

Chapter 26 - Structs

1. What is a Struct?

Before it grew to be C++, the language C included a means of combining data object variables into one object type. This was done with the use of *structs*. C++ includes *structs*. To the eye of a C++ programmer, a *struct* looks something like an underdeveloped *class* that contains only *public data objects*. In fact, this is a fairly accurate description of what *structs* are in C++. Like *classes*, *struct* definitions are templates for objects. Here is the form of defining a *struct*:

```
struct struct-name {
    data-member-declaration;
    data-member-declaration;
    :
    :
};
```

Every member object of a *struct* is a *public data member*. There are no *member functions* in a *struct*. There is no means of declaring a *private member* of a *struct*.

Here is an example struct:

```
struct phonecard {
    char fname[15];
    char lname[15];
    char phone[15];
};
```

(Note: Previous chapters ignored the issues involved in arrays of characters. One of the problems that beginning C++ programmers encounter is the fact that null terminated strings and strings are special case arrays of characters. Both C and C++ handle them differently than other types of arrays, which creates problems when trying to create data structures that use strings. Null terminated strings can be used in an array by placing strings inside of a struct and building arrays of the struct type, many of the problems associated with null terminated strings go away, without having to create an entire class to achieve the same outcome.) (Note: You can create arrays of string class objects.)

The struct *phonecard* can be used to declare objects. For example:

```
phonecard example;
```

Each object of a type struct contains an instance of each of the data members of the struct. Each data-member of a struct object is accessed with the following form:

```
struct-object-name.data-member
```

Here are some examples of the *phonecard* object example being used:

```
cin >> example.fname;
strcpy(example.lname, "Fred");
cout << example.phone << endl;
```

Here is an example program that declares and uses objects of the above struct definition:

```
#include <iostream>
#include <string.h>
using namespace std;

struct phonecard {
    char fname[15];
    char lname[15];
    char phone[15];
};

void main() {
    phonecard example;

    strcpy(example.fname, "Sam");
    strcpy(example.lname, "Trowel");
    strcpy(example.phone, "337-555-9393");

    cout << "    First Name: " << example.fname << endl
         << "    Last Name: " << example.lname << endl
         << "Phone Number: " << example.phone << endl;
}
```

When run, this program produces the following output:

```
    First Name: Sam
    Last Name: Spade
    Phone Number: 337-555-9393
```

2. Using a Struct Type

The struct *phonecard* could be used to create an array of phone number information. For example:

```
phonecard phonenumber[1000];
```

The example array *phonelist* can hold up to 1000 names and phone numbers. To access a struct data member, the array name and an index is used between the array name and the struct data member, as in the following examples:

```
cin >> phonelist[55].fname;
strcpy(phonelist[x].lname,"Roda");
```

The array *phonelist* could be a data member of a class, as in this example class:

```
#include <iostream>
#include <string.h>
using namespace std;

int const maxlen=1000;

struct phonecard {
    char fname[15];
    char lname[15];
    char phone[15];
};

class phonebook {
public:
    phonebook( );    // loads the phone book data from a file on startup

    ~phonebook( ); // returns the phone book data to the file on exit

    void newcard( ); // gets information from the user and makes a new phonecard in
                    // the list

    void findcards( ); // gets the last name from the user and displays all cards
                       // with that last name; displays the index of each card

    void showall( ); // lists all cards one screen at a time

private:
    phonecard phonelist[maxlen]; // the phone number list
    int currentlen;              // current number of phonecards in use
};
```

Exercises

1. What is a struct?
2. How is a struct like a class?
3. How is a struct not like a class?
4. May the contents of a struct be accessed?
5. Create a struct to hold information about an item in an inventory. Fields should include item name, the cost of the item, the quantity on hand and the sales price of the item.
6. Declare an object of the struct type of (5).
7. Assign values to all fields in the object of (6).
8. Write a function that accepts an argument of type of the struct from (5). The function should output all values stored in the struct to the monitor screen.
9. Write a call to the function of (8) that passes the object of (7) to it.

Programming Exercise 26.1

Complete the class *phonebook* by writing the member functions. Use the class in a program that loops until the user elects to exit.

Programming Exercise 26.2

Simplify the constructor and destructor member functions of class *phonebook* by including operator overloading for file input and file output of objects of *struct* type *phonecard*. Using the overloading, input and output data for an entire *struct* of type *phonecard* with each input and output from and to the data file.