HW6
Out: November 1, Wednesday -- DUE: November 9, Thursday 12:00pm
EC327 Introduction to Software Engineering – Fall 2023

Total: 100 points

- *Comment your code and use a clean, easily understandable coding convention. A few coding style guides for your reference:*

http://geosoft.no/development/cppstyle.html
https://google.github.io/styleguide/cppguide.html

Submission:

1. Failure to follow submission guidelines (filenames, program I/O, and others) WILL result in a significant loss of points on the assignment as we use autograders.
2. Completed assignments **must** be submitted on GradeScope.
3. Submit the required files for each question:
   MyString.cpp MyString.h Q2.cpp Stack.h Stack.cpp Q3.cpp
4. Please make sure your code compiles and runs as intended on the **engineering grid**. Code that does not compile will NOT be graded and will receive a 0.

Late Homework Submissions:

Recall the late submission policy and the fixed penalty of 20% for submitting up to two days after the deadline. Also recall that we only grade your latest submission (e.g., if you submit both on time and after the deadline, only your late submission is graded).

Please write C++ code for solving the following problems.

Q1. Files to submit: MyString.h and MyString.cpp -- *Operator Overloading.* **[30 points]**
Implement the following operators on the provided *MyString* class. Please directly update the provided files MyString.cpp and MyString.h.
Operators to overload:

[ ], +, <<, >>, ==

The functionality of these operators should be the same as their functionality in the standard string class (e.g., operator + concatenates two strings).

Write test code to verify the operators are implemented correctly. You need to test the functionality of each operator in a main function, but you won't be submitting your test code**.**

Q2. - Files to submit: Q2.cpp – *Inheritance and Polymorphism* **[30 points]**
An Animal class is provided in Q2.cpp. Derive a Dog class based on the Animal class and implement it in the same file. A Dog object should have the following additional features:

- Another private string data field called *breed*.
- Dog objects should be constructed with a *name, weight, and breed*.
- printInfo function should print: "Dog <name> is a <breed> and weighs <weight> lbs".
- The > operator should be able to compare the weights of two Dog objects and return TRUE only if the first operand Dog object has a larger weight than the second one.

Make the necessary changes to the Animal class such that the Dog class can be derived from it correctly. Animal class data fields, constructors, and function names should not be changed. Comment your changes with "//changed" at the end of each changed line.

Make sure to implement the Dog class as a derived class (parent class is Animal). Your code must work correctly with the test code provided in Q2.cpp. You should expand the test cases as you see fit.

Q3. Files to submit: Stack.cpp Stack.h Q3.cpp -- *Templates and Stack* **[40 points]**
Implement a generic stack as follows:

```
template<typename T>
class Stack
{
   public:
       Stack();                    //constructor
       bool isEmpty() const;       //returns true if stack is empty, false otherwise
       void push(T value);         //add a new item of type T and value value to the
                                   //stack if there is still space on the stack
       T pop();                    //return top element on the stack,
                                   //remove the returned item from stack
       int getSize() const;        //return the size of the stack
   private:
       T * elements;               //pointer to elements of the stack
       int size;                   //current size of stack
};
```

Write a C++ function (that is not a member of Stack) that uses your new generic stack to detect whether the parenthesis order in each given expression is correct. You can use a single stack or multiple stacks. The input string may consist of parenthesis ( ), four operators (+, -, *, /), and integers. Print an error message to console ("Error: Unknown Input") if other unknown characters are used in the expression (printing should be done inside your function). See some examples for correct and incorrect expressions:

(1 + (3 * 5) / (2 - 1)) / 2	Correct
10 * (10 / 3) + 1 (+ 2 * (2 - 7))	Correct
3 + (5 / 2)) * 1	Incorrect: # of left and right parentheses are not matching
)3+ 5(* 2 +20/3	Incorrect: right parathesis ")" appears before any left parenthesis "("

Please use the following function header:

```
//returns true if the parentheses in the expression are correct, false otherwise.
bool parCheck(const string & expression);
```

You do not have to handle expressions with more than 25 elements (an element is a number, operator, or parenthesis). Assume the operands are always positive integers. Note that you do not need to evaluate the expressions; i.e., **you only need to consider the parenthesis ordering**.

Write test code (in your `main()`) that lets the user to enter an expression and prints "Correct Expression" or "Incorrect Expression" on console based on the parCheck function output. Implement the test code and the `parCheck()` function in **Q3.cpp**. Define your Stack class in the file **Stack.h**, and implement it in **Stack.cpp**. Note that `parCheck()` is not a member function of Stack.