Sabancı University

Faculty of Engineering and Natural Sciences

CS204 Advanced Programming

Fall 2019-2020

Homework 3 – Two Languages

Due: **20/10/2019, 11:55pm**

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You HAVE TO write down the code on your own.

You CANNOT HELP any friend while coding.

Plagiarism will not be tolerated!

Introduction

This homework's aim is to make you familiar with the logic behind **queues** and **stacks** (and a bit of **simply linked lists** again) by practically using them in a real-case scenario (you will probably encounter similar cases as a future programmer).

A Tale of Two Languages

We have created a new programming language called CS++ which has a very simple syntax and another language called Machine Language. Details about the languages will be explained later in the document. The program you will implement will take an input of CS++ language and transform it to Machine Language code. After the transformation, it will evaluate the Machine Language code and run it step-by-step.

CS++ Syntax

With the programming language one can define up to 20 variables. And users can define only integer variables. Syntax is similar to C++, but there is no other function than the **main** function. Below, a sample CS++ program is given.

```
main:
        int a1
        int a2
        int b1 = 5
        a1 = 4
        a2 = ( ( b1 + 4 ) * a1 )
        print a2
return
```

The CS++ code above will be translated to machine code below – comments (in red) are added just for your sake, they are not required in your solution:

```
mov CSG2 5 //store the value 5 in register CSG2 – we will explain the registers later
mov CSG0 4 // store the value 4 in register CSG0
add CSG2 4 //perform addition with b1 and 4
push CSA //CSA is a special register that stores the results of all arithmetic operations
pop //pop the last element from the stack – it will be stored in a special register CSP
mul CSP CSG0 //multiply the values of CSP and CSG0
push CSA //push the result to the stack
pop //pop it back to CSP
mov CSG1 CSP //set the value of CSG1 to that of CSP
push CSG1  //push the value of CSG1 to stack
print //pop and print the top value in the stack
ret //return
```

The details of the machine code are given below.

Registers:

In the machine code there are small storage units to store the variables, which are called registers. In real life, a processor register (CPU register) is one of a small set of data holding places that are part of the computer processor. You will be using these registers to store the variables defined in the CS++ program given as input. You have 25 registers. 5 of them have special purposes and the rest is for you to store the defined variables. The names and purposes of the registers are given below:

- 25 Registers
    - CSA (Add/mul/sub operation result will be stored in)
    - CSB (After Div operation, result will be here)
    - CSC (After Div operation, remainder will be here)
    - CSG0 (General use)
    - CSG1 (General use)
    - CSG2 (General use)
    - CSG3 (General use)
    - CSG4 (General use)
    - CSG5 (General use)

      …

      …
    - CSG19 (General use)
    - CSP (After pop operation, popped value from stack is stored in here)
    - CST (Can be used to store values for a temporary time)

Instructions

The code written in CS++ is translated into simple instructions to be processed by the CPU. And we have 9 simple instructions, which are defined clearly below:

- Instructions:
    - push
    - pop
    - mov
    - add
    - sub

- mul
- div
- print
- ret

Instruction Definitions:

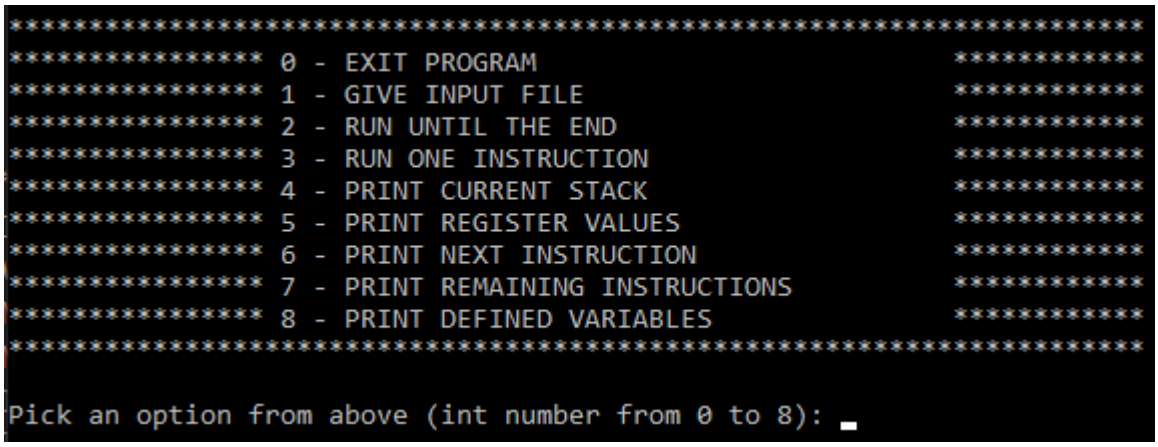
- push:
  - Pushes the parameter to the stack (memory) which you will generate to simulate computer behavior. If the given parameter is register or a user-defined variable, only the value of it will be pushed to stack.
- mov:
  - Takes two parameters and assigns the value of the first parameter to second.
  - For example **mov CSG0 6** means, CSG0 = 6.
  - And **mov CSG0 CSG1** means, CSG0 = CSG1.
- pop:
  - Pops the item at the top of the stack and stores it in CSP.
  - **pop** means pop the item at the top of stack then store it in **CSP** register (CSP = 4).
- add:
  - Add takes two parameters; it sums up the values of parameters and stores the result in the **CSA** register.
  - **add CSG0 CSG1,** in the example, sums up the values stored in CSG0 (let's say it is 6) and CSG1 (let's say it is 4) and stores the result in CSA (which becomes 10).
- sub:
  - Sub takes two parameters. It subtracts the second parameter from the first one and stores the result in the **CSA** register. For example:
    - **mov CSG0 6**
    - **mov CSG1 4**
    - **sub CSG0 CSG1**
  - The example operation given subtracts the value stored in CSG1 from CSG0 and stores the result in **CSA** (CSA becomes 2).
- mul:
  - This instruction takes two parameters (values or registers). It multiplies the values and stores the result of it in the **CSA** register.
    - For example**, mul 5 12** operation will multiply 5 with 12 and stores the result (60) in **CSA.**
- div:
  - It takes two parameters, lets just call them param1 and param2. Then divides param1 to param2. It is an integer division and the result of it will be stored in **CSB**. Also remainder will be calculated and stored in the **CSC**.
  - For example: **div 21 5**
  - After the instruction is processed:
    - CSB = 4
    - CSC = 1
- print:
  - The print function pops the item at the top of the stack and prints it. Sample usage:
    - **push 4**
    - **print**
  - The example above will print 4.

- ret:

- There is no special thing about **ret**. When you see **ret**, it means program is ended. Program must exit at that point

The main menu

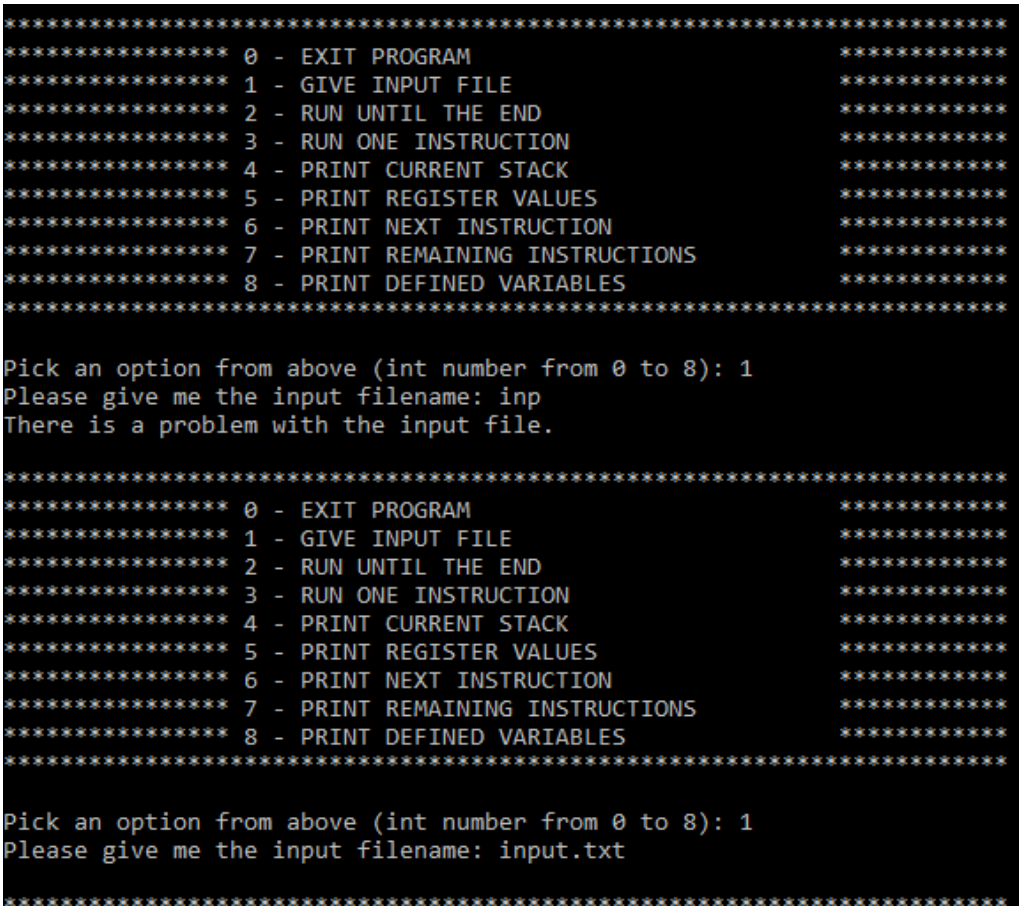
Part of the program is given to you in the zip file. In the main menu there will be 8 option presented to the user. The main menu is as follows:



Even though part of program is given to you, you can edit any part of it as long as the output is the same. Below we describe each of the main menu options (with option 0 being obvious and already implemented).

1) Give Input File

It will ask the user for an input CS++ file to transform into Machine Language code. The steps of the generated machine code will be stored in a **queue.** If the input file couldn't be opened it will print an error and return to main menu. **There won't be any syntax errors in the given CS++ file. You don't need to check that – you can assume that everything is as they are supposed to be, e.g., the variables are defined only once, if they are used they are already defined etc.** Just focus on the conversion and running the steps. The erroneous and successful outputs are shown below.



Sample input file used for demonstration is given below:

input.txt

```
main:
        int a1
        int a2 = 4
        int a3
        int a4 = ( ( 5 + 13 ) / ( 4 - 2 ) )
        print a4
        int a5
        int a6 = a2
        print a6
        a1 = 5
        a2 = 3
        print a2
        a3 = 8
        a5 = 12
        print a5
        a6 = 9
        int a7
        a7 = ( ( ( a3 * ( a1 + a2 ) ) - a4 ) / ( a5 % a6 ) )
        print a7
return
```

As seen in the input file, code starts with main, then int variables are defined. Variables can be initialized when they are defined also values can be assigned to variables after definition. On the right side of assignment operator (=), there can be integer value (e.g. a2 = 4), there can be another variable (e.g. a6 = a2) and there can be a computation consists of simple arithmetic operations: addition, subtraction, multiplication, division and mod. Calculations will be in infix form as seen in the line:

- a7 = (((a3 * (a1 + a2)) - a4) / (a5 % a6))

There is an empty space (" ") between every token in the calculation. So, you can get every variable name, operator, parenthesis with stringstream. In an equation, all arithmetic operations are always parenthesized. So you know when to perform an operation.

The Machine Code version of the ***input.txt*** is given below:

```
mov CSG1 4
add 5 13
push CSA
sub 4 2
push CSA
pop
mov CST CSP
pop
div CSP CST
push CSB
pop
mov CSG3 CSP
push CSG3
print
mov CSG5 CSG1
push CSG5
print
mov CSG0 5
mov CSG1 3
push CSG1
print
mov CSG2 8
mov CSG4 12
push CSG4
print
mov CSG5 9
add CSG0 CSG1
push CSA
pop
mul CSG2 CSP
push CSA
pop
sub CSP CSG3
push CSA
div CSG4 CSG5
push CSC
```

pop
mov CST CSP
pop
div CSP CST
push CSB
pop
mov CSG6 CSP
push CSG6
print
ret

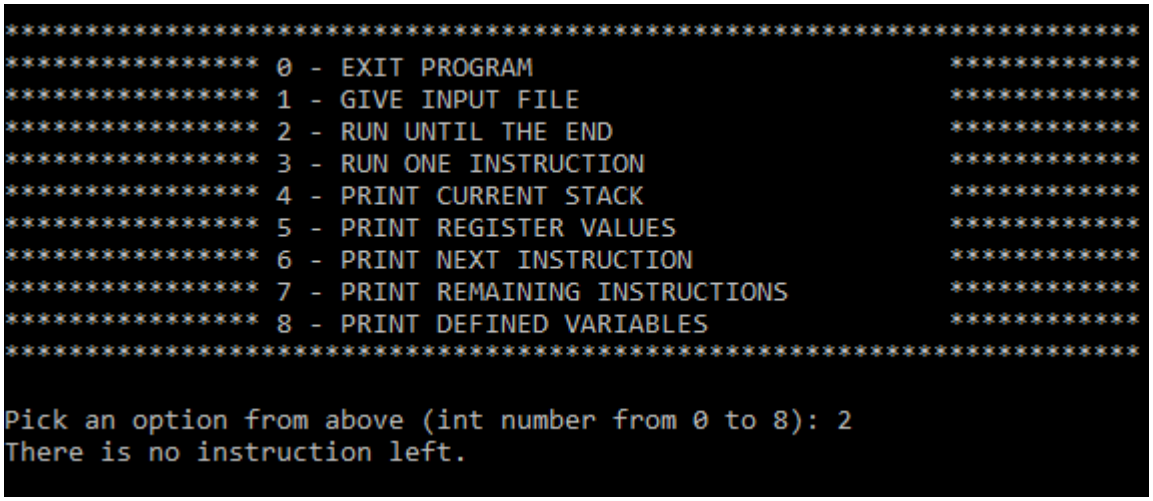
After the user uses 2nd and 3rd options, the Machine Code instructions will be executed, so that there might be values on memory stack, the instruction queue may not be empty, registers may be storing some values; so that when the user selects option 1, these structures must be cleared.

2) Run Until the End

With this option, your program will run the Machine Code you generated from CS++ code in the first option. The output for this option with the input file (*input.txt*) is given below.

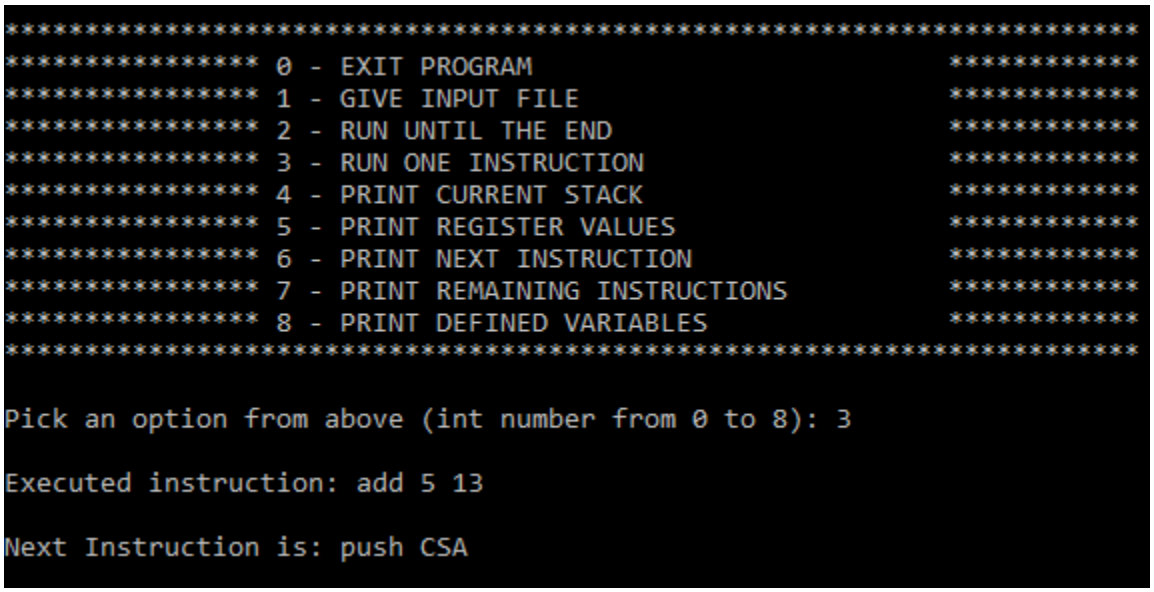
```
**************************************************************************
********** 0 - EXIT PROGRAM                                     **********
********** 1 - GIVE INPUT FILE                                  **********
********** 2 - RUN UNTIL THE END                                **********
********** 3 - RUN ONE INSTRUCTION                              **********
********** 4 - PRINT CURRENT STACK                              **********
********** 5 - PRINT REGISTER VALUES                            **********
********** 6 - PRINT NEXT INSTRUCTION                           **********
********** 7 - PRINT REMAINING INSTRUCTIONS                     **********
********** 8 - PRINT DEFINED VARIABLES                          **********
**************************************************************************

Pick an option from above (int number from 0 to 8): 2
Print Output: 9
Print Output: 4
Print Output: 3
Print Output: 12
Print Output: 18
```

After the instructions of Machine Code are executed, the program should return to the main menu. Since there is no instruction left in the queue, the instruction queue should be empty. After option 2 is called, if user tries to select option 2 again, it should state that there are no more instructions left in the queue.

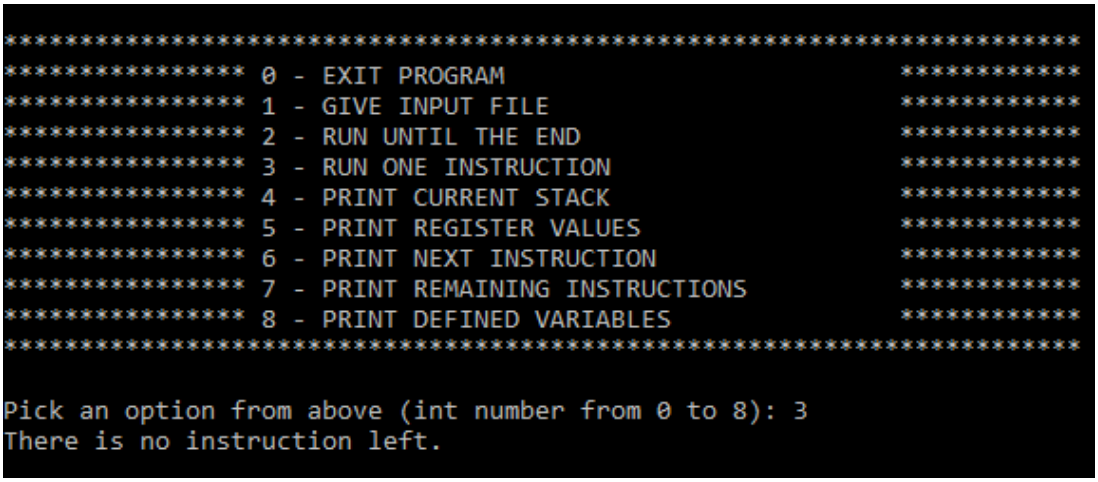


3) Run One Instruction

This option will be used to run only one instruction. You will dequeue one instruction from instruction queue and execute it.



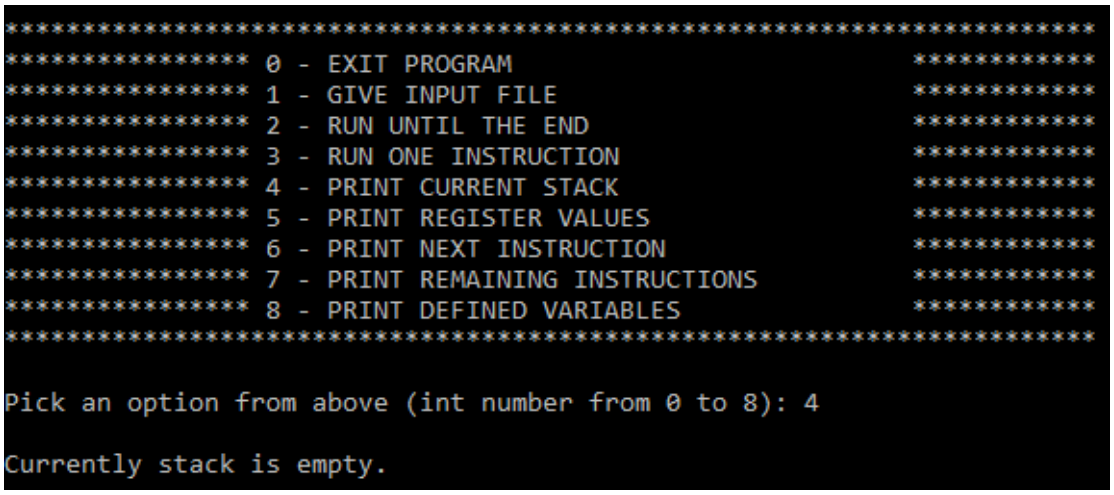
When there is no instruction left on the queue, it should print a proper message and go back to main menu.



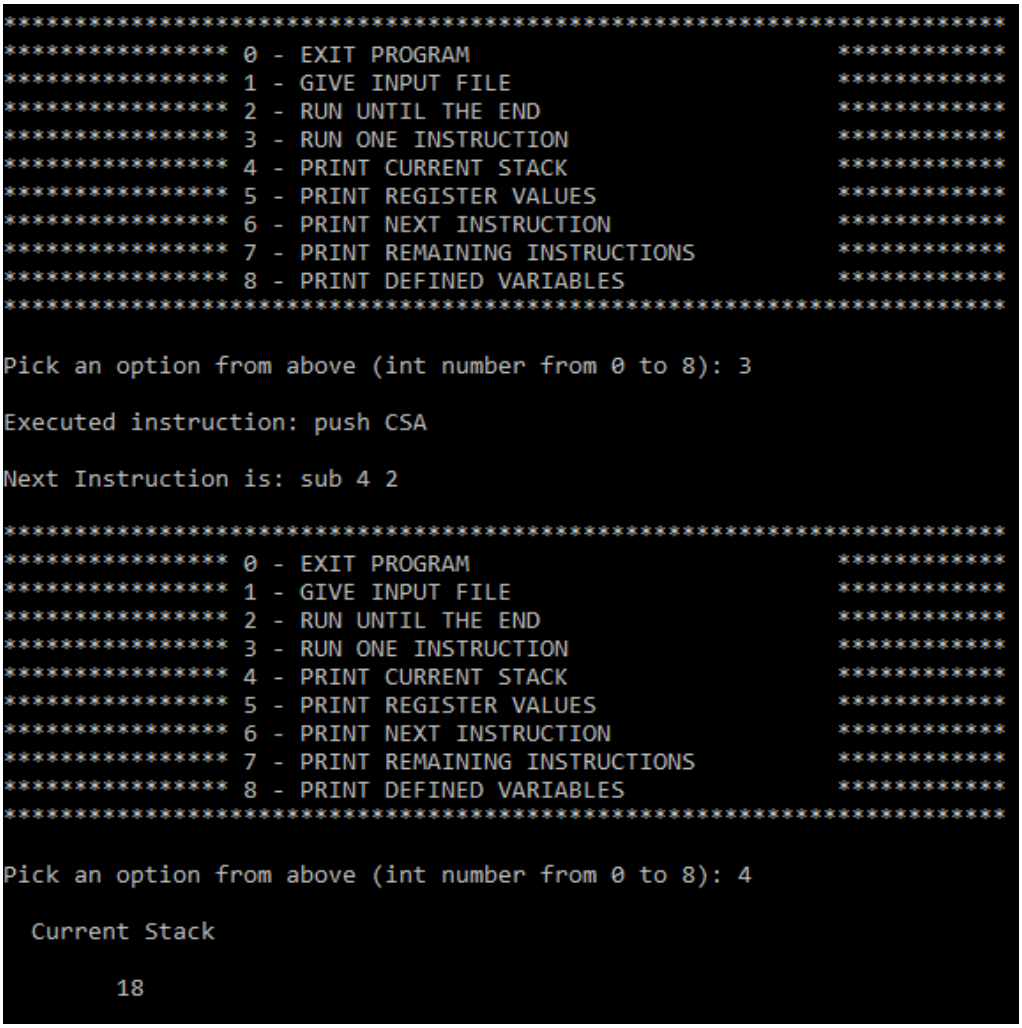
4) Print Current Stack

At this option, the memory stack will be printed. The stack is used when running the Machine Code instructions. When **push** instruction is executed, the parameter given to push is pushed to memory stack. (e.g. push 5, push CSA, push CSG1, push CSP).

When the stack is empty, it should print a proper message.

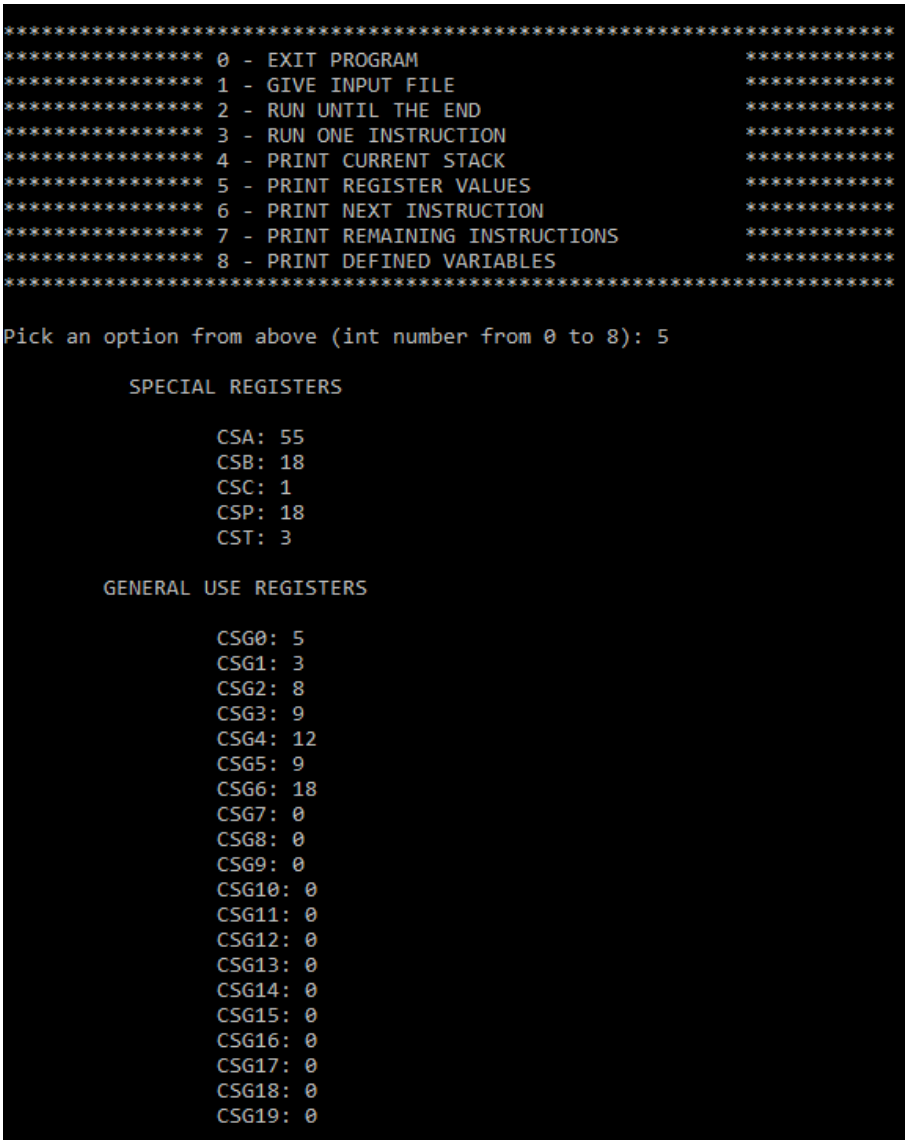


Otherwise it should print the elements in the stack. Below is an example after 3rd instruction is executed in the instruction queue generated by ***input.txt***.



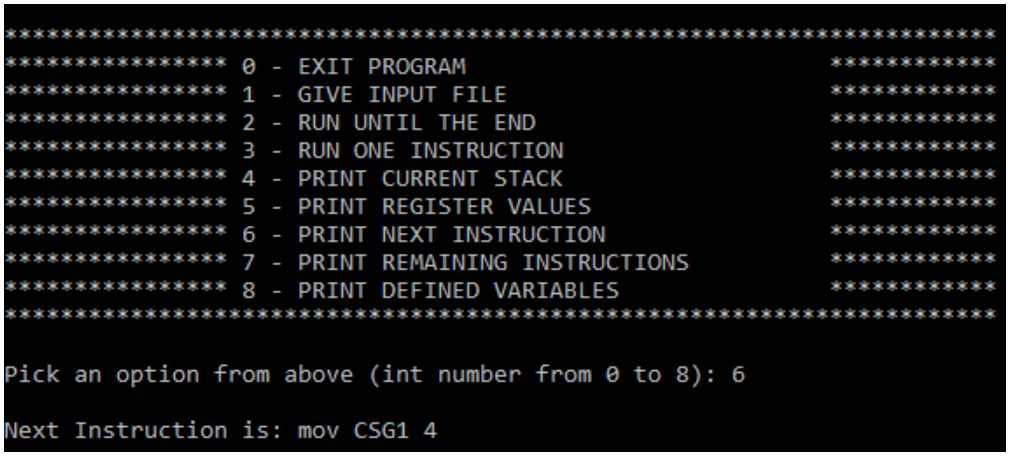
5) Print Register Values

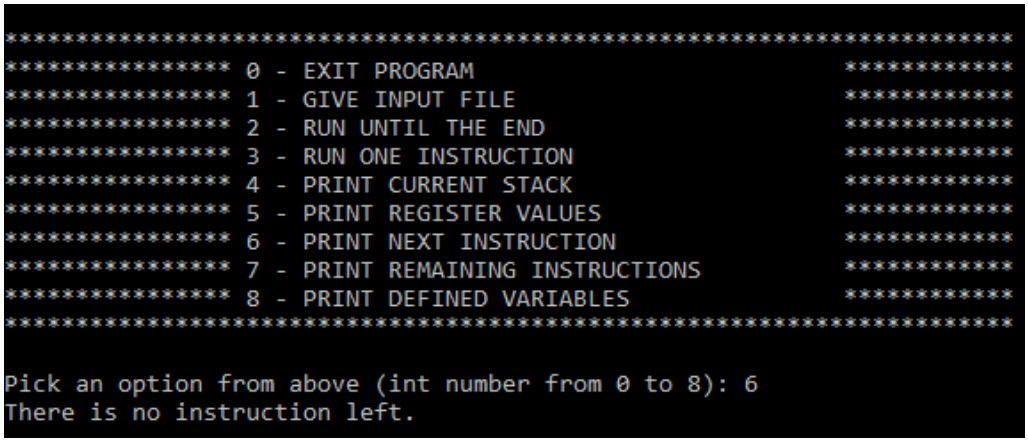
With this option, program will print the Special Use Registers and General Use Registers and the values stored in them. The Machine Code generated by ***input.txt*** is executed till the end, and the registers are printed as follows.



6) Print Next Instruction

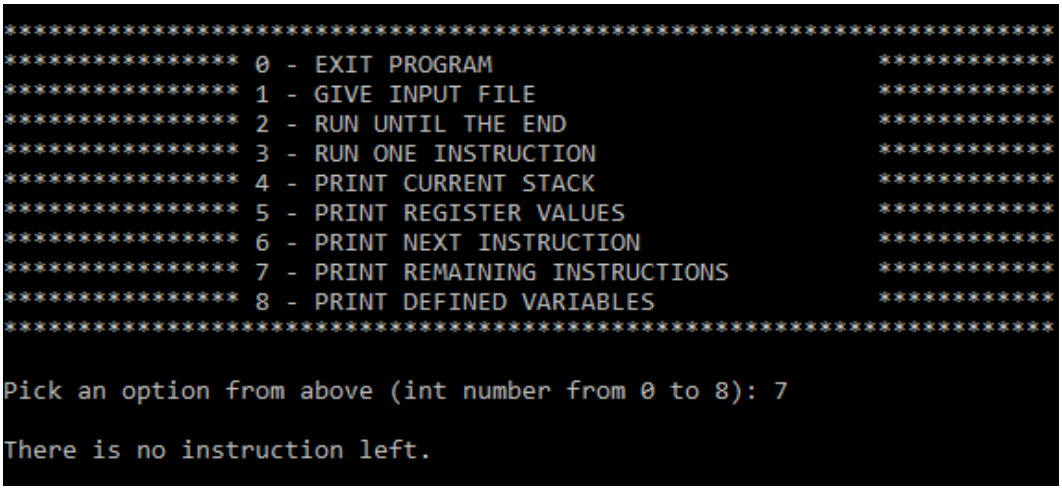
The next instruction to be executed will be printed. If there is no instruction left, proper message should be printed and go back to main menu.





7) Print Remaining Instructions

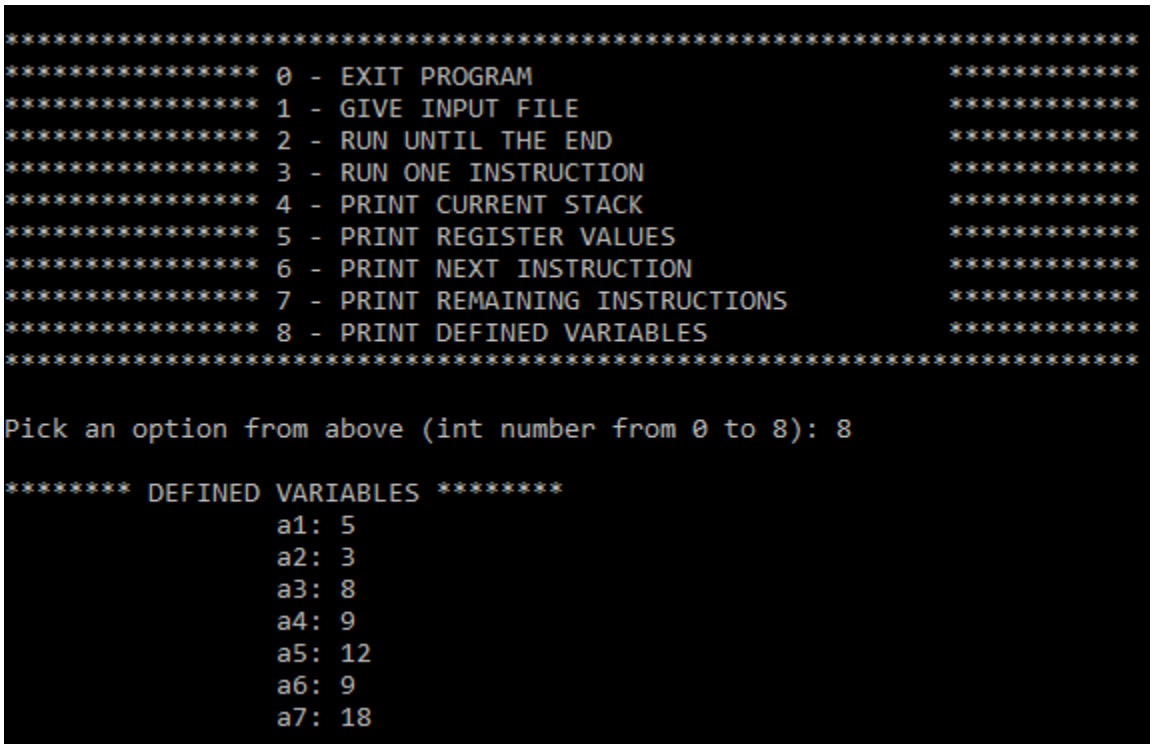
It will print the remaining instructions in the instruction queue. If there is no instruction left in the queue it should print a proper message and go back to main menu.



```
**************************************************************************
**************** 0 - EXIT PROGRAM                                 **********
**************** 1 - GIVE INPUT FILE                              **********
**************** 2 - RUN UNTIL THE END                            **********
**************** 3 - RUN ONE INSTRUCTION                          **********
**************** 4 - PRINT CURRENT STACK                          **********
**************** 5 - PRINT REGISTER VALUES                        **********
**************** 6 - PRINT NEXT INSTRUCTION                       **********
**************** 7 - PRINT REMAINING INSTRUCTIONS                 **********
**************** 8 - PRINT DEFINED VARIABLES                      **********
**************************************************************************

Pick an option from above (int number from 0 to 8): 7

******** INSTRUCTIONS ********
[1] mov CSG1 4
[2] add 5 13
[3] push CSA
[4] sub 4 2
[5] push CSA
[6] pop
[7] mov CST CSP
[8] pop
[9] div CSP CST
[10] push CSB
[11] pop
[12] mov CSG3 CSP
[13] push CSG3
[14] print
[15] mov CSG5 CSG1
[16] push CSG5
[17] print
[18] mov CSG0 5
[19] mov CSG1 3
[20] push CSG1
[21] print
[22] mov CSG2 8
[23] mov CSG4 12
[24] push CSG4
[25] print
[26] mov CSG5 9
[27] add CSG0 CSG1
[28] push CSA
[29] pop
[30] mul CSG2 CSP
[31] push CSA
[32] pop
[33] sub CSP CSG3
[34] push CSA
[35] div CSG4 CSG5
[36] push CSC
[37] pop
[38] mov CST CSP
[39] pop
[40] div CSP CST
[41] push CSB
[42] pop
[43] mov CSG6 CSP
[44] push CSG6
[45] print
[46] ret
```

8) Print Defined Variables

This option will print the variables defined in the input CS++ file and values of the variables.



Some Important Rules

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate.

When we grade your homeworks we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we are going to run your programs in *Release* mode and **we may test your programs with very large test cases**.

What and where to submit (PLEASE READ, IMPORTANT)

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.
Name your cpp file that contains your program as follows:

"SUCourseUserName_YourLastname_YourName_HWnumber.cpp"

Your SUCourse user name is actually your SUNet username that is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is valent, name is Valentina, and last name is Tereşkova, then the file name must be:

Valent_Tereskova_Valentina_hw1.cpp

Do not add any other character or phrase to the file name. Make sure that this file is the latest version of your homework program. Compress this cpp file using WINZIP or WINRAR programs. Please use "zip" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains your cpp file.

You will receive no credits if your compressed zip file does not expand or it does not contain the correct file. The naming convention of the zip file is the same as the cpp file (except the extension of the file of course). The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourName_HWnumber.zip

For example zubzipler_Zipleroglu_Zubeyir_hw1.zip is a valid name, but

hw1_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
CS204 Team (M. Yusa Erguven, Kamer Kaya)