COMP 451

FORMAN CHRISTIAN COLLEGE

(A CHARTERED UNIVERSITY)

COMPILER CONSTRUCTION

Class Project

Spring 2024

It's an open books and open notes task. This programming task can be done in groups. A maximum of two students is allowed in a group. Groups must be formed within the section. Students from different sections of this course cannot create a group. You CANNOT share your code with any other group. 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 project. The report should carry following sections:

- Introduction about the problem in hand.
- Your short code segments followed by a detailed description explaining how you built up the logic of the given code snippet.
- If you have used some user defined functions in your code make sure to given a detailed description about their working.
- Additional functionalities and / or exclusions (if any) should be stated with separate heading in bold face font.
- Code description should be augmented (if possible) by the screen shots of the output for that piece of code to make your description clearer and more concise.
- Start early. **NO** additional time in any case what so ever will be granted.
- *Your code will be checked for plagiarism on Turnitin. All students who will have a similarity index larger than 30% will be awarded ZERO in this assessment.*

Hard Deadline: Submit your code file on Moodle course page on or before Sunday Jun 02, 2024 11:59 pm. The hard copy of the project MUST be submitted on Monday Jun 03 in the lab session.

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

COMP451_<Student_ RollNumbers>_classProject

For example, COMP451_12345678_classProject

Submissions through email will NOT be considered for grading.

Project Task - 1 [20 Marks]

Write a program in C which can accept input from the command line argument. The input should be an expression that can have arithmetic operators like +, -, * and / applied on some identifiers. A sample input is shown:

b*c/d+a-b$

a+b+c-d*e$

Input should be provided on command line from the user.

Note that we can have a maximum of FIVE operands in our input string. Each input must have a $ symbol as sentinel value.

Here we assume that all four arithmetic operators are valid tokens while we can have any lower-case single alphabet as operand. All other characters are considered as invalid tokens.

Your program should scan the input string, and identify tokens.

The output of this program should be as follows:

Sample Run-1

```
$ gcc lexer.c -o lexer
$ ./lexer a+b-c*d/a$
Program finds following tokens in the expression:
Expression received:  a+b-c*d/e$
int literal found: a
Arithmetic operator: +
int literal found: b
Arithmetic operator: -
int literal found: c
Arithmetic operator: *
int literal found: d
Arithmetic operator: /
int literal found: d
```

Sample Run-2

```
$ gcc lexer.c -o lexer
$ ./lexer a+b*/a$
Program finds following tokens in the expression:
Expression received:  a+b*/a$
int literal found: a
Arithmetic operator: +
int literal found: b
Arithmetic operator: *
Arithmetic operator: /
int literal found: a
```

Sample Run-3

```
$ gcc lexer.c -o lexer
$ ./lexer a+b@c*a$
Program finds following tokens in the expression:
Expression received:  a+b@c*a$
int literal found: a
Arithmetic operator: +
int literal found: b
Invalid token encountered. Program terminated prematurely.
```

Project Task - 2 [40 Marks]

Now that you have identified all the tokens in the expression, write a program that will accept the valid stream of tokens (along with the sentinel value), apply a parsing algorithm based on a CFG and provide the output of the expression while displaying the stack implementation table.

For this purpose, we will use following grammar:

E → E + E | E - E | E * E | E / E | id

Note that the grammar above is ambiguous as well as carries left recursion. You must get rid of all these issues before you use it in your program.

We will use this grammar in operator precedence parser. For our parser we can make use of the following operator precedence relation table:

	Id	+	-	*	/	$
Id		>	>	>	>	>
+	<	>	>	<	<	>
-	<	>	>	<	<	>
*	<	>	>	>	>	>
/	<	>	>	>	>	>
$	<	<	<	<	<	

Your program should follow the following flow:

First your program should recognize the identifiers in the input stream of tokens (provided as command line argument) and ask user to enter the value for each.

Next, your task is to write a program that can list down the stack implementation table for the given input using this parser.

While the table is listed down you can apply your logic such that for each pop in the action column the output stack should be updated accordingly providing the result of expression if the string is accepted.

A sample run is shown on next page.

Sample Run

```
$ gcc parser.c -o parser
$ ./parser a+b*c$
Enter integer values of the following identifiers:
Value of a: 2
Value of b: 5
Value of c: 10
The stack implementation table for operator precedence parser for the given
expression is as follows:
```

Stack	Input	Action
$	a+b*c$	
a$	+b*c$	Push
$	+b*c$	Pop
+$	b*c$	Push
b+$	*c$	Push
+$	*c$	Pop
*+$	c$	Push
c*+$	$	Push
*+$	$	Pop
+$	$	Pop
$	$	Pop
$	$	Accepted

```
The output of the given expression is: 52
```

Note that in case of any syntactical error the program should terminate displaying an appropriate error.

Project Task - 3 [20 Marks]

In this section, you will write an SDT for the same grammar, making appropriate changes as per the type of your parser. The SDT should enable you to add code in part 2 so that now the program should not only print the stack implementation table for the input string but should also generate the assembly code (only if the input string is valid). For the sake of simplicity, assume that you have an unlimited supply of registers with naming convention of the register being an R followed by an integer. That is R0, R1, R2, …

For this project you can assume that you are only concerned with the following instructions:

Load Immediate	LI	<Register>, <Variable>
ADDITION	ADD	<Destination Register>, <Source Retister 1>,<Source Register 2>
SUBTRACTION	SUB	<Destination Register>, <Source Retister 1>,<Source Register 2>
MULTIPLICATION	MUL	<Destination Register>, <Source Retister 1>,<Source Register 2>
DIVISION	DIV	<Destination Register>, <Source Retister 1>,<Source Register 2>

You need to first load the variables into registers, and then can write the assembly file instructions in a separate file. (just my algo. You may have your own). Once you have finished with the table, you would have written the assembly code in a file as well. Now you can simply read the file and display the code on console.

As an example if we continue with the same sample shown in task2, the output for this section with the same input string should be as follows:

Sample Run

```
$ gcc parser.c -o parser
$ ./parser a+b*c$
Enter integer values of the following identifiers:
Value of a: 2
Value of b: 5
Value of c: 10
The stack implementation table for operator precedence parser for the given
expression is as follows:
```

Stack	Input	Action
$	a+b*c$	
a$	+b*c$	Push
$	+b*c$	Pop
+$	b*c$	Push
b+$	*c$	Push
+$	*c$	Pop
*+$	c$	Push
c*+$	$	Push
*+$	$	Pop
+$	$	Pop
$	$	Pop
$	$	Accepted

```
The output of the given expression is: 52
```

The Equivalent Assembly code

```
LI    R1,a
LI    R2,b
LI    R3,c
MUL   R0, R2, R3
Add  R4,R1,R0
```

Submission Criteria:

You are supposed to submit three programs, one against each task.

Place all your files in a folder.

Give it appropriate name. Strictly follow the naming convention.

Zip the folder with the same naming convention and upload it on MOODLE.