# Assignment 2 - 100 points

## Part 1

## Assignment Overview

In Part 1 of this assignment, you will write a main program and several classes to create and print a small database of health care provider data. The assignment has been split into two parts to encourage you to code your program in an incremental fashion, a technique that will be increasingly important as the semester goes on.

## Purpose

This assignment reviews object-oriented programming concepts such as classes, methods, constructors, accessor methods, and access modifiers. It makes use of an array of objects as a class data member, and introduces the concept of object serialization or "binary I/O".

## Set Up

1. As in Assignment 1, you should create a subdirectory to hold your files for Assignment 2.
2. In that directory, make a symbolic link to the data file for this part of the assignment:
3. ln -s /home/turing/t90kjm1/CS241/Data/Fall2017/Assign2/providerdb

(Note that this is a platform-specific binary file. It will most likely not work anywhere other than our UNIX system.)

1. In this assignment, you will be creating several source code and header files, as described below. You can create each of these files separately using the nano editor, just as you did on Assignment 1.

This assignment description assumes that you are writing Assignment 2 from scratch. However, the Provider class used in this assignment is largely identical to the class from Assignment 1, so you should be able to reuse some of your code from that assignment.

## Program

For this assignment, you will need to write three source code files as well as two header files. Each of these files is relatively short, but many inexperienced programmers are overwhelmed by the idea of writing a program as multiple files. "Where do I start?!!" is a common refrain. This assignment sheet attempts to walk you through the steps of writing a multi-file program.

The steps outlined below should not be thought of as a purely linear process, but rather an iterative one - For example, work a little on Step 1, then a little on Step 2, then test what you've written (Step 3).

### Step 1: Write the Provider class declaration

The Provider class represents information about a health care Provider. The code for the Provider class will be placed in two separate files, which is the norm for non-template C++ classes.

The header file for a class contains the class declaration, including declarations of any data members and prototypes for the methods of the class. The name of the header file should be of the form ClassName.h(for example, Provider.h for the header file of the Provider class).

A skeleton for the Provider.h file is given below. As shown, a header file should begin and end with [header guards](http://faculty.cs.niu.edu/~mcmahon/CS241/c241man/node90.html) to prevent it from accidentally being #included more than once in the same source code file (which would produce duplicate symbol definition errors). The symbol name used in the header guards can be any valid C++ name that is not already in use in your program or the C/C++ libraries. Using a name of the format CLASSNAME\_H (like PROVIDER\_H in the code below) is recommended to avoid naming conflicts.

#ifndef PROVIDER\_H

#define PROVIDER\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// FILE: Provider.h

// AUTHOR: your name

// LOGON ID: your z-ID

// DUE DATE: due date of assignment

//

// PURPOSE: Declaration for the Provider class, which represents

// information about a health care provider.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class Provider

{

// Data members and method prototypes for the Provider class go here

.

.

.

};

#endif

Data Members

The data members for the Provider class are the same as those used in Assignment 1:

* Provider number (a character array with room for 6 characters PLUS the null character).
* Specialty (a character array with room for 40 characters PLUS the null character).
* Name (a character array with room for 40 characters PLUS the null character). The name will store a concatenated version of the provider's last name, first name, middle initial (if present), and title
* First part of the provider address (a character array with room for 40 characters PLUS the null character)
* Second part of the provider address (a character array with room for 30 characters PLUS the null character)
* City (a character array with room for 20 characters PLUS the null character)
* State (a character array with room for 2 characters PLUS the null character)
* Zip code (a character array with room for 5 characters PLUS the null character)
* Phone number (a character array with room for 14 characters PLUS the null character)

**Note: Make sure you code your data members in THE EXACT ORDER LISTED ABOVE and with THE EXACT SAME DATA TYPES AND SIZES.** The code for this class will be reused on the next assignment, and if you only make the name array 40 characters long instead of 41 or make the city array 31 characters instead of 21, your program for the next assignment will not work correctly.

Method Prototypes

The Provider class declaration should (eventually) contain public prototypes for all of the methods in the Provider.cpp source code file described in **Step 2** below.

### Step 2: Write the Provider class implementation

The source code file for a class contains the method definitions for the class. The name of the source code file should be of the form ClassName.cpp or ClassName.cc (for example, Provider.cpp for the source code file of the Provider class).

The Provider class implementation should (eventually) contain definitions for all of the methods described below. Make sure to #include "Provider.h" at the top of this file.

* Default Constructor

The constructor you wrote for Assignment 1 should work with no modifications required.

* print()

This method is the same as the one from Assignment 1, with one minor change: since the method will not change any data members of the object that calls it, the method should be declared as const.

To declare a method of a class to be const, put the keyword const at the end of the method's prototype and at the end of the first line of its definition. For example:

// Method prototype to be coded as part of the Provider class declaration in Provider.h

void print() const;

// Method definition to be coded in Provider.cpp

void Provider::print() const

{

// Code for print() method goes here

...

}

* getProviderNumber()

This accessor method takes no parameters. It should return the provider number data member. In C++, the usual return data type specified when you are returning the name of a character array is const char\* or "pointer to a character constant" (since returning an array's name will convert the name into a pointer to the first element of the array, which in this case is data type char. This method should be const, since it does not change any data members.

* getSpecialty()

This accessor method takes no parameters. It will have a return data type of const char\*. It should return the specialty data member. This method should be const, since it does not change any data members.

* getName()

This accessor method takes no parameters. It will have a return data type of const char\*. It should return the name data member. This method should be const, since it does not change any data members.

### Step 3: Write the ProviderDB class declaration

The ProviderDB class represents a database of Provider objects. Like the Provider class, the code for this class will be placed in two separate files.

Place the class declaration in a header file called ProviderDB.h. Like the file Provider.h you wrote in Step 1, this file should begin and end with [header guards](http://faculty.cs.niu.edu/~mcmahon/CS241/c241man/node90.html) to prevent it from accidentally being #included more than once in the same source code file.

After the header guard at the top of the file but before the class definition, make sure to #include "Provider.h".

Data Members

The ProviderDB class should have the following two private data members:

* An array of exactly 40 Provider objects
* An integer that specifies the number of providers actually stored in the array

**Note: Once again, make sure you code your data members in THE EXACT ORDER LISTED ABOVE and with THE EXACT SAME DATA TYPES.** If you make the array 41 elements long instead of 40, the input data will not be read correctly.

Method Prototypes

The ProviderDB class declaration should (eventually) contain public prototypes for all of the methods in the ProviderDB.cpp source code file described in **Step 4** below.

### Step 4: Write the ProviderDB class implementation

The ProviderDB class implementation should (eventually) contain definitions for all of the methods described below. Make sure to #include "ProviderDB.h" at the top of this file.

* ProviderDB default constructor

This "default" constructor for the ProviderDB class takes no parameters. Like all C++ constructors, it does not have a return data type.

This constructor is called to create an empty database, so this method should set the number of providers data member to 0.

(If you initialize the number of providers data member to 0 when you declare it, this method's body can be empty. You still need to code the method with an empty body.)

* Alternate ProviderDB constructor

Write a second constructor for the ProviderDB class that takes one parameter: a pointer to a constant character (which will point to an array of characters that contains the name of an existing database file). Like all C++ constructors, this constructor does not have a return data type.

This constructor should do the following:

* 1. Declare an input file stream variable (the code below assumes it is named inFile).
  2. Open the file stream for binary input. The file to open is the one whose name was passed into the constructor as a C string variable. Check to make sure the file was opened successfully as usual.
  3. Read the database file into your ProviderDB object. You can do this with a single statement:
  4. inFile.read((char\*) this, sizeof(ProviderDB));
  5. Close the file stream.
* print()

This method takes no parameters and returns nothing. It does not change any data members, so it should be declared as const

This method should first print a descriptive header line (e.g., "Health Care Provider Listing"). It should then loop through the array of Provider objects and print each of the elements that contain provider data. Here we see some of the power of object-oriented programming: since each element of the array is an object, we can call a method for that object. We've already written a print() method for the Provider class, so printing an element of the array is as easy as calling print() for the array element. The syntax for calling a method using an array element that is an object is pretty straightforward:

arrayName[subscript].methodName(arguments);

### Step 5: Write the main program

The main() function should be placed in a file named assign2.cpp. Since most of the logic of the program is embedded in the two classes you wrote, the main() routine logic is extremely simple.

* Create a ProviderDB object using the alternate constructor you wrote. Pass the filename string "providerdb" as an argument to the constructor.
* Call the print() method for the ProviderDB object.

## Makefile

The makefile for Assignment 2 is given below. Since we now have three source code files and two header files, the number of makefile rules and the size of dependy lists have increased.

#

# PROGRAM: assign2

# PROGRAMMER: your name

# LOGON ID: your z-id

# DATE DUE: due date of program

#

# Compiler variables

CCFLAGS = -Wall -std=c++11

# Rule to link object code files to create executable file

assign2: assign2.o Provider.o ProviderDB.o

g++ $(CCFLAGS) -o assign2 assign2.o Provider.o ProviderDB.o

# Rules to compile source code file to object code

assign2.o: assign2.cpp ProviderDB.h

g++ $(CCFLAGS) -c assign2.cpp

Provider.o: Provider.cpp Provider.h

g++ $(CCFLAGS) -c Provider.cpp

ProviderDB.o: ProviderDB.cpp ProviderDB.h

g++ $(CCFLAGS) -c ProviderDB.cpp

# Header file dependency

ProviderDB.h: Provider.h

# Pseudo-target to remove object code and executable files

clean:

-rm \*.o assign2

## Output

Output for this part of Assignment 2 should be identical to the output from Assignment 1.

## Other Points

* You do not need to submit this part of the assignment, only Part 2.
* Make sure to document your program according to the standards listed in the Course Notes book. In particular, each class method or function should have a documentation box describing its purpose, the input parameters (if any), and the return value (if any). There should also be a documentation box for the program as a whole.

# Assignment 2 - 100 points

## Part 2

## Assignment Overview

In Part 2 of this assignment, you will add some functionality to the ProviderDB class and add some logic to your main() routine to test that functionality.

## Purpose

This part of the assignment introduces the insertion sort algorithm.

### Step 1: Add methods to ProviderDB

Add the following three methods to the ProviderDB class

* sortByProviderNumber()

This method takes no parameters and returns nothing.

This method should sort the array of Provider objects in ascending order by provider number using the [insertion sort algorithm](http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/insertion_sort.html).

The sort code linked to above sorts an array of integers called numbers of size size. You will need to make a substantial number of changes to that code to make it work in this program:

* 1. This will be a method, not a function. Change the parameters for the method to those described above.
  2. In the method body, change the data type of bucket to Provider. This temporary storage will be used to swap elements of the array of Provider objects.
  3. In the method body, change any occurrence of numbers to the name of your array of Provider objects and size to numProviders (or whatever you called the data member that tracks the number of valid Provider objects stored in the array).
  4. The comparison of providerArray[j-1] and bucket will need to use the C string library function strcmp() to perform the comparison. Also, you'll need to use the getProviderNumber() method to access the providerNumber data member within each Provider object. The final version of the inner for loop should look something like this:
  5. for (j = i; (j > 0) && (strcmp(providerArray[j-1].getProviderNumber(), bucket.getProviderNumber()) > 0); j--)
  6. ...
  7. It is legal to assign one Provider object to another; you don't need to write code to copy individual data members.
* sortBySpecialty()

This method takes no parameters and returns nothing.

This method should sort the array of Provider objects in descending order by specialty using the insertion sort algorithm.

* sortByName()

This method takes no parameters and returns nothing.

This method should sort the array of Provider objects in ascending order by name using the insertion sort algorithm.

### Step 2: Add some method calls to the main program

The main() routine logic will now look like this:

* Create a ProviderDB object using the alternate constructor you wrote. Pass the filename string "providerdb" as an argument to the constructor.
* Call the print() method for the ProviderDB object.
* Call the sortByProviderNumber() method for the ProviderDB object.
* Call the print() method for the ProviderDB object.
* Call the sortBySpecialty() method for the ProviderDB object.
* Call the print() method for the ProviderDB object.
* Call the sortByName() method for the ProviderDB object.
* Call the print() method for the ProviderDB object.

## Comparison Functions and Function Pointers (10 points extra credit)

The basic version of this assignment outlined above asks you to write three different sort methods, all of which are practically identical. This section describes a more sophisticated (and more difficult) alternate approach to sorting that allows you to use a single generic sort function. This approach is optional, but will be worth extra credit if you are able to get it to work.

A C++ function definition is stored somewhere in the computer's memory. The location of the function is called the function's address. When a function is called, your C++ program branches to the address of that function and begins executing the statements in the function body.

A pointer in C++ is a variable that holds the address of another thing. Generally this "other thing" will be another variable, but it can also be a function. A pointer to a variable allows you to access the value stored in that variable using the pointer. A pointer to a function allows you to call that function using the pointer.

A declaration for a function pointer looks like this in C++:

return-type (\*pointer-name)(parameter-data-types)

So, for exammple, a declaration for a pointer to a function that takes two integers as arguments and returns a Boolean value would look like this:

bool (\*compare)(int, int)

You can obtain the address of a function using the unary & ("address-of") operator. The address can be assigned to a function pointer or (more commonly) can be passed to a function and copied into a function pointer parameter:

compare = &myFunction;

Once the pointer compare contains the address of a function, we can use it to call the function - just use the pointer name exactly like a function name:

if (compare(num1, num2))

{

// Code to execute if the function pointed to by compare returns true

}

Let's say that we want to sort an array of Student objects in ascending order by the data member name using the selection sort algorithm. If the sort code is defined in a method of a Course class that contains an array of Student objects named studentArray and an integer numStudents that specifies the number of objects in the array that have been filled with valid data, the method might look like this:

void Course::sort(bool (\*compare)(const Student&, const Student&))

{

int i, j, min;

Student temp;

for (i = 0; i < (numStudents - 1); i++)

{

min = i;

for (j = i+1; j < numStudents; j++)

{

if (compare(studentArray[j], studentArray[min]))

min = j;

}

temp = studentArray[min];

studentArray[min] = studentArray[i];

studentArray[i] = temp;

}

}

In Course.cpp, we can define a comparison function that will let us check whether one Student object's name member is less than that of the other:

bool nameLess(const Student& s1, const Student& s2)

{

if (strcmp(s1.getName(), s2.getName()) < 0)

return true;

else

return false;

}

Notes:

1. Object types are normally passed by reference in C++ to avoid having to copy the object. Since this function will not alter either of the Student objects passed to it, we can declare each of parameters as a reference to a constant Student object.
2. The names are C strings, so they must be compared using strcmp(). They are also private data, so we need to use the public getName() accessor method to access them.
3. You can make this code shorter by simply returning the conditional expression, since that will itself evaluate to true or false.

We put the prototype for this comparison function in Course.h, outside the declaration of the Course class:

bool nameLess(const Student&, const Student&);

Once all this is in place, we can call the sort() method on a Course object, supplying the address of the comparison function as the argument:

myCourse.sort(&nameLess);

To sort the course's students in a different fashion, no modifications are required to the actual sort() method. Instead, we can just write another comparison function. For example, if we wanted to be able to sort in descending order by the students' GPA, we could add this function to Course.cpp (and its prototype to Course.h):

bool gpaGreater(const Student& s1, const Student& s2)

{

return (s1.getGPA() > s2.getGPA());

}

Now the behavior of our sort() method can vary depending on what comaprison function we pass to it:

cout << "Students listed in ascending order by name:\n\n";

myCourse.sort(&nameLess);

myCourse.print();

cout << "Students listed in descending order by GPA:\n\n";

myCourse.sort(&gpaGreater);

myCourse.print();

And because the comparison functions are not actually part of the Course class, it's easy for another programmer to add new comparison functions as desired.

## Other Points

* As always, programs that do not compile on turing/hopper automatically receive 0 points.
* Make sure to document your program according to the standards listed in the Course Notes book. In particular, each function should have a documentation box describing the purpose of the function, the input parameters, and the return value (if any). There should also be a documentation box for the program as a whole.
* Submit the final version of your program using the electronic submission guidelines posted on the course web site and described in class.