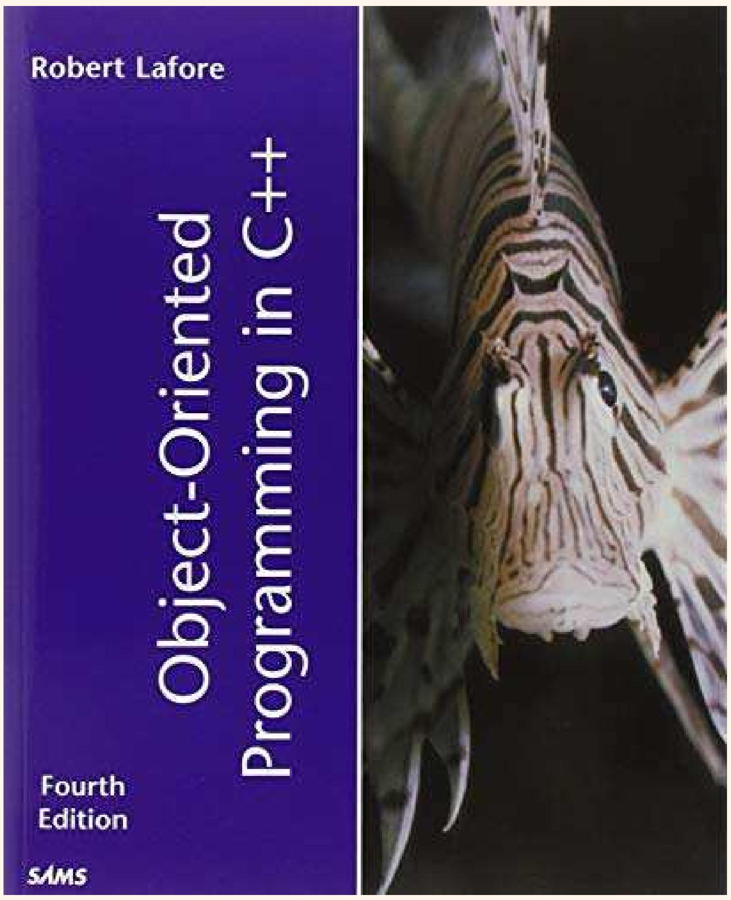


Chapter 4

Structures

Animated Version

Topics

- Structures
- Enumerations

Structures

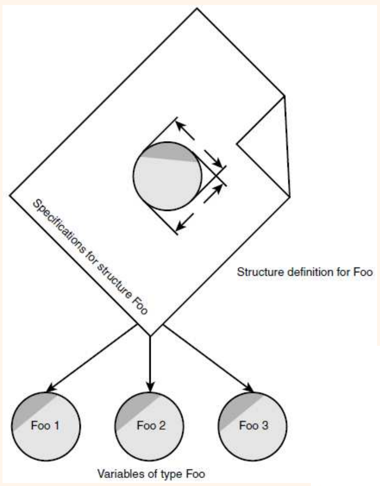
- ❑ As groceries are organized into bags, employees into departments, and words into sentences, it’s often convenient to organize simple variables into more complex entities.
  - ❑ The C++ construction called the structure is one way to do this.
- ❑ A structure is a collection of simple variables.
  - ❑ The variables in a structure can be of different types: Some can be int, some can be float, and so on.
  - ❑ The data items in a structure are called the *members* of the structure.
- ❑ Structures is an important building block in the understanding of objects and classes.
- ❑ In fact, the syntax of a structure is almost identical to that of a class.
  - ❑ A structure (as typically used) is a collection of data, while a class is a collection of both data and functions.

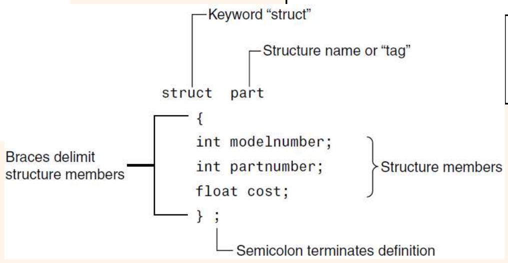
A Simple Structure

- Structure definition
  - – serves only as a blueprint
  - – does not set aside any space in memory

```
// parts.cpp
// uses parts inventory to demonstrate structures
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
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;
   }
```

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


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


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

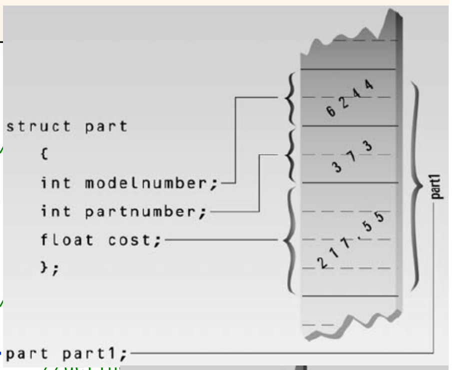
```
struct  part
   {
   int modelnumber;
   int partnumber;
   float cost;
   } ;
```

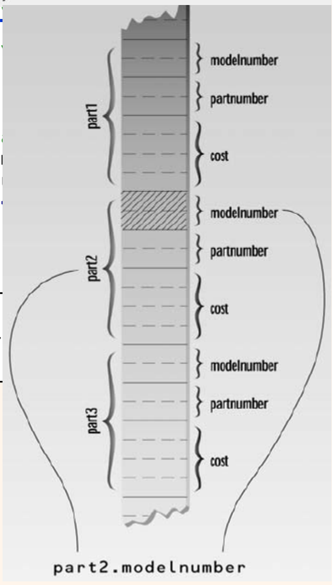
A Simple Structure (2)

- Defining a Structure Variable
  - the format for defining a structure variable is the same as that for defining a basic built-in data type
  - int var1;
  - In C: struct part part1;
- Accessing Structure Members
  - three parts:
    - the name of the structure variable;
    - the dot operator, which consists of a period (.);
    - and the member name
  - dot operator is member access operator.

```
// parts.cpp
// uses parts inventory to
#include <iostream>
using namespace std;
/////////////////////////
struct part
   {
   int modelnumber;
   int partnumber;
   float cost;
   };
/////////////////////////
int main()
   {
   part part1;

   part1.modelnumber = 6244;  //give
     members
   part1.partnumber = 373;
   part1.cost = 217.55F;
                              //displ
   cout << "Model "  << part1.modelnu
   cout << ", part "   << part1.partn
   cout << ", costs $" << part1.cost
   return 0;
   }
```

```
Output:
Model 6244, part 37
```





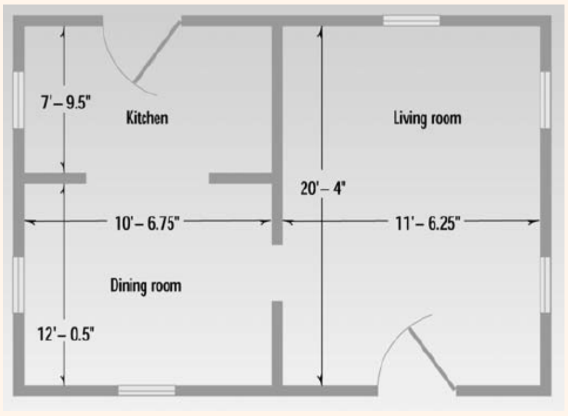
Other Structure Features

- Initialization
- Structure Variables in Assignment Statements
  - one structure variable can be assigned to another only when they are of the same structure type.

```
// partinit.cpp
// shows initialization of structure variables
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
     /
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;
   }
```

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

A Measurement Example

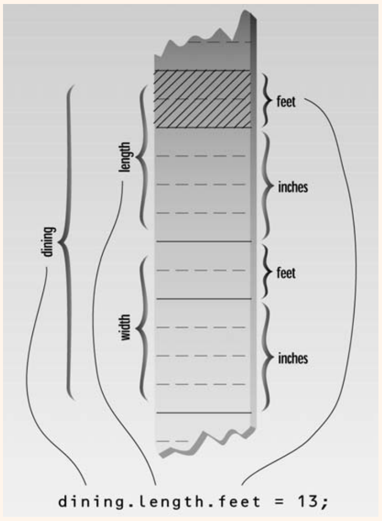


```
// englstrc.cpp
// demonstrates structures using English measurements
#include <iostream>
using namespace std;
///////////////////////////////////////////////////////////////
    /
struct Distance                  //English distance
   {
   int feet;
   float inches;
   };
//////////////////////
    /
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 1
      }
   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;
   }
```

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

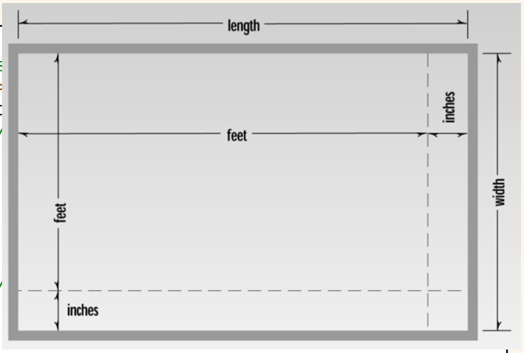
Structures Within Structures

- Accessing Nested Structure Members



- Initializing Nested Structures
  - `Room dining = { {13, 6.5}, {10, 0.0} };`
- Depth of Nesting: any depth

```
// englarea.cpp
// demonstrates nes
#include <iostream>
using namespace std
/////////////////
    /
struct Distance
   {
   int feet;
   float inches;
   };
/////////////////
    /
struct Room
   {
   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;
   }
```



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

A Card Game Example

```cpp
// cards.cpp
// demonstrates structures using playing cards
#include <iostream>
using namespace std;

const int clubs = 0;                        //suits
const int diamonds = 1;
const int hearts = 2;
const int spades = 3;

const int jack = 11;                        //face cards
const int queen = 12;
const int king = 13;
const int ace = 14;
////////////////////////////////////////////////////////////////
struct card
   {
   int number;   //2 to 10, jack, queen, king, ace
   int suit;     //clubs, diamonds, hearts, spades
   };
////////////////////////////////////////////////////////////////
int main()
   {
   card temp, chosen, prize;                //define cards
   int position;

   card card1 = { 7, clubs };               //initialize card1
   cout << "Card 1 is the 7 of clubs\n";

   card card2 = { jack, hearts };           //initialize card2
   cout << "Card 2 is the jack of hearts\n";

   card card3 = { ace, spades };            //initialize card3
   cout << "Card 3 is the ace of spades\n";

   prize = card3;            //copy this card, to remember it
   cout << "I'm swapping card 1 and card 3\n";
   temp = card3; card3 = card1; card1 = temp;

   cout << "I'm swapping card 2 and card 3\n";
   temp = card3; card3 = card2; card2 = temp;

   cout << "I'm swapping card 1 and card 2\n";
   temp = card2; card2 = card1; card1 = temp;

   cout << "Now, where (1, 2, or 3) is the ace of spades?
     ";
   cin >> position;

   switch (position)
      {
      case 1: chosen = card1; break;
      case 2: chosen = card2; break;
      case 3: chosen = card3; break;
      }
// compare cards
   if(chosen.number == prize.number &&
                chosen.suit == prize.suit)
      cout << "That's right!  You win!\n";
   else
      cout << "Sorry. You lose.\n";
   return 0;
   }
```

```
Output:
Card 1 is the 7 of clubs
Card 2 is the jack of hearts
Card 3 is the ace of spades
I’m swapping card 1 and card 3
I’m swapping card 2 and card 3
I’m swapping card 1 and card 2
Now, where (1, 2, or 3) is the ace of
spades? 3
Sorry. You lose.
```

Structures and Classes

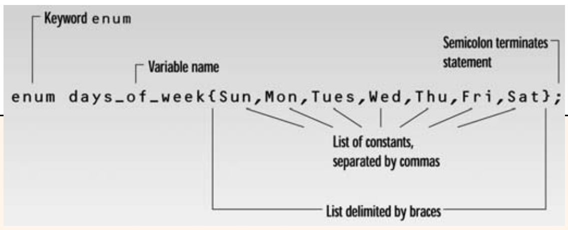
- Structures are usually used to hold data only, and classes are used to hold both data and functions.
- However, in C++, structures can in fact hold both data and functions. (In C they can hold only data.)
- The syntactical distinction between structures and classes in C++ is minimal, so they can in theory be used almost interchangeably.
- But most C++ programmers use structures as we have in this chapter, exclusively for data.
- Classes are usually used to hold both data and functions.

Enumerations

- Enumerated types work when you know in advance a finite (usually short) list of values that a data type can take on.
- An enumeration is a list of all possible values.
- In an enum you must give a specific name to every possible value.
- An enum declaration defines the set of all names that will be permissible values of the type. These permissible values are called enumerators
- You can't use values that weren't listed in the declaration.

```
// dayenum.cpp
// demonstrates enum types
#include <iostream>
using namespace std;
                                      //specify enum type
enum days_of_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

int main()
   {
   days_of_week day1, day2;   //define variables
                              //of type days_of_week
   day1 = Mon;                //give values to
   day2 = Thu;                //variables

   int diff = day2 - day1;    //can do integer
    arithmetic
   cout << "Days between = " << diff << endl;

   if(day1 < day2)            //can do comparisons
      cout << "day1 comes before day2\n";
   return 0;
   }
```

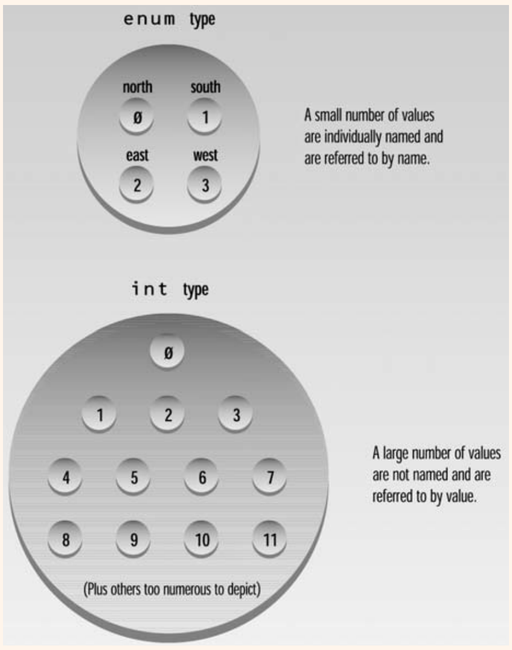
```
Output:
Days between = 3
day1 comes before day2
```



Keyword enum
Variable name
Semicolon terminates statement
enum days_of_week{Sun,Mon,Tues,Wed,Thu,Fri,Sat};
List of constants, separated by commas
List delimited by braces

Enumerations (2)

- treated internally as integers
- Ordinarily the first name in the list is given the value 0, the next name is given the value 1, and so on.
- But compiler warning if:
  - `day1 = 5;`



enum type
north
0
south
1
east
2
west
3
A small number of values are individually named and are referred to by name.
int type
0
1
2
3
4
5
6
7
8
9
10
11
(Plus others too numerous to depict)
A large number of values are not named and are referred to by value.

Enumerations (3): Organizing the Cards

```
// cardenum.cpp
// demonstrates enumerations
#include <iostream>
using namespace std;

const int jack = 11;         //2 through 10 are unnamed integers
const int queen = 12;
const int king = 13;
const int ace = 14;

enum Suit { clubs, diamonds, hearts, spades };
/////////////////////////////////////////////////////////////////
struct card
   {
   int number;               //2 to 10, jack, queen, king, ace
   Suit suit;                //clubs, diamonds, hearts, spades
   };
/////////////////////////////////////////////////////////////////
int main()
   {
   card temp, chosen, prize;                 //define cards
   int position;

   card card1 = { 7, clubs };                //initialize card1
   cout << "Card 1 is the seven of clubs\n";

   card card2 = { jack, hearts };            //initialize card2
   cout << "Card 2 is the jack of hearts\n";

   card card3 = { ace, spades };             //initialize card3
   cout << "Card 3 is the ace of spades\n";

   prize = card3;               //copy this card, to remember it
```

```
cout << "I'm swapping card 1 and card 3\n";
temp = card3; card3 = card1; card1 = temp;

cout << "I'm swapping card 2 and card 3\n";
temp = card3; card3 = card2; card2 = temp;

cout << "I'm swapping card 1 and card 2\n";
temp = card2; card2 = card1; card1 = temp;

cout << "Now, where (1, 2, or 3) is the ace of spades?
  ";
cin >> position;

switch (position)
   {
   case 1: chosen = card1; break;
   case 2: chosen = card2; break;
   case 3: chosen = card3; break;
   }
//compare cards
if(chosen.number == prize.number &&
              chosen.suit == prize.suit)
   cout << "That's right!  You win!\n";
else
   cout << "Sorry. You lose.\n";
return 0;
}
```

Output:

```
Card 1 is the 7 of clubs
Card 2 is the jack of hearts
Card 3 is the ace of spades
I'm swapping card 1 and card 3
I'm swapping card 2 and card 3
I'm swapping card 1 and card 2
Now, where (1, 2, or 3) is the ace of
spades? 3
Sorry. You lose.
```

Enumerations (4)

- Specifying Integer Values
  - `enum Suit { clubs=1, diamonds, hearts, spades };`
- Not Perfect

```
enum direction { north, south, east, west };
direction dir1 = south;
cout << dir1;
```

  - C++ I/O treats variables of enum types as integers, so the output would be 1 instead of `south`.
- Other Examples

```
enum months { Jan, Feb, Mar, Apr, May, Jun,
              Jul, Aug, Sep, Oct, Nov, Dec };

enum switch { off, on };

enum meridian { am, pm };

enum chess { pawn, knight, bishop, rook, queen, king };

enum coins { penny, nickel, dime, quarter, half-dollar, dollar };
```

Summary (1)

- Structures are an important component of C++, since their syntax is the same as that of classes.
  - In fact, classes are (syntactically, at least) nothing more than structures that include functions.
  - Structures are typically used to group several data items together to form a single entity.
  - A structure definition lists the variables that make up the structure. Other definitions then set aside memory for structure variables.
  - Structure variables are treated as indivisible units in some situations (such as setting one structure variable equal to another), but in other situations their members are accessed individually (often using the dot operator).

Summary (2)

- An enumeration is a programmer-defined type that is limited to a fixed list of values.
  - A declaration gives the type a name and specifies the permissible values, which are called enumerators.
  - Definitions can then create variables of this type. Internally the compiler treats enumeration variables as integers.
- Structures should not be confused with enumerations.
  - Structures are a powerful and flexible way of grouping a diverse collection of data into a single entity.
  - An enumeration allows the definition of variables that can take on a fixed set of values that are listed (enumerated) in the type's declaration.