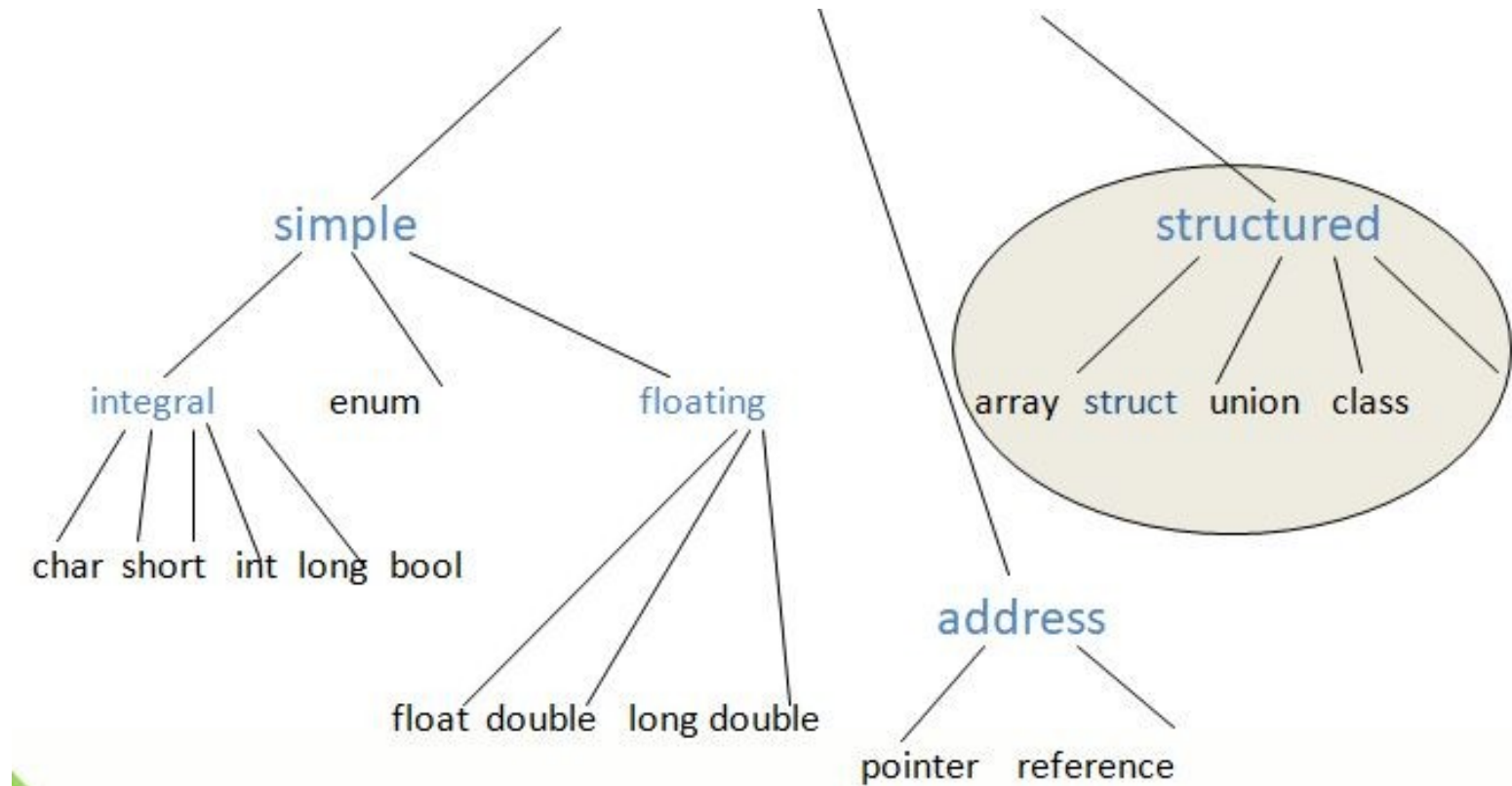


Computer Programming(ECEg1052)

Chapter-6

Structure

C++ Data Types



Introduction To Structure

- ▶ A structure is a collection of variables referenced under one name, providing a convenient means of keeping related information together.
 - ▶ A **Structure** is a collection of related data items, possibly different types.
 - ▶ **Heterogeneous** in that it can be composed of data of different types.
 - ▶ In contrast, **array** is **homogeneous**, it can contain only data of the same type.
- ▶ A structure declaration forms a template that may be used to create structure objects (that is, instances of a structure).
- ▶ The variables that make up the structure are called **members**.
 - ▶ **Structure members** are also commonly referred to as **elements or fields**.

Cont...

- ▶ Examples:
 - ▶ **Student record**: studentId, name, major, gender, startYear, ...
 - ▶ **Bank account**: accountNumber, name, currency, balance, ...
 - ▶ **Address book**: name, address, telephone number, ...
- ▶ In database applications, structures are called **records**.

Definition Of A Structure

- ▶ The keyword `struct` tells the compiler that a structure is being declared
- ▶ Defining a structure is giving the compiler a blueprint for creating your type.
- ▶ When you create a variable based on the structure definition, all of the member variables are created automatically and grouped under the name you gave.
- ▶ Variable have unique name that is different from its type.

Cont...

- ▶ Individual components of a **struct** type are called **members** (or **fields**).
- ▶ Members can be of **different types** (simple, array or **struct**).
- ▶ A **struct** is named as a whole while individual members are named using field identifiers.
- ▶ Complex data structures can be formed by defining **arrays of struct's**.

Cont...

- ▶ The `struct` declaration **names a type** and **names the members** of the `struct`.
- ▶ Note that each member name is given a type.
 - ▶ Also, member names must be unique within a `struct` type.
- ▶ It does not allocate memory when you make a `struct` declaration.
- ▶ None of the `struct` members are associated with memory until we declare a `struct` variable.

Cont...

```
struct <struct-type>{  
    <type> <identifier_list>;  
    <type> <identifier_list>;  
    ...  
};
```

} Each identifier defines a member of the structure.

► Example:

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

} The “Date” structure has 3 members, day, month & year.

Cont...

1. Student structure:

```
struct StudentInfo{  
    int Id;  
    int age;  
    char Gender;  
    double CGA;  
};
```

The “StudentInfo”
structure has 4 members
of different types.

2. Student grade structure :

```
struct StudentGrade{  
    char Name[15];  
    char Course[9];  
    int Lab[5];  
    int Homework[3];  
    int Exam[2];  
};
```

The “StudentGrade”
structure has 5
members of
different array
types.

Cont...

3. Bank Account structure

```
struct BankAccount{  
  char Name[15];  
  int AccountNo[10];  
  double balance;  
  Date Birthday;  
};
```

The “BankAccount” structure has simple, array and structure types as members.

4. Student Record structure

```
struct StudentRecord{  
  char Name[15];  
  int Id;  
  char Dept[5];  
  char Gender;  
};
```

The “StudentRecord” structure has 4 members.

struct type Declaration...

- ▶ If the **struct** type declaration precedes all functions it will be visible throughout the rest of the file.
- ▶ If it is placed within a function, only that function can use it.
- ▶ It is common to place **struct** type declarations with TypeNames in a (.h) header file and `#include` that file.
- ▶ It is possible for members of different **struct** types to have the same identifiers.
 - ▶ □ Also a non-struct variable may have the same identifier as a structure member.

Structure Variable Declaration

- ▶ Declaration of a variable of struct type:

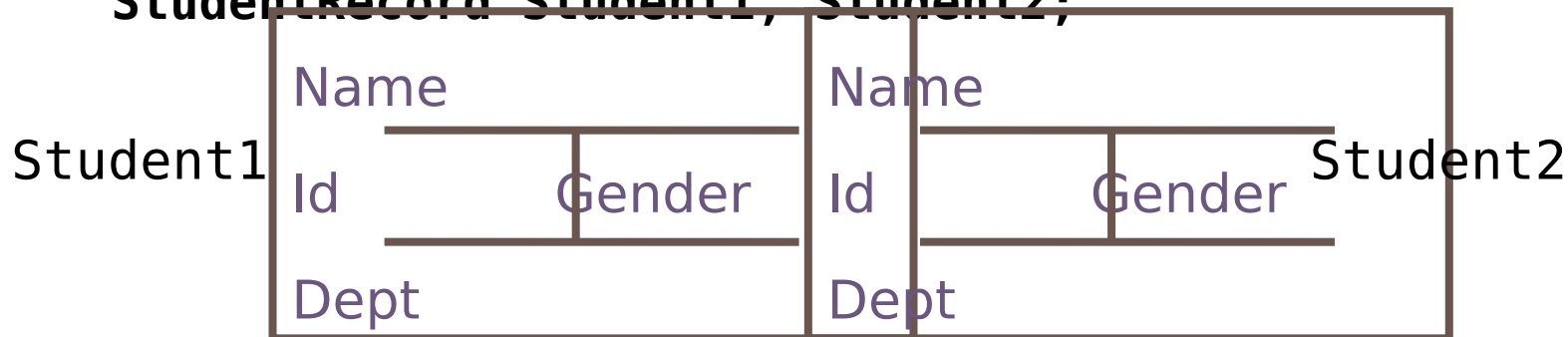
<struct-type> <identifier_list>;

OR

struct <struct-type> <identifier_list>;

- ▶ Example:

StudentRecord Student1, Student2;



Student1 and **Student2** are variables of **StudentRecord** type.

Accessing **struct** Members

- ▶ The members of a **struct** type variable are accessed with the dot (.) operator:

<struct-variable>.<member_name>;

- ▶ Dot (period) is the member selection operator.
- ▶ Example full demonstration

```
struct Person {  
    char name[50];  
    int age;  
    float salary; };  
  
int main()  
{  
    Person p1;  
    Cout<<"enter your full name"<<endl;  
    cin.get(p1.name, 50);  
}
```

Cont...

```
cout<<"enter your age"<<endl;
cin>>p1.age;
cout<<"enter your salary"<<endl;
cin>>p1.salary;
cout<<"these is your information"<<endl;

cout<<"Name"<<"="<<p1.name<<"Age"<<"="<<
p1.age<<endl;
    cout<<"Salary"<<"="<<p1.salary<<endl;
    getch();
    return 0;
}
```

Aggregate **struct** Operation

- ▶ **Aggregate Operation** is an operation on a data structure as a whole.
- ▶ I/O, arithmetic, and comparisons of entire **struct** variables are NOT ALLOWED!
- ▶ Valid operations on an entire **struct** type variable:
 - ▶ assignment to another **struct** variable of same type
 - ▶ pass to a function as argument (by value or by reference)
 - ▶ return as value of a function

struct-to-struct assignment

- ▶ The values contained in one **struct** type variable can be assigned to another variable of the same **struct** type.
- ▶ Example:

```
strcpy(Student1.Name, "Almaz Kebede");
```

```
Student1.Id = 12345;
```

```
strcpy(Student1.Dept, "PreEng");
```

```
Student1.gender = 'M';
```

```
Student2 = Student1;
```

Nested Structures

- ▶ **Nested or hierarchical structures.**
 - ▶ The type of a `struct` member can be another `struct` type.
 - ▶ are very useful when there is much detailed information in each record.
- ▶ Example: Information about each machine in a shop contains:
 - ▶ an `idNumber`,
 - ▶ a written description,
 - ▶ the purchase date,
 - ▶ the cost,
 - ▶ and a history (including failure rate, number of days down, and date of last service).

Cont...

```
struct DateType
{   int   month ;           // Assume 1 .. 12
    int   day ; // Assume 1 .. 31
    int   year ;           // Assume 1900 .. 2050
};
```

```
struct StatisticsType
{   float failRate ;
    DateType lastServiced ; // DateType is a struct type
    int downDays ;
};
```

```
struct MachineRec
{   int   idNumber ;
    string description ;
    StatisticsType history ; // StatisticsType is a struct type
    DateType purchaseDate ;
    float cost ;
};
```

```
MachineRec machine ;
```

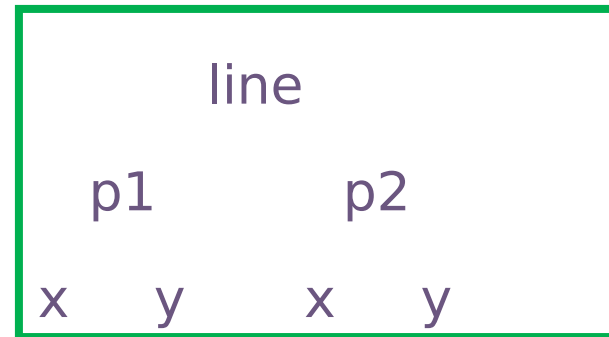
Cont...

► Examples 2:

```
struct point{  
  double x, y;  
};  
point P;
```

```
struct line{  
  point p1, p2;  
};  
line L;
```

```
struct triangle{  
  point p1, p2, p3;  
};  
triangle T;
```



Cont...

```
struct employeeType
{
    string firstname;
    string middlename;
    string lastname;
    string empID;
    string address1;
    string address2;
    string city;
    string state;
    string zip;
    int hiremonth;
    int hireday;
    int hireyear;
    int quitmonth;
    int quitday;
    int quityear;
    string phone;
    string cellphone;
    string fax;
    string pager;
    string email;
    string deptID;
    double salary;
};
```

```
struct nameType
{
    string first;
    string middle;
    string last;
};

struct addressType
{
    string address1;
    string address2;
    string city;
    string state;
    string zip;
};

struct dateType
{
    int month;
    int day;
    int year;
};

struct contactType
{
    string phone;
    string cellphone;
    string fax;
    string pager;
    string email;
};
```

```
struct employeeType
{
    nameType name;
    string empID;
    addressType address;
    dateType hireDate;
    dateType quitDate;
    contactType contact;
    string deptID;
    double salary;
};
```

Cont...

newEmployee		
name	first	
	middle	
	last	
emplID		
address	address 1	
	address 2	
	city	
	state	
	zip	
hireDate	month	
	day	
	year	
quitDate	month	
	day	
	year	
contact	phone	
	cellphone	
	fax	
	pager	
	email	
deptID		
salary		

employeeType newEmployee;

To sets the salary of newEmployee to 45678.00.

`newEmployee.salary = 45678.00;`

The statements:

`newEmployee.name.first = "Mary";`

`newEmployee.name.middle = "Beth";`

`newEmployee.name.last = "Simmons";`

The statement:

`cin >> newEmployee.name.first;`

reads and stores a string into newEmployee.name.first.

The statement:

`newEmployee.salary = newEmployee.salary * 1.05;`

updates the salary of newEmployee

`employeeType employees[100];`

The **for** loop:

```
for (int j = 0; j < 100; j++)
```

```
    cin >> employees[j].name.first >> employees[j].name.middle
```

```
    >> employees[j].name.last;
```

Arrays of structures

- ▶ The most common usage of structures is in arrays of structures.
- ▶ To declare an array of structures,
- ▶ First define a structure and then declare an array variable of that type.
- ▶ Example, to declare a 100-element array of structures of type **StudentRecord** write, `struct StudentRecord student_info[100];`
- ▶ To access a specific structure, index the structure name. For example, to print the Name code of structure 3, write
`cout<<student_info[2]. Name ;`
- ▶ Arrays of structures begin indexing at 0.

Cont...

```
1.    #include<iostream.h>
2.    #include<conio.h>
3.    struct stud{
4.        int studid;
5.        char sname[50];
6.        int year;
7.    };
8.    int main(){
9.        stud st[5];
10.       clrscr();
11.       for (int j=0;j<5;j++){
12.           cout<<"enter the ID credintials of stud"<<j<<endl;
13.           cin>>st[j].studid;
14.           cout<<"enter the name of stud"<<j<<endl;
15.           cin>>st[j].sname;
16.           cout<<"enter the year of stud"<<j<<endl;
17.           cin>>st[j].year;
18.       }
19.       clrscr();
20.       cout<<"StudId" <<"\t" <<"Sname" <<"\t"<<"Year" <<endl;
21.       for(int k=0;k<5;k++){
22.           cout<<st[k].studid<<"\t"<<st[k].sname<<"\t"<<st[k].year<<endl;
23.       }
24.       getch();
25.       return 0;
26.    }
```



Passing Structures to Functions

- ▶ When you pass a member of a structure to a function, you are actually passing the value of that member to the function.
- ▶ For example, consider this structure:

struct Example1

Here are examples of each member being passed to a function:

char x;	func(ex.x); /* passes character value of x */
int y;	func2(ex.y); /* passes integer value of y */
float z;	func3(ex.z); /* passes float value of z */
char s[10];	func4(ex.s); /* passes address of string s */
} ex;	func(ex.s[2]); /* passes character value of s[2] */

Passing Entire Structures to Functions

- ▶ When a structure is used as an argument to a function, the entire structure is passed using the standard **call-by-value** method.
 - ▶ Any changes made to the contents of the structure inside the function **do not affect** the structure used as an argument
- ▶ When using a structure as a parameter, remember that the type of the argument **must match** the type of the parameter.
- ▶ For example, in the following program both the argument **arg** and the parameter **parm** are declared as the same type of structure..

Cont...

```
/* Define a structure type. */
struct struct_type {
    int a, b;
    char ch;
} ;
void f1(struct struct_type parm);
int main(void)
{
    struct_type arg;
    arg.a = 1000;
    f1(arg);
    return 0;
}
void f1(struct struct_type parm)
{
    cout<<parm.a;
}
```

Structure Pointers

- ▶ C/C++ allows pointers to structures just as it allows pointers to any other type of variable.
- ▶ **Structure pointers** are declared by placing * in front of a structure variable's name.
- ▶ For example, assuming the previously defined structure **StudentRecord**, the following declares **studptr** as a pointer to data of that type:

```
struct StudentRecord *studptr;
```

Using Structure Pointers

- ▶ There are **two primary uses** for structure pointers:
 - ▶ **To pass a structure to a function** using call by reference,
 - ▶ **To create linked lists** and other **dynamic data structures** that rely on dynamic allocation.
- ▶ When a pointer to a structure is passed to a function, only the address of the structure is pushed on the stack.
- ▶ **Advantages:**
 - ▶ For very fast function calls.
 - ▶ When a function needs to reference the actual structure used as the argument, instead of a copy.
- ▶ By passing a pointer, the function can modify the contents of the structure used in the call.

Address of Structure Variable

- ▶ To find the address of a structure variable, place the & operator before the structure's name.
- ▶ For example, given the following fragment:

```
struct bal {  
    float balance;  
    char name[80];  
} person;  
struct bal *p; /* declare a structure pointer */  
then  
p = &person;
```

- ▶ places the address of the structure **person** into the pointer **p**.

Access The Members of A Structure using Pointer

- ▶ To access the members of a structure using a pointer, you must use the **->** operator.

- ▶ For example, this references the **balance** field:

`p->balance`

- ▶ The **->** is usually called the **arrow operator**, and consists of the minus sign followed by a greater-than sign.
- ▶ The arrow is used in place of the dot operator when you are accessing a structure member through a pointer to the structure.

Example

- This simple program, which prints the hours, minutes, and seconds on your screen.

```
struct my_time {  
    int hours;  
    int minutes;  
    int seconds;  
};  
void display(struct my_time *t);  
int main(void)  
{  
    struct my_time systime;  
    systime.hours = 19;  
    systime.minutes = 43;  
    systime.seconds = 57;  
  
    display(&systime);  
    return 0;  
}  
void display(struct my_time *t)  
{  
    cout<<t->hours<<":"<<t->minutes<<":"<<t->seconds;  
}
```

Assignment

- ▶ Reading assignment
 - ▶ Enumeration user define data type
- ▶ Assignment type individual assignment

