FORMAN CHRISTIAN COLLEGE (A CHARTERED UNIVERSITY)

Compiler Construction - COMP 451 A

Spring 22

Class Project

Muhammad Sameed Gilani - 231488347

INTRODUCTION:

The Project was a continuation of lab 9. Where we had to design an LL(1) parser for the grammar,

A → BwA ---------------1

A → e -----------------2

B → CxB ---------------3

B → yC ----------------4

C → z -----------------5

An LL(1) parser is a predictive parser. It works top down, left to right and uses left most derivation. The 1 in the brackets is the look ahead symbols.

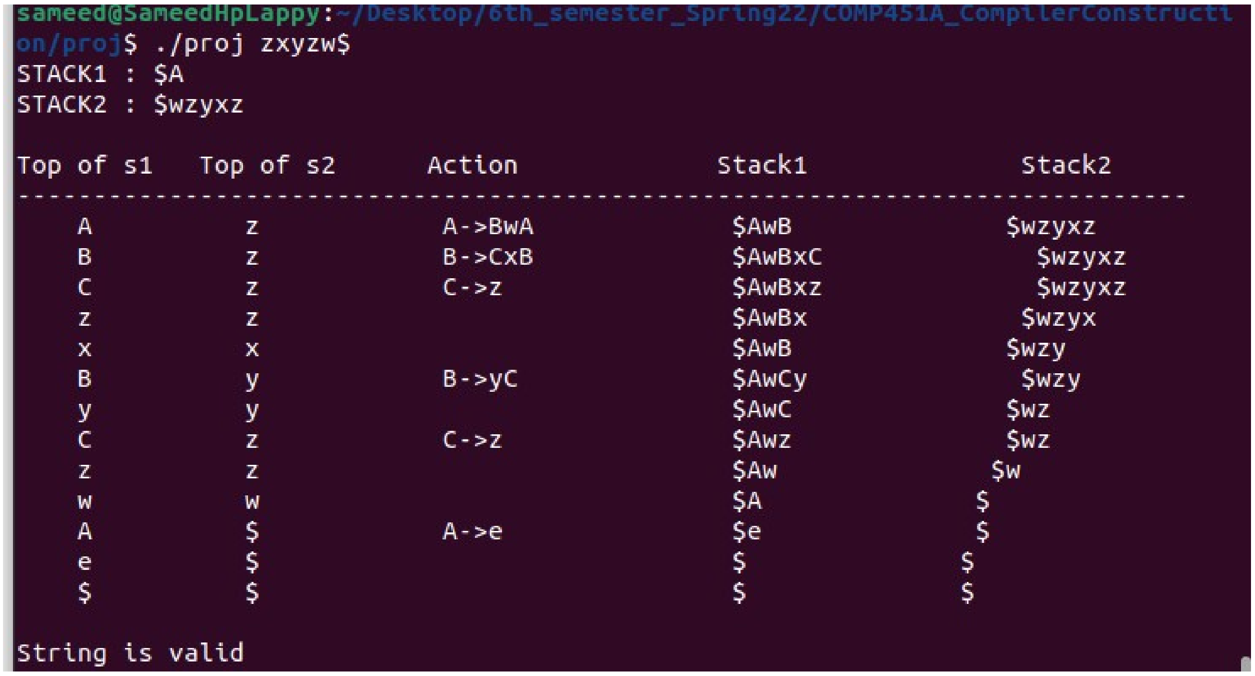
The given grammar does not have left recursion or common prefix problems.

We proceed with the given first/follow table and the parse table, to write a program for LL(1) parser for the given grammar.

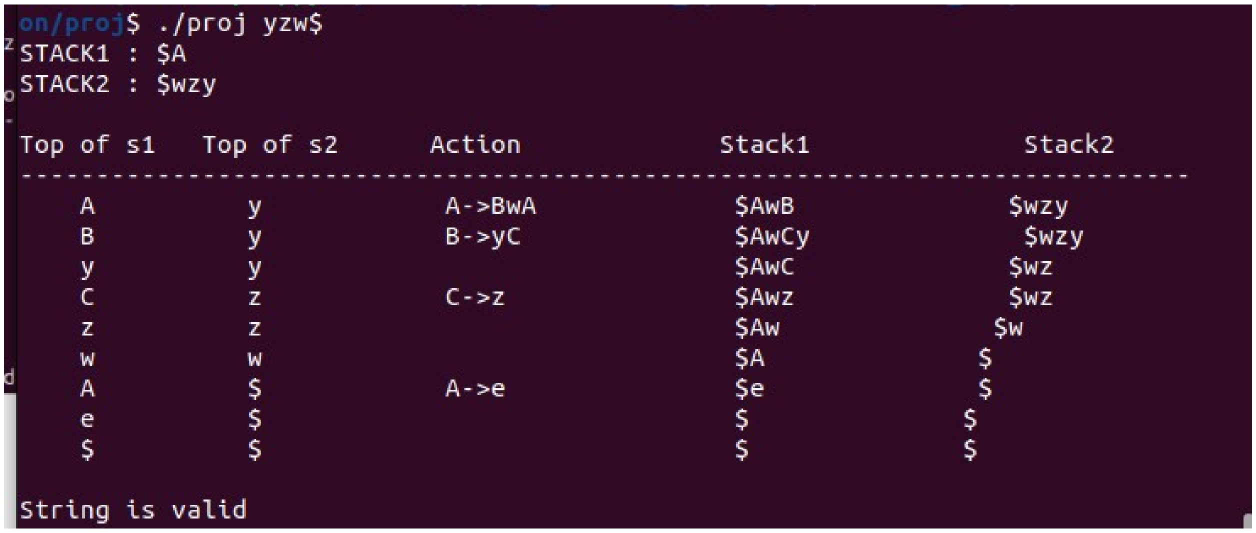
LOGIC:

- We begin with prototype definitions of all the user defined stack functions.
  Then define all the variables and arrays we will use. We also define a string array which stores all possible actions (Taken from the parse table) that the parser can perform.
- We take an input string from the command line (Input has to end with $ and be <10 in length or the program will end. 10 length according to the given stack size)
  Then in init_stack() we initialize stack 1 with the start symbol and stack 2 with the user input. We insert the input in reverse, with $ first in the stack.
- In main() we use peek1() and peek2() to get the top elements of both the stacks and store them. Then we pass these as argument to compare_tops(). This function will return a number, according to the inputs. This number corresponds to a production from the string array we made earlier.
- We do this in a while(1) loops. We have conditions setup to validate and invalidate the string and perform other actions according to the LL(1) parser.
  If both stacks have $ on top, string is valid and program ends
  If compare_tops() sends an error, string is invalid and program ends
  If both stacks have the same symbol on top, they are popped
  If stack1 has an e(epsilon) on top, it is popped
- There is also a user defined function, pushProduction(int code);
  This function, according to the output of compare_tops, pushes the appropriate production on top of stack1.
  It first pops the top symbol and then pushes that symbols production, symbol by symbol on to stack1.
- At the end of the loop, it prints the top of stack1 and stack2, the action performed and stack1 and stack2 as a whole.
- The loop keeps repeating until the validity or invalidity condition are met and the program is terminated.

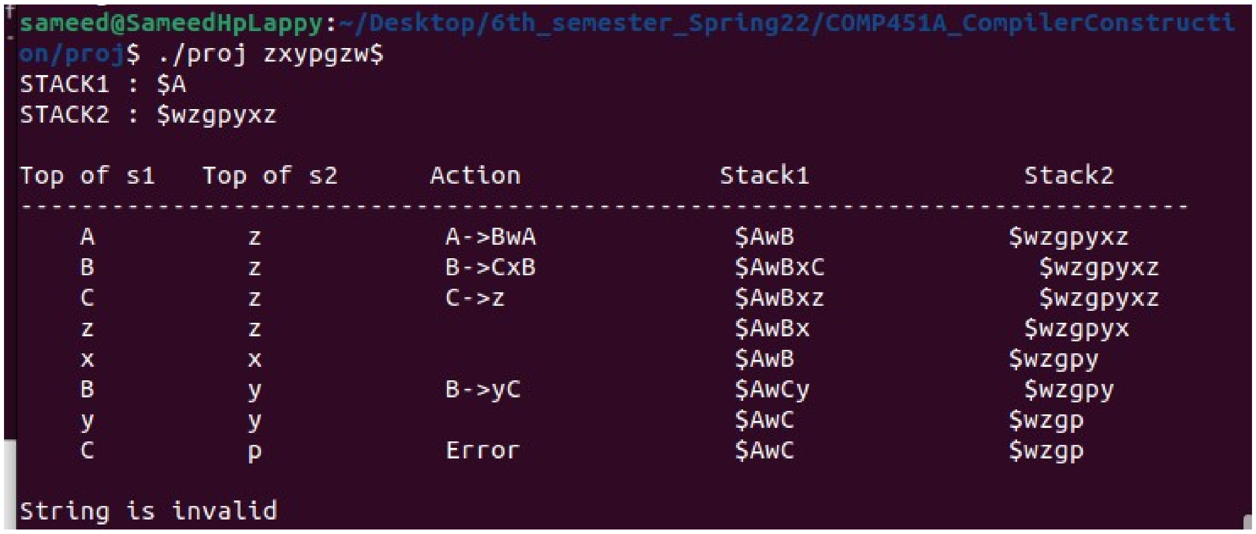
Sample Output:



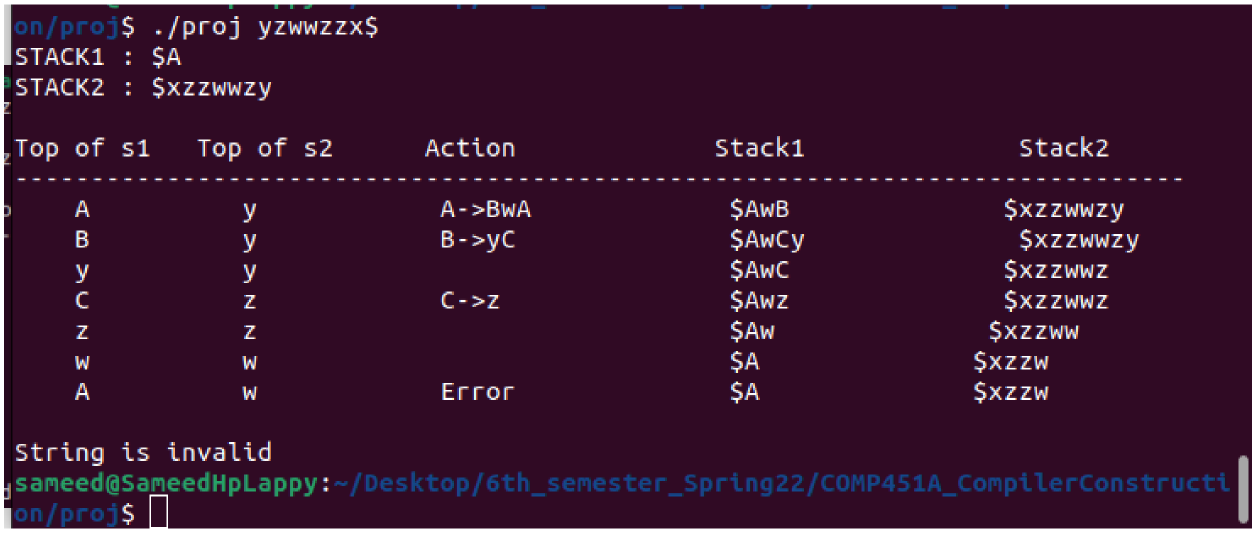
```
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/proj$ ./proj zxyzw$
STACK1 : $A
STACK2 : $wzyxz

Top of s1   Top of s2      Action              Stack1              Stack2
-------------------------------------------------------------------------------
    A           z            A->BwA              $AwB                $wzyxz
    B           z            B->CxB              $AwBxC                $wzyxz
    C           z            C->z                $AwBxz                $wzyxz
    z           z                                $AwBx                $wzyx
    x           x                                $AwB                $wzy
    B           y            B->yC               $AwCy                $wzy
    y           y                                $AwC                $wz
    C           z            C->z                $Awz                $wz
    z           z                                $Aw                $w
    w           w                                $A                $
    A           $            A->e                $e                $
    e           $                                $               $
    $           $                                $               $

String is valid
```



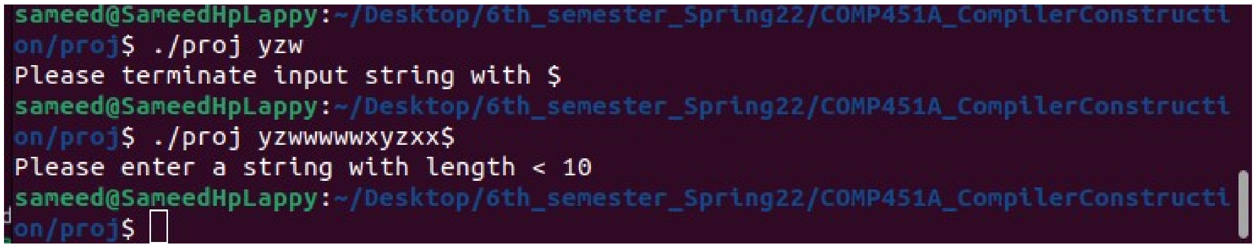
```
on/proj$ ./proj yzw$
STACK1 : $A
STACK2 : $wzy

Top of s1   Top of s2      Action              Stack1              Stack2
-------------------------------------------------------------------------------
    A           y            A->BwA              $AwB                $wzy
    B           y            B->yC               $AwCy                $wzy
    y           y                                $AwC                $wz
    C           z            C->z                $Awz                $wz
    z           z                                $Aw                $w
    w           w                                $A                $
    A           $            A->e                $e                $
    e           $                                $               $
    $           $                                $               $

String is valid
```



```
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/proj$ ./proj zxypgzw$
STACK1 : $A
STACK2 : $wzgpyxz

Top of s1   Top of s2      Action              Stack1              Stack2
-----------------------------------------------------------------------------
    A           z            A->BwA              $AwB                $wzgpyxz
    B           z            B->CxB              $AwBxC                $wzgpyxz
    C           z            C->z                $AwBxz                $wzgpyxz
    z           z                                $AwBx               $wzgpyx
    x           x                                $AwB               $wzgpy
    B           y            B->yC               $AwCy               $wzgpy
    y           y                                $AwC               $wzgp
    C           p            Error               $AwC               $wzgp

String is invalid
```



```
on/proj$ ./proj yzwwzzx$
STACK1 : $A
STACK2 : $xzzwwzy

Top of s1   Top of s2      Action              Stack1              Stack2
-----------------------------------------------------------------------------
    A           y            A->BwA              $AwB               $xzzwwzy
    B           y            B->yC               $AwCy               $xzzwwzy
    y           y                                $AwC               $xzzwwz
    C           z            C->z                $Awz               $xzzwwz
    z           z                                $Aw               $xzzww
    w           w                                $A               $xzzw
    A           w            Error               $A               $xzzw

String is invalid
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/proj$ 
```



```
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/proj$ ./proj yzw
Please terminate input string with $
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/proj$ ./proj yzwwwwwxyzxx$
Please enter a string with length < 10
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/proj$ 
```

CODE:

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


int compare_tops(char,char);


//functions to manage stacks

int isfull1();

int isempty1();

char push1(char);

char pop1();

char peek1();


int isfull2();

int isempty2();

char push2(char);

char pop2();

char peek2();

void init_Stacks(char a[]);

void pushProduction(int code);


//stacks

char stack1[10];

char stack2[10];

int top1 = -1;

int top2 = -1;

char a,b;

char NT[] = {'A','B','C'};
char buff[10];

char action[9][10] = {
    "A->BwA",
    "A->e  ",
    "B->CxB",
    "B->yC ",
    "C->z  ",
    "Error ",
    "      ",
    "      ",
    "      "
};


int main(int argc, char *argv[]){

    //check for command line inputs
    if(argc!= 2){
        printf("Incorrect number of arguments !");
        exit(0);
    }

    //place first arg in s1 and second in s2.
    //make sure $ is the last symbol in input while it
    //appears at bottom of the stack.

    init_Stacks(argv[1]);
    //display the header

    printf("STACK1 : %s\n",stack1);

    printf("STACK2 : %s\n",stack2);

    printf("\nTop of s1   Top of s2      Action              Stack1
Stack2\n");

printf("-------------------------------------------------------------------
---------------\n");


    //push top of each stack


    while(1){

        char top_s1 = peek1();

        char top_s2 = peek2();

        int ret = compare_tops(top_s1,top_s2);


        if(top_s1 == '$' && top_s2 == '$'){

            printf("    %c          %c             %s              %s
%s\n",top_s1,top_s2,action[ret],stack1,stack2);

            printf("\nString is valid\n");

            exit(0);

        }


        if(ret == 8){

            pop1();

        }
        if(ret == 5){
            printf("    %c          %c            %s              %s
%s\n",top_s1,top_s2,action[ret],stack1,stack2);
            printf("\nString is invalid\n");
            exit(0);
        }

        pushProduction(ret);
        if(ret == 7){
            pop1();
            pop2();
        }


        printf("    %c          %c            %s              %s
%s\n",top_s1,top_s2,action[ret],stack1,stack2);

    }

    //based on return value from compare_tops() function, print an
    //appropriate output
    return 0;

}




void init_Stacks(char argv1[]){
    //this function populates the stacks with the user input
    if(strlen(argv1) > 10){
        printf("Please enter a string with length < 10\n");
        exit(10);
    }

    if((argv1[strlen(argv1)-1] == '$')){

        int j = 0;

        while(j < strlen(argv1)+1){
            push2(argv1[strlen(argv1)-j]);
            j++;
        }

        j = 0;

        push1('$');
        push1('A');
    }
    else{
        printf("Please terminate input string with $\n");
        exit(0);
    }


}

int compare_tops(char a,char b){
```

```
//This function compares top of both stacks and returns an integer
which

//is then used in main to determine what action to be printed on the

//output based on the parsing table. Consider it as a long switch or
if-

//elseif statement ladder.


    if(a == 'A' && b == 'y') return 0;

    else if (a == 'A' && b == 'z') return 0;

    else if (a == 'A' && b == '$') return 1;

    else if (a == 'B' && b == 'y') return 3;

    else if (a == 'B' && b == 'z') return 2;

    else if (a == 'C' && b == 'z') return 4;

    else if (a == '$' && b == '$') return 6;

    else if (a == b) return 7;

    else if (a == 'e' && b == '$') return 8;

    else return 5;

}


void pushProduction(int code){

    // This function pushes appropriate production on stack1 after
popping the top symbol. According to the top of stack1


    if(code == 0){

        pop1();

        push1('A');

        push1('w');

        push1('B');

    }
    else if(code == 1){

        pop1();

        push1('e');

    }

    else if(code == 2){

        pop1();

        push1('B');

        push1('x');

        push1('C');

    }

    else if(code == 3){

        pop1();

        push1('C');

        push1('y');

    }

    else if(code == 4){

        pop1();

        push1('z');

    }



}


// Following are functions to manage the stacks


int isempty1() {

    if(strlen(stack1) == 0) return 1;

    else return 0;

}
```

```
int isfull1() {
    if(strlen(stack1) == 10) return 1;
    else return 0;
}
char peek1() {
    return stack1[strlen(stack1)-1];
}
char pop1() {
    char popped;
                                popped = stack1[strlen(stack1)-1];
                                stack1[strlen(stack1)-1] = '\0';
                                return popped;
}
char push1(char data) {
    if (isfull1() == 0){
        stack1[strlen(stack1)] = data;
    }
    return stack1[strlen(stack1)];
}

int isempty2() {
    if(strlen(stack2) == 0) return 1;
    else return 0;
}
int isfull2() {
    if(strlen(stack2) == 10) return 1;
    else return 0;
}
char peek2() {
    return stack2[strlen(stack2)-1];
}
char pop2() {
    char popped;
                                popped = stack2[strlen(stack2)-1];
                                stack2[strlen(stack2)-1] = '\0';
                                return popped;
}
char push2(char data) {
    if (isfull2() == 0){
        stack2[strlen(stack2)] = data;
    }
    return stack2[strlen(stack2)];
}
```