Vivekanand Education Society's Institute of Technology
Department of Computer Engineering

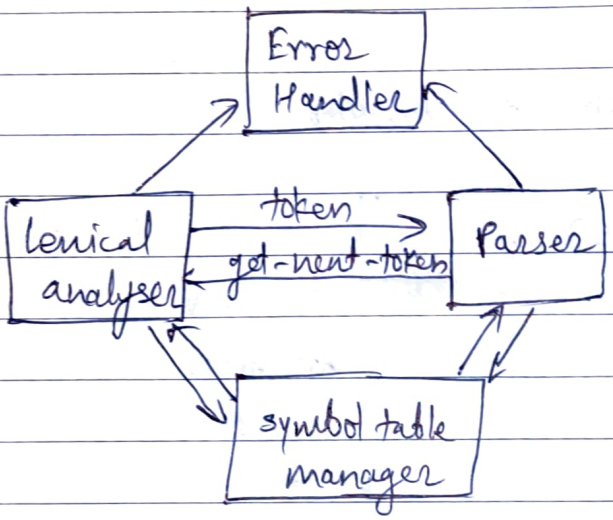
Subject: -SPCC

Class:- T.E. (D12) **Semester:- VI** **Div:- A**

Roll No: 21	Name: Amit V. Joshi		
Exp. No: 2	Title: Assignment 2		
DOP:		DOS:	08/04/2021
GRADE:	LAB OUTCOMES :	SIGNATURE:	

SPCC Assignment-2

Q1) As the first phase of compiler, lexical analyser is used to read input characters of the source program, group them into lexemes and produce as output a sequence of tokens for each lexeme in the source program. The stream of tokens is sent to parsers for syntax analysis. When lexical analyser discovers a lexeme constituting an identifier, it enters that lexeme into the symbol table. It is suggested in diagram below.



It also performs other ~~task~~ like stripping out comments and white spaces that is used to seperate tokens in the input. Other task is keeping line no in mind ~~table~~ to help user by showing line no with error.

Q2)

Token	Lexeme.	Token.	Lexeme.
Data type.	int	Greater than operator	>
identifier.	max	Ternary operator.	? :
Left parenthesis.	(	Delimiter	;
identifier.	x	Right Brace	}
Identifier	y		
Right parenthesis	)		
left brace	{		
Key word.	return.		

Q3) i) Synthesized attribute :-

It is an attribute of the non-terminal on the LHS of a production synthesized attributes represent information being passed up the parse tree. The attributes represent information being passed up the parse tree. The attribute can take value only from its children

Eg: A → BC, A should get value from B or C

ii) Inherited attribute :-

Attribute of non-terminal on RHS of a production is called inherited attribute. The attribute can take value either from parent or its sibling.

Eg: A → BC, B's attribute is dependant on A's or C's attribute.

S- attributed SDT.	L- attributed SDT.
- It only uses synthesized attributes.	It uses both synthesized and inherited attributes with an exception that inherited value can only be from left sibling only.
- Evaluated in bottom-up parsing	- Evaluated in depth first & left to right parsing.
- semantic actions placed in right most place of RHS	- Semantic actions placed anywhere.
- If S- attributed, then it is L- attributed also.	- If L- attributed, then not necessary that it is S- attributed.

Q4) Input Buffering

The lexical analyser scans i/p from left to right. It uses two pointers - Begin and forward.

Initially they are both at the first character of input string.

The forward pointer moves ahead to search end of lexeme

Eg. 

In this, as soon as it encounters blankspace, int is identified.

Next, the white space is ignored and both Begin & Fwd are set at next token.

Reading from secondary memory is costly therefore buffering techniques is used.

There are 2 types of buffer :-

i) One buffer scheme : Only one buffer is used

ii) Two buffer scheme : Two ~~diffe~~ buffers are used such that when one buffer ends, the next is scanned. Only drawback is that if length of lexeme is longer than buffer then it cannot be scanned completely.

Q5) $E \rightarrow E_1 + E_2$ { if (E_1.type == E_2.type) && (E_1.type = int) then E.type = int else error; }

/ $E_1 == E_2$ { if (E_1.type = E_2.type) && (E_1.type = int/boolean) then E.type = boolean else error; }

/ (E) { E.type = E_1.type }

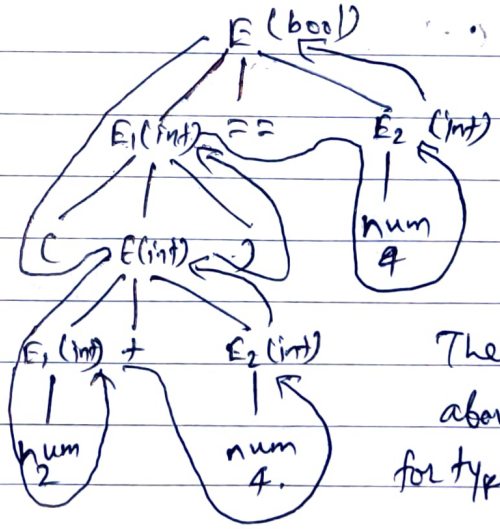
/ num { E.type = int }

/ True { E.type = boolean }

/ False { E.type = boolean }

Eg : (2+4) == 4

Parse tree :-



Therefore above en is correct for type checking.

Q6). i)

1) if (a<b) goto (3)
2. goto (11)
3. if (c<d) goto (5)
4. goto (8)
5. $T_1 = y+2$
6. $x = T_1$
7. goto (10)
8. $T_2 = y-2$
9. $x = T_2$
10. goto (1)
11.

ii) 1. if (a > b) goto 3
2. goto (8)
3. if (x <= y) goto (5)
4. goto (8)
5. $T_1 = b + c$
6. $A = T_1$
7. goto (10)
8. $T_2 = q * r$
9. $p = T_2$
10.

Q7). Grammer after removing left recursion :-

$S \rightarrow A$ | First (S) = {a} | Follow (S) = {$}
$A \rightarrow aBA'$ | First (A) = {a} | Follow (A) = {$}
$A' \rightarrow dA' / \epsilon$ | First (A') = {d, ε} | Follow (A') = {$}
$B \rightarrow bBC / f$ | First (B) = {b, f} | Follow (B) = {d, g, $}
$C \rightarrow g.$ | First (C) = {g} | Follow (C) = {d, g, $}

	a	b	d	f	g	$
S	S→A					
A	A→aBA'					
A'			A'→dA'			A'→ε
B		B→bBC		B→f		
C					C→g.	

Parser Table.