CS 202 Lab 8

Learning Targets

- I can construct a dynamically allocated array within a class.
- I can construct a templated function.

Grading

This lab is graded on a "pass / needs work" basis. No partial credit is given. You must pass the autograder tests to earn full credit. Labs are **due** the Friday of the week the lab was assigned. Late work can be submitted, but it is not eligible for regrade. If you submit a reasonable attempt at the lab by the due date, you can request a regrade by completing the google form linked in the syllabus. All resubmissions are due by the day before the test that covers the lab at 11:59 PM.

Purpose

The purpose of this lab is to give you the opportunity to modify a program to use a template function, and to modify a class to use a dynamically allocated array.

This repository contains arrayManager.h, arrayManager.cpp, fraction.cpp, fraction.h, main.cpp, helpers.h, helpers.cpp, and makefile. The fraction class **should not be modified**. We will make changes to the other files.

Name your executable `math`

When you are done, you will submit arrayManager.cpp, arrayManager.h, fraction.cpp, fraction.h, helpers.h, main.cpp, and your makefile.

Step 1

Template functions are used when we have multiple functions doing the exact same work with different data types, such as the function overloaded add and multiply functions in helpers.

Templates are a bit tricky when it comes to compilation. We either need to predeclare all possible data types used by the template function, *or* we need to remove all declarations and change them to definitions(either as a prototype are a .h file) and then update the makefile so that an object is not made from the .h file containing the template function.

In this step, we are going to remove all declarations to the add and multiply functions in helpers.h, and replace them with the template function definitions. We will then update the makefile so that the template function properly integrates into main.cpp.

- In helpers.h, delete everything except for the file guards. We want to keep them so that we don't accidentally include the same file multiple times, but we don't need the declarations since we're going to define them in the header file.
- Copy *one* of the function definitions for add in helpers.cpp into helpers.h
- In helpers.h, modify the function definition for add so that instead of taking a specific data type for the array pointer or returning a specific data type, we replace it with a an unknown template type (T). The general syntax for function templates is (**note, the <> here should not be replaced, that top line is the exact syntax you want**):

```
template <typename T>
returnType functionName(dataType1 P1, dataType2 P2){
    //do stuff
}
```

In this example, we would replace anything we're unsure of or anything that consistently changes (such as the return type of add and the parameter type of the array) with T. This means that you will also need to update the function body of add so that instead of sum being a specific data type, it is a generalized type (T). Here's a more concrete example:

```
template <typename T>
T getMax(T P1, T P2){
    T max;
    if(P1 > P2){
        max = P1;
    }
    else{
        max = P2;
    }
    return max;
}
```

- Copy *one* of the function definitions for multiply in helpers.cpp into helpers.h
- In helpers.h, modify the function definition for multiply so that instead of taking a specific data type for the array pointer or returning a specific data type, we replace it with an unknown template type (T), just like we did with add.
- Delete helpers.cpp. If you're scared to do it just yet before compiling, rename it something else.
- Modify your makefile. We can't create an object out of a file that only contains templates, since we don't know their type. That's why we moved the definitions from helpers.cpp to helpers.h. In the makefile, delete the recipe that creates helpers.o. Then, modify the math executable generation recipe so that helpers.o is also removed.
- Make clean (to be safe, because sometimes templates don't update within objects if you don't). Compile your code. You will get 6 undefined symbol errors (or something similar). That's

because we haven't told main.cpp's calls to add and multiply what data types were working with.
- Alter lines 15 - 20 so that all calls to add and multiply have <datatype> after the function name but before the parenthesis for the parameters, like so:

```
int var1=1, var2 = 2;
int max = getMax<int>(var1, var2);
```

- Make clean (again, to be safe) and compile. It should work now. Run it to be sure. If you didn't delete helpers.cpp before, do it now.

Step 2

In this step, we will modify the arrayManager class so that it uses a dynamically allocated array of ints, floats, and Fractions. Remember, if we dynamically allocate an array (or anything else for that matter) in a class, we need to make sure we're doing deep copies in our constructors, and that we have an assignment operator overload and a destructor (Rule of 3).

- In arrayManager.h, delete MAX_ARR_SIZE; we won't need it. Instead we'll track each array's max size with another property.
- In arrayManager.h, change each of the arrays to be array pointers.
- In arrayManager.h declare maxNumInts, maxNumFloats, and maxNumFractions in addition to numInts, numFloats, numFractions. The last 3 track the actual number of items in the named array, while the first 3 track the absolute maximum number of items we can store in the array without having to resize.
- In arrayManager.h, add getters for each of the maxNum properties you just declared. You **do not** need to update the constructor declarations.
- In arrayManager.h , declare a destructor.
- In arrayManager.h, declare an assignment operator overload
- In arrayManager.cpp's default constructor definition, set all of the maxNum properties to be 2.
- In arrayManager.cpp's default constructor definition, allocate memory to the integers, floats, and fractions array using the new keyword, and the maximum size you just set. The general syntax will be (**replace <data type> with the actual datatype)**:

```
property = new <data type>[maxNum];
```

- In arrayManager.cpp's integers parameterized constructor, set the maxNumInts to be the same as the numInts you passed in- this will ensure your array starts without wasted space if this constructor is used. Then, allocate memory to the integers property using the same syntax as in the previous step. Finally, create a loop that runs from 0 to number of integers in the integers array, and copy each item from the passed integer array pointer into the integers array property;
- Repeat the previous step for the float and Fraction parameterized constructor

- In the parameterized constructor that takes all 3 types of arrays and their counts, set the maximumNumInts, maximumNumFloats, and maximumNumFractions to be the same as the passed values for numInts, numFloats, and numFractions. Allocate memory to integers, floats, and fractions using the new keyword and syntax used in the previous 3 steps. Copy all of the passed ints, passed floats, and passed fractions into their respective array properties by copying from 0 to numItems in the array. If you've done this correctly, it will look like you have copied code from the other three parameterized constructors into this constructor.
- In arrayManager.cpp's copy constructor, set your max values using the passed ArrayManager. Then dynamically allocate memory to the integers, floats, and fractions arrays using the maximum values you just copied, using the same syntax as was used in the parameterized constructor. Copy all of the passed objects integers, floats, and fractions into the current object's integers, floats, and fractions arrays by copying from 0 to numItems in the array.
- In arrayManager.cpp, implement the getters for the 3 new max properties that you declared in arrayManager.h
- In arrayManager.cpp, in each of the addToArr methods, do the following:
  - Check if the number of items of that data type (for example, numInts) is equal to or greater than its max (maxNumInts);
  - Increment the max value by 1 (add 1 to maxNumInts, for example)
  - Create a temporary array pointer of that data type (int*, for example) and dynamically allocate memory to the array using the new keyword and the max size you just incremented (see general syntax above if you don't remember how)
  - Copy all of the items in the array of that data type (integers for example) into the temporary array by iterating from 0 to the number of items in the array that you're copying from.
  - Free the integers array by using the delete keyword, square brackets, and the array of that data type, like so:

```
delete [] integers;
```

  - Assign the array of that data type to be the temporary pointer, so that we have all of our elements copied into the larger array stored at the property's pointer.
  - Insert the passed item into the array property of that datatype at the number of items in the array and increment the number of items in the array to update how many items are stored in it.
- In arrayManager.cpp, implement the destructor. If there are more than 0 items in the array, delete it. You should have 3 if statements that check (one for integers, floats, and fractions), and if there are more than 0, free the array memory using the syntax from the previous step.
- In arrayManager.cpp, implement the assignment operator overload. You can copy and paste the copy constructor into the bode of the overload, and then return the dereferenced this keyword, just as you did in previous labs.
- Compile and run your code. The values output by the program should be the exact same as the were before making any changes to your code:

```
Integer Array Sum: 10
Float Array Sum: 5
Fraction Array Sum: 10/2

Integer Array Product: 1
Float Array Product: 0.000976562
Fraction Array Product: 1/1024
```

Run tests to be sure you pass all tests.

Once you are passing all tests, compress your .h .cpp, and makefiles into a .zip or .tar.gz, and submit them on WebCampus.

Remember: the labs are step by step instructions meant to prepare you for your programming assignments. While you are allowed to resubmit labs, it's better to finish them as they are assigned so that you don't struggle with the programming assignments and tests.