University of Illinois at Urbana-Champaign
Department of Computer Science

# MP2: Advanced C++

CS 225 Data Structures
Spring Semester, 1999

**Handed out: Friday, February 5, 1999**
**Due date: Thursday, February 11, 1999 at 11:59 PM**

## 1 Introduction

In this MP, you will mainly focus on C++ syntax. First, you will gain some practice with the `String` and `Array` classes you saw during your second discussion section. Second, you will both write and use template functions and a template class. Finally, you will also edit a provided `Makefile` slightly, in order to get some more practice with writing a `Makefile`.

## 2 The programming assignment

To begin with, you will need to copy the given files into your own directory. The files are located in:

```
~cs225/src/mp2        (MP code)
~cs225/src/mp2/test   (test files)
```

### 2.1 The given code

The first part of the coding assignment involves writing a template class which will be called `NamedItem`. The idea behind this class is that there are situations where you have a collection of objects of some type, and you might perhaps want to assign a name to each of those objects. So, in order to insure that every object has an associated name, we will place the object and the name inside another, larger object. Our larger object will thus store the name and the smaller object as data members.

Since we don't know what type the smaller object is, we will allow it to be a generic type, which means that the larger object's type must be a template type (since its member data is a generic type).

The specification for the class is as follows:

`NamedItem`

`Private Data Members`

There are three private data members for the `NamedItem` class. First, you have two *pointers to* `String` objects. These two pointers will point to `String` objects holding, respectively, the first and last name of the individual associated with this object. Of course, you can only declare the pointers themselves as member data – the allocation of the `String` objects must wait until the constructor.

The third data member is an *object* of the generic type indicated by your template declaration.

`Public Functions`

There are nine public functions to write for `NamedItem`.

- First, you need a default constructor. This constructor will allocate empty `String` objects for the member pointers to point to. The object of the generic type is automatically initialized with its default constructor and so nothing else needs to be done with it here. (This statement needs to be expanded on, but we will do that after the MP is due.)
- You also want two additional constructors, both of which take three parameters. The first constructor will take two `char*` parameters and a parameter of the generic type, and the second constructor will take two `String` parameters and a parameter of the generic type. The `char*` value or `String` value (depending on the constructor) can then be used to initialize a new `String` object (take note of the constructors of `String`). The first parameter will initialize the first name, the second parameter will initialize the last name, and then the third parameter – the parameter of the generic type – will be used to initialize the generic object of your class.
- Because this class allocates dynamic memory, you will need a copy constructor, a destructor, and an overloaded assignment operator.
- Finally, you want three return functions, called `GetFirstName()`, `GetLastName()`, and `GetItem()`. The first two return constant references to the `String` objects pointed to by the member pointers. The third function will likewise return a constant reference, but to the object of generic type.

## 2.2 Testing the new classes

Next, you will notice that we have provided a `main.C` file. Your next immediate task is to understand what is going on in the code in the `main.C` file. Then, you need to write two template functions that will appear in a separate file. The declarations for these functions will appear in a file called `mp2fns.h`, and the definitions will appear in a file called `mp2fns.C`.

Your first function will be called `RetrieveFromArray` and will take as parameters a `String` object, a constant reference to an "`Array` of `NamedItem` objects of a generic type", and a reference to an object of that same generic type. This function will search the "last name" fields of the objects in the `Array`, and if any of the objects is found to have a last name equal to the `String` object that was passed in as a parameter, then the reference to a generically-typed object, the one passed in as a parameter, will be set equal to the "item" field of this object whose last name matches the `String` parameter. If you do find such an object, eventually you should return 1. Otherwise, you should return 0. So, you are basically searching the `Array` for a last name, and if it is not found, you return 0, but if it is found, you set the third parameter to hold the item associated with that last name, and return 1.

You can see how this function is called near the end of the `main.C` file.

Next, you must complete the second template function that we want you to write. This function takes only one parameter, a `NamedItem` of a generic type. This function should concatenate three `String` objects together: the first name of the `NamedItem` object, the last name of the `NamedItem` object, and in-between them, a single blank space. This concatenated `String` should then be returned (by value, not by reference) as the return type of the function.

You can also see how this function is called near the end of the `main.C` file.

## 3 Altering the `Makefile`

Finally, you need to add some lines to the given `Makefile` so that all the files you are using are compiled for the MP. Right now, the `Makefile` has code to support the compilation of the given files `asserts.*`, `array.*`, and `string.*`. You need to add lines that allow it to compile the remaining files: `named.*`, `mp2fns.*`, and `main.C`. There are a few places you will need to add information to the `Makefile`. You can refer to last week's tutorial to refresh your memory concerning what the various parts of the `Makefile` do, and once you have done this, an examiniation of the given `Makefile` should make it clear what parts still need to be completed or added to. Also, don't forget that the indentation is a `TAB`, and **not!!!!!** eight spaces. That fact is *very* important, because if you use spaces instead of a TAB, the `Makefile` will not work.

## 4 Compiling and testing your code

Once you have written the `NamedItem` class and the two template functions and correctly edited the `Makefile`, your program should compile and produce output that matches the output in the `test.1.std` file.

## 5 Handing in your code

To handin your MP2 code, the first thing you need to do is rename your final `Makefile` to the filename `Makefile.txt`. Then, use the following command to handin:

```
handin cs225 mp2 Makefile.txt named.h named.C mp2fns.h mp2fns.C
```