Mohd. Manzar Iqbal
19212017
MCA - sem-IV

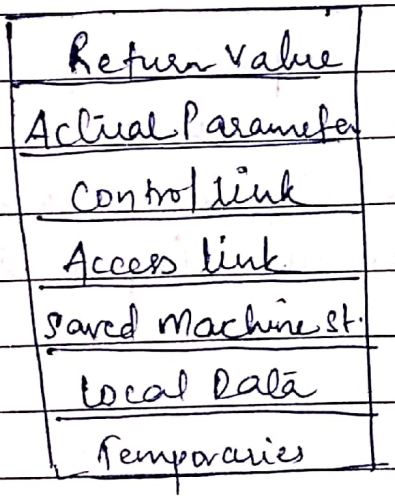
03/03/21

Date:
Page:

Mid-term Exam
MCA-623 compiler design

(Q.5) Activation Record:-

- Activation record is pushed into the ~~xxxxxx~~ stack when a procedure is called and it is popped when the control returns to the caller function.
- Activation record is used to manage the information needed by single execution of a procedure.
- When it is called (activation begins) then the procedure name will push on the stack and when it returns then it will be popped.



Diag of content of activation Records

Return Value :- It stores the return values.

Local Data :- Stores local data of the called procedure

Machine Status :- stores machine status such as register, program counter, etc. before the procedure is called.

Control Link :- stores the address of activation record of the called procedure

Access Link :- stores the information of data which is outside the local scope.

Actual Parameters : stores the actual parameters. i.e parameters which are used to send input to the called procedure.

Q.4 Predictive Parsing is a special form of recursive descent parsing, where no backtracking is required so this can predict which production to use to replace the input string

Algorithm :-

```
begin
  let X be the top stack symbol and the
  next input symbol;
  if X is a terminal or $ then
    if X = a then
      pop X from stack and remove a from
      input
    else ERROR()
```

```
else                    /* X is a non-terminal */
    if M[X, a] = X → Y1 Y2 ... Yk then
    begin
    POP X from the stack;
    push Yk ... Y2 Y1 onto the stack, Y1 on top
    end
else
    ERROR ();
end
until X = $             /* stack has emptied */
```

(Q.3)

$E \rightarrow TX$

$X \rightarrow +E \mid \epsilon$

$T \rightarrow (E) \mid int\ Y$

$Y \rightarrow *T \mid \epsilon$

The first can be computed from the grammer is the scale for calculating the first function

(1) for a production scale $x \rightarrow \epsilon$

first $(x) = \{\epsilon\}$

(2) for any terminal symbol 'a'

first $(a) = \{a\}$

(3) for a production scale $x \rightarrow y_1\ y_2\ y_3$

calculating first (x)

(1) If $\epsilon \notin$ first (y_1) then first (x) = first (y_1)

(2) If $\epsilon \in$ first (y_1) then First (x) =

$\{$ First $(y_1) - \epsilon\} \cup$ First $(y_2\ y_3)$

The grammer $E \rightarrow TX$

$X \rightarrow +E \mid \epsilon$

$T \rightarrow (E) \mid int\ y$

$Y = *T \mid \epsilon$

First $(() = \{($

First $()) = \{)\}$

First $(int) = \{int\}$

First $(+) = \{+\}$

First $(*) = \{*\}$

First $(T) = \{int, (\}$

First $(E) = \{int, (\}$

First $(X) = \{+, \epsilon\}$

First $(Y) = \{*, \epsilon\}$

$E \rightarrow TX$

$T \rightarrow (E) \mid int\ Y$

$X \rightarrow +E \mid \epsilon$

$Y \rightarrow *T \mid \epsilon$

Follow (+) = { int, (}
Follow (( ) = { int, (}
Follow (X) = { $,) }
Follow ( )) = { +,), $ }
Follow (int) = { *, +,), $ }

Follow (*) = { int, (}
Follow (E) = {), $ }
Follow (T) = { +,), $ }
Follow (Y) = { +,), $ }

(Q.1) **Tokens**:- In programming language keywords, constants, identifiers, strings, number, operators and punctuations symbols can be considered as tokens.

Pattern:- A pattern explains what can be a token and these patterns are defined by means of regular expressions.

Lexeme:- A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

tokens in the given program:-

int dividend, divisor, quotient, remainder;

operator → /, %, .

Lexeme:-

No. of token / lexeme :-

```
(5) int main() {
(9) int dividend, divisor, quotient, remainder
(5) printf ("Enter dividend");
(8) scanf ("%d", &dividend);
(5) printf ("Enter divisor :");
(8) scanf ("%d", &divisor);
(6) quotient = dividend / divisor;
(6) remainder = dividend % divisor;
(7) printf ("Quotient = %d", quotient);
(7) printf ("Remainder = %d", remainder);
(3) return 0;
(1) }
```

So total token = 5 + 9 + 5 + 8 + 6 + 6 + 7 + 7 + 5 + 8 + 3 + 1

= 70

So there are 70 tokens / lexeme in this c code.

(Q.2) :- **Left Recursion** :- A production of grammar is said to have left recursion if the leftmost variable of its RHS is as same as variable of its LHS. A grammar containing a production having left recursion is called as left recursive Grammar.