Report for final assignment

Subarna Khorshed

Ug02-48-18-010

Assignment 1 : Suppose, a given C source program has been scanned, filtered, lexically analyzed and tokenized as that were done in earlier sessions. In addition, line numbers have been assigned to the source code lines for generating proper error messages. As the first step to Syntax Analysis, we now perform detection of simple syntax errors like duplication of tokens except parentheses or braces, unbalanced braces or parentheses problem, unmatched 'else' problem, etc. Duplicate identifier declarations must also be detected with the help of the Symbol Table.

Code Screenshot:

```c
#include <stdio.h>
#include <stdlib.h>

/* Functions */
void check_comment (char) ; // checks for both types of comments, then passes on control to below comments
void block_comment () ;  //  handles block or multiline comments
void single_comment () ;  // handles single line comments

/* 2 file pointers - 1st is for the file in which we check for comments,
and 2nd is the file in which we copy the code after removing comments  */
FILE *fp , *fp2;

int main(void)
{
    char c;
    fp = fopen ("test.txt","r") ;  // open the first file in read mode
    fp2 = fopen ("mynewfile.txt","w") ;   // open the second file in write mode

    while((c=fgetc(fp))!=EOF)      // read the file character by character
        check_comment(c);  // check for each character if it seems like the beginning of a comment

     //  close both the files at the end of the program
    fclose(fp);
    fclose(fp2);

    return 0;
}


//handles both types of comments
void check_comment(char c)
{
    char d;
```

Assignment 5

Implementing the grammar in C

Code screenshot:


```c
                if(str[i]==temp[i])
                    return;
                else if(str[i]!=temp[i] && temp[i]>=65 && temp[i]<=90)
                    break;
            }
            break;
        }
    }
    if(temp[i]>=65 && temp[i]<=90)
        findter();
}

int main()
{
   FILE *f;

    for(i=0;i<10;i++)
        pro[i].n=0;

    f=fopen("in.txt","r");
    while(!feof(f))
    {
        fscanf(f,"%s",pro[n].lhs);
        if(n>0)
        {
            if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )
            {
                pro[n].lhs[0]='\0';
                fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);
                pro[n-1].n++;
                continue;
            }
        }
```

6.1: Find the FIRST and FOLLOW sets of each of the non-terminals .

Solve:

Assignment - 6

1. Find the FIRST and FOLLOW sets of each of the non-terminals.

Here $Y \rightarrow b$ and $Y \rightarrow \varepsilon$

So, we can write: $Y \rightarrow b \mid \varepsilon$

Same for Z

$Z \rightarrow cX$ and $Z \rightarrow \varepsilon$

So, $Z \rightarrow cX \mid \varepsilon$

So, we have,

$S \rightarrow aXd$, $X \rightarrow YZ$, $Y \rightarrow b \mid \varepsilon$, $Z \rightarrow cX$

Determining FIRST sets of each of the non-terminals.

$FIRST(S) = \{a\}$

$FIRST(X) = FIRST(Y)$
$= \{b, \varepsilon\}$

$FIRST(Y) = \{b, \varepsilon\}$

$FIRST(Z) = \{c, \varepsilon\}$

Determing FOLLOW sets of each of the non-terminals.

~~Follow~~

FOLLOW $(S) = \{\$\}$

FOLLOW $(X) = \{\$; d; \text{FOLLOW}(Z)\}$

$= \{\$, d, \text{FOLLOW}(X)\}$

$= \{\$, d\}$.

FOLLOW $(Y) = \{\$, \text{FIRST}(Z)\}$

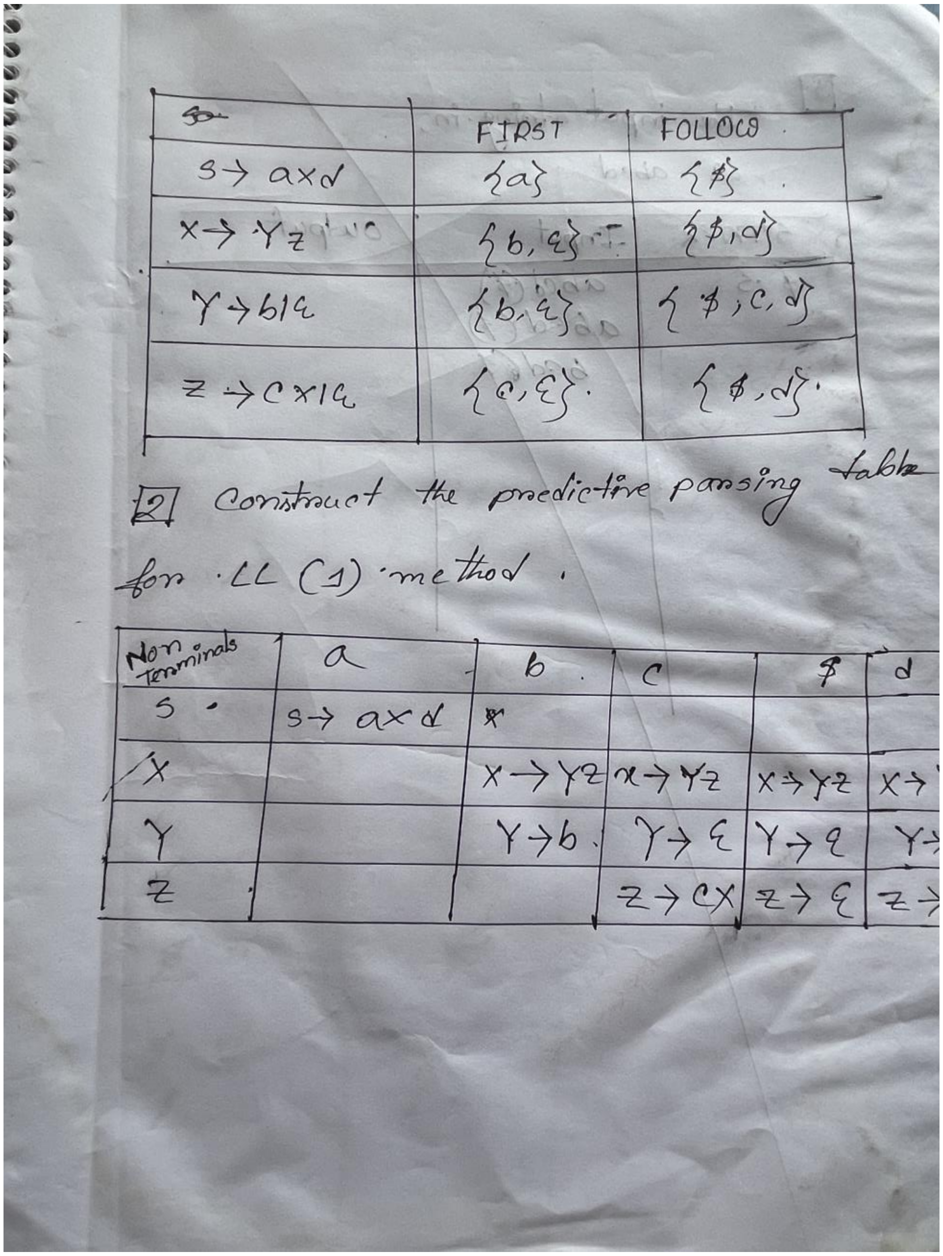
$= \{\$, c, \varepsilon\}$

$= \{\$, c, \text{FOLLOW}(X)\}$

$= \{\$, c, d\}$

FOLLOW $(Z) = \{\$, \text{FOLLOW}(X)$

$= \{\$; d\}$.



	FIRST	FOLLOW
$S \to axd$	$\{a\}$	$\{\$\}$
$X \to YZ$	$\{b, \varepsilon\}$	$\{\$, d\}$
$Y \to b \mid \varepsilon$	$\{b, \varepsilon\}$	$\{\$, c, d\}$
$Z \to cX \mid \varepsilon$	$\{c, \varepsilon\}$	$\{\$, d\}$

[2] Construct the predictive parsing table for LL (1) method.

Non Terminals	a	b	c	$	d
S	$S \to axd$				
X		$X \to YZ$	$X \to YZ$	$X \to YZ$	$X \to$
Y		$Y \to b$	$Y \to \varepsilon$	$Y \to \varepsilon$	$Y \to$
Z			$Z \to cX$	$Z \to \varepsilon$	$Z \to$

6.2: Construct the predictive parsing table for LL(1) method.

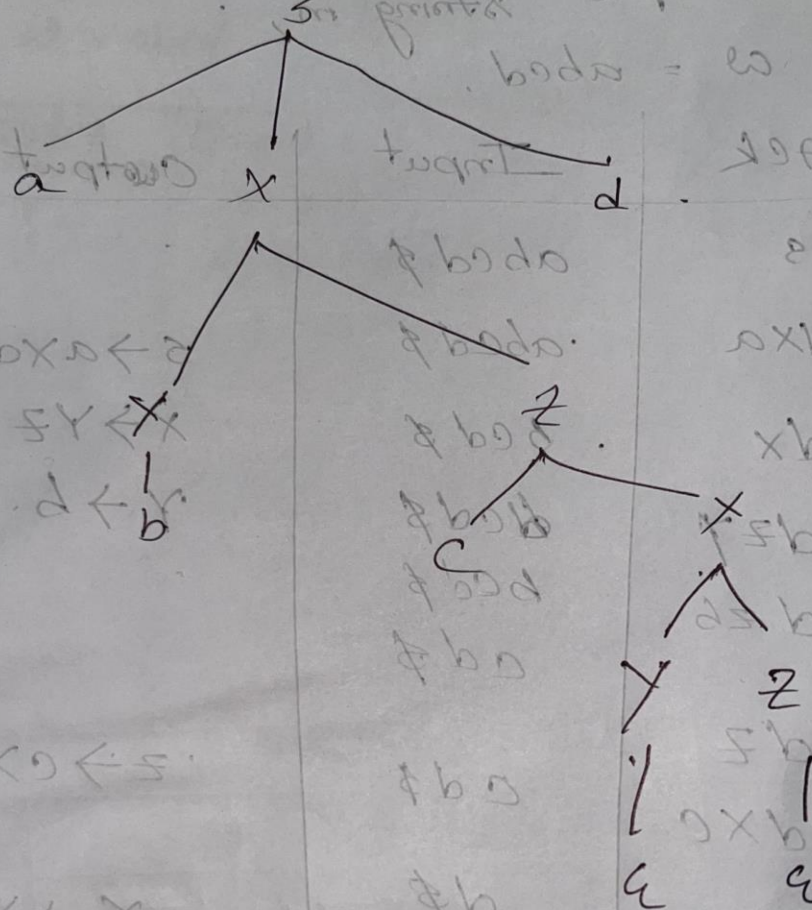
Solve:

3.

The input string is,
$w = abcd$.

STACK	Input	Output
$ S	abcd$	
$ dXa	abcd$	$S \to aXd$
$ dX	bcd$	$X \to YZ$
$ dZY	bcd$	$Y \to b$
$ dZb	bcd$	
	cd$	
$ dZ	cd$	$Z \to cX$
$ dXc	d$	
$ dX	d$	$X \to YZ$
$ dZY	d$	$Y \to \epsilon$
$ dZ	d$	$Z \to \epsilon$
$ d	$	
$		

Now the parse tree :—



3.

The input string in,

ω = abcd.

STACK	Input	Output
\$ S	abcd\$	
\$ dXa	abcd\$	$S \to aXd$
\$ dX	bcd\$	$X \to YZ$.
\$ dZY	bcd\$	$Y \to b$.
\$ dZb	bcd\$	
	cd\$	
\$ dZ	cd\$	$Z \to cX$
\$ dXc		
\$ dX	d\$	$X \to YZ$
	d\$	
\$ dZY	d\$	$Y \to \epsilon$
\$ dZ	d\$	$Z \to \epsilon$
\$ d		
\$	\$	

6.3: **Demonstrate the moves of the LL(1) parser on the given input.**

Solve of 6.3

Here the 1st L represents that the scanning of the Input will be done from Left to Right manner and the second L shows that in this parsing technique we are going to use Left most Derivation Tree. And finally, the 1 represents the number of look-ahead, which means how many symbols are you going to see when you want to make a decision.

Algorithm to construct LL(1) Parsing Table:

Step 1: First check for left recursion in the grammar, if there is left recursion in the grammar remove that and go to step 2.

Step 2: Calculate First() and Follow() for all non-terminals.

First(): If there is a variable, and from that variable, if we try to drive all the strings then the beginning Terminal Symbol is called the First.

Follow(): What is the Terminal Symbol which follows a variable in the process of derivation.

Step 3: For each production A –> α. (A tends to alpha)

Find First(α) and for each terminal in First(α), make entry A –> α in the table.

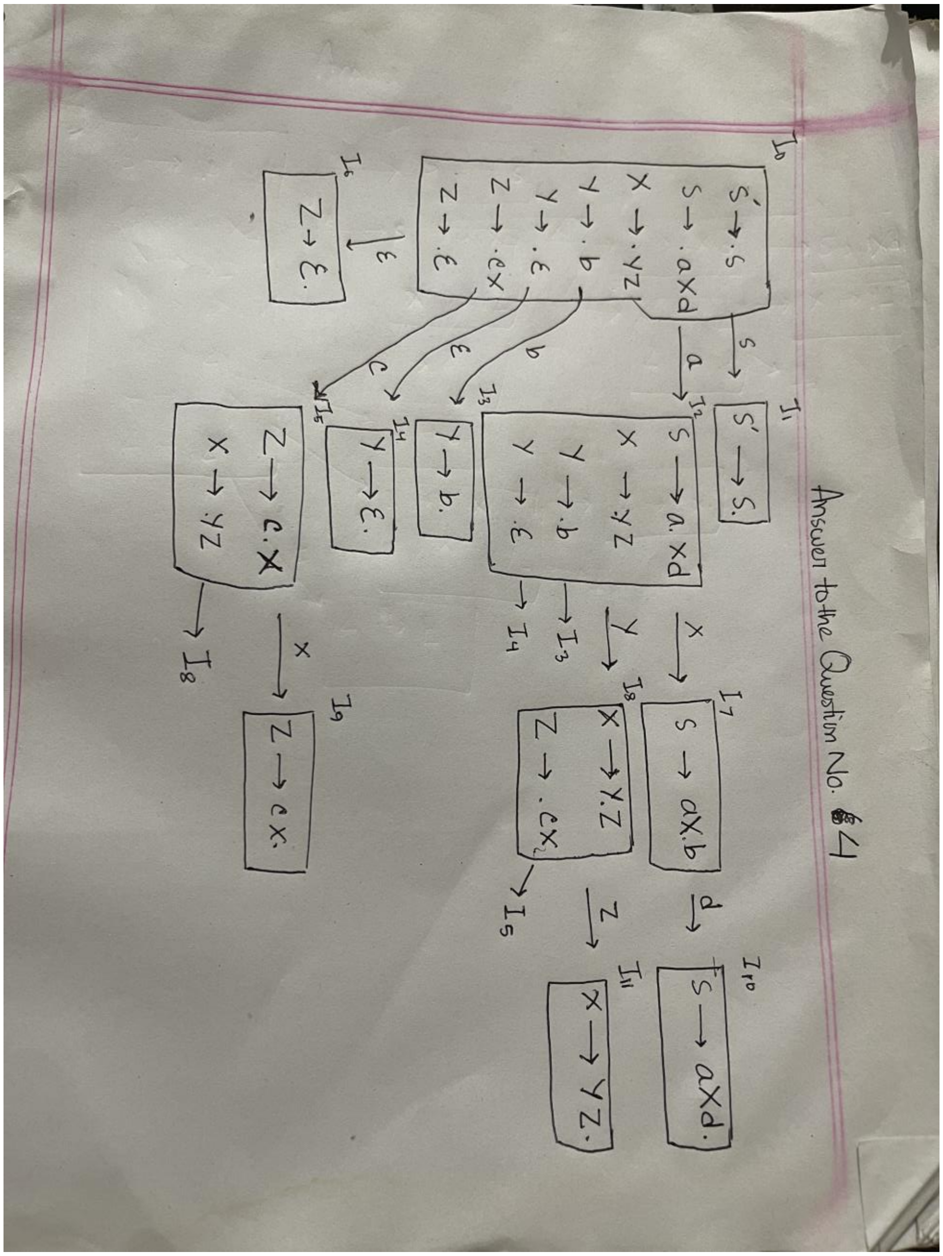
If First(α) contains ε (epsilon) as terminal than, find the Follow(A) and for each terminal in Follow(A), make entry A –> α in the table.

If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A –> α in the table for the $.

To construct the parsing table, we have two functions:

In the table, rows will contain the Non-Terminals and the column will contain the Terminal Symbols. All the Null Productions of the Grammars will go under the Follow elements and the remaining productions will lie under the elements of the First set.

6.4. Construct the LR(0) automaton for the grammar.



6.6. Demonstrate the moves of the LR(1) parser on the given input.

Solve:

LR (1) Parsing

Various steps involved in the LR (1) Parsing:

For the given input string write a context free grammar.

Check the ambiguity of the grammar.

Add Augment production in the given grammar.

Create Canonical collection of LR (0) items.

Draw a data flow diagram (DFA).

Construct a LR (1) parsing table.