PIC 10C Section 1 - Homework # 1 (due Friday, October 1, by 11:59 pm)

You should upload each .cpp and/or .h file separately and submit them to CCLE before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.

Be sure you upload files with the precise name that you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine. Also be sure your code compiles and runs on Visual Studio 2019.

TEMPLATED MAXIMUM HEAP

This homework is a review of basic templates and a chance to implement the maximum heap data structure. In the end, you will submit **max_heap.h** and "Honesty.txt" as described in the syllabus. Failure to provide the Honesty file will automatically result in a zero.

Recall that a **maximum heap** is a data structure that always has the "largest" value at the top. This value can be popped off. We shall allow the user to specify a custom means of sorting the values.

For this homework, the **only header files you are allowed to use** are:

- **utility**
- **vector**
- **stdexcept**

The main purpose of this homework is to review prerequisite material. A visual representation and explanation for how to implement a maximum heap is given in the Data Structures notes from PIC 10B; relevant template information appears in the Template notes from PIC 10B.

Please refer to the syllabus for how the work will be graded: note that more than half of the marks come from good coding practices and code documentation.

Here are the requirements of the class:

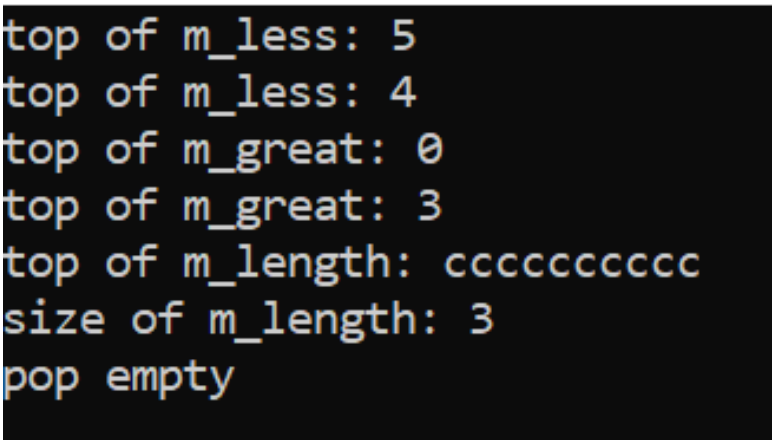
The **max_heap** class must:

- be defined within the **pic10c** namespace;
- be templated by the type of data it stores, **T**, and a comparison operator, **compare_type**, which should be **std::less<T>** by default;
- store a **std::vector<T>** called **values** and a **compare_type** called **pred** as member variables;
- have a single constructor with a single argument such that if no arguments are provided, **pred** will be set to compare_type{} and otherwise uses the value provided as the value for **pred**;
- have an **insert** function that, when passed an object of type **T** (be careful about lvalues and rvalues for efficiency), places the object into the data structure;
- have a **size** function to return the number of elements in the structure;
- have a **top** function that returns the "maximum" value from the heap (as defined by **pred**); and
- have a **pop** function that, when called, removes the "maximum" value from the heap (as defined by **pred**), throwing a **std::logic_error** if the heap is empty at the call.

A test case, code and output, are provided below.

```
#include "max_heap.h"

#include <string>
#include <iostream>

int main() {
  // plain vanilla max heap, does things by <
  pic10c::max_heap<int> m_less;
  m_less.insert(3);
  m_less.insert(4);
  m_less.insert(5);
  m_less.insert(0);

  std::cout << "top of m_less: " << m_less.top() << '\n';
  m_less.pop();
  std::cout << "top of m_less: " << m_less.top() << '\n';
```

```
  // more fancy, replaces < with > so sorting is reversed
  pic10c::max_heap<int, std::greater<int>> m_great;
  m_great.insert(3);
  m_great.insert(4);
  m_great.insert(5);
  m_great.insert(0);

  std::cout << "top of m_great: " << m_great.top() << '\n';
  m_great.pop();
  std::cout << "top of m_great: " << m_great.top() << '\n';

  // lambda that will compare two strings by size
  // you'll need at least C++14 settings for this to compile!
  auto by_length = [](const auto& s1, const auto& s2) {
    return s1.size() < s2.size();
  };

  // this max heap will sort things by the length of the strings it stores
  pic10c::max_heap<std::string, decltype(by_length)> m_length{ by_length };
  m_length.insert("hello");
  m_length.insert("ccccccccc");
  m_length.insert(std::string{});


  std::cout << "top of m_length: " << m_length.top() << '\n';
  std::cout << "size of m_length: " << m_length.size() << '\n';

  pic10c::max_heap<int> uh_oh;
  try { // go ahead, try to do a pop
    uh_oh.pop();
  }
  catch (const std::logic_error & E) { // and we'll catch the exception if it comes up
    std::cout << E.what() << '\n';
  }

  return 0;
}
```


top of m_less: 5
top of m_less: 4
top of m_great: 0
top of m_great: 3
top of m_length: ccccccccccc
size of m_length: 3
pop empty