INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

EEC-101
Programming with C++

Module-3:
Structures

About Subject

- Aggregate Data-types:
  - Arrays
  - Pointers
  - Structures
  - Dynamic data and Pointers
  - Dynamic arrays

Declaration of Structure

- In C++, collection of heterogeneous data types can be grouped to form a structure.

- When this is done, the entire collection can be referred to by a structure name.

- Structures (also called structs) are a way to group several related variables into one place.

Each variable in the structure is known as a **member** of the structure.

- The individual components, which are called fields or members, can be accessed and processed separately. (syntax: struct_name.member)

- Distinctions between arrays and structures:
  1. All the elements of the array have the same data type, whereas in structure the components or fields can have different data types.
  2. Another distinction is that a component of an array is referred to by its position whereas in structure each component has a unique name
- Both have the similarity that they are defined with a finite number of components.

Structures

- The ‘struct’ keyword is used to create a structure. The general syntax to create a structure is as shown below:

```
struct <type name>
{
Member 1;
Member 2; ….

} <variable list>;
```

- The variables declared in the variable list are of type <type name>
- Members of a struct can be struct themselves leading to hierarchical structures.

Example

The user_defined_name is usually used.

Example:

```
struct date {
     int day;
     int month;
     int year;
};
date join;
```

```
struct date {
     int day;
     string month;
     int year;
};
date join;
```

Structure Variables

- A structure variable can either be declared with structure declaration or as a separate declaration like basic types.
- In these declarations, s1 is a variable of type student.

```
struct  student {
    int rollno;
    string name;
    int age;
    char gender;
    int height;
    int weight;
} s1;
```

```
struct  student {
  int rollno;
  string name;
  int age;
  char gender;
  int height;
  int weight;
}
struct student s1;
student s2;    // int x;
```

In C++, the struct keyword is optional before variable declaration of. In C, it is mandatory.

Example: Members

- Each member of structure is specified by a variable name followed by a period(.) and the member name.

- The . (dot) is a structure member operator which can be referred simply as the **dot operator.**

structurename . membername

Example: Members

```
#include<iostream>
using namespace std;
struct date{
    int day ;
    string month ;
    int year ;
} ;
int main ( )
{
date join {1,"Jan", 2024};
cout<<join. day ;
cout<<join .month ;
cout<<join .year ;
cout<<endl;
join. year=2020;
cout<<join. day ;
cout<<join .month ;
cout<<join .year ;
cout<<endl;
} // end of main ( )
```

- date is a structure consisting of three members, which can be referred to in the program as shown.

- The date of a day may be assigned as :
- join.day = 1;
- join.month = Jan;
- join.year = 2024;

1 Jan 2024
1 Jan 2020

```
#include<iostream>
using namespace std;
struct date {
    int day  ;
    string month ;
    int year ;
} ;
int main ( )
{
date join {1,"Jan", 2024};
cout<<join. day ;
cout<<join .month ;
cout<<join .year ;
cout<<endl;
join. year=2020;
cout<<join. day ;
cout<<join .month ;
cout<<join .year ;
cout<<endl;
} // end of main ( )
```

- The C++ complier will not read or write an entire structure as a single command like this :

  cin >> join ; // error

  cout << join ; // error

- It will read or write the members of a structure separately as :

  // read a structure

  cin >> join.day;

  cin >> join.month;

  cin >> join.year;

  // writing a structure on the screen

  cout << join.day ;

  cout << join.month ;

  cout << join.year ;

Example

WAP to define a structure named DATE which includes d, m, y. Declare a called DOB variable of type DATE. Initialize and display

```
using namespace std;
int main ()
{
struct sample {
int x;
float y; } a ;
a.x = 10;
a.y=20.20;

cout<<" content of x="<< a.x << endl;
cout << "content of y="<< a.y << endl;

struct date{
int d;
string m;
int y; } ;
date DOB;
DOB.d = 1;
DOB.m="JAN";
DOB.y=2020;

cout<<" DOB of xxx is on"<< DOB.d<<"st ";
cout<<DOB.m<<" "<<DOB.y;
return 0;
```

```
 content of x=10
content of y=20.2
 DOB of xxx is on1st JAN 2020
Process returned 0 (0x0)   execution time : 0.053 s
Press any key to continue.
```

Example

```
#include <iostream>
using namespace std;
 struct sample
 {
 int x ;
 float y ;
 };
sample a,b ;
int main ()
{
a.x = 10 ;
a.y = 20.20 ;
a.x = 10 ;
a.y = 20.20 ;
cout << " content of  a.x="<< a.x << endl ;
cout << " content of  a.y="<< a.y << endl ;

b.x = 14 ;
b.y = 44.20 ;
cout << " content of  b.x="<< b.x << endl ;
cout << " content of  b.y="<< b.y << endl ;

a=b;

cout << " content of  a.x="<< a.x << endl ;
cout << " content of  b.y="<< a.y << endl ;
system("pause");
return 0;
}
```

```
 content of  a.y=20.2
 content of  b.x=14
 content of  b.y=44.2
 content of  a.x=14
 content of  b.y=44.2
Press any key to continue . . .
```

int A[5];
int B[5];

A=B; // Not allowed for Arrays

Initialization of Structure

A structure can be initialized in the same way as any other data type using an initializer list.

Similar to int x[5]={1,2,3,4,5}

```
struct student {
long int rollno;
int age;
char gender;
float height;
float weight;
};
student s1 = {24115000, 18, 'M', 167.9, 75.6};
```

Arrays of Structure

- An array is a group of identical elements which are stored in consecutive memory locations in a common heading or a variable.

- A similar type of structure with a common heading or a common variable name is called arrays of structures.

```
//Example
struct student {
                    int rollno;
                    int age;
                    char gender;
                    float height;
                    float weight;
};
          student   s1; // one variable of type student
          student s[300]; // array of variables of type student
```

- student [300] is a structure variable.
- It may accommodate 300 structures of type student
- Each record may be accessed and processed separately like individual element of an array.

```
//Example
struct student {
                    int rollno;
                    int age;
                    char gender;
                    float height;
                    float weight;
};
 student   s1; // one variable of type student
student s[300]; // array of variables of type student
```

Array within a Structure

- A member of a structure can also be an array data type

```
//Example
struct student {
    char name [20];
     int roll;
    int subj [7];
};
student s1;
```

```
//Array within a structure and array of structure
#include <iostream>
using namespace std;
const int MAX = 40;
struct school {
    char name [25];
    long int rollno;
    int age;
    char gender;
    float height;
    float weight;
    };
school student [MAX]={ { "Raj", 2451000, 18, 'M', 167.9,60},
                       {"Simran", 2451300, 18, 'F', 156.7, 45.5}};
int main()
{
cout<<"Name\t"<<"Roll No "<<"age\t"<<"gender "<<endl;
for (int i=0;i<3;i++){
    cout<<student [i].name<<"\t"<<student [i] . rollno<<"\t";
    cout<<student [i]. age<<"\t"<<student [i]. gender<<endl;
}
return 0;}
```

student[0].age

Garbage

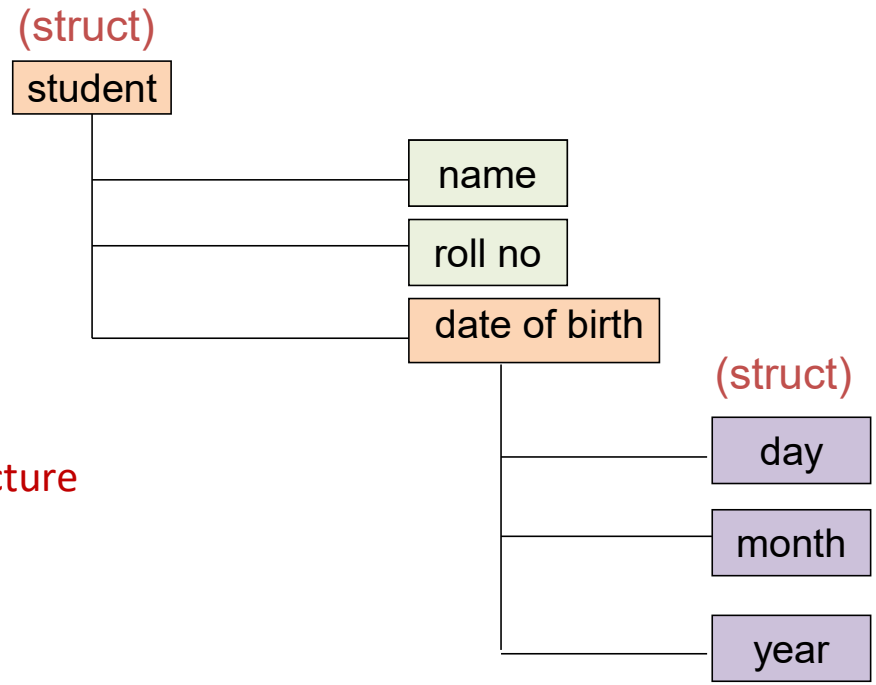
```
Name    Roll No age     gender
Raj     2451000 18      M
Simran  2451300 18      F
        0       0

Process returned 0 (0x0)   execution time : 0.023 s
Press any key to continue.
```

Structure within a Structure (Nested Structure)

- It is permissible to use a structure as a member of another structure.
- When a structure is declared as the member of another structure, it is called as a nested structure or structure within a structure.

```
struct date {
        int day;
        int month;
        int year;
        };
 struct student  {
        char  name[20];
        long int rollno;
        date dob; //structure is member of another structure
        };
student s1;
```



- To process the individual elements in a nested structure
  - first represent the structure variable name and the first structure
  - Then the field name of the first structure

e.g.

s1. rollno=200000;

s1.dob.day = 3;

s1.dob.month = 3;

s1.dob.year = 2011;

```
struct date {
int day;
int month;
int year;
};
 struct student  {
                char  name[20];
                long int rollno;
                date dob;
                };
student s1;
```

Example

```
int main()
{
    student e;

    cout<<"Enter the roll: ";
    cin>>e.roll;
    cout<<"Enter the Name: ";
    cin>>e.name;

    cout<<"Enter the City: ";
    cin>>e.address.city;
    cout<<"Enter the State: ";
    cin>>e.address.state;

    cout<<"\n\nRoll no: "<<e.roll;
    cout<<"\nEmployee Name: "<<e.name;
    cout<<"\nAddress: ";
    cout<<e.address.city<<", "<<e.address.state;
    cout<<endl;
    return 0;
}
```

struct ~~address~~ { string city, state; string }

```
Enter the roll: 1
Enter the Name: JamesThomason
Enter the City: Roorkee
Enter the State: Uttarakhand


Roll no: 1
Employee Name: JamesThomason
Address: Roorkee, Uttarakhand

Process returned 0 (0x0)   execution time : 71.090 s
Press any key to continue.
```

struct student {
int rollo;
string name;
~~address~~;
}

Hierarchical structure

```
struct machineRec {
int idNumber;
string description;
int purchaseDay;
int purchaseMnth;
int purchaseYr;
int lastServicedDay;
int lastServicedMnth;
int lastServicedYr;
float failRate;
int downDays;
float cost; };
```

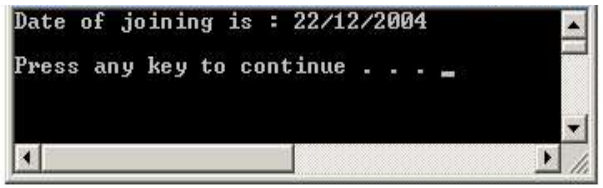
```
struct dateType
{ int day, month, year; };
struct statsType {
dateType lastServiced;
float failRate;
int downDays; };
struct machineRec {
int idNumber;
string description;
statsType history;
dateType purchaseDate;
float cost; };
```

Functions and Structures

- Function is a very powerful technique to decompose a complex problem into separate manageable parts or modules.
- Each part is called a function and is very much used to convert a complicated program into simple ones.
- Functions can be compiled separately, they can be tested individually and then invoked in the main program.
- A structure can be passed to a function as a single variable.
- Functions can also be members of a structure.

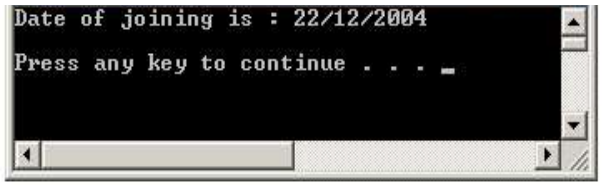
- The field or data member should be same throughout the program either in the main or in a function.

```
//Function and structure
#include <iostream>
using namespace std;
 struct date {
    int day;
    int month;
    int year;
};
date join;
void dispdate(date ); //function prototype
int main()
{
join.day   = 22;
join.month = 12;
join.year  = 2004;
dispdate(join);//Function calling
system("pause");
return 0;
}
//Function declaration
void dispdate(date join)
{
cout<<"Date of joining is : "<<join.day<<"/"<< join.month<<"/"<<join.year;
cout<<endl<<endl;
}
```

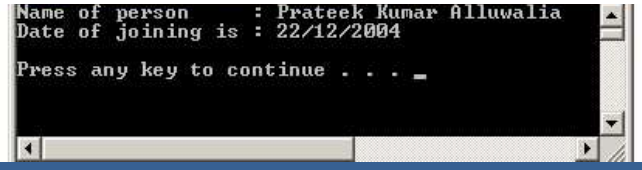


Example

```
//Function and structure
#include <iostream>
using namespace std;
 struct date {
    int day;
    int month;
    int year;
};
date join;
void dispdate(date ); //function prototype
int main()
{
join.day   = 22;
join.month = 12;
join.year  = 2004;
dispdate(join);//Function calling
system("pause");
return 0;
}
//Function declaration
void dispdate(date join)
{
cout<<"Date of joining is : "<<join.day<<"/"<< join.month<<"/"<<join.year;
cout<<endl<<endl;
}
```



```
1  //Function and structure, FUNCTION WITHIN STRUCTURE with nested structure
2  #include <iostream>
3  using namespace std;
4
5  struct date {
6      int day;
7      int month;
8      int year;
9  };
10 struct employee {
11     char* name; // if taken char name[30] then line 21 gives error
12     date doj; // nested structre
13     //Function declaration within structure
14     void dispdate()
15     {   cout<<"Name of person      : "<<name<<endl;
16         cout<<"Date of joining is : "<<doj.day<<"/"<< doj.month<<"/"<<doj.year;
17     cout<<endl<<endl;}
18 };
19 int main() {
20     employee join; // declare a varibale of type employee
21     join.name = "Prateek Kumar Alluwalia";//works with char* name
22     //cin>>join.name;// will give error with char* name, works with char name[30]
23     join.doj.day   = 22;
24     join.doj.month = 12;
25     join.doj.year  = 2004;
26 join.dispdate();//Function calling
27 system("pause");return 0;}
28
```



Pointers and Structures

- Member of a structure could be an ordinary data type such as int , float, char or even structure also.
- Pointer variables can be declared as members of a structure.
- A pointer is a variable which holds the memory address of a variable of basic data types such as int, float or sometimes an array.
- A pointer can be used to hold the address of a structure variable also.
- This pointer variable is used to construct complex databases using data structures such as linked lists, double-linked lists, and binary trees.

The following declaration is valid

```
struct sample {
 int x ;
 float y ;
 char s ;
 } ;
sample one;
sample *ptr  ;
ptr= &one;
```

where ptr is a pointer variable holding the address of the structure sample which is having three members such as int x, float y and char s.

```
struct sample {
 int x ;
 float y ;
 char s ;
 } ;
struct sample one;
struct sample *ptr  ;
ptr= &one;
```

one.x = 50;

- The pointer structure variable can be accessed by two ways as follows.

  (*pointer_structure_name).field_name = variable;

  (*ptr).x= variable;

- The parentheses are essential because the structure member period (.) has higher precedence over indirection operator(*).

- The pointer structure to structure can also be expressed as

  pointer_structure_name ->field_name = variable;

  *ptr->.x= variable;

Pointers and Structures

```
//A program to assign some values to the member of a structure
//using an indirection operator
//pointers and structures
#include <iostream>
using namespace std;
int main ()
{
struct example {
int x;
int y ;
};
example one ;
example *ptr;
ptr = &one ;
(*ptr).x = 10;
(*ptr).y = 20 ;
cout << "contents of x = "<< (*ptr ).x <<endl;
cout << "contents of y = "<< (*ptr ).y <<endl;
system("pause");
return 0;
}
```

```
contents of x = 10
contents of y = 20
Press any key to continue . . .
```

Pointers and Structures

```
//A program to assign some values to the member of a structure
//using an indirection operator
//pointers and structures
#include <iostream>
using namespace std;
int main ()
{
struct example {
int x;
int y ;
};
example one ;
example *ptr;
ptr = &one ;
//(*ptr).x = 10;
//(*ptr).y = 20 ;// can also be done as follows
ptr->x=100;
ptr->y=200;
cout << "contents of x = "<< (*ptr ).x <<endl;
cout << "contents of y = "<< (*ptr ).y <<endl;
system("pause");
return 0;
```

```
contents of x = 100
contents of y = 200
Press any key to continue . . .
```

```
//pointers and structures
#include <iostream>
using namespace std;
int main ()
{
struct example {
int x;
int y ;
};
example one ;
example *ptr;
ptr = &one ;
//(*ptr).x = 10;
//(*ptr).y = 20 ;// can also be done as follows
cout<<"enter x and y";
cin>>ptr->x;
cin>>ptr->y;
cout << "contents of x = "<< (*ptr ).x <<endl;
cout << "contents of y = "<< (*ptr ).y <<endl;
system("pause");
return 0;
}
```

```
enter x and y450
-32978
contents of x = 450
contents of y = -32978
Press any key to continue . . .
```

```
/*A program to declare a pointer variable as a member
of a structure and to display the contents of the
structure on screen and the address of the contents */
#include <iostream>
using namespace std;
int main()
{
struct sample {
int* ptr1;
float* ptr2;
};
sample one;
sample *strPtr = &one;
int value1 = 900;
float value2=111.6;

strPtr->ptr1= &value1;
strPtr->ptr2 = &value2;
cout<<"contents of the strPtr member ="<<endl;
cout <<*(strPtr->ptr1)<<endl;
cout << *(strPtr->ptr2) <<endl;
// following is equivalent
//cout <<*strPtr->ptr1<<endl;
//cout << *strPtr->ptr2 <<endl;
cout<<"Address of the strPtr member ="<<endl;
cout <<(strPtr->ptr1)<<endl;
cout << (strPtr->ptr2) <<endl;

system("pause");
return 0;
}
```

```
contents of the strPtr member =
900
111.6
Address of the strPtr member =
0x61fdfc
0x61fdf8
Press any key to continue . . .
```

Thanks