Homework 2: Templates and the standard library

Instructor: Mehmet Emre

CS 32 Spring '22

Due: 4/6 12:30pm
Name & Perm #: Bharat Kathi (5938444)
Homework buddy (leave blank if you worked alone):
Reading: "Templates and the STL", PS 8.3, 17, 18

1 (10 pts)

In the lecture, we talked about STL being a misnomer. STL was a historical predecessor of parts of the C++ standard library. Nowadays, we would refer to those parts by name. What are the four components of STL that are now part of the standard library?

Answer: .
Algorithms, Containers, Iterators, Functions

2 (8 pts)

Suppose we have a C++ class called `Student`. There are many ways one can create a collection `Student` objects. Among these: one can use a normal C++ array or an STL `vector`, and one could create a collection of objects, or of pointers to objects (presumably allocated on the heap.)

1. (2 pts) Write a line of C++ code that declares an array `a` that can hold 10 objects of type `Student`.

   Answer: .

   ```
   Student a[10];
   ```

2. (2 pts) Write a line of C++ code that declares an array `b` that can hold 10 pointers to objects of type `Student`.

   Answer: .

   ```
   Student* b[10];
   ```

3. (2 pts) Write a line of C++ code that declares a standard library vector `c` that can hold 10 objects of type `Student`.

Answer: .

```
vector<Student> c(10);
```

4. (2 pts) Write a line of C++ code that declares a standard library vector `d` that can hold 10 pointers to objects of type `Student`.

   Answer: .

```
vector<Student*> d(10);
```

3 (20 pts)

In the previous problem, you were asked to write four different declarations, 1, 2, 3, 4. Each of these has "pros" and "cons". There are also differences among them that we might describe as "neutral", for whether they are a pro/con depends on the context. For the "pros" and "cons" below, please indicate which of the options above (1, 2, 3, 4) the statement applies to. Note that in some cases, you may have to indicate more than one number.

Circle the numbers to which the statement applies. If you mess up, cross out all the letters and list the letters in the space to the right.

Hero or zero?	Choices
Con: Additional `#include` directives are required to use this approach.	– – 3 4
Pro: The item declared can be expanded beyond 10 items if needed.	– – 3 4
Pro: You can declare this item without actually creating any Student objects.	1 2 3 4
Con: The class MUST have a default constructor, or this declaration is not permitted.	– – – –
Pro: All space for the item and the Students it contains are co-located in memory– on the stack if the item is declared as a local.	1 2 – –

Answer: The remaining numbers are my selections in the table above.

4 (3 pts)

In your own words, what is the "problem" for which "templates for functions" is a solution? I'm looking for a **brief** description–a single sentence, or at most 2-3 sentences that get to the *central point*, a description of the *problem*, not a detailed description of everything you know about templates, and certainly *not* a sentence copied, word-for-word, from either textbook.

Answer: .

Templates are useful to abstract certain functions that can be used for multiple data types. This way you only need to implement your function once, but are able to use it with any data type.

5 (3 pts)

The C++ syntax for function templates includes `template <class Item>`. The name `Item` in the example above is preceeded by the keyword `class`. One might infer from that that `Item` should be the name of some `class`, e.g. `class Student`, or `class Roster`, etc. But that is not a correct assumption. Explain why **briefly**.

Answer: .

`Item` is just a placeholder class for you to use in your class implementations. Then when you actually use the class and create instances of it, you can pass in whichever class you want to be used inside your template.

6 (6 pts)

Let's say you have a function `int calc(int operand1, int operand2, char op)` where `op` is either `+` or `-`. Instead of overloading this function for each numerical datatype in C++ (e.g. also defining it for `float`, `double`, etc., you want to create a generic function that is far more flexible. In the space below, define `calc` as a template function. If op is neither `+` nor `-` , print an error message on `std::cerr` and call the system function `exit(1);` (you may assume that `#include <cstdlib>` has already been done.)[1]

Answer: .

```
template<class T>
T calc(T operand1, T operand2, char op) {
  if (op == '+') {
    return operand1 + operand2;
  }
  else if (op == '-') {
    return operand 1 - operand2;
  }
  else {
    std::cerr << "invalid operation received, must be + or -";
    exit(1);
  }
}
```

[1]Note that once we've covered exceptions, that would be a better approach than the `cerr/exit` technique.