**CSCI 241 Assignment 1  
100 points**

**Purpose**

This assignment is an exercise in writing, compiling, and executing a C++ program on the departmental UNIX system. It also covers the basics of object-oriented programming and the manipulation of C and C++ strings.

**Set Up**

1. Log in to UNIX.
2. Change to your csci241 directory:
3. cd csci241
4. Create a directory to hold your files for Assignment 1:
5. mkdir Assign1
6. Change to your Assign1 directory:
7. cd Assign1
8. To create a symbolic link to the input file for this assignment, type:
9. ln -s ~t90kjm1/CS241/Data/Fall2017/Assign1/providers.csv
10. You can use the nano text editor to create and edit your source code file, by typing:
11. nano assign1.cpp

If the file assign1.cpp does not exist, an empty text file will be opened for you. If it does exist, it will be opened for editing. You can type your C++ source code into the file. To save, type Ctrl-o. To save and exit, type Ctrl-x.

**Makefile**

The file named makefile tells the make utility how to build the final executable file from a collection of C++ source code and header files. The makefile for this assignment is given in its entirety below. Makefiles for future assignments will follow this basic pattern. Like any other text file on Unix, this file can be edited and saved using nano or another text editor.

**IMPORTANT NOTE:** The commands that appear in the makefile below **MUST** be indented as shown. Furthermore, the indentation **MUST** be done using a tab character, not spaces. If you don't indent your makefile commands, or indent using spaces, your makefile **WILL NOT WORK**.

#

# PROGRAM: assign1

# PROGRAMMER: your name

# LOGON ID: your z-id

# DATE DUE: due date of program

#

# Compiler variables

CCFLAGS = -ansi -Wall -std=c++11

# Rule to link object code files to create executable file

assign1: assign1.o

g++ $(CCFLAGS) -o assign1 assign1.o

# Rule to compile source code file to object code

assign1.o: assign1.cpp

g++ $(CCFLAGS) -c assign1.cpp

# Pseudo-target to remove object code and executable files

clean:

-rm \*.o assign1

Once you've written the file makefile, you can simply type the command make to compile and link your program. To clean up the object code and executable files before submitting your assignment, use the command make clean.

To run the executable file created by the make command, type:

./assign1

**Program**

The goal of this program is simple: you need to read a file of records into an array of objects, then loop through the array and print the objects.

**Input**

The input file for this assignment is a **comma-separated values** (CSV) file. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

The CSV file in this assignment contains an unknown number of records (maximum of 40) containing data about medical providers associated with a health maintenance organization (HMO). Here are a couple of sample records from the input file to help illustrate the record format:

290764,ADULT/INTERNAL MEDICINE,PATEL,ILAXI,,DO,KISHHEALTH PHYSICIANS GROUP - DEKALB,1850 GATEWAY DR,SYCAMORE,IL,60178,(815) 758-8671

145397,OTOLARYNGOLOGY,SCIANNA,JOSEPH,M.,MD,NORTHERN IL ENT,2127 MIDLANDS CT SUITE 203,SYCAMORE,IL,60178,(815) 758-8106

The fields for a record (from left to right) are as follows:

1. Provider number
2. Provider specialty
3. Provider last name
4. Provider first name
5. Provider middle initial (optional)
6. Provider title
7. Provider address (part 1)
8. Provider address (part 2)
9. Provider city
10. Provider state
11. Provider zip code
12. Provider phone number

Reading the records of a CSV file will be a bit more difficult than reading records that have their fields separated by whitespace. The >> operator can not be used to read this data.

Techniques for reading the input data will be discussed more fully in class, but here are brief descriptions of two possible options:

*Option 1*

1. Read each of the fields of the record except the phone number as C++ strings using the [getline() function](http://www.cplusplus.com/reference/string/string/getline/) with a comma delimiter
2. Read the phone number as a C++ string using the [getline() function](http://www.cplusplus.com/reference/string/string/getline/) with the default (newline) delimiter
3. Create a new C++ string for the name by concatenating the last name, first name, middle initial (if present), and title using C++ string concatenation
4. Use the [c\_str() method](http://www.cplusplus.com/reference/string/string/c_str/) of the string class to convert the C++ string objects to C strings

*Option 2*

1. Read each of the fields of the record except the phone number as C strings using the [getline() method](http://www.cplusplus.com/reference/istream/istream/getline/) of the istream class with a comma specified as the delimiter
2. Read the phone number as a C string using the [getline() method](http://www.cplusplus.com/reference/istream/istream/getline/) of the istream class with the default (newline) delimiter
3. Create a new C string for the name by concatenating the last name, first name, middle initial (if present), and title using the strcpy() and strcat() functions

**The Provider Class**

The Provider class will be used to represent information about an HMO medical provider. A Provider object will contain the data from one record in the input CSV file. You will need to write a class declaration for this class, as well as definitions for any methods that the class contains.

*Data Members*

The Provider class should have the following private data members:

* 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)

*C++11 Initialization Option for Data Members*

C++11 adds the ability to initialize the non-static data members of a class at the time you declare them using a "brace-or-equal" syntax. This is very convenient, and can eliminate most or all of the code from your default constructor. Here are a few examples of the kind of initializations you can do in a class declaration:

class Foo

{

// Data members

private:

// Numeric types

int x = 0; // Initialize x to 0

double y = 9.9; // Initialize y to 9.9

bool found = false; // Initialize found to false

// C strings

char text[21]{}; // Initialize text to an

// empty string

char city[21] = {}; // Initialize city to an

// empty string

char name[11]{'J', 'o', 'h', 'n', '\0'}; // Initialize name to "John"

char state[3] = {'I', 'L', '\0'}; // Iniitialize state to "IL"

// C++ strings

string s{"Hello"}; // Initialize s to "Hello"

etc.

};

Doing this does not eliminate the need to write a default constructor, but it may take the place of some or all of the code for that constructor.

Feel free to use this option if you want to.

*Method Prototypes*

The Provider class declaration should also contain public prototypes for all of the methods you write for the Provider class.

*Method Definitions*

At minimum, you will need to write the following method definitions for the Provider class:

* Provider default constructor

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

This method should set each of the data members to a "null string". This can be done by copying a null string literal ("") into the character array using strcpy() or by setting the first element of the array to a null character ('\0').

(If you're working in C++11 and you initialized the data members at declaration as described above under *C++11 Initialization Option for Data Members*, this method's body can be empty. You still need to code the method though, even though it won't actually do anything. That's because the compiler will attempt to call this method to initialize each object in an array of Provider objects, and if it can't find the definition you will get a syntax error.)

* print()

This method takes no parameters and returns nothing. The method should print the data members in a neatly formatted fashion on the console using cout. See **Output** below for how to format the output produced by this method.

In addition to the methods listed above, you will need some way of getting the data from an input record that you've read into a Provider object. There are basically two ways to do this:

1. Write a set method for each data member of the class. These methods can be used to assign new values to the data members of an object.
2. Write an alternate constructor that takes a parameter for each data member of the class. In the constructor, the parameters passed in can be copied into the data members of the object.

Note that the second technique can only create a new object, not alter the data members of an existing one. However, if you create a new Provider object, you can then assign the object to an existing one, thereby overwriting all of the existing object's data members with new values at once.

**Output**

Here's a sample of what the output for this program should look like for the two example records listed above under **Input**:

#290764

PATEL, ILAXI, DO

ADULT/INTERNAL MEDICINE

KISHHEALTH PHYSICIANS GROUP - DEKALB

1850 GATEWAY DR

SYCAMORE, IL 60178

(815) 758-8671

#145397

SCIANNA, JOSEPH M., MD

OTOLARYNGOLOGY

NORTHERN IL ENT

2127 MIDLANDS CT SUITE 203

SYCAMORE, IL 60178

(815) 758-8106

**Notes**

* The code for this program should be grouped into functions in a logical fashion. One possible breakdown would be to write a function that reads the input data into the array of objects, and another function that loops through the array of objects and prints them. Both of these functions could be called from main().

Programs submitted with all of the code in the main() function may lose points for coding technique.

* Document your functions and methods using the guidelines found in the [CSCI 241 Documentation Standards](http://faculty.cs.niu.edu/~mcmahon/CS241/241DocStandards.html).
* As always, programs that do not compile on turing / hopper automatically receive 0 points.
* Submit your program using the electronic submission guidelines posted on the course web site and described in class.