Policies and Guidelines

- Assignment deadlines come with a ***24 hours grace period***. If you submit the assignment during the grace period, it will be accepted without penalty. Submissions after the grace period will be accepted with a ***50% penalty until the last day of classes on Wednesday, May 11, 2022, at 11:59 pm***.
- Every lab has required submission guidelines. Please read the submission requirements carefully. Any deviations from specifications on lab projects will result in point deductions or incomplete grades.
- This lab assumes you are using the [bingweb.binghamton.com] server. Although you may work on your computer, we cannot troubleshoot system-specific issues. You will need to make the necessary changes in the lab description for your system (i.e., vim is a Unix editor, so replace it with Notepad on Windows).
- If your code does not run on the bingweb Linux server, it is considered non-functioning, ***even if it runs on your personal computer***. Always check that your code runs on the bingweb Linux server before submitting.
- To get help, please attend the lecture and your lab section. During the lecture, I will teach the course material and answer questions. Lab sections are the UCA's office hours to answer lab-related questions. Although we will answer questions, provide clarification, and give general help where possible, we will not help you debug specific code within 24 hours of the deadline. We will not provide any help after the deadline.
- ***Instructors will not 'fix' your code to see what works***. If your code does not run due to a syntax error, it is considered non-functioning.
- You cannot resubmit 'fixed' code after the deadline, regardless of how small the error is.
- When you run an SSH client, you get a terminal window logged in to the University's Linux server. Keep in mind that others in the University may need to use these same resources, so please use these resources wisely. You should also follow the rules and restrictions associated with the server.
- In general, an SSH terminal in which you use simple line-oriented commands requires minimal resources from a server. More complicated or buggy commands, such as programs caught in endless loops, may consume significantly more resources. Please try to be careful and interrupt and cancel (e.g., Ctrl+C) programs that behave badly.

Academic Honesty Expectations and Violation Penalty

- ***This lab is an individual lab project***. You must do the vast majority of the work on your own. It is permissible to consult with classmates to ask general questions about the

project, help discover and fix specific bugs, and talk about high-level approaches in general terms. ***It is not permissible to give or receive answers or solution details from fellow students***.

- You may research online for additional resources; however, ***you may not use code that was written specifically to solve the problem you have been given***. ***You may not have anyone else help you write the code or solve the problem***. You may use code snippets found online, providing that they are appropriately and clearly cited within your submitted code.
- ***If you use a distributed version control service such as GitHub to maintain your lab project, you should always set your project repository to private***. If you make your lab project repository public during the semester or even after the semester, it may cause a violation of the academic honesty policy if other students find and use your solution.
- If you are involved in plagiarism or cheating, you will receive a score of "0" for the involved project for the first offense. You will receive a grade of "F" for the course and will be reported to the university for any additional offense.
- Students are required to strictly follow the rules and guidelines laid out in the Watson School Student Academic Honesty Code at the following link: http://www.binghamton.edu/watson/about/honesty-policy.pdf
- Please also review Computer Science Faculty Letter to Students Regarding Academic Honesty at the following link: http://www.cs.binghamton.edu/~ghyan/courses/CSHonestyLetterToStudents.pdf
- Cheating and copying will **NOT** be tolerated. Anything submitted as a programming assignment must be the student's original work.
- We reserve the right to use MOSS (https://theory.stanford.edu/~aiken/moss/) to detect plagiarism in the programming assignments.

Lab-3 Setup

Prepare Your Development Environment

Follow the below instructions to set up your development environment:

- Whether you are on-campus or off-campus, please follow the instructions in the following link to connect to the university's network through "SSL VPN": https://www.binghamton.edu/its/about/organization/operations-and-infrastructure/networking/off-campus-connecting.html
- Binghamton University students can use a facility called a "Secure Socket Shell" or SSH for short in order to log on to a Binghamton University Linux server machine using a terminal window on their machines. For this, they will need software to enable opening an SSH window. Please follow the instructions below for your operating system:

    - **Windows:** If you are on a Windows machine, a free product called MobaXterm is available for you to use. Other SSH client software is available, but MobaXterm is easy, and everything you need is available in the free edition. Download and install the free edition from https://mobaxterm.mobatek.net/. Once you have installed MobaXterm, start it, and create a new session by clicking on the "Session" icon. Then, start an "SSH" session by specifying the Binghamton University Server [bingweb.binghamton.edu] as the remote host and your PODS userid as the username. Leave the "Port" at the default of 22. None of the advanced settings need to be modified.
    - **Mac OS and Linux:** If you are on a Mac OS or Linux machine, an ssh client is already installed on your computer. Open a terminal and type the below command to access the Linux Server, where <userid> is your PODS userid. The first time you do this, ssh will ask if you accept the host's key fingerprint. I say "yes" at this point. After the first time, this question will not appear:
      ssh <userid>@bingweb.binghamton.edu
- Your account on the Linux server uses a U-Drive that is assigned to you by the University. I find it very useful to mount your network U-Drive on your machine. This way, you can edit files locally, using your local editor, and compile/run on the Linux server using the SSH window. Please follow the instructions in the following link to mount your network U-Drive: https://www.binghamton.edu/its/helpdesk/u-drive.html
- Feel free to use the editor of your choice.

Create "lab-3" directory on U-Drive

- Open a new terminal window to SSH into your account on the [bingweb.binghamton.com] server. When logged in, make sure you are in your home directory by typing the following command if you are not already there. The "~" is just a shortcut symbol that means home directory:
  cd ~
- Create a directory named "lab-3" by typing:
  mkdir lab-3
- Change your current directory to the "lab-3" directory by typing:
  cd lab-3
- Confirm that you are in the "lab-3" directory by typing:
  pwd
- Download Lab-3 files from Brightspace and move them to the “lab-3” directory on your U-Drive.

Lab-3 Content

Lab-3 consists of six files. Three files are provided to you, and the other three you should implement. The files are:

- lab3.pdf (provided)
- README.md (provided)
- lab3.c (provided)
- lab3staticstack.h (should be implemented)
- lab3staticstack.c (should be implemented)
- lab3.txt (should be implemented)

Lab-3 Implementation Guidelines

You should write your code in “lab3staticstack.c” and “lab3staticstack.h”. ***Do not write your code in the "lab3.c" file. This file should only include the “main()” function and any other function used to test your implementation. We will NOT grade this file. We will use our version of this file, which includes different and more comprehensive tests, to test your code***.

To compile, make sure all files are in the "lab-3" directory, then run the command:
gcc lab3.c lab3staticstack.c -o lab3.out

Because header files are included in the source “.c” files, they are not included in the "gcc" command to get compiled.

To run “lab3.out”, type the command:
./lab3.out

Implementation

Header File

Add a header file to your project directory “lab-3”. To do this, follow the following steps:

- Create a new header file in your project directory called "lab3staticstack.h".
- Add include guards to the header file.
- Copy and paste the following to your header file:
  ```
  #include <stdio.h>
  #define STR_SIZE 128
  #define STACK_SIZE 64
  ```

Source Code File

In Lab-3, you should implement six functions in “lab3staticstack.c” and declare them in “lab3staticstack.h”. Each of these functions is described below. Now, begin by adding a new source code file to your project directory “lab-3”. To do this, follow the following steps:

- Create a new source code file in your project directory called "lab3staticstack.c".
- Include the header file “lab3staticstack.h” in the source code file “lab3staticstack.c”.
- Copy and paste the following lines to your source code file “lab3staticstack.c” after the include statement. These lines will define the “stack” array and the “top” variable as static global variables in your source code file and limit access to them from other source code files:
  ```
  static char stack[STACK_SIZE] = "";
  static int top = -1;
  ```
- Implement the functions described below in “lab3staticstack.c”.

In Lab-3, we will implement a program that:

- Read a string of characters from an input file.
- Add and remove the string characters to a static stack implementation.

The header file defines a preprocessor directive “STR_SIZE”, which is the maximum number of characters the program accepts as an input. It also defines “STACK_SIZE”, which is the maximum number of characters that can be added to the stack.

void readFile(char filename[], char buffer[])

You can use your Lab-2 implementation of this function.

Implement the function "*void readFile(char filename[], char buffer[])*", which reads a string from an input file “filename”, and saves it in an array of characters “buffer”. The function implementation should adhere to the following requirements:

- The function should open the file in reading mode “r”.
- The function should read the file character-by-character using the “fgetc()” function.
- The function should stop reading the file if any of the following holds:
  - The end of the file is detected.
  - The end of the line is detected.
  - The “buffer” array has been filled with “STR_SIZE - 1” characters.
- The function should add the termination character “\0” to the end of the string in the array “buffer”.
- The function should close the file when the reading is finished.

int push(const char c)

Implement the function "*int push(const char c)*", which adds the constant character “c” to the top of the stack. The function should return one of the following integer values:

- **“–1”:** if the stack is full and, therefore, the character can’t be added to the stack.
- **“0”:** if the character is successfully added to the top of the stack.

char pop()

Implement the function "*char pop()*", which removes and returns a character from the top of the stack. The function should return one of the following character values:

- **“\0”:** if the stack is empty and, therefore, no character is available on the stack to be removed and returned. In this case, the null character “\0” should be returned.
- If the stack is not empty, then the character at the top of the stack should be removed and returned. The next character on the stack should become the new top of the stack.

int status()

Implement the function "*int status()*", which returns the status of the stack. The function does not remove any character from the stack. It should only return one of the following integer values:

- **“-1”:** if the stack is empty.
- **“1”:** if the stack is full.
- **“0”:** if the stack is neither empty nor full.

char viewTop()

Implement the function "*char viewTop()*", which returns the character at the top of the stack. The function does not remove any character from the stack. It should only return one of the following character values:

- **“\0”:** if the stack is empty and, therefore, no character is available on the stack to be returned. In this case, the null character “\0” should be returned.
- If the stack is not empty, then the character at the top of the stack should be returned. No character should be removed from the stack.

void writeFile(char filename[])

Implement the function "*void writeFile(char filename[])*", which writes the current stack content to an output file “filename”. The function should write the stack content (characters) using Last In First Out (LIFO) order. The function implementation should adhere to the following requirements:

- The function should open the file in appending mode “a”.
- The function should not make any changes to the stack.
- The function should add a new line to the file before writing the stack content.
- The function should write the stack content to the file character-by-character using the “fprintf()” function in Last In First Out (LIFO) order.
- The function should add the character “|” after each character is written to the file. However, the character “|” should not be added after the last character on the stack is written to file.
- The function should close the file when the writing is finished.

For example,

- If the stack has the following characters on it, where the “s” character is the top of the stack
    - CS212: Programming II for Engineers
- The output file content should be
    - s|r|e|e|n|i|g|n|E| |r|o|f| |I|I| |g|n|i|m|m|a|r|g|o|r|P| |:|2|1|2|S|C

Save your program, compile and run it, and fix errors if you get any.

Valgrind

Run your code with Valgrind on the Linux server, and ensure you have no memory errors.

To run your code with Valgrind, make sure all files are in the "lab-3" directory, then compile your code using this command:
gcc -g lab3.c lab3staticstack.c -o lab3.out

Now run “lab3.out” with Valgrind using this command:
valgrind --tool=memcheck --leak-check=full ./lab3.out

If you have any memory errors, correct them before submitting.

Submission

- Complete the "README.md". You may write “n/a” where applicable (bugs, references, etc.). There is no need to keep the "< >" symbols---they’re just there to signify that you should replace the text there.
- Please only submit the following files for Lab-3 on Brightspace using the same file name and extension as here:
    - lab3staticstack.h
    - lab3staticstack.c

    - README.md
- You must submit your lab before the deadline (including the 24 hours grace period) to be considered on time. Otherwise, it will be considered late even if your lab was completely working before the deadline. ***Please note that on Brightspace, we maintain all your submissions but only grade your most recent submission.***
- That's it! We've completed our work for this lab.

Grading Rubric

Total: 100 points

- Header file “lab3staticstack.h”: 5 points (2.5 points if partially completed)
- Compile and run without issues: 5 points (2.5 points if only compiles, but crashes when run)
- Pass Valgrind with no memory errors: 5 points (No partial points)
- readFile(): no points (from Lab-2).
- push()
    - Implementation: 10 points
    - Tests: 10 points
- pop()
    - Implementation: 10 points
    - Tests: 10 points
- status()
    - Implementation: 5 points (No partial points, No points if tests fail)
    - Tests: 5 points (No partial points)
- viewTop()
    - Implementation: 5 points (No partial points, No points if tests fail)
    - Tests: 5 points (No partial points)
- writeFileReverse()
    - Implementation: 10 points
    - Tests: 10 points
- Submission: 5 points (2.5 point if partially completed)