Lab 12: Structures

Structures are a way to create new types that combine two or more existing types. A simple example would be a new type that consists of two integers (existing data types). Such a type might be useful for fractions: one integer holds the numerator, the other holds the denominator.

Structures are defined using the `struct` keyword, and a new programmer-defined type is typically defined using the `typedef` keyword. For example, the code:

```c
struct fraction
{
    int numerator;
    int denominator;
};

typedef struct fraction Fraction;
```

defines the `Fraction` type described above; and by using the `typedef` keyword renames the type from `struct fraction` to just `Fraction`. The numerator and denominator are variables of type integer; they are often referred to as fields or members of the structure.

So we can now create variables, pointers, and arrays using the newly defined `Fraction` type and the normal syntax. For example, the code:

```c
Fraction f1;
Fraction arrayOfFractions[10];
Fraction *fracP;
```
declares (creates) a variable called `f1` of the `Fraction` type, an array of ten fractions called `arrayOfFractions`, and a pointer to a fraction called `fracP`.

However, we can not (yet) do things with fractions that we can do with standard types. For example, commands like:

```c
cout << f1 << endl;
f1 = arrayOfFractions[0] + arrayOfFractions[1];
cin >> f1;
```
are all illegal because C++ doesn’t know how we want fractions printed, added or entered. Instead we need to tell C++ exactly how to do these things. For example, if we wanted fractions to be printed like \( \frac{n}{d} \) we could write:

```c
cout << f1.numerator << "/" << f1.denominator << endl;
```
in this case we are asking C++ to print f1.numerator and f1.denominator, both of which are integers and C++ know how to print integers.

Note the use of the *dot (.)* notation to access the *members* of the structure. The dot notation is used for both accessing and setting the values of the members. For example, to have the user enter a fraction we could write:

```cpp
cout << "Please enter the numerator";
cin >> f1.numerator;
cout << "Please enter the denominator";
cin >> f1.denominator;
```

This works because C++ knows how to use cin with integers.

Structures can be passed to and returned from functions just like any other type of variable. The structure name (in these examples *Fraction*) is used as the argument type. So, for example:

```cpp
Fraction func( Fraction f, int k );
```

defines a function called *func* that takes a fraction structure and an integer as arguments and returns a fraction.

Because we cannot directly use common C++ elements like cin and cout with structures, it is common to write functions that do their job for us. For example, writing a function to print a fraction or a function that adds two fractions together and returns their sum as a fraction.

**Exercise 12.1**

1. Write a structure that will store a name and ID number. The structure should have two members: a character array or string to store the name and an integer to store the ID.

2. Write a function that allows the user to enter a name and ID number a structure.

3. Write a function that prints the structure. You can chose how the name and ID are printed.

4. Write a program that creates an array of 5 of these name and ID structures, allows the user to enter up to 5 names and IDs, and then prints the list of names and IDs. The names and IDs should be entered and printed using the two functions written in steps two and three.

Turn in the program and sample output.