CSc 300 – Assignment #2 – Gamradt – Due: 10-06-23 (Late: 10-13-23)

Create a user-defined Abstract Data Type (ADT) name **Stack**

- Use an appropriate set of C++ header/implementation files as discussed in class
- **Stack** is implemented as a **dynamically allocated array**
  - See **C++ Pointers** under D2L Lecture Notes
- **Stack** consists of 0 or more **Element** values
  - **Element** is an exportable standard **string** data type

The **Stack** ADT must define and implement the following data types and operations.

- Do not add to or modify the public interface (exportable components – public components).
- Do not add to or modify any attributes or data types (storage components).

<u>Exportable Operations: (declared .h file and defined .cpp file)</u>

Stack	default constructor function – creates an initialized empty stack – size 2 **(+)**
Stack	parameterized constructor – creates an initialized empty stack – size user specified **(+)**
Stack	copy constructor – creates a duplicate copy of an existing stack **(*)** reuse the pop and push standard methods when populating the new stack
~Stack	destructor function – removes all elements from the stack stack instance state before going out of scope – initialized empty stack reuse the pop standard method when emptying the existing stack
push	inserts a new element to the top of the stack
pop	removes an existing element from the top of the stack
peek	access an existing element from the top of the stack **(*)** state of stack is not altered reuse the pop and push standard methods when populating the new stack
view	displays the contents of the stack from the top to the bottom **(*)** view function uses a non-destructive implementation reuse the pop and push standard methods when populating the new stack

(+) Merge into 1 constructor function using the techniques discussed in class
(*) Before an element can be accessed and processed it must first be removed from the top of the stack

<u>User-Defined Data Types:</u>
Element

Stack Required Output Format: (view)

```
TOP -> BOTTOM                                    // Output for an empty Stack instance
TOP -> CSc -> 300 -> Data -> Structures -> BOTTOM   // Output for a populated Stack instance
```

Required header file (.h). **// only partially specified**

```
//  General description of the ADT and supported operations – exportable operations only
//  Do not include any implementation details

#ifndef _STACK_H                                        // Guard
#define _STACK_H

#include <iostream>

typedef std::string Element;
typedef Element * ElementPtr;

class Stack {
        public:                                         // exportable
//  General description of each of the ADT operations/methods/functions – exportable operations only
                Stack();                                //  merge into 1 using the techniques
                Stack( const int );                     //  discussed in class
                Stack( Stack & );
                ~Stack();
                void push( const Element );
                Element pop( );
                Element peek( );
                void view( );
        private:                                        // non-exportable
//  No private member documentation – implementation details are hidden/abstracted away
                const int STACK_SIZE;                   // requires initialization
                ElementPtr stackArray;
                int top;
};

#endif                                                  // Guard
```

Stack ADT include sequence: // Never include .cpp files

$$\begin{tikzcd}
\text{main.cpp} \arrow[r] & \text{Stack.h} & \text{Sack.cpp} \arrow[l]
\end{tikzcd}$$

Stack ADT incremental building sequence: // Using make

1. Place all files in the project folder // I would use Gamradt2
2. make // Process Makefile – generate executable
3. ./output

Make sure that you completely document the header/implementation files.

- The header (.h) file tells the user exactly how to use your ADT
  - General descriptions only – do not include implementation details
- The implementation file (.cpp) tells the implementer/programmer exactly how the ADT works
  - Detailed descriptions – include implementation details
- See **Documentation Requirements** – D2L Handouts Folder

I will write a test program that will include your **Stack** ADT so all header/implementation files tested must use common names. You **MUST** use:

- the **EXACT** same names for each data type and function in the header/implementation files
- the **EXACT** same function argument sequence in the header/implementation files

Use **PITA** everywhere possible

- Prefer Initialization to Assignment

Remember that a stack uses the basic operations of **push** and **pop** to support all additional operations.

- Apply function **Reuse** wherever possible
  - E.g., copy constructor, destructor, peek, view, …

Project Folder:	Lastname2	// I would use Gamradt2
• Stack.h	**Stack** class header file	
• Stack.cpp	**Stack** class implementation file	
• main.cpp	driver program file	// I will use my own
• Makefile	appropriate set of incremental build rules	// "1" module

Push your assignment solution to your GitHub account, then send me a access to the assignment repository

- E.g., CSc300 // CSc300
  - Remember that a 20% reduction is applied for not using GitHub
  - See **Assignment Requirements** – D2L Handouts Folder

List the class number, your lastname, and assignment number as the e-mail message subject:
SUBJECT: csc300 – Lastname – a2 // I would use "… Gamradt …"