Lab 11 - CMPS 1044, Computer Science I

Structs and Arrays of Structs

Lesson objectives:

1. Review format of structs and their use with arrays.
2. Learn how to fill an array of structs with data from a flat file
3. Extract data from this data from an array of structs vertically
4. Extract data from this data from an array of structs horizontally

A struct (short for structure) is a group of data elements grouped together under one name. These data elements, aka members, can have different types and be of different lengths. Structs are declared in C++ using the following syntax:

```
struct structure_name
{
      memberType memberName;
      memberType memberName;
      ...
};

structure_name objectName; // name of the variable that contains the struct
```

A struct declaration creates a new ***type*** that can be used to declare variables in the program. That is, the struct name is used in the same was as `int` or `float` or `double`. For example, the following code first defines the structure `regularPolygon` and then creates two variables, `square` and `triangle`, of this type.

```
struct regularPolygon
{
      int number_of_sides;
      float length_of_sides;
};

regularPolygon square;
regularPolygon triangle;
```

The variables `square` and `triangle` now represent 2 values each, not one. The 2 values are the attributes or members and, in this case, are represented by the names `number_of_sides` and `length_of_sides`. The attributes (members) of any variable defined as a struct are accessed using the **dot operator**. Here, the variable name and attribute name are written with a . (dot) separating them. Thus, the 4 values that we have declared are referred to as follows:

```
square.number_of_sides
square.length_of_sides
triangle.number_of_sides
triangle.length_of_sides
```

Note that the identifiers `number_of_sides` and `length_of_sides` are **NEVER** used independently; only as components of `square` & `triangle`.

The variables `square` and `triangle` are now defined to be `regularPolygon`. Each has two attributes, `number_of_sides` and `length_of_sides`, neither of which have been initialized at this point. The full names are used as any other variable. The following are examples of their initialization:

```
square.number_of_sides = 4;
square.length_of_sides=1;
```

One convenient feature of struct objects is the ability to assign one object to another. Assuming the above declaration we can do this:

```
regularPolygon square2;
square2 = square;     // assigns ALL the attributes of square to the
                      // corresponding ones in square2
```

Most often, structs are used in conjunction with arrays. This allows the programmer to create a collection of records (structs) that can be accessed in a variety of ways. A simple database can be easily constructed in this fashion. The following declaration sets up this scenario.

```
struct Student
{
      string lastName;
      string firstName;
      int classification; //0-3 for freshman thru senior
      float gpa;          //grade point average
};
Student  list[100];   // allocates array of 100 uninitialized
                      // Student slots.
```

This array is to be filled from a file containing the student list. This file is written in the following format. The attributes are separated by a space, one artist per line.

```
4  // This is the number of artists that follow.  Read it in first
Presley Elvis 0 2.5
Diddley Bo 2 3.1
Nelson Ricky 3 3.6
Berry Chuck 1 2.3
```

The code that can read in the above is as follows.

```
#include <iostream>
#include <fstream>
#include <cstdlib>     // For the exit function
#include <string>
using namespace std;
int main()
{
      ifstream infile;

      struct Student {
            string lastName;
            string firstName;
            int classification;
            float gpa;
      };
```

```
	Student list[100];

	int num; // Number of records in file
	infile.open("names.txt");
	if (!infile) // file could not be opened
	{
		cerr << "Error: file could not be opened" << '\n';
		exit(1);
	}

	infile >> num; // Read in the number of records in the file

	// Read in the file line by line
	for (int i = 0; i < num; i++)
	{
		infile >> list[i].lastName >> list[i].firstName >>
			list[i].classification >> list[i].gpa;
	}

	// Print out the third record for as test case
	cout << list[3].firstName << " " << list[3].lastName << " "
	    << list[3].classification << " " << list[3].gpa << '\n';
	system("pause");
	return 0;
}
```

Enter the above program and use the text file `names.txt` given to you by your instructor. Compile and test the program. When running properly move on to the following section

LAB 10 – Assignment

Modify the program you have been working on to include the following:

1. Write a loop at the end of the above program that prints out all students who have a gpa above 3.0. Just print the last name, first name and gpa for each student.
2. Write another loop that will calculate the average gpa of all the students in the list.
3. Write a third loop that will count the number of freshmen, sophomores, juniors, and seniors. Print out the results in this format together with the total count.
4. Include a header, create an output file, print your header and results to the output file, and include at least 3 comments in the body of your program.

Sample output (your numbers will be different):

```
Freshmen: 28 Sophomores: 23 Juniors: 22 Seniors: 18 Total: 91
```