1.1 Structures in C++

A structure is a collection of simple variables. The variables in a structure can be of different types: int, float, and so on. The data items in a structure are called the *members* of the structure. In fact, the syntax of a structure is almost identical to that of a class. A structure is a collection of data, while a class is a collection of both data and functions. Structures in C++ similar to records *in* Pascal.

1.2 A Simple Structure (user-defined data types)

The company makes several kinds of widgets, so the widget model number is the first member of the structure.

The number of the part itself is the next member, and the final member is the part's cost. The next program defines the structure part, defines a structure variable of that type called part1. Assigns values to its members, and then displays these values.

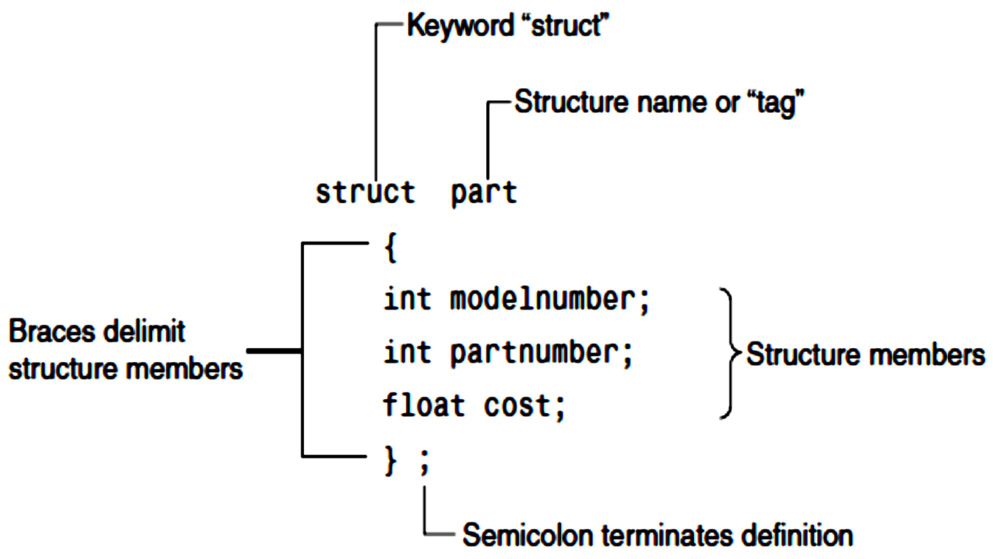
```
#include <iostream.h>
struct part //declare a structure
{
 int modelnumber; //ID number of widget
 int partnumber; //ID number of widget part
 float cost; //cost of part
};
int main()
{
part part1; //define a structure variable
part1.modelnumber = 6244; //give values to structure members
part1.partnumber = 373;
part1.cost = 217.55F;
//display structure members
cout << “Model “ << part1.modelnumber;
cout << “, part “ << part1.partnumber;
cout << “, costs $” << part1.cost << endl;
```

return 0; }

The program's output looks like this:

Model 6244, part 373, costs $217.55

1.2.1 Syntax of the Structure Definition

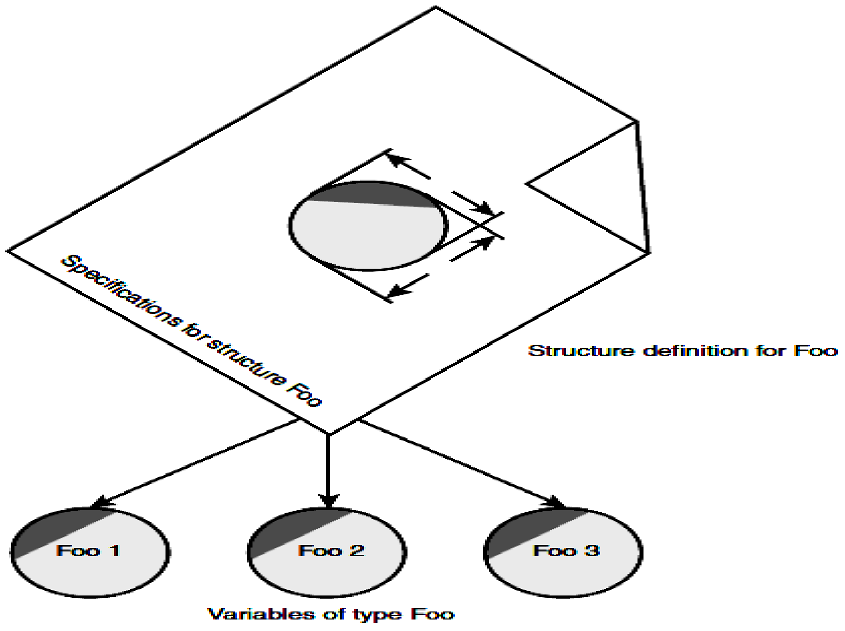

Keyword "struct"
Structure name or "tag"
Braces delimit structure members
Structure members
Semicolon terminates definition

```
struct  part
{
int modelnumber;
int partnumber;
float cost;
} ;
```

1.2.1 Use of the Structure Definition

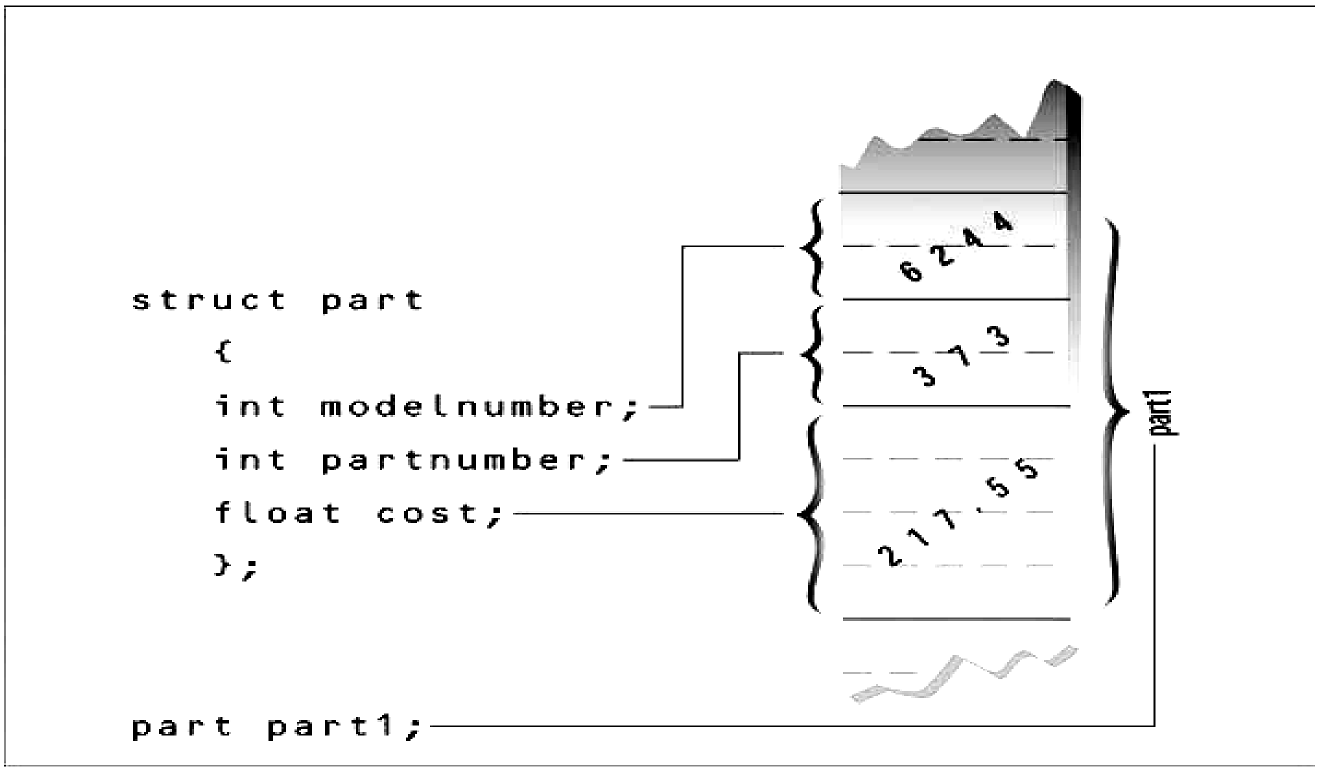
The structure definition serves only as a blueprint for the creation of variables of type part. It does not itself create any structure variables; that is, it does not set aside any space in memory or even name any variables.

This is unlike the definition of a simple variable, which does set aside memory. A structure definition is specification for how structure variables will look when they are defined. This is shown in Figure below:


Specifications for structure Foo
Structure definition for Foo
Foo 1
Foo 2
Foo 3
Variables of type Foo

1.2.2 Defining a Structure Variable

The first statement in main() part part1; defines a variable, called part1, of type structure part. This definition reserves space in memory for part1. Figure below shows how part1 looks inmemory.



1.2.3 Accessing Structure Members

members can be accessed using the dot operator.

part1.modelnumber = 6244;

The structure member is written in three parts:

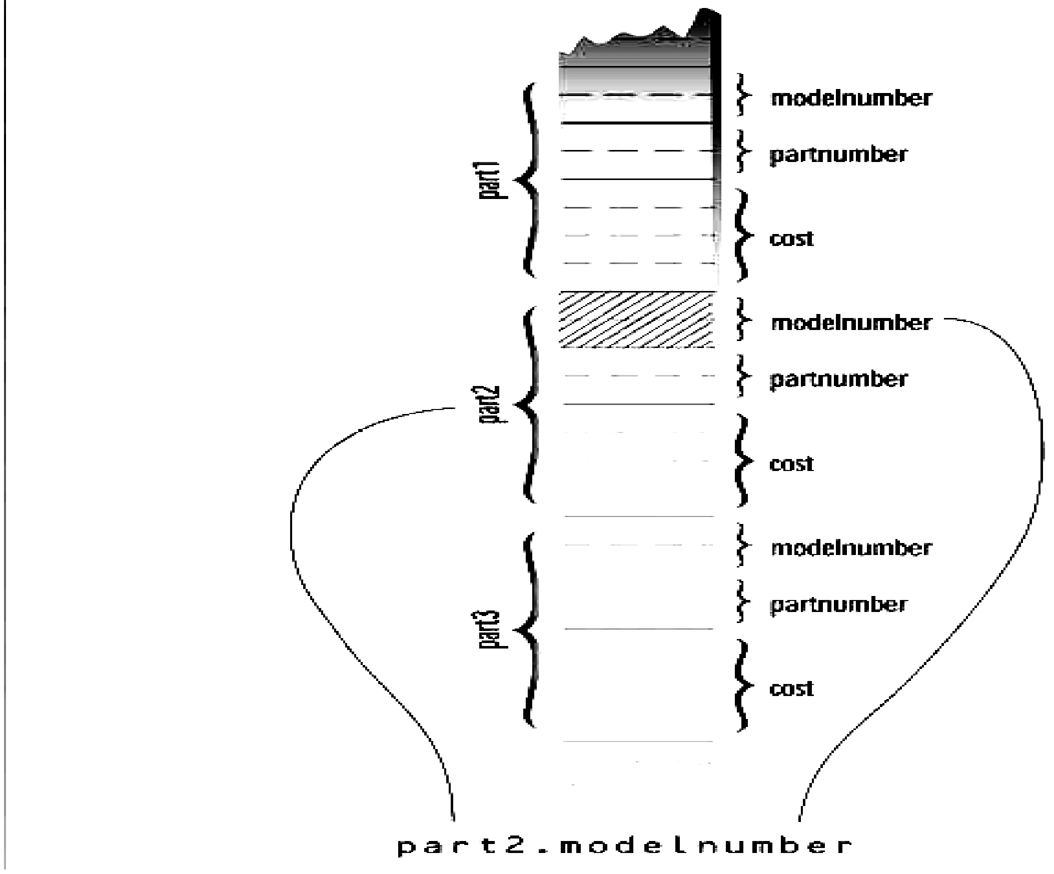
1- the name of the structure variable (part1);

2- dot operator, which consists of a period (.);

3- member name (modelnumber).

The first component of an expression involving the dot operator is the name of the specific structure variable (part1 in this case), not the name of the structure definition (part).

The variable name must be used to distinguish one variable from another, such as part1, part2, and so on, as shown in Figure below:



Structure members are treated just like other variables. In the statement

part1.modelnumber = 6244.

The member is given the value 6244 using a normal assignment operator.
The program also shows members used in cout statements such as

cout << "\nModel " << part1.modelnumber;

These statements output the values of the structure members.

1.2.4 Initializing Structure Members

The next example shows how structure members can be initialized when the structure variable is defined. It also demonstrates that you can have more than one variable of a given structure type

```
#include <iostream.h>
struct part //specify a structure
{ int modelnumber; //ID number of widget
  int partnumber; //ID number of widget part
  float cost; //cost of part };
int main()
{ //initialize variable
part part1 = { 6244, 373, 217.55F };
part part2; //define variable
//display first variable
cout << “Model “ << part1.modelnumber;
cout << “, part “ << part1.partnumber;
cout << “, costs $” << part1.cost << endl;
part2 = part1; //assign first variable to second
//display second variable
cout << “Model “ << part2.modelnumber;
cout << “, part “ << part2.partnumber;
cout << “, costs $” << part2.cost << endl;
return 0;
}
```

Here’s the output:

```
Model 6244, part 373, costs $217.55
Model 6244, part 373, costs $217.55
```

1.5 A Measurement Example

Suppose you want to create a drawing or architectural program that uses the English system. It will be convenient to store distances as two numbers, representing feet and inches. The next example, gives an idea of how this could be done using a structure.

This program will show how two measurements of type Distance can be added together.

```
#include <iostream.h>
struct Distance //English distance
```

```
{int feet;
float inches;
};
int main()
{Distance d1, d3; //define two lengths
 Distance d2 = { 11, 6.25 }; //define & initialize one length
 //get length d1 from user
 cout << “\nEnter feet: “;      cin >> d1.feet;
 cout << “Enter inches: “;    cin >> d1.inches;
 //add lengths d1 and d2 to get d3
 d3.inches = d1.inches + d2.inches; //add the inches
 d3.feet = 0; //(for possible carry)
 if(d3.inches >= 12.0) //if total exceeds 12.0,
{ //then decrease inches by 12.0
d3.inches -= 12.0; //and
d3.feet++; //increase feet by }
d3.feet += d1.feet + d2.feet; //add the feet
//display all lengths
cout << d1.feet << “\’-” << d1.inches << “\” + “;
cout << d2.feet << “\’-” << d2.inches << “\” = “;
cout << d3.feet << “\’-” << d3.inches << “\”\n”;
return 0;}
```

Here’s some output:

```
Enter feet: 10
Enter inches: 6.75
10’-6.75” + 11’-6.25” = 22’-1”
```

Notice that we can’t add the two distances with a program statement like d3 = d1 + d2; // can’t do this in previous program. Why not? Because there is no routine built into C++ that knows how to add variables of type Distance. The + operator works with built-in types like float, but not with types we define ourselves, like Distance.

1.6 Structures Within Structures

You can nest structures within other structures. In the next program we want to create a data structure that stores the dimensions of a typical room: its length and width.

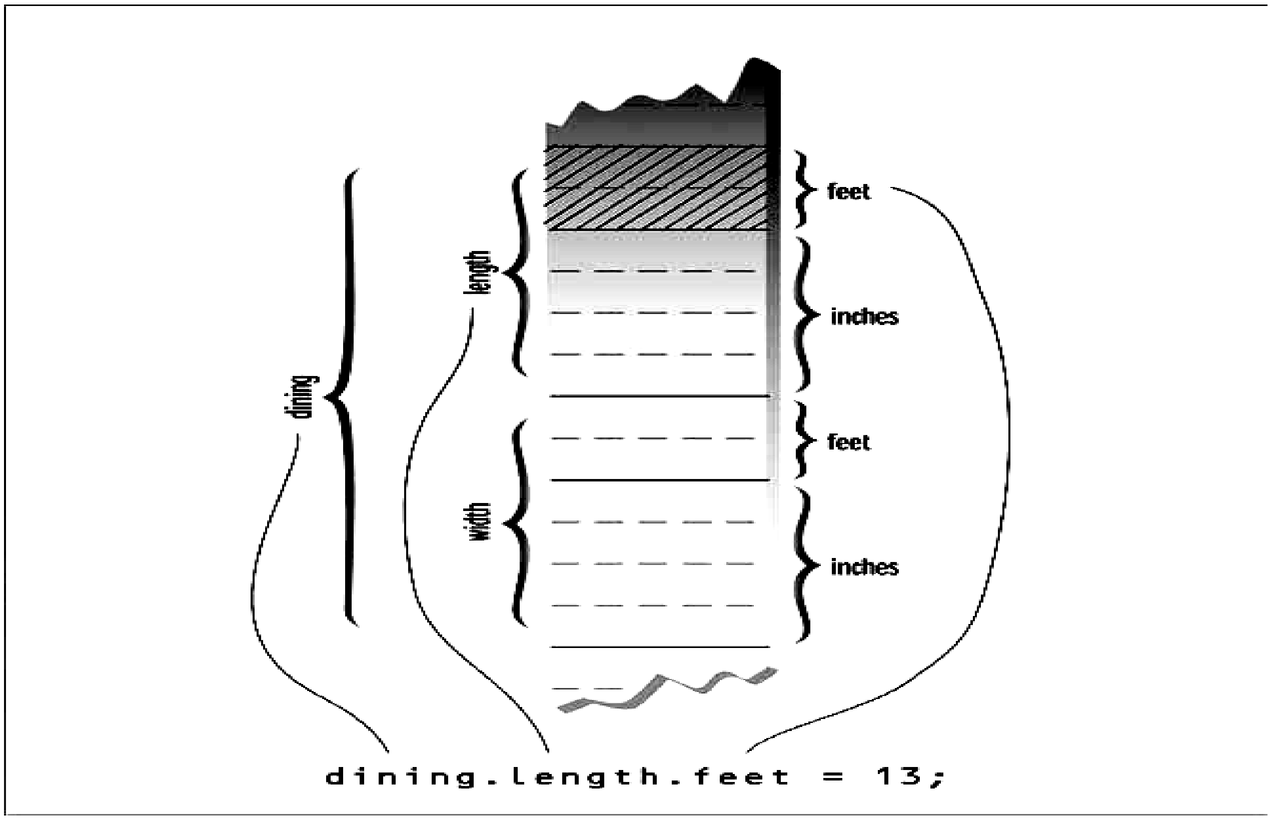
```
#include <iostream.h>
struct Distance  //English distance
{
 int feet;
float inches;
};
struct Room    //rectangular area
{
Distance length; //length of rectangle
Distance width; //width of rectangle
};
int main()
{Room dining; //define a room
dining.length.feet = 13; //assign values to room
dining.length.inches = 6.5;
dining.width.feet = 10;
dining.width.inches = 0.0;
//convert length & width
float l = dining.length.feet + dining.length.inches/12;
float w = dining.width.feet + dining.width.inches/12;
//find area and display it
cout << "Dining room area is " << l * w << " square feet\n" ;
return 0;
}
```

1.7 Accessing Nested Structure Members

Because one structure is nested inside another, we must apply the dot operator twice to access the structure members.

dining.length.feet = 13;

In this statement, dining is the name of the structure variable, as before; length is the name of a member in the outer structure (Room); and feet is the name of a member of the inner structure (Distance). The statement means "take the feet member of the length member of the variable dining and assign it the value 13." Figure below shows how this works.



Once values have been assigned to members of dining, the program calculates the floor area of the room, as shown in Figure below. To find the area, the program converts the length and width from variables of type Distance to variables of type float, l, and w, representing distances in feet. The values of l and w are found by adding the feet member of Distance to the inches member divided by 12.

The feet member is converted to type float automatically before the addition is performed, and the result is type float. The l and w variables are then multiplied together to obtain the area.

1.7.1 User-Defined Type Conversions

Note that the program converts two distances of type Distance to two distances of type float: the variables l and w. In effect it also converts the room's area, which is stored as a structure of type Room (which is defined as two structures of type Distance), to a single floating-point number representing the area in square feet.

Here's the output:

```
Dining room area is 135.416672 square feet
```

Converting a value of one type to a value of another is an important aspect of programs that employ user-defined data types.

1.7.2 Initializing Nested Structures

How do you initialize a structure variable that itself contains structures? The following statement initializes the variable dining to the same values it is given in the program:

```
Room dining = { {13, 6.5}, {10, 0.0} };
```

Each structure of type Distance, which is embedded in Room, is initialized separately. Remember that this involves surrounding the values with braces and separating them with commas.

The first Distance is initialized to: {13, 6.5}
and the second to: {10, 0.0}

<u>Exercises</u>

1. A phone number, such as (212) 767-8900, can be thought of as having three parts: thearea code (212), the exchange (767), and the number (8900). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone. Create two structure variables of type phone. Initialize one, and have the user input a number for the other one. Then display both numbers. The interchange might look like this:

```
Enter your area code, exchange, and number: 415 555 1212
```

```
My number is (212) 767-8900
Your number is (415) 555-1212
```

Solutions to Exercises

1.

```
#include <iostream.h>
struct phone
{
int area; //area code (3 digits)
int exchange; //exchange (3 digits)
int number; //number (4 digits)
};
int main()
{
phone ph1 = { 212, 767, 8900 }; //initialize phone number
phone ph2; //define phone number
// get phone no from user
cout << “\nEnter your area code, exchange, and number”;
cout << “\n(Don’t use leading zeros): “;
cin >> ph2.area >> ph2.exchange >> ph2.number;
cout << “\nMy number is “ //display numbers << ‘(‘ <<   ph1.area
<< “) “ << ph1.exchange << ‘-’ << ph1.number;
cout << “\nYour number is “<< ‘(‘ << ph2.area << “)
“<<ph2.exchange << ‘-’ << ph2.number << endl;
return 0;
}
```