Hacettepe University

Computer Engineering Department

BBM203 Software Laboratory

Fall 2016

Date Issued	**:** 7.11.2016
Date Due	**:** 20.11.2016
Advisors	**:** Dr. Sevil Şen, Dr. Burcu Can Buğlalılar, Dr. Adnan Özsoy, Dr. Aydın Kaya

Summary

In this experiment you will learn how to use dynamic data structure (stacks and queues) on C programming language.

Problem

You will implement a stack-based calculator that works on hexadecimal and integer numbers. Stacks can be used for evaluating arithmetic expressions. To make this work, you should take advantage of C struct and pointer facility. The stack elements will store operands and operators. Your calculator must evaluate Integer and Hexadecimal expressions. All operators are character type.

Your program takes input file and handles the arithmetic expressions one by one. Output file contains the results of expressions present at input file.

```
exp2  [input file name] [output file name]
```

There are two commands in input file: calculate and print. When a row starts with calculate command, your program handles the following arithmetic expression as hex or integer. There two expression types and these will be indicated by hex/integer expression. Arithmetic expressions will be given between two quotation marks. You can see the sample expressions and valid/invalid expressions below. Your program must handle both valid and invalid expressions.

```
calculate hex “AA+0CE*2”        -- valid input, evaluate expression
calculate hex "aa+0Ce"          -- valid input, evaluate expression
calculate integer “-200+200”    -- valid input, evaluate expression
calculate hex "-1A"             -- valid input, evaluate expression
calculate hex "+1A"             -- valid input, evaluate expression
calculate integer “(1+22”       -- expression error
calculate integer “(1+(22)))”   -- expression error
calculate integer “1.1”         -- expression error
calculate integer “1 1”         -- expression error
calculate integer "1-*1"        -- expression error
calculate integer "2+-(2)"         -- valid input, evaluate expression
calculate integer "2*-2"        -- valid input, evaluate expression
calculate integer "2/-2"        -- valid input, evaluate expression
```

If an arithmetic expression is valid, you will store expression type (hex/integer) and result. If this expression has invalid structure, you will store just the “Error” string. You will store the expressions’ results on a queue data structure, until a “print” command occurs. If an input row includes print command, you will write all the queue content to output file.

calculate hex “AA+0CE*2”	(add to queue: hex | 246)	Queue has 1 element
calculate hex "aa+0Ce"	(add to queue: hex | 178)	Queue has 2 element
calculate integer "10+(8*(9-11+3))"	(add to queue:: integer | 18)	Queue has 3 element
calculate hex “((AB+12)*12”	(add to queue: error)	Queue has 4 element
print	Print all to output file	Queue has 0 element
calculate integer “5+5”	(add to queue:: integer | 10)	Queue has 1 element

There is a simple input file on first column of table. Queue content is shown on second column and queue element count is shown on third column. At the beginning, your queue is empty. There are four consecutive calculate commands. On each command you will add expression type and value to your queue data structure. On the fifth command you see “print” expression. Now you pull elements one by one and write them to output file one by one. The last calculate command will be handled and stored in queue but will not be printed on output file. Final output file content is shown below.

```
hex 246
hex 178
integer 18
error
```

Please do not provide additional information on output file. You have to strictly comply with output file format.

Calculator Properties

- Nested parenthesis can be used.
- Negative numbers can be used.
- Arithmetic priority is important. Parathesis are used for changing priority.
- Check missing parathesis.
- Write hex number in upper-case.
- White space on output is not important.
- Here is arithmetic operator list in order of priority:
    1. * : Multipication (3*2 = 6) and
       / : Division (21/3 = 7)
    2. + : Addition (-3+67 = 70) and
       - : Subtraction (-45-2 = -49)
- All strings are case in-sensitive.

How to use stacks for arithmetic expressions?

You can examine Chapter 13.1 of "Ada 95: The Craft of Object-Oriented Programming" for detailed information. This part will be shared in piazza group.

			Stack for	Stack for	
	Input	Symbol	Operands	Operators	Action
1)	2*(3+4).			#	(start state)
2)	*(3+4).	2	2	#	Push 2
3)	(3+4).	*	2	# *	* > #; push *
4)	3+4).	(	2	# * (	Push (
5)	+4).	3	2 3	# * (	Push 3
6)	4).	+	2 3	# * (+	+ > (; push +
7)	).	4	2 3 4	# * (+	Push 4
8)	.	)	2 7	# * (	Apply +
9)	.	)	2 7	# *	Remove (
10)	.		14	#	. < *; apply *

Notes:

- Please use understandable variable and function names. Respect to C naming conventions.
- Your stack and queue data structures must be dynamic.
- Provide comment part for each of your functions. Also provide comment lines on critical parts of your codes, if necessary.
- Please strictly respect to output file formats.
- Submission details will be shared on piazza group.

Remarks:

- You will use online submission system to submit your experiments.
- https://submit.cs.hacettepe.edu.tr/ Deadline is: 23:59. No other submission method
  (such as; CD or email) will be accepted. Late deliveries will not be accepted, either.
- Do not submit any file via e-mail related with this assignment.
- SAVE all your work until the assignment is graded.
- The assignment must be original, INDIVIDUAL work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but DO NOT SHARE answers, algorithms or source codes.
- You can ask your questions through course's Piazza group and you are supposed to be aware of everything discussed in the group.
  - https://piazza.com/hacettepe.edu.tr/fall2016/bbm203