Programming In C++

Course 2: Lecture 8, Structured data

Prepared By Dr. Ali Mohsin

Nineveh University- Faculty of IT- department of software

Combining Data into Structures

C++ allows you to group several variables together into a single item
known as a structure.

The limitation of arrays, however, is that all the elements must be of the same data type.

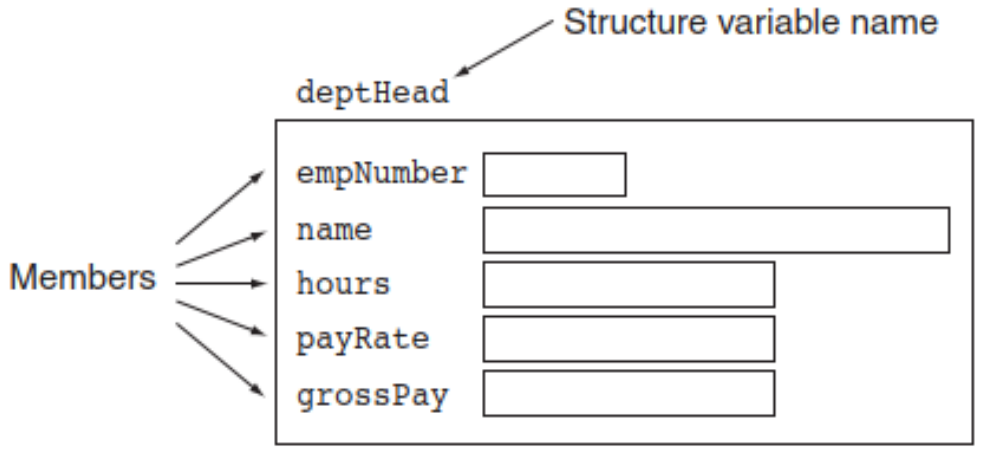
Sometimes a relationship exists between items of different types. For example, a payroll system might keep the variables shown in the following Table. These variables hold data for a single employee.

Variable Definition	Data Held
`int empNumber;`	Employee number
`string name;`	Employee's name
`double hours;`	Hours worked
`double payRate;`	Hourly pay rate
`double grossPay;`	Gross pay

Combining Data into Structures

Before a structure can be used, it must be declared. Here is the general format of a structure declaration:

```
struct PayRoll
{
 int empNumber; // Employee number
 char name[40]; // Employee's name
 double hours; // Hours worked
 double payRate; // Hourly pay rate
   double grossPay;  // Gross pay
};
```



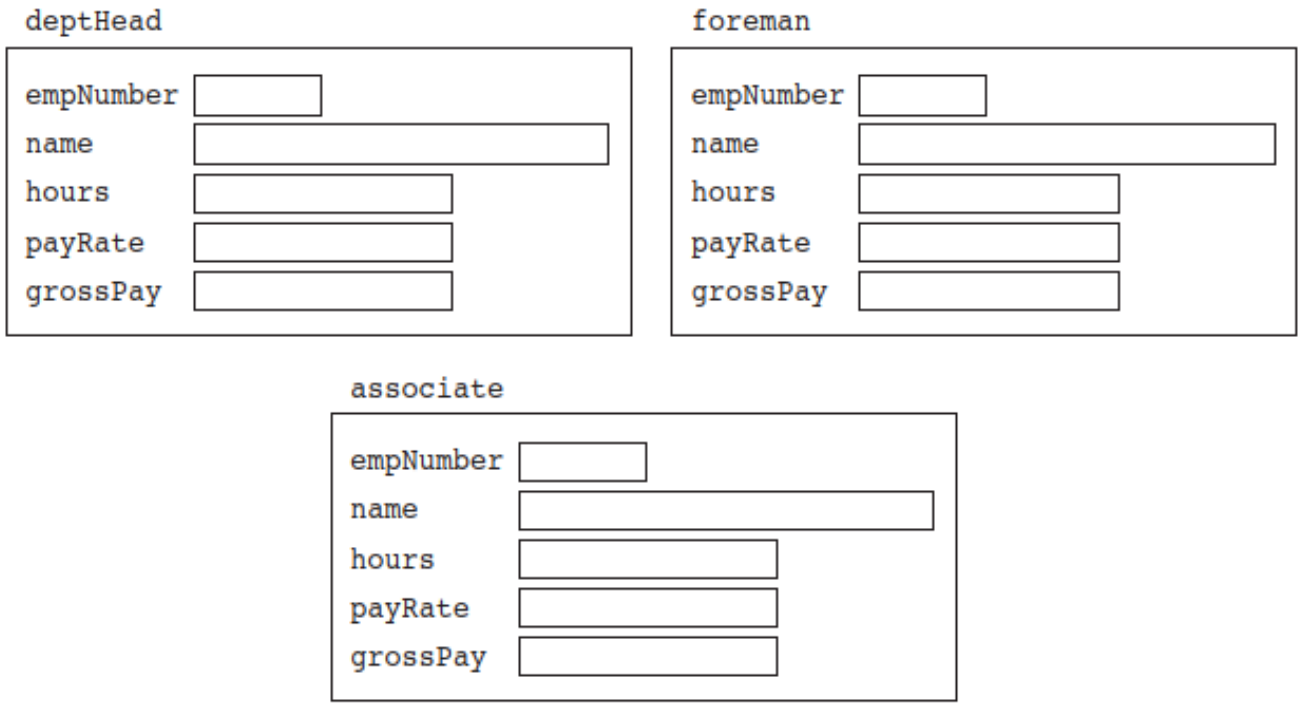
You can define variables of this type with simple definition statements, just as you would with any other data type. For example, the following statement defines a variable named deptHead:

PayRoll deptHead;

Combining Data into Structures

he following statement defines three PayRoll variables: deptHead, foreman, and associate:

PayRoll deptHead, foreman, associate;



Accessing Structure Members

The dot operator (.) allows you to access structure members in a program.

deptHead.empNumber = 475;

In this statement, the number 475 is assigned to the empNumber member of deptHead.

The dot operator connects the name of the member variable with the name of the structure variable it belongs to.

```
deptHead.empNumber = 475;
foreman.empNumber = 897;
associate.empNumber = 729;
```

Example

```
#include <iostream>
#include <cmath> // For the pow function
#include <iomanip>
using namespace std;

// Constant for pi.
const double PI = 3.14159;

struct Circle
{
    double radius;     // A circle's radius
    double diameter;   // A circle's diameter
    double area;       // A circle's area
};
```

```
int main()
{
    Circle c;    // Define a structure variable

    // Get the circle's diameter.
    cout << "Enter the diameter of a circle: ";
    cin >> c.diameter;

    // Calculate the circle's radius.
    c.radius = c.diameter / 2;

    // Calculate the circle's area.
    c.area = PI * pow(c.radius, 2.0);

    // Display the circle data.
    cout << fixed << showpoint << setprecision(2);
    cout << "The radius and area of the circle are:\n";
    cout << "Radius: " << c.radius << endl;
    cout << "Area: " << c.area << endl;
    return 0;
}
```

Comparing Structure Variables

You cannot perform comparison operations directly on structure variables.

For example, assume that circle1 and circle2 are Circle structure variables. The following statement will cause an error.

```
if (circle1 == circle2)  // Error!
```

In order to compare two structures, you must compare the individual members, as shown in the following code.

```
if (circle1.radius == circle2.radius &&
    circle1.diameter == circle2.diameter &&
    circle1.area == circle2.area)
```

Initializing a Structure

Assume the following structure declaration exists in a program:

```
struct CityInfo
{
    string cityName;
    string state;
    long population;
    int distance;
};
```

A variable may then be defined with an initialization list, as shown in the following:

```
CityInfo location = {"Asheville", "NC", 50000, 28};
```

Initializing a Structure

You do not have to provide initializers for all the members of a structure variable.

For example, the following statement only initializes the cityName member of location:

```
CityInfo location = {"Tampa"};
```

The following
statement only initializes the

cityName and state members, while leaving population and distance uninitialized:

```
CityInfo location = {"Atlanta", "GA"};
```

```
CityInfo location = {"Knoxville", "TN", , 90};  // Illegal!
```

Arrays of Structures

An array of structures is defined like any other array. Assume the following structure declaration exists in a program:

```
struct BookInfo
{
   string title;
   string author;
   string publisher;
   double price;
};

BookInfo bookList[20];
```

```
for (int index = 0; index < 20; index++)
{
   cout << bookList[index].title << endl;
   cout << bookList[index].author << endl;
   cout << bookList[index].publisher << endl;
   cout << bookList[index].price << endl << endl;
}
```

Example

Structure with a name called student contains(name, age, array of marks with 5 elements, av). The program read the required information and computes the average. You should declare array of structure for 10 students.

Then, the program adds 5 marks to students who their average less than 50.

```
#include <iostream>
#include <cstring>
using namespace std;
struct student{
char name[30];
int age;
float mark[5];
float av;
};
```

```
int main()
{
   student st[2];
   for (int i=0;i<2;i++){
      cin>>st[i].name;
      cin>>st[i].age;
      float s=0;
      for(int j=0;j<5;j++){
         cin>>st[i].mark[j];
         s+=st[i].mark[j];
      }
      st[i].av=s/5.0;
      cout<<"Enter the next student info"<<endl;
   }
```

```
   for(int i=0;i<2;i++)
      if(st[i].av<50)
      st[i].av=st[i].av+5;

      for(int i=0;i<2;i++){
         cout<<st[i].name<<" "<<st[i].av;
         cout<<endl;
      }
   return 0;
}
```

The End