

Mariya Eggensperger
CST 370, Spring 2017
Dr. Feiling Jia
Design/Analysis
of Algorithms

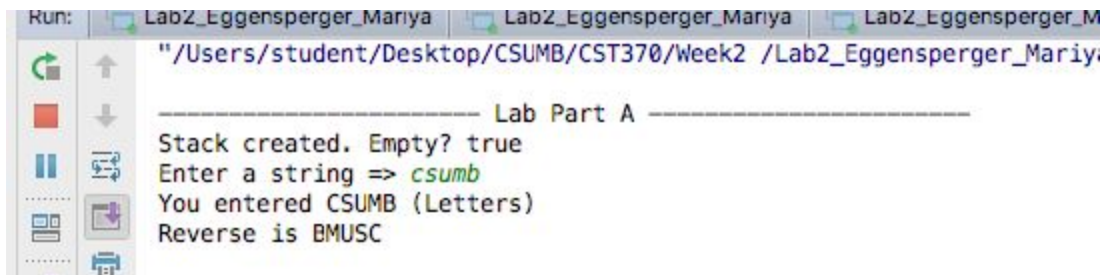
Lab 2 Submission : Stacks

(a) Download Stack.h, Stack.cpp, and Sample_Stack_tester.cpp from iLearn

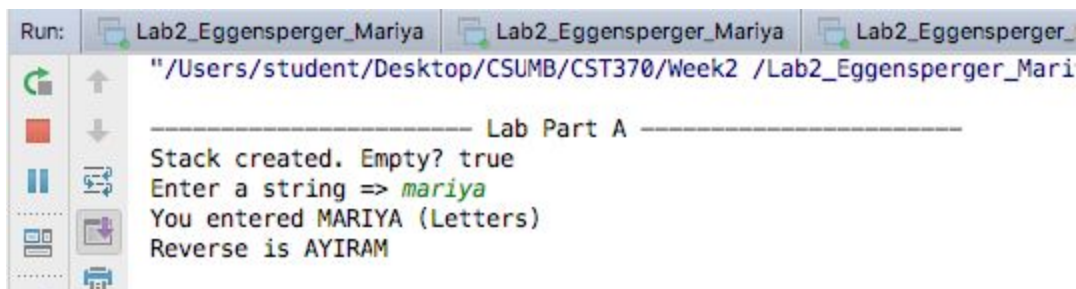
Make a project with the three files. Then, modify the programs so that the program can read a sequence of characters and reverse it using the stack. The output of your program should look as follows:

```
Enter a string => CSUMB  
You entered CSUMB  
Reverse is BMUSC
```

Figure 1 This is the first run for the character input program. The program first checks whether a new stack was created and that the new stack is empty for input. The program then prompts the user to enter a sequence of characters. If the user inputs lowercase letters, the program transforms the letters to upper (uppercase) and then reverses the input as seen in these console output(s).

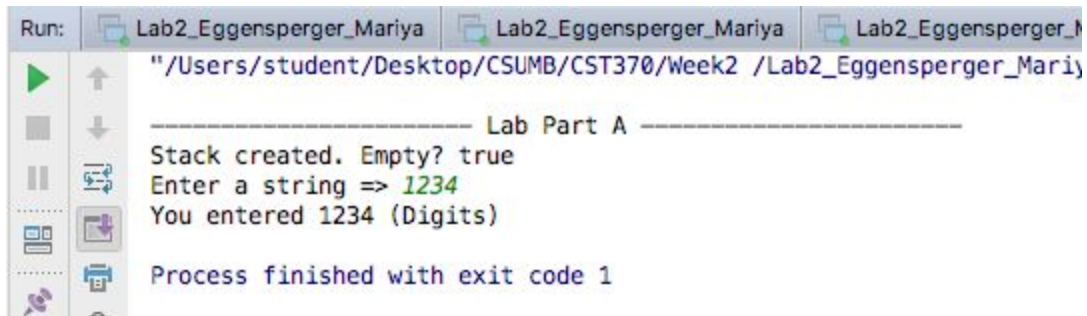


```
Run: Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya  
"/Users/student/Desktop/CSUMB/CST370/Week2 /Lab2_Eggensperger_Mariya  
----- Lab Part A -----  
Stack created. Empty? true  
Enter a string => csumb  
You entered CSUMB (Letters)  
Reverse is BMUSC
```



```
Run: Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya  
"/Users/student/Desktop/CSUMB/CST370/Week2 /Lab2_Eggensperger_Mariya  
----- Lab Part A -----  
Stack created. Empty? true  
Enter a string => mariya  
You entered MARIYA (Letters)  
Reverse is AYIRAM
```

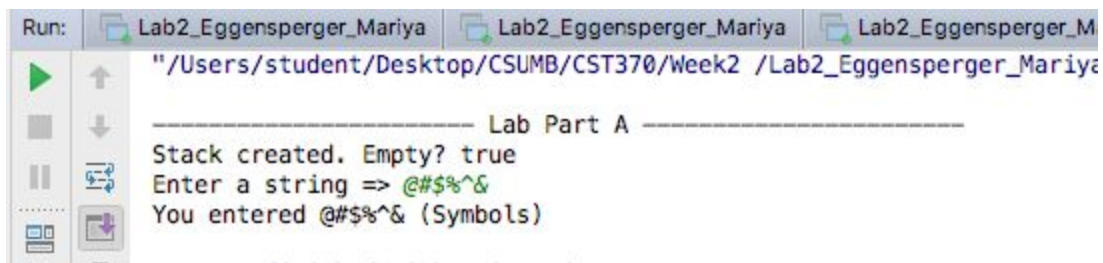
Figure 2 This is the second run for the character input program. The program first checks whether a new stack was created and that it is empty for input. However, if the user inputs digits or symbols, the program is terminated with a code `exit(1)`, since these are not valid character inputs.



```
Run: Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya
"/Users/student/Desktop/CSUMB/CST370/Week2 /Lab2_Eggensperger_Mariya

----- Lab Part A -----
Stack created. Empty? true
Enter a string => 1234
You entered 1234 (Digits)

Process finished with exit code 1
```



```
Run: Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya Lab2_Eggensperger_Mariya
"/Users/student/Desktop/CSUMB/CST370/Week2 /Lab2_Eggensperger_Mariya

----- Lab Part A -----
Stack created. Empty? true
Enter a string => @#$$^&
You entered @#$$^& (Symbols)

Process finished with exit code 1
```

(b) Download Stack.h, Stack.cpp, and Sample_Stack_tester.cpp from iLearn.

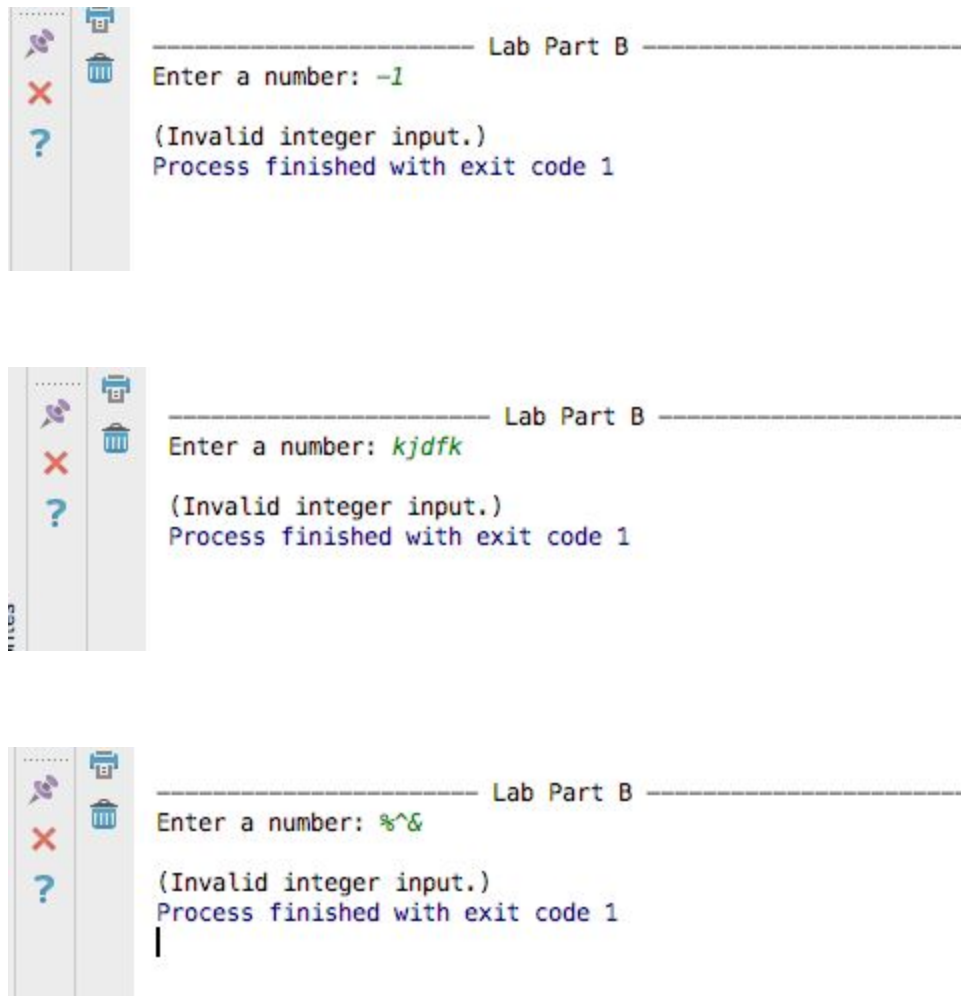
Make a project with the three files. Then, modify the programs so that the program can convert a positive integer to a binary representation. The output of your program should look as follows:

```
Enter a number: 5
Decimal: 5
Binary: 101
```

This is another sample execution:

```
Enter a number: 26
Decimal: 26
Binary: 11010
```

Figure 3 As per lab instructions, the program requests that that user input a set of **positive** integers. If the user inputs a set of negative integers, letters or symbols, an error message is displayed and the program terminates with exit(1).



The figure consists of three vertically stacked screenshots of a terminal window. Each screenshot has a vertical toolbar on the left with icons for a magnifying glass, a red 'X', a blue question mark, a printer, and a trash can. The terminal output for each screenshot is as follows:

```

----- Lab Part B -----
Enter a number: -1
(Invalid integer input.)
Process finished with exit code 1

----- Lab Part B -----
Enter a number: kjdfk
(Invalid integer input.)
Process finished with exit code 1

----- Lab Part B -----
Enter a number: %^&
(Invalid integer input.)
Process finished with exit code 1
|

```

Figure 4 Only after a valid input is made with positive integers, the program runs an algorithm to convert the integer to a binary representation. The algorithm is such that, while *number* is not equal to zero, calculate the *remainder* that results when the *number* is divided by 2. Place the *remainder* on the top of the *stack_of_remainders* and replace the *number* by the integer quotient of *number*/2. End while. Then, while the stack of remainders is not empty, retrieve and remove the *remainder* from the top of the stack of remainders and append the *remainder to the output* produced. End while. Below are two runs as instructed in the lab.



Lab Part B

Enter a number: 5
Decimal: 5
Binary: 101



Lab Part B

Enter a number: 26
Decimal: 26
Binary: 11010