Name:_____

1. A *structure* is a C (and C++) construct that allows multiple variables of potentially different types to be grouped together. For example, we can define the template for a `Student` structure as:

```
struct Student {
    int studentID;
    string name;
    short yearInSchool;
    double gpa;
}; // end Student struct
```

In lab you wrote a `getStudent` function `getStudent` to interactively read information about a student from the keyboard. One way to return the `Student` information back to the main is by using a pass-by-reference Student parameter, but another way would be to return the `Student` as the return type. A call from the main might look like:

```
Student myStudent;
myStudent = getStudent( );
```

a) Write this `getStudent` function that returns a `Student` as the return type.

b) Why would using a pass-by-reference Student parameter for the `getStudent` function be more efficient?

c) A third approach would be to pass a pointer to a Student structure to the the function as:

```
Student * newStudentPtr;
newStudentPtr = new Student;        // dynamically allocate a Student record
getStudent(newStudentPtr);          // call by sending the pointer newStudentPtr
```

In the `getStudent` function definition, show how would you read a value for the `studentID` member?

```
void getStudent(Student * myStudentPtr) {
    cout << "Enter Student's ID: ";
    cin >>
    ...
}
```

d) Alternatively, if you have a pointer to a structure as in the previous `getStudent` function, you can use the "follow the pointer operator" `->` as shown below. How would you read a value for the `name` member?

```
void getStudent(Student * myStudentPtr) {
      cout << "Enter Student's ID: ";
      cin >> myStudentPtr->studentID;

      cout << "Enter Student's name: ";
      cin >>
      ...
}
```

2. An *enumerated type* in C++ is a data type whose legal values are a set of named constant integers. An example would be:

```
      enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY };
```

A variable of type Day can only be assigned one of these constant values.

```
int myInt, weekDay;
Day today, tomorrow;

today = MONDAY;
```

In C++ enumerated types are inplemented using integer. In the type Day, Sunday is a named constant integer with the value 0, Monday is a named constant integer with the value 1, etc.

a) The assignment statements:

```
      today = MONDAY;
      myInt = today;    // myInt gets the integer value 1
```

are both legal, but

```
      myInt = 1;
      today = myInt;
```

causes an error. Why?

b) What would be printed by the following code?

```
#include <iostream>
using namespace std;

enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY };

int main() {
      int myInt, weekDay;
      double totalHours, hoursWorked[] = { 10, 5, 5, 9, 8, 9, 12 };
      Day today;

      today = MONDAY;
      cout << "today's value: " << today << endl;

      today = MONDAY;
      myInt = today;    // myInt gets the integer value 1
      cout << "myInt's value: " << myInt << endl;
      myInt = 1;
      //today = myInt;  CAUSES AN ERROR
      today = static_cast<Day>(myInt+1);
      cout << "today's value: " << today << endl;

      totalHours = 0;
      for (weekDay = MONDAY; weekDay <= FRIDAY; weekDay++) {
            totalHours = totalHours + hoursWorked[weekDay];
      } // end for
      cout << "Week Day total hours:  " << totalHours << endl;
      cout << "Week-End total hours:  " << hoursWorked[SATURDAY]+hoursWorked[SUNDAY] <<endl;
} // end main
```