Single Structures

- **Structure:** similar to a class with no member methods, only data members; allows the storage of different data types
- Can be used to represent a record in a file
- Keyword **`struct`** is used to declare a structure
- Declaration lists the data types, data names, and arrangement of data items

Single Structures (continued)

Example of a structure named "birth":

```
struct
{
   int month;
   int day;
   int year;
} birth;
```

- Data items are also called **fields** or **members** of the structure
- **Populating the structure:** assigning data values to the data items

Single Structures (continued)

Program 13.1

```
// a program that defines and populates a record
#include <iostream>
using namespace std;

int main()
{
  struct
  {
    int month;
    int day;
    int year;
  } birth;

  birth.month = 12;
  birth.day = 28;
  birth.year = 86;

  cout << "My birth date is "
       << birth.month << '/'
       << birth.day   << '/'
       << birth.year  << endl;

  return 0;
}
```

```
My birth date is 12/28/86
```

Single Structures (continued)

```
struct {int month; int day; int year;} birth;
```

```
struct
{
  int month;
  int day;
  int year;
} birth, current;
```

Single Structures (continued)

- Structure can be defined without a variable name by using a data type name; this does not actually create a structure but establishes a user-defined type

Example:

```
struct Date
{
  int month;
  int day;
  int year;
};
```

Single Structures (continued)

Program 13.2

```
#include <iostream>
using namespace std;

struct Date    // this is a global declaration
{
  int month;
  int day;
  int year;
};

int main()
{
  Date birth;

  birth.month = 12;
  birth.day = 28;
  birth.year = 86;

  cout << "My birth date is " << birth.month << '/'
       << birth.day   << '/'
       << birth.year  << endl;

  return 0;
}
```

```
Date birth = {12, 28, 86};
```

Single Structures (continued)

Example structure with mixed data types:

```
struct PayRec
{
  string name;
  int idNum;
  double regRate;
  double otRate;
};
```

Example initialization of this structure:

```
PayRec employee = {"H. Price",12387,15.89,25.50};
```

Single Structures (continued)

- Structure can include members of any valid C++ data type, including arrays and other structures
- For nested structures, each structure name must be used, followed by a period to access the data member

Arrays of Structures

- An array of structures allows the representation of a set of similar records
- Declare the structure first, then declare the array as the structure type

Example:

```
struct PayRec
 {int idnum; string name; double rate;};
PayRec employee[10];
```

- Each element of the **employee** array is a **PayRec** structure

Arrays of Structures

- An array of structures allows the representation of a set of similar records
- Declare the structure first, then declare the array as the structure type

Example:

```
struct PayRec
 {int idnum; string name; double rate;};
PayRec employee[10];
```

- Each element of the **employee** array is a **PayRec** structure

Arrays of Structures (continued)

Figure 13.2 A List of Structures

	Employee Number	Employee Name	Employee Pay Rate
1st structure →	32479	Abrams, B.	6.72
2nd structure →	33623	Bohm, P.	7.54
3rd structure →	34145	Donaldson, S.	5.56
4th structure →	35987	Ernst, T.	5.43
5th structure →	36203	Gwodz, K.	8.72
6th structure →	36417	Hanson, H.	7.64
7th structure →	37634	Monroe, G.	5.29
8th structure →	38321	Price, S.	9.67
9th structure →	39435	Robbins, L.	8.50
10th structure →	39567	Williams, B.	7.20

Arrays of Structures (continued)

- Reference a data item of the array element's structure by using the array index and a period, followed by the data item name:

Example:

```
employee[0].rate
```

- Standard array processing can be used on the list of structures

Arrays of Structures (continued)

Program 13.3

```
const int NUMRECS = 5;   // maximum number of records

struct PayRec   // this is a global declaration
{
  int id;
  string name;
  double rate;
};

int main()
{
  int i;
  PayRec employee[NUMRECS] = {
                              { 32479, "Abrams, B.", 6.72 },
                              { 33623, "Bohm, P.", 7.54},
                              { 34145, "Donaldson, S.",  5.56},
                              { 35987, "Ernst, T.", 5.43 },
                              { 36203, "Gwodz, K.", 8.72 }
                              };
  cout << endl;   // start on a new line
  cout << setiosflags(ios::left);  // left justify the output
  for ( i = 0; i < NUMRECS; i++)
    cout << setw(7)  << employee[i].id
         << setw(15) << employee[i].name
         << setw(6)  << employee[i].rate << endl;

  return 0;
}
```

```
32479  Abrams, B.      6.72
33623  Bohm, P.        7.54
34145  Donaldson, S.   5.56
35987  Ernst, T.       5.43
36203  Gwodz, K.       8.72
```

Structures as Function Arguments (continued)

Program 13.4

```
struct Employee       // declare a global type
{
  int idNum;
  double payRate;
  double hours;
};

double calcNet(Employee);   // function prototype

int main()
{
  Employee emp = {6782, 8.93, 40.5};
  double netPay;

  netPay = calcNet(emp);       // pass copies of the values in emp
    // set output formats
  cout << setw(10)
       << setiosflags(ios::fixed)
       << setiosflags(ios::showpoint)
       << setprecision(2);

  cout << "The net pay for employee " << emp.idNum
       << " is $" << netPay << endl;

  return 0;
}

double calcNet(Employee temp) // temp is of data type Employee
{
  return temp.payRate * temp.hours;
}
```

```
The net pay for employee 6782 is $361.66
```

Structures as Function Arguments (continued)

- Structure can also be passed by reference to allow the function to directly modify it

Example:

```
double calcNet(Employee& temp);
    // function declaration
netpay = calcNet(emp); // pass a reference
```

- A pointer can also be used:

```
double calcNet(Employee *pt); // function
                              // declaration
netPay = calcNet(&emp); // pass an address
```

Structures as Function Arguments (continued)

Program 13.4a

```
#include <iostream>
#include <iomanip>
using namespace std;

struct Employee       // declare a global data type
{
  int idNum;
  double payRate;
  double hours;
};

double calcNet(Employee&);   // function prototype

int main()
{
  Employee emp = {6782, 8.93, 40.5};
  double netPay;

  netPay = calcNet(emp);       // pass a reference
    // set output formats
  cout << setw(10)
       << setiosflags(ios::fixed)
       << setiosflags(ios::showpoint)
       << setprecision(2);

  cout << "The net pay for employee " << emp.idNum
       << " is $" <<  netPay << endl;

  return 0;
}

double calcNet(Employee& temp)   // temp is a reference variable
{
  return temp.payRate * temp.hours;
}
```

Structures as Function Arguments (continued)

- To reference a structure member using pointers, place the pointer variable in parentheses with the indirection operator

Example:

```
(*pt).payRate
```

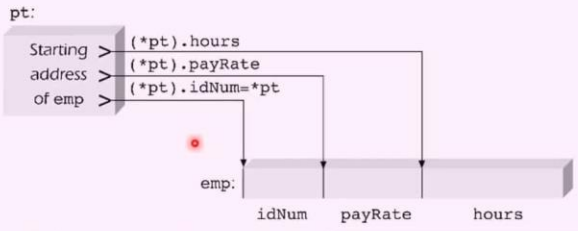


Figure 13.3 A pointer can be used to access structure members

Structures as Function Arguments (continued)

- Two operators can be used:
  - When accessing the structure directly, use the period notation:
    `structureName.memberName`
  - When using a pointer to the structure, use the arrow notation:
    `pointerName->memberName`

Example:

`(*pt).payRate` OR **`pt->payRate`**

```
(*pt).idNum can be replaced by pt->idNum
(*pt).payRate can be replaced by pt->payRate
(*pt).hours  can be replaced by pt->hours
```

Structures as Function Arguments (continued)

Program 13.5

```
#include <iostream>
#include <iomanip>
using namespace std;

struct Employee   // declare a global data type
{
  int idNum;
  double payRate;
  double hours;
};

double calcNet(Employee *);   //function prototype
```

```
int main()
{
  Employee emp = {6782, 8.93, 40.5};
  double netPay;
  netPay = calcNet(&emp);       // pass an address

    // set output formats
  cout << setw(10)
       << setiosflags(ios::fixed)
       << setiosflags(ios::showpoint)
       << setprecision(2);

  cout << "The net pay for employee " << emp.idNum
       << " is $" << netPay << endl;

  return 0;
}

double calcNet(Employee *pt)   // pt is a pointer to a
{                              // structure of Employee type
  return(pt->payRate * pt->hours);
}
```

Structures as Function Arguments (continued)

- Increment and decrement operators can be used with pointers to structure data members

Example:

```
++pt->hours  // adds one to the hours member
(++pt)->hours // increments pointer first,
              // then accesses hours member
```

- A structure can also be the return type of a function

Structures as Function Arguments (continued)

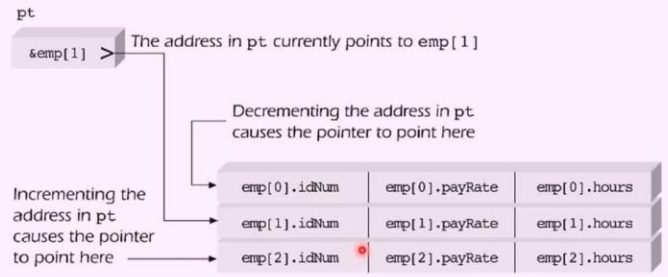


Figure 13.4 Changing pointer addresses

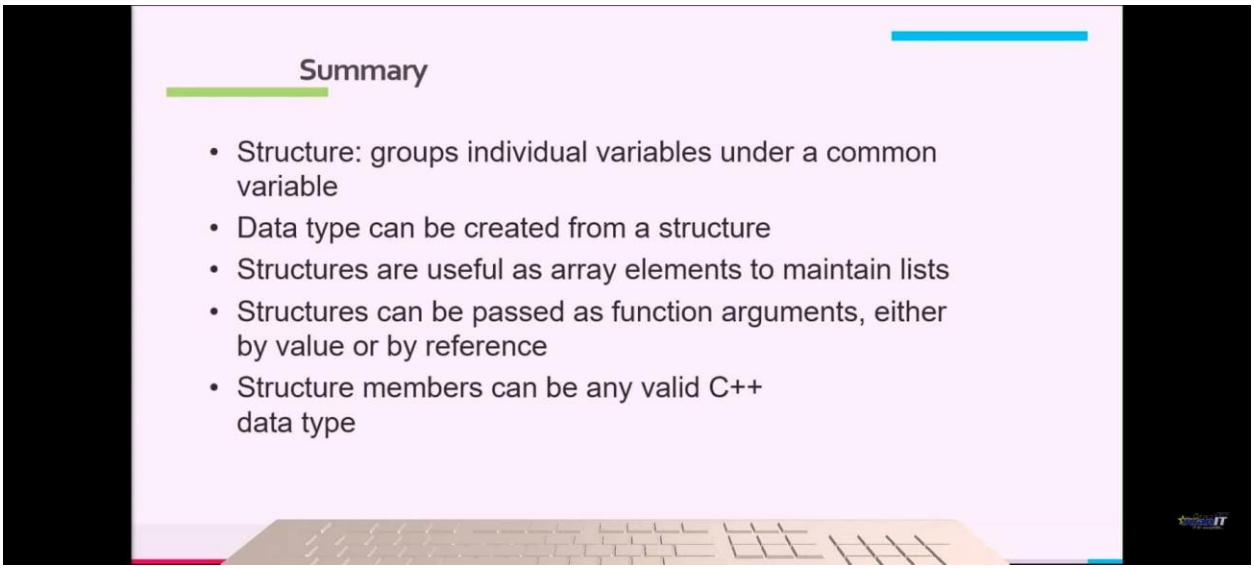
Structures as Function Arguments (continued)

Program 13.6

```
#include <iostream>
#include <iomanip>
using namespace std;

struct Employee      // declare a global data type
{
  int idNum;
  double payRate;
  double hours;
};

Employee getVals();  // function prototype

int main()
{
  Employee emp;

  emp = getVals();
  cout << '\nThe employee id number is ' << emp.idNum
       << '\nThe employee pay rate is $' << emp.payRate
       << '\nThe employee hours are ' << emp.hours << endl;

  return 0;
}

Employee getVals() // return an Employee structure
{
  Employee next;

  next.idNum = 6789;
  next.payRate = 16.25;
  next.hours = 38.0;

  return next;
}
```

The following output is displayed when Program 13.6 runs:

```
The employee id number is 6789
The employee pay rate is $16.25
The employee hours are 38
```


Summary
• Structure: groups individual variables under a common variable
• Data type can be created from a structure
• Structures are useful as array elements to maintain lists
• Structures can be passed as function arguments, either by value or by reference
• Structure members can be any valid C++ data type