Introduction

Arrays are one of the most widely used data structures in programming languages. One disadvantage of using such a data type is that it can only hold multiple items of the same type (homogeneous collection of data). Structures overcome this problem by allowing the programmer to have an unlimited number of items of different data types in a single unit. The C++ programming language allows programmers to define program-specific data types (User-defined data types) through the use of **structures and classes**. Instances of these data types are known as **Structure/Class Variables** (objects). In C++, a **struct/class** contain Data "Content" and Function "Process/Method" as well and are extensions of the "C" Language **struct** datatype. In "C" **struct** cannot contain functions in it.

Defining a Structure

Structure groups various data members of same/different data types into a single unit and when a variable of type struct is created it forms a record.

Data members in the record are defined by naming a <data type>, followed by one or more <variable-name> (separated by commas). A semicolon is used to separate members of different data types in the same way as we declare normal variables of different data type.

Syntax:

```
struct [<struct type name>]
{
    [<data type> <member-name[, member-name, ...]>] ;
    [<data type> <member-name[, member-name, ...]>] ;
    ...
} [<structure variables>] ;
```

Valid Sample definitions of Structure:

Struct Sample 1	

To define a Structure Employee with data items as Eno (Employee number), Ename(Employee Name), and Salary	`struct Employee`
	`{`
	`  int Eno;`
	`  char EName[20];`
	`  float Salary;`
	`};`

Struct Sample 2	

To define a Structure Student to hold Rno (Roll number), Name (student name), Marks in 5 subjects and Total Marks	`struct Student`
	`{`
	`  int Rno;`
	`  char Name[20];`
	`  float Marks[5],Total;`
	`};`

Struct Sample 3	

To define a Structure POINT to hold Coordinates X and Y	`struct POINT`
	`{`
	`  int X,Y;`
	`};`

	Important Note: The struct definition Scope must be terminated with a semicolon (;)

Though both <struct type name> and <structure variables> are optional, one of the two must appear. A C++ programmer should not use structure variables along with the struct definition as it allows global variable declaration, which is discouraged in Object Oriented Programming approach(This is explained in the following samples):

Struct Sample 4

```
struct BOOK
{
  long AccessionNo;
  char Title[20],Author[20];
  float Price;
} B1,B2;
```

Struct Sample 5: Unnamed Structure

```
struct
{
  long TrainNo;
  char TrainName[20];
  char From[20],To[20];
} Y,T[5];
```

Here, in Struct Sample 4, along with definition of BOOK structure, two global variables B1 and B2 are declared. Struct Sample 5 demonstrates unnamed structure with a global declaration of a structure variable and a structure array (concept of a structure array is explained later).

Declaring Structure Variables

Syntax:

```
<struct type> <Structure Variable List>;
```

Assuming the definition of Employee structure in Struct Sample 1, an example of declaration and initialization of structure variables can be as follows:

```
Employee E;
Employee F={201,"Akriti Saxena",23000},G={205,"Jaya Sen"};
/*Note that the order of values and their type matches the order in
struct definition. We may also skip the last value(s), they will be
assigned as garbage values by default(as shown for variable G)*/
```

Accessing Members

As a structure variable consists of various parts the only way to work on the data members is to use the struct variable name followed by a period sign (.) followed by the data member name.

Example 1 (based on Struct Sample 4)

```
BOOK New,Old={12435,"Power of minds","Aladin",4500};
cout<<"Acno  :"<<Old.AccessionNo<<endl;
cout<<"Title :"<<Old.Title<<" Author:"<<Old.Author<<endl;
cout<<"Price :"<<Old.Price<<endl;
```

Output:

```
Acno  :12435
Title :Power of minds Author:Aladin
Price :4500
```

Example 2 (based on Struct Sample 2)

```
Student St={23,"Arun",78,80,90,77,98,0};
cout<<"RollNo:"<<St.Rno<<" Name:"<<St.Name<<endl;
cout<<"Marks:";
for (int I=0;I<5;I++)
{cout<<St.Marks[I]<<",";St.Total+=St.Marks[I];}
cout<<"Total Marks:"<<St.Total<<endl;
```

Output:

```
RollNo:23 Name:Arun
Marks:78,80,90,77,98,
Total Marks: 423
```

Manipulating contents of Structure variable

Example 3 (based on Struct Sample 3)

```
POINT P,Q={23,40};
cout<<"("<<Q.X<<","<<Q.Y<<")"<<endl;
P=Q;        // Assignment of one structure variable to another.
cout<<"("<<P.X<<","<<P.Y<<")"<<endl;
P.X++;
Q.Y+=5;
cout<<"("<<P.X<<","<<P.Y<<")"<<endl;
cout<<"("<<Q.X<<","<<Q.Y<<")"<<endl;
```

Output:

```
(23,40)
(23,40)
(24,40)
(23,45)
```

→ The only operation possible on complete structure variable is assignment of one structure variable to another as explained in **Example 3** above.

Passing Structure to a function

Example 4 (based on Struct Sample 3)

```
void EnterCoord(POINT &A);
void ShowCoord(POINT A);
float Slope(POINT A,POINT B);
void main()
{
  POINT P,Q;
  cout<<"Coordinate 1:";
  EnterCoord(P);      //** Note that this is a call by reference
  cout<<"Coordinate 2:";
  EnterCoord(Q);      //** Note that this is a call by reference
  ShowCoord(P);
  ShowCoord(Q);
  float Slp=Slope(P,Q);
  cout<<"Slope between two points:"<<Slp<<endl;
}
void EnterCoord(POINT &A)
{
  cout<<"X:";cin>>A.X;
  cout<<"Y:";cin>>A.Y;
}
void ShowCoord(POINT A)
{
  cout<<"("<<A.X<<","<<A.Y<<")"<<endl;
}
float Slope(POINT A,POINT B)
{
  float S=((float)(B.Y-A.Y))/(B.X-A.X);
  return S;
}
```

Output:

```
Coordinate 1:
X:3                          //Assuming user enters 3
Y:7                          //Assuming user enters 7
Coordinate 2:
X:1                          //Assuming user enters 1
Y:7                          //Assuming user enters 7
(3,7)
(1,7)
Slope between two points:0
```

Example 5:

```
struct COMPLEX
{
  float Real,Imag;
};
void Enter(COMPLEX &);
void Disp(COMPLEX );
COMPLEX Add(COMPLEX ,COMPLEX );
COMPLEX Sub(COMPLEX ,COMPLEX );
void main()
{
  COMPLEX A,B,Res1,Res2;
  cout<<"First Complex Number?
  Enter(A);
  cout<<"Second Complex Number?
  Enter(B);
  Res1=Add(A,B);
  Res2=Sub(A,B);
  cout<<"Complex Number 1"<<endl;
  Disp(A);
  cout<<"Complex Number 2"<<endl;
  Disp(B);
  cout<<"Sum"<<endl;
  Disp(Res1);
  cout<<"Difference"<<endl;
  Disp(Res2);
}
void Enter(COMPLEX &C)// Reference
{
  cout<<"Real Part:";cin>>C.Real;
  cout<<"Imag. Part:";cin>>C.Imag;
}
void Disp(COMPLEX C) // Value
{
  cout<<C.Real<<"+("<<C.Imag<<")i"<<endl;
}
COMPLEX Add(COMPLEX C1,COMPLEX C2) //Returning a Structure
{
  COMPLEX C;
  C.Real=C1.Real+C2.Real;
```

```
    C.Imag=C1.Imag+C2.Imag;
    return C;
}
COMPLEX Sub(COMPLEX C1,COMPLEX C2) //Returning Structure
{
    COMPLEX C;
    C.Real = C1.Real-C2.Real;
    C.Imag = C1.Imag-C2.Imag;
    return C;
}
```

Notice that in all the above examples, structure data members which are accessed using a '.' operator behave just like any normal variables of that type and can be worked upon with ease, the only difference being that since they belong to structure, the structure variable name has to be mentioned before the '.'operator.

Array of structure

An element of an array of structure would be equivalent to a single variable of that structure type (which contains the data of different type- equivalent to a record of information). A number of such elements constituting an array would thus be a number of records. To access each record we need the subscript number and to access each data item of that record we need the '.' followed by the name of the data item.

Example 6 (Based on Struct Sample 1):

```
//Declaration of an array of structure
Employee E[2];

//Entering content of an element of array of structure
cout<< "Enter details of an Employee:";
cin>>E[0].Eno;
gets(E[0].Ename);
cin>>E[0].Salary;

//Assigning content of an element to another element - same as
//assigning a structure variable to another
E[1]=E[0];

//Initialisation of an array of structure
Employee F[]={{101,"Aruna",3200},
              {105,"Priyam",4500},
              {105,"Priyam",4500},
              {105,"Priyam",4500}};

//Displaying content of an array of structure
for(int I=0;I<4;I++)
  cout<<F[I].Eno<<":"<<F[I].Ename<<":"<<F[I].Salary<<endl;
```

So instead of using three different arrays with homogeneous data (i.e., one for employee number, one for names and one for salary, instead, it is much simpler to use a single array of structure to hold the details of many employees under one single name having three different parts (Eno, Name and Salary).

Passing array of structure to a function

Example 7 (Based on Struct Sample 1):

```
void Enter(Employee [],int);
void SearchEno(Employee [],int );
void SortEno(Employee [],int );
void SortName(Employee [],int );
void SortSalary(Employee [],int );
void Show(Employee [],int);
void main()
{
  Employee Emp[10]; int N;
  cout<<"Enter Total no. of employees"(<=10):";cin>>N;
  Enter(Emp,N);
  Show(Emp,N);
  SearchEno(Emp,N);
  SortEno(Emp,N);
  cout<<"Sorted data (Eno ascending order):"<<endl;
  Show(Emp,N);
  SortName(Emp,N);
  cout<<"Sorted data (Name ascending order):"<<endl;
  Show(Emp,N);
  SortSalary(Emp,N);
  cout<<"Sorted data (Salary descending order):"<<endl;
  Show(Emp,N);
}
//To allow user to enter the content in the array
void Enter(Employee E[],int N)
{
  for (int I=0;I<N;I++)
  {
    cout<<"Eno    :";cin>>E[I].Eno;
    cout<<"Ename :";gets(E[I].Ename;
    cout<<"Salary:";cin>>E[I].Salary;
  }
}
//To search for an employee with an Eno using linear search
void Search(Employee E[],int N)
{
  int Enos,Found=0;
  count<<"Ëno to be searched:";cin>>Enos;
  for (int I=0;I<N;I++)
  {
    if (E[I].Eno==Enos)
    {
      cout<<"Ename :"<<E[I].Ename<<" Salary:"<<E[I].Salary<<endl;
      Found++;
    }
  }
  if (Found==0) cout<<"Sorry! Eno not found!"<<endl;
}
```

```
//To sort the array of employee in ascending order of Eno
void SortEno(Employee E[],int N)
{
  for (int I=0;I<N-1;I++)
    for (int J=0;J<N-I-1;J++)
    {
      if (E[J].Eno>E[J+1].Eno)//Comparison of Eno of jth and j+1th record*
      {
        Employee Temp=E[J];//Note Temp is of type Employee to
        E[J]=E[J+1];        // Swap of jth and j+1th record
        E[J+1]=T;
      }
    }
}
//To sort the array of employee in ascending order of Name
void SortName(Employee E[],int N)
{
  for (int I=0;I<N-1;I++)
    for (int J=0;J<N-I-1;J++)
    {
      if (strcmpi(E[J].Ename,E[J+1].Ename)>0)
      {
        Employee Temp=E[J];//Note Temp is of type Employee to
        E[J]=E[J+1];        // Swap of jth and j+1th record
        E[J+1]=T;
      }
    }
}

//To sort the array of employee in descending order of Salary
void SortSalary(Employee E[],int N)
{
  for (int I=0;I<N-1;I++)
    for (int J=0;J<N-I-1;J++)
    {
      if (E[J].Salary < E[J+1].Salary)
      {
        Employee Temp=E[J];//Note Temp is of type Employee to
        E[J]=E[J+1];        // Swap of jth and j+1th record
        E[J+1]=T;
      }
    }
}

//To display content of the array of employee
void Show(Employee E[],int N)
{
  for (int I=0;I<N;I++)
   cout<<E[I].Eno<<":"<<E[I].Ename<<":"<<E[I].Salary<<endl;
}
```