Power Engineering School

M.CS201 “Programming language”

Lecture 10

Lecturer:

Prof. Dr. T.Uranchimeg

Agenda

- ***Structures***
- ***Defining Structures***
- ***Declaring Structures***
- ***Complex structures***

Simple Structures

A *structure* is a collection of one or more variables grouped under a single name for easy manipulation. The variables in a structure, unlike those in an array, can be of different variable types. A structure can contain any of C's data types, including arrays and other structures. Each variable within a structure is called a *member* of the structure. The next section shows a simple example.

Defining and Declaring Structures

If you're writing a graphics program, your code needs to deal with the coordinates of points on the screen. Screen coordinates are written as an x value, giving the horizontal position, and a y value, giving the vertical position.

Example

You can define a structure named coord that contains both the x and y values of a screen location as follows:

```
struct coord {
    int x;
    int y;
};
```

The Example

The struct keyword, which identifies the beginning of a structure definition, must be followed immediately by the structure name, or *tag* (which follows the same rules as other C variable names). Within the braces following the structure name is a list of the structure's member variables. You must give a variable type and name for each member.

The explain

The preceding statements define a structure type named coord that contains two integer variables, x and y. They do not, however, actually create any instances of the structure coord. In other words, they don't *declare* (set aside storage for) any structures. There are two ways to declare structures.

The structure definition

One is to follow the structure definition with a list of one or more variable names, as is done here:

```
struct coord {
    int x;
    int y;
} first, second;
```

The explain

These statements define the structure type coord and declare two structures, first and second, of type coord. first and second are each *instances* of type coord; first contains two integer members named x and y, and so does second.

Accessing Structure Members

Individual structure members can be used like other variables of the same type. Structure members are accessed using the *structure member operator* (.), also called the *dot operator*, between the structure name and the member name.

The Example

Thus, to have the structure named first refer to a screen location that has coordinates x=50, y=100, you could write

```
first.x = 50;
first.y = 100;
```

Displaying members

To display the screen locations stored in the structure second, you could write

printf("%d,%d", second.x, second.y);

At this point, you might be wondering what the advantage is of using structures rather than individual variables. One major advantage is that you can copy information between structures of the same type with a simple equation statement.

Cont.

Continuing with the preceding example, the statement

first = second;

is equivalent to this statement:

first.x = second.x;

first.y = second.y;

Complex structures

When your program uses complex structures with many members, this notation can be a great time-saver. Other advantages of structures will become apparent as you learn some advanced techniques. In general, you'll find structures to be useful whenever information of different variable types needs to be treated as a group.

Example

For example, in a mailing list database, each entry could be a structure, and each piece of information (name, address, city, and so on) could be a structure member.

The struct Keyword

```
struct tag {
    structure_member(s);
    /* additional statements may go
   here */
} instance;
```

The explain

The struct keyword is used to declare structures. A structure is a collection of one or more variables (*structure_members*) that have been grouped under a single name for easy manipulation. The variables don't have to be of the same variable type, nor do they have to be simple variables. Structures also can hold arrays, pointers, and other structures.

The keyword struct

The keyword struct identifies the beginning of a structure definition. It's followed by a tag that is the name given to the structure. Following the tag are the structure members, enclosed in braces. An *instance*, the actual declaration of a structure, can also be defined. If you define the structure without the instance, it's just a template that can be used later in a program to declare structures.

Template format

```
struct tag {
    structure_member(s);
    /* additional statements may go here */
};
```

Using format

To use the template, you use the following format:

struct *tag instance;*

To use this format, you must have previously declared a structure with the given tag.

Example 1

```
/* Declare a structure template called SSN */
struct SSN {
     int first_three;
     char dash1;
     int second_two;
     char dash2;
     int last_four;
}
/* Use the structure template */
struct SSN customer_ssn;
```

Example 2

```
/* Declare a structure and instance together */
struct date {
     char month[2];
     char day[2];
     char year[4];
} current_date;
```

Example 3

```
/* Declare and initialize a structure */
struct time {
     int hours;
     int minutes;
     int seconds;
} time_of_birth = { 8, 45, 0 };
```

More-Complex Structures

Now that you have been introduced to simple structures, you can get to the more interesting and complex types of structures. These are structures that contain other structures as members and structures that contain arrays as members.

Structures That Contain Structures

As mentioned earlier, a C structure can contain any of C's data types. For example, a structure can contain other structures. The previous example can be extended to illustrate this.

The explain

Assume that your graphics program needs to deal with rectangles. A rectangle can be defined by the coordinates of two diagonally opposite corners. You've already seen how to define a structure that can hold the two coordinates required for a single point. You need two such structures to define a rectangle.

Example

You can define a structure as follows (assuming, of course, that you have already defined the type coord structure):

```
struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
};
```

The explain

This statement defines a structure of type rectangle that contains two structures of type coord. These two type coord structures are named topleft and bottomrt.

Cont.

The preceding statement defines only the type rectangle structure. To declare a structure, you must then include a statement such as

struct rectangle mybox;

Example

You could have combined the definition and declaration, as you did before for the type coord:

```
struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
} mybox;
```

Example

To access the actual data locations (the type int members), you must apply the member operator (.) twice. Thus, the expression

mybox.topleft.x

refers to the x member of the topleft member of the type rectangle structure named mybox.

Example

To define a rectangle with coordinates (0,10),(100,200), you would write

```
mybox.topleft.x = 0;
mybox.topleft.y = 10;
mybox.bottomrt.x = 100;
mybox.bottomrt.y = 200;
```

A demonstration of structures that contain other structures

```
#include <stdio.h>

int length, width;
long area;
struct coord{
     int x;
     int y;
};
struct rectangle{
     struct coord topleft;
     struct coord bottomrt;
} mybox;
```

Example

```
main()
{
  printf("\nEnter the top left x coordinate: ");
  scanf("%d", &mybox.topleft.x);
  printf("\nEnter the top left y coordinate: ");
  scanf("%d", &mybox.topleft.y);
  printf("\nEnter the bottom right x coordinate: ");
  scanf("%d", &mybox.bottomrt.x);
  printf("\nEnter the bottom right y coordinate: ");
  scanf("%d", &mybox.bottomrt.y);
  width = mybox.bottomrt.x - mybox.topleft.x;
  length = mybox.bottomrt.y - mybox.topleft.y;
```

Example

```
    area = width * length;
    printf("\nThe area is %ld units.\n",area);
    return 0;
}
```

```
Enter the top left x coordinate: 1
Enter the top left y coordinate: 1
Enter the bottom right x coordinate: 10
Enter the bottom right y coordinate: 10
The area is 81 units.
```

Summary

- ***Structures***
- ***Defining Structures***
- ***Declaring Structures***
- ***Complex structures***

Any questions?

Thank you for attention