E.type = real
E → id	E.type = lookup (id.entry)
E → E1 op E2	E.type = if E1.type = integer & E2.type = integer

c) check following grammer is LL (1) or not

→ X → YZ
Y → m | n | ϵ
Z → m

	m ~~First~~	n ~~Follow~~	$
X	X → YZ		
Y	Y → m	Y → n	Y → ϵ
Z	Z → m		

Q4

a) What is an ambiguous grammer ?

→ 1) Every ambiguous grammer failed to be LR & thus is not in any of the classes of grammer

2) We can construct the LR parser for an ambiguous grammer by adding new productions to the grammer i.

3) Using precedence & associativity to resolve parsing actions conflicts :

eg. $E \rightarrow E+E \mid E*E \mid (E) \mid id$

4) The above grammer is ambiguous as in this the associativity & precedence are not specified. The unambiguous grammer for the above grammer can be constructed by giving + a lower precedence than *, & makes both operator left associative.

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

b. Explain closure function for constructing SLR parsing table.

→ - If I is set of items of a grammer G then closure (I) is the items constructed from I by the two rules :

1. Initially, every items in I is added to closure (I).
2. If $A \to \alpha \cdot B\beta$ is in closure (I) & $B \to \gamma$ is production, then add the item $B \to \cdot\gamma$ to I, if it is not already there. Apply this rule until no more new items can be added to closure (I)

```
closure (I)
{
    repeat
    {   J = I;
        for each item in J (A → α.Bβ) &
        each production B → γ of G such that
        B → .γ is not in J.
        Add B → .γ to J
    }
    Until no
}
```

Q5

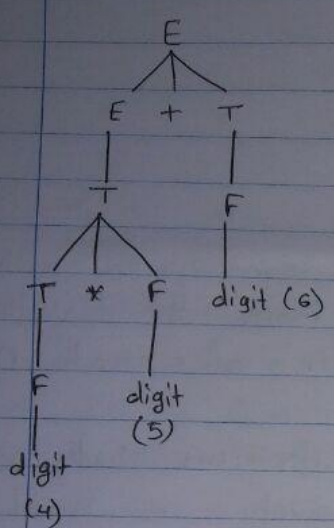
a) Explain the following terms

1) Synthesized Attributes.

- The value of a synthesized attribute at a node is computed from the values of attribute at the children of that node in the parse tree.
- A ~~tree~~ syntax directed definition that uses synthesized attribute exclusively is said to be an S-attributed definition.
- A parse tree for an S-attributed definition can always be annotated by evaluating the semantic rules for the attribute at each node by bottom up approach

2) Inherited Attribute -

- The inherited attribute are the attribute in which the value at a node is defined in term of attributes at the parent and sibling of that node.
- Inherited attribute are helpful for expressing the dependence of a programming language construct.



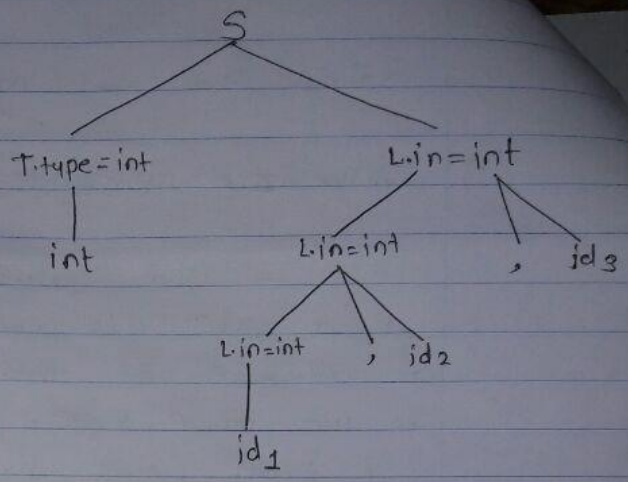


Fig. Synthesized Attribute

fig. Inherited attribute

b. Explain Advantage of intermidiate code.

→ ① Commonly used intermediate representation are -
   a) graphical representation
   b) Postfix Notation
   c) Three address Code.

② In the graphical representation a syntax tree & directed Acyclic graph can be created for the given expression

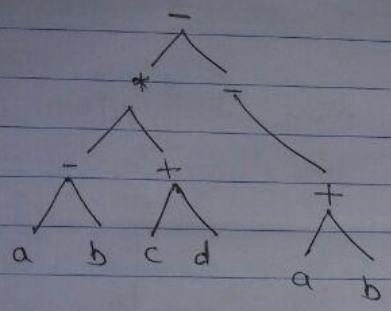


Fig. Syntax tree

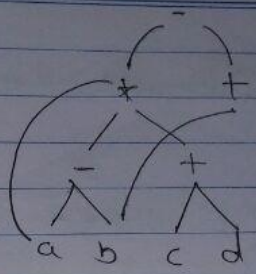


Fig. DAG.

③ From representation (i) we can write postfix notation as ab-cd+*ab+-

④ The three address code can be:

$t_1 = a - b$ $t_2 = c + d$ $t_3 = t_1 * t_2$
$t_4 = a + b$ $t_3 = t_3 + t_4$

C Generate three address code & quadruple for following $a = b * - c + b * - c$

→ Given -

$a = b * - c + b * - c$

t_1 = Unary minus
$t_2 = b * t_1$
$t_3 = b * t_1 = t_2$
$t_4 = t_2 * t_3$

- Quadruple Representation

	OP	Arg 1	Arg 2	Result
(0)	Unary -	c		t_1
(1)	*	b	t_1	t_2
(2)	*	b	t_1	t_3
(3)	+	t_2	t_3	t_4.

Q6

a) Explain L-attributed Definition.

→ ① It depends upon the attribute of the Symbol $X_1, X_2, X_3 \ldots X_{j-1}$ to the left of X_j

② It also depends upon the inherited attribute A.

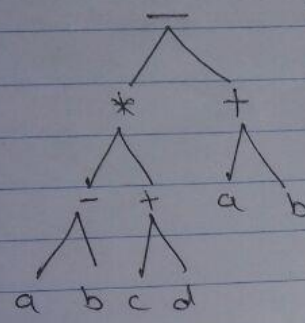
③ These are the definitions in which all inherited attribute are such that their value depend only on -

① inherited attribute of parent
② attribute of left siblings

Explain Syntax tree & DAG

Syntax ~~fite~~ tree -

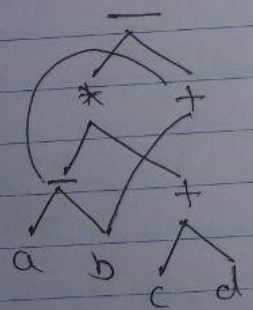
- In the graphical representation a syntax tree is users.



- In the syntax tree, operate & keywords appear an interior node that would be the parent of the leaves in the parse tree
- Syntax directed translation can be based on syntax trees as well as & parse trees.

DAG -

DAG is directed Acyclic Graph use for graphical representation.



- The node in a DAG has more than one parent but in syntax tree subtree are duplicated.
- In many application, nodes are implemented as records stored in an array.