CS 121 Worksheet - Week 8: Structures

**Notes on Structs (look at the code on pages 2 and 3 while looking at the below):**

* Structures are objects that have a collection of variables (and functions)
  + - The variables inside of a struct are called **member variables** (they are a member of the struct and belong to that struct)
* Two ways to initialize a structure: After declaration (initialize each variable) or by *constructor*
* Without a *constructor*, you cannot initialize member variable **inside** of the structure.
  + - Example syntax of using a constructor to initialize member variables on page 3
* Access a member variable of a structure by using the dot operator.
* Pass structs through functions just like any other object:
  + - Example: Assume that Movie is a struct

*void print\_movie(const Movie& m);*

* + - It's typically preferred to call structure by-reference (since it's a *collection* of data types, thus the byte-size of a structure is normally bigger than any primitive data type)
* Creating an array of structures is also like any other type:
  + - Example: Assume that *Movie* is a struct with no constructor and *SIZE* is a constant integer

*Movie movie\_array[SIZE];* //static array of Movie

*vector<Movie> movie\_vect;* //vector of Movie

* + - **WARNING:** Do not create an array/vector of a struct if you have a constructor (for now). We'll learn why and how to do so later on.
    - You can think of an array of struct as a better alternative to working with parallel arrays (in some situations at least). For example (assuming students have a name, gpa, and id):

*Student student\_list[SIZE];*

is equivalent to having the following parallel arrays:

*string name[SIZE];*

*int id[SIZE];*

*double gpa[SIZE];*

We only have **ONE** array for a struct array and **THREE** arrays for parallel arrays.

* + - * To then access each Student, we call it out using a subscript. For instance, if we wanted to change the third student's name to "Bobby", id to *17793,* and gpa to *3.5* (assuming the student is made already in the array) we do the following:

*student\_list[2].name = "Bobby";*

*student\_list[2].id = 17793;*

*student\_list[2].gpa = 3.5;*

In a parallel array we would have:

*name[2] = "Bobby";*

*id[2] = 17793;*

*gpa[2] = 3.5;*

Same amount of lines, however parallel arrays can get confusing when adding more variables.

Simple program expressing usage of struct:

#include <iostream>

#include <string>

using namespace std;

//Defining struct called Student

struct Student

{

//Student characteristics:

string name;

int id;

double gpa;

//...other member variables...

};

int main()

{

//Declaring the struct Student and naming its identifier as s

Student s;

//Initialize contents of Student s using dot operator

s.name = "John Smith"; //initializing name in Student s

s.id = 1795334; //" " id in Student s

s.gpa = 3.5; //" " gpa in Student s

//Print contents/characteristics of s using dot operator

cout << "Student info: " << endl;

cout << "Name: " << s.name << endl;

cout << "ID: " << s.id << endl;

cout << "GPA: " << s.gpa << endl;

//...code...

system("pause");

return 0;

}

The struct *Student* (above) can be thought of as the following:

Student Properties:

- Has a name (string of characters)

- Has an ID (integer)

- Has a GPA (double)

Same program as above except using constructor to initialize student:

#include <iostream>

#include <string>

using namespace std;

//Defining struct called Student

struct Student

{

//Define Student using a constructor

Student(string n, int i, double g)

{

name = n;

id = i;

gpa = g;

}

//Student characteristics:

string name;

int id;

double gpa;

//...other member variables...

};

int main()

{

//Declaring the struct using a constructor

Student s("John Smith", 1795334, 3.5);

//Print contents/characteristics of s using dot operator

cout << "Student info: " << endl;

cout << "Name: " << s.name << endl;

cout << "ID: " << s.id << endl;

cout << "GPA: " << s.gpa << endl;

//...code...

return 0;

}

-----------------------------------------------------------------------------------------**Questions:**

1. Convert the following objects into structures. Have at least three characteristics/member variable*s* for each. Make sure to list what the characteristics are that you plan to use, and afterwards write each one out in correct struct syntax. **Do not** use a constructor in your struct syntax (follow the first code example).
   * + A car
     + A movie
     + <Come up with another object other than a car, movie, or student>
2. Using one of the above structures you made, write the syntax to both **declare** and **initialize** your struct object. Make sure to initialize **all** of its member variables.
3. Using the initialized struct from question two, create a *void* function to print out its contents to console. Give an example of what calling the function would look like (by passing your previous struct through). **NOTE:** You should only be passing one value (the struct).
4. Using a struct **different** than the one used for questions two and three, write a code segment that creates and initializes an array or vector of your chosen struct. Use a *for* loop to initialize each position of the array, grabbing data from either the console or a file (your choice). If you choose to use an array in your answer, have the size of your array to be five.
5. (Unrelated to structs) Given a vector of integers called *int­\_list*, write a code segment that prints an integer "num" found in *int­\_list*. Print out the subscript of where num is first found. **HINT:** You will need a boolean and a *for* loop.