COMP 451

FORMAN CHRISTIAN COLLEGE

(A CHARTERED UNIVERSITY)

COMPILER CONSTRUCTION

CLASS PROJECT

It's an open books and open notes assignment. Use of Internet is allowed. This assignment should be done in complete isolation. No group formation is allowed. You MUST NOT share your code with any other student. Any such attempt will result in a ZERO grade in this instrument.

Grading Criteria

Working Code: 80%

Properly formatted Report: 20%

Important: You need to submit a well formatted and well written report for this assignment. The report should carry following sections:

- Introduction about the problem in hand especially well written information about the preprocessor and its functions.
- Your code followed by a detailed description explaining how you built up the logic of the program. Additional functionalities and / or exclusions (if any) should be stated with separate heading in bold face font.
- Code description should be accompanied by the screen shots of your output with at least three separate inputs.
- Start early. **NO** additional time in any case what so ever will be granted.
- Viva will be conducted for this assignment. Date will be announced later in class.

Hard Deadline: Report along with the code file/s should be submitted on Moodle course page on or before Sunday Jun 26, 2022 before 11:59 pm.

Make sure to create a zip file of your submission and give it a name using given format:

COMP451_A_RollNumber_ClassProject

Submissions through email will NOT be considered for grading.

COMP 451

Project Task [80 Marks]

This project is a sequel of our last lab. The project description is as follows:

We assume that for this project our grammar is as shown:

```
A → BwA ---------------1
A → e -----------------2
B → CxB ---------------3
B → yC ----------------4
C → z -----------------5
```

Note that each production is numbered in the grammar. Also the symbol 'e' stands for epsilon.

FIRST and FOLLOW of each variable is given:

Variable	FIRST	FOLLOW
A	y, z, e	$
B	y, z	w
C	z	x, w

The LL(1) parse table is also shown below:

	w	x	y	z	$
A			A → BwA	A → BwA	A → e
B			B → yC	B → CxB	
C				C → z	

A boiler plate code is provided. You MUST strictly use the following code template. Must not deviate from it.

```
#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[],char b[]);
//stacks
char stack1[10];
char stack2[10];
int top1 = -1;
int top2 = -1;
char a,b;
char NT[] = {'A','B','C'};
char buff[10];
int main(int argc, char *argv[])
{
      //check for command line inputs
      if(argc!= 3)
      {
      }
      //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],argv[2]);

      //display the header
      printf("Top of s1     Top of s2        Action\n");
      printf("---------------------------------------------------\n");
      //push top of each stack
      char top_s1 = pop1();
      char top_s2 = pop2();
      int ret = compare_tops(top_s1,top_s2);
      //based on return value from compare_tops() function, print an
      //appropriate output
      return 0;
}

void init_Stacks(char argv1[], char argv2[])
{
      //this function populates the stacks with the user input
}
```

```
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.
}
int isempty1() {}
int isfull1() {}
char peek1() {}
char pop1() {}
char push1(char data) {}
int isempty2() {}
int isfull2() {}
char peek2() {}
char pop2() {}
char push2(char data) {}
```

In order to complete this project, you need to first complete the lab (if not completed in the lab session). The project will be built on top of the code for the lab. In the lab you were given the task to print appropriate production given two inputs on command line arguments. (Go through the hand out if you have missed the lab).

For this project you will write C code that completely simulates the LL(1) parser for the given grammar. Note that we have not changed the grammar for the project.

Your program should accept a string from user on command line.

The string must be terminated with a '$' symbol.

If user omits the $ symbol, the program should produce an exception displaying a message and terminating the process.

Your program should initialize two stacks. One with the input string and other with start symbol. Note that both stacks should have $ symbol as first element.

Your program should then compare the top of stacks and proceed as per the algorithm of LL(1) parser discussed in class.

If the input string is valid, your program should accept it, otherwise reject it.

The output of the program should be a **"complete"** stack implementation table for the user provided input string.

Make sure that your program should be able to tackle all kinds of strings in an appropriate way.