UAH Electrical and Computer Engineering Department
Project03 (3 points) Version 1.0

Submit Your Solution to the Canvas Dropbox by 5:00pm on Friday, February 22, 2019

Important Note

All CPE 212 projects are *automatically graded* in order to provide you timely feedback. So, it is critical that you follow all directions for the preparation and submission of your projects for grading.

Failure to follow the directions may result in zero credit (0 points).

Hint

Match the output of your program to that of the sample solution!!!
Directions for running the sample solution appear on the following pages.
Directions for running the preview script appear on the following pages.

Project03 Goals

A fundamental software engineering skill is the design, implementation, and testing of a software component that must be integrated into a larger software product. The software component must conform to a previously agreed upon interface format. As part of this assignment, you will practice this skill by completing the implementation of **two stack classes** (**MiniStack** and **BigStack**) that will be integrated into a larger software product provided by the instructor. Along the way you will review basic C++ programming and Linux file management skills required for successful completion of CPE 212.

Project03 Overview

For this project, you will ***complete*** the provided partial C++ program that utilizes classes to implement a stack. The **MiniStack** class is an array-based implementation of the Stack Abstract Data Type (ADT) where the array size is limited to five elements of type **char**. **MiniStack** objects are used by **BigStack** to create a doubly-linked-node, stack-of-**MiniStacks** implementation of the Stack ADT. So with this project, you practice implementing both an array-based stack (**MiniStack**) and a linked-node stack (**BigStack**).

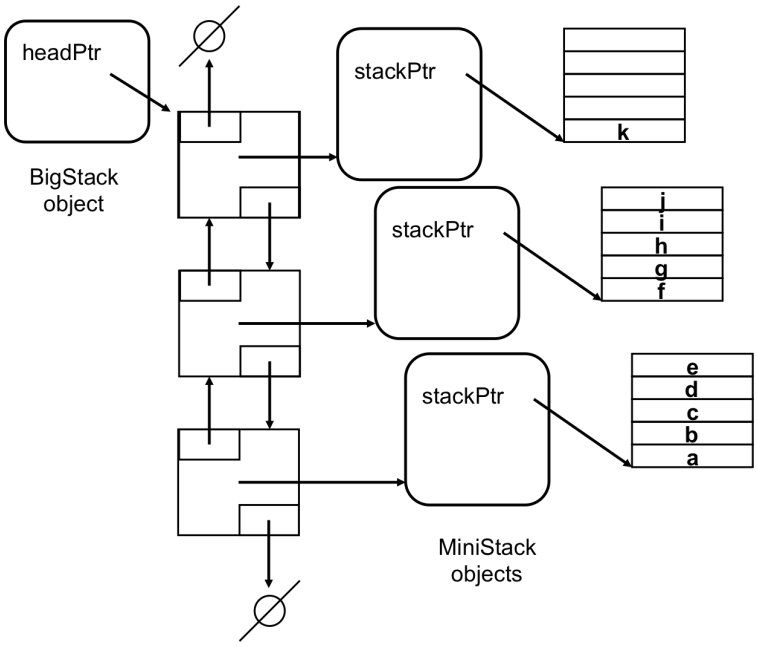
The **Project03_Materials.zip** file includes four source code files and a **makefile**.

Filename	Project03_Materials.zip	Description
main.cpp	Included	Driver program to test **MiniStack** and **BigStack** classes
ministack.h	Included	Specification file for **MiniStack** class
ministack.cpp	**Write and submit this file**	Implementation file for **MiniStack** class
bigstack.h	Included	Specification file for **BigStack** class
bigstack.cpp	**Write and submit this file**	Implementation file for **BigStack** class
p03input1.txt	Included	Tests class **MiniStack** member functions
p03input2.txt	Included	Tests class **BigStack** member functions (assuming **MiniStack** works)
p03input3.txt	Included	Tests class **BigStack** member functions (assuming **MiniStack** works)

MiniStack operates as a regular array-based stack object, and one may develop and test it in isolation from **BigStack** provided that the **BigStack** function stubs are in place. As data values are pushed onto

BigStack, it must create a new node that will use a **MiniStack** object to store the incoming data. Once that **MiniStack** object is filled to capacity, **BigStack** will have to create a new node to store any additional values pushed onto **BigStack.**

In the figure below, the character '**a**' was the first character added to the **BigStack** object (it is now at the bottom of the **BigStack**), and the character '**k**' was the last character added (it is now at the top of the **BigStack**). So, four more characters could be pushed onto this **BigStack** object before another node/MiniStack object is needed. Note that **BigStack** nodes have both **nextPtr** and **previousPtr** links.



Getting Started

- Create empty function stubs for **MiniStack** and **BigStack** member functions.
- Use **make** to compile your framework.
- Once the framework compiles and executes, implement the member functions of **MiniStack**
- Use **p03input1.txt** to verify correct operation of just **MiniStack**.
- Now implement the member functions of **BigStack**.
  Note that as a client of **MiniStack,** the class **BigStack** will have to utilize (call/invoke) the public member functions of **MiniStack** to store and manipulate data in the array linked to each **MiniStack** object. So, be sure **MiniStack** is fully functional first.
- Use **p03input2.txt** and **p03input3.txt** to verify correct operation

Hints

-- The code for an array implementation of the stack ADT is in your notes!!! You will need to modify the code for the linked-list implementation to include both **nextPtr** and **previousPtr** links.
-- Implement **MiniStack** first and test it thoroughly before attempting **BigStack**
-- If **MiniStack** does not work, **BigStack** will not work
-- The outputs of **MiniStack** are grouped within **( )** to help you visualize where the pushed characters have been stored. The output of **BigStack** is bracketed by **[ ]** and may consist of zero or more **MiniStack** ( ) groupings.

Step #1 - Unzipping Project Materials on blackhawk

Use the Firefox browser to access Canvas and download the **Project03_Materials.zip** file into your **Project03** directory. At terminal window prompt, use the unzip utility to uncompress the files. For example, to unzip the files into your current directory:

```
unzip  Project03_Materials.zip
```

Since this project is worth three points, you have been given three input files to test your program.

Step #2 – Create Function Stubs for the Missing Support Functions

Open a Linux terminal window and navigate to the directory containing the **Project03** materials downloaded from Canvas (you should be there already if you just completed Step #1 above).

Use the following command to create the missing files by typing the following at the Linux prompt. Linux is case sensitive so pay attention to upper versus lower case letters.

```
gedit  ministack.cpp  bigstack.cpp  &
```

This command starts the **gedit** text editor running as a background process (**&**). With the editor running in the background, you may use the same terminal window to compile and test your program.

-- Add stubs for **MiniStack** and **BigStack** functions to **ministack.cpp** and **bigstack.cpp** and save the files.

In the terminal window, compile your program by typing the word **make** at the prompt and press the Enter key. If you see error messages, use the text editor to make corrections to the function stubs and switch back to the terminal window to type **make** again to recompile your program. Once your program compiles, you may execute it by typing: `./project03  inputfilename`

Now that your program compiles and executes using function stubs, you may implement and test your solution function by function. Start with the functions associated with opening files and loading data.

Running the Sample Solution on blackhawk [shows the desired output for specified input file]

The best description of what your code should do is the Sample Solution for the project.

Run the sample solution by typing the following at **blackhawk** terminal window command prompt where `inputfilename` is the name of one of the provided input files (for example, p03input1.txt).

```
/home/work/cpe212/project03/p03  inputfilename
```

Your current working directory must contain the input files for this to work.

Running the Preview Script on blackhawk [analyzes outputs for all provided input files]

Run the preview script by typing the following in a **blackhawk** terminal window command prompt

```
/home/work/cpe212data/project03/preview03.bash
```

This script will run both the **Sample Solution** AND your **project03** executable program on the complete set of input files, and it compares the outputs of the two programs line by line to identify errors in your program's outputs. Make sure that the output of your program exactly matches the output of the Sample Solution.

- ***To use the preview script, your executable must be on blackhawk***
- ***The sample input files must be in your current working directory BEFORE you execute the preview script!!!***