Question Bank-2
Based on Array & Structure

Q1-Q38 in Question Bank-1

Q39 Define Structure?

Structure is a collection of **heterogeneous** data type stored under one name and memory Is allocated contiguously.

Q40 Differentiate between Array & Structure?

Array	**Structure**
Array is collection of Homogenous data type	Structure is a collection of heterogeneous data
Array is passed as Reference parameter by default	Structure is passed as Value parameter by default
Array initializer can't be empty	Structure initializer can be empty

Q41 Give Two Similarities between Array & Structure

- Both can hold many values under same name.
- Memory is allocated contiguously for both array and structure

Q41 Define Open Array? Give its two Usages?

- Open Array is used to initialize an array elements at the time of creation.
- Open Array is used as formal parameter

Q42. Create a structure Date having following member

DD integer
MM integer
YY Integer

Create a structure Player with the following data member

Name String of 20 Characters
Runs Integer
Centuries Integer
Batting_Avg double
Debut Date Type

Use initializer to assign value to array of 3 Player

```
struct Date { int DD,MM,YY};

struct Player
{
char Name[20];
int Runs ,Centuries;
double Batting_Avg;
Date Debut;
};
Player P[3]={
        {"Yuvraj",    9000,  15,    50.5,  {10,06,2004} },
        {"Rohit",     7000,  10,    52.6,  {03,03,2006} },
        {"Virat",     8000,  20,    60.6,  {12,07,2008} }
        };
```

Q43 Give the Output

```
#include <iostream.h>
    struct PLAY
        {  int Score, Bonus;  };

        void Calculate(PLAY &P, int N=10)
        {       P.Score++;
                P.Bonus+=N;
        }
        void main()
        {
                PLAY PL={15,30};
                Calculate(PL,5);
                        cout<<PL.Score<<":"<<PL.Bonus<<endl;
                        Calculate(PL);

    cout<<PL.Score<<":"<<PL.Bonus<<endl;
    Calculate(PL,20);
                cout<<PL.Score<<":"<<PL.Bonus<<endl;
        }
```

Output

16,35
17,45
18,65

Explaination

PL={15,30}; means **PL.Scores**=15 **PL.Bonus**=30
Calculate (PL,5) will invoke a function Calculate and pass Actual Parameter **PL** as **Reference to P** and pass 5 as value to N so value of P and PL will work on same memory hence **P.Scores=15** and **P.Bonus=30**
P.Scores++ will increase the value of Scores by 1 so value of scores become **16**
P.Scores=16

P.Bonus+=N means **P.Bonus**=P.Bonus+N so 30+5 =**35**

P.Bonus=35

cout<<PL.Score<<":"<<PL.Bonus<<endl; will display **16,35**

as P & PL are reference and working on same memory address so it shares same value

II Time

Calculate(PL) since this time function Calculate is invoked with one Parameter PL which will be reference to P Second Parameter will use default Value that N=10

SO P.Scores=16 P.Bonus=35

P.Scores++ will increase the value of Scores by 1 so value of scores become **17**

P.Scores=17

P.Bonus+=N means **P.Bonus**=P.Bonus+N so 35+10 =**45**

P.Bonus=35

cout<<PL.Score<<":"<<PL.Bonus<<endl; will display **17,45**

as P & PL are reference and working on same memory address so it shares same value

III Time

Calculate (PL,20) will invoke a function Calculate and pass Actual Parameter **PL** as **Reference to P** and pass 20 as value to N so value of P and PL will work on same memory hence **P.Scores=17** and **P.Bonus=45**

P.Scores++ will increase the value of Scores by 1 so value of scores become **18**

P.Scores=18

P.Bonus+=N means **P.Bonus**=P.Bonus+N so 45+20 =**65**

P.Bonus=35

cout<<PL.Score<<":"<<PL.Bonus<<endl; will display **18,65**

as P & PL are reference and working on same memory address so it shares same value

Q44 What is the role of an initializer in an array of structure?

An initializer assigns values to the elements of an array of structure, when the array of structure is being created.

What happens when number values inside the initializer

i) is less than the number of elements in the array
Remaining elements will be 0 (zero) for int/float/double type and nul character for char type.

ii) is more than the number of elements in the array Compuiler will flag syntax error

Q45. Write any two differences between buitl-in functions random() and randomize().

random()	randomize()
• Return type int	• Return type void
• Has an int type parameter	• Has no parameter

Q46. Write any three differences between macro identifier and constant identifier.

Macro Identifier	Constant Identifier
• Created with compiler directive #define	• Created with keyword const
• Has no data type	• Has a data type
• No memory is allocated	• Memory is allocated
• Cannot use scope resolution operator with macro name	• Can use scope resolution operator with macro name
• Compiled code has no macro name	• Compiled code has constant name

Q47. Write a C++ expression to do the following:

i) Generate a four digits positive random integer
1000+random(9000)

i) Generate an uppercase random character
65+random(26) //'A'+random(26)

iii) Generate a double digit negative integer
-10-random(90) //-(10+random(10))

Q48. Write the output generated by the C++ program sement given below:

```
char str[]="nOvEmbErDeCeMbER";
for (int k=0; str[k]; k++)
    if (str[k]>='A' && str[k]<='Z')
        str[k]++;
    else
        k%2 ? --str[k] : str[k]-=32;
cout<<str<<endl;
```

Output: NPVFMaFqEdDdNaFS

Q49. Write the output generated by the C++ program segment given below:

```
int arr[8]={59, 77, 98, 39, 48, 99, 87, 65};
int s1=0, s2=0;
for (int x=0; x<8; x+=2)
{       int a=arr[x], b=arr[x+1];
        s1+=a/10;
        s2+=b%10;
        arr[x]=a/10+b/10;
        arr[x+1]=a%10+b%10;
}
for (int k=0; k<8; k++)
    cout<<arr[k]<<'*';
cout<<s1<<'*'<<s2<<endl;
```

Output: 12*16*12*17*13*17*14*12*26*30

Q50. Write C++ functions for following:

a) Sort an array of double using bubble sort. Array name and number of elements in the array are passed as parameters to the function. Return type of the function is **void**.

```
void bubblesort(double a[], int n)
{       for (int k=1; k<n; k++)
        for (int x=0; x<n-k; x++)
            if (a[x]>a[x+1])
            {
                double t=a[x];
                a[x]=a[x+1];
                a[x+1]=t;    }
}
```

b) Insert an integer in an array of integers. Array name, number of elements currently in the array, integer to be inserted and the position for insertion are passed as parameters to the function. Assume array size is a macro identifier SIZE. Return type of the function is **void**.

```
void arrayinsert(int a[ ], int &n, int pos, int item)
{   if (n==SIZE)
        cout<<"Array Overflow\n";
    else
    {
        for (int k=n-1; k>=pos; k--)
            a[k+1]=a[k];
        a[pos]=item;
        n++;    }
}
```

c) Count and display the number of even integers (integer divisible by 2), sum of even integers and average of even integers stored in an array of integers. Array name and number of elements in the array are passed as parameters to the function. Return type of the function is **void**.

```
void countsumavg(int a[], int n)
    {
        int s=0, c=0;                                   //double s=0; int c=0;
        for (int k=0; k<n; k++)
            if (a[k]%2==0)
            {
                s+=a[k];
                c++; }
        double avg=double(s)/n;    //double avg=s/n;
        cout<<"Count="<<c<<"\nSum="<<s<<"\nAverage="<<avg<<endl;
    }
```

a) To check whether a string is a Palindrome or not. The string that is to be checked is passed as a parameter to the function. Return type of the function is **void**.
Input: NITIN **Output**: Palindrome

Input: NUTAN **Output**: Not Palindrome

```
void chkstrpalin(char str[])
{
    int len=0;
    while (str[len])
        len++;
    int ri=len-1, palin=1;
    for (int le=0; le<ri && palin==1; le++, ri--)
        if (str[le]!=str[ri])
            palin=0;
    /*
        int palin=1;
        for (int k=0; k<len/2 && palin==1; k++)
            if (str[k]!=str[len-k-1])
                palin=0;
        OR
        int le=0, ri=len-1, palin=1;
        while (le<ri && palin==1)
            if (str[le++]!=str[ri--])
                palin=0;
    */
```

```
    if (palin==1)
        cout<<str<<" Palindrome\n";
    else
        cout<<str<<" Not Palindrome\n";
}
```

Q 51. Create a structure employee with following data members:

eno Employee Number (int type)
name Employee Name (string type)
basic Basic Salary (double type)
hrent House Rent (double type)
perks Extra allowance (double type)
gross Gross Salary (double type)

- Write a function empinput(), to input eno, name and basic of an employee passed as a parameter to the function.
- Write a function empsalary(), to calculate hrent, gsal and perks of an employee passed as a parameter to the function. Data members hrent and perks are calculated according to the table given below:

Basic Salary	**House Rent**	**Extra allowance**
>=100000	30% of Basic Salary	15% of Basic Salary
>100000 and <=400000	25% of Basic Salary	10% of Basic Salary
>400000	20% of Basic Salary	5% of Basic Salary

Gross Salary = Basic Salary + House Rent + Extra allowance

- **Write a main() function to create a variable of the type employee, call the function empinput(), empsalary() and display values stored in the variable of the type employee.**

```
#include<iostream.h>
#include<stdio.h>
struct employee
{
    int eno; char name[20]; double basic, hrent, perks, gross;
};
void empinput(employee &a)
{       cout<<"Code? "; cin>>a.eno;
        cout<<"Name? "; gets(a.name);
        cout<<"Basic? "; cin>>a.basic;
}
void empsalary(employee &a)
{       if (a.basic<=100000)
        {       a.hrent=0.30*a.basic;
                a.perks=0.15*a.basic; }
    else
    if (a.basic<=400000)
    {   a.hrent=0.25*a.basic;
        a.perks=0.10*a.basic;
        }
```

```
        else
        if (a.basic<=400000)
        {    a.hrent=0.20*a.basic;
             a.perks=0.05*a.basic;
        }
        a.gross=a.basic+a.hrent+a.perks;
    }
    void main()
    {
        employee emp;
        empinput(emp);
        empsalary(emp);
        cout<<"Code ="<<emp.eno<<endl;
        cout<<"Name ="<<emp.name<<endl;
        cout<<"Basic="<<emp.basic<<endl;
        cout<<"HRent="<<emp.hrent<<endl;
        cout<<"Perks="<<emp.perks<<endl;
        cout<<"Gross="<<emp.gross<<endl;
    }
```

Q54.Create a structure mobile with following members:

model	model name (string of 20 characters)
storage	internal storage (string type like "16 GB", "32 GB", "64 GB", …)
mpix	mega pixel (double type)
price	price (double type)

Write a C++ function to display details of all mobiles where internal storage is 32 GB and mega pixel exceeds 16 from an array of mobile. Array name, number of elements are passed as parameters to the function. At the end display number of such mobiles found. Return type of the function is **void**.

```
struct mobile
{
    char model[20], storage[8]; double mpix, price;

};
void count(mobile a[], int n)
{
    int c=0;
    for (int k=0; k<n; k++)
        if (strcmpi(a[k].storage, "32 GB")==0 && a[k].mpix>16)
        {          cout<<a[k].model<<","<<a[k].storage<<"," <<a[k].mpix<<","<<a[k].price<<endl;
        c++;
        }
    cout<<"Storage=32 GB and MPix>16="<<c<<endl;
}
```

Q55. What is use of keyword typedef? Show the use of typedef with a suitable example.

Keyword **typedef** is used to create a user defined data type.

```
typedef char string[20]; //string is the user defined data type
string myname;      /myname is a variable of the type string
```

Q56.**Explain the concept of function overloading with suitable examples.**

Function overloading: two or more functions having same name but differentiated on the basis of formal parameter(s).

```
void print(char c) { cout<<"Character="<<c<<endl; } void print(int i) {
cout<<"Integer="<<i<<endl; } void print(char s[]) { cout<<"String="<<s<<endl;
} void main()
{       print(2593);
        print("MANGAF");
        print('A');
}
```

Q57 Explain the concept of function with default parameter with a suitable example.

Function with default parameter: Formal parameter is assigned some fixed/constant value so that the function can be invoked with any actual parameter. If actual parameter is present, it replaces the default value of the formal parameter. Default parameter always starts from right.

```
void display(char c='$', int n=10){
        for (int k=0; k<n; k++)
                cout<<c;
                cout<<endl;       }
void main()
{       display(); display('*');
        display('@', 20);       }
```

Q58. Declare a structure **student** with the following members:

roll roll number
name student name
marks marks obtain out of 100
grade single letter (grade is calculated according to the table given below):

Marks	Grade
>=90	A
>=80 and <90	B
>=70 and <80	C
>=55 and <70	D
>=40 and <55	E
<40	F

Write a C++ function **resinput**(), that accepts **student** as a parameter and use the function to input values for data members roll, name and marks. Use the same function to calculate grade.

Write another C++ function **resdisplay**(), to display the content of structure variable of the type **student** on the screen. Structure variable of the type **student** is passed as a parameter to the function. Create a variable of the type **student** in the **main**() function. Invoke the function **resinput**() and **resdisplay**().

```
struct student
{       int roll; char name[20]; double marks; char grade;
};
void resinput(student &a)
```

```
{
    cout<<"Roll ? "; cin>>a.roll; cout<<"Name ? "; gets(a.name);
    cout<<"Marks? "; cin>>a.marks;
    if (a.marks>=90)
        a.grade='A';
    else
    if (a.marks>=80)     //Or, if (a.marks>=80 && a.marks<90)
        a.grade='B';
    else
    if (a.marks>=70)     //Or, if (a.marks>=70 && a.marks<80)
        a.grade='C';
    else
    if (a.marks>=55)     //Or, if (a.marks>=55 && a.marks<70)
        a.grade='D';
    else
    if (a.marks>=40)     //Or, if (a.marks>=40 && a.marks<55)
        a.grade='E';
    else a.grade='F';
}
void resdisplay(student a)
{  cout<<"Roll ="<<a.roll<<endl; cout<<"Name "<<a.name<<endl;
   cout<<"Marks="<<a.marks<<endl; cout<<"Grade="<<a.grade<<endl;  }
void main()
{
    student t; resinput(t);
    resdisplay(t);   }
```

Q58.

```
struct product
{           int code; char pname[20];
            double price;
};
```

a) Write a C++ function to sort an array of **product** using **insertion** sort on **price** in descending order. Array name and number of elements in the array are passed as parameters to the function.

```
void insertionsort(product a[ ], int n)
{
    for (int k=1; k<n; k++)
    {
        product t=a[k];
        int x=k-1;
        while (x>=0 && t.price>a[x].price)
        {
            a[x+1]=a[x];
            x--;
        }
        a[x+1]=t;
    }
}
```

b) Write a C++ function to sort an array of **product** using **bubble** sort on **pname**. Array name and number of elements in the array are passed as parameters to the function.

```
void bubblesort(product a[ ], int n)
{
    for (int k=1; k<n; k++)
        for (int x=0; x<n-k; x++)
            if (strcmpi(a[x].pname, a[x+1].pname)>0)
            {
                product t=a[x];
                a[x]=a[x+1]; a[x+1]=t;
            }
}
```

c) Write a C++ function to sort an array of **product** using **selection** sort on **code**. Array name and number of elements in the array are passed as parameters to the function.

```
void selectionsort(product a[ ], int n)
{
    for (int k=0; k<n-1; k++)
    {
        product t=a[k];
        int p=k;
        for (int x=k+1; x<n; x++)
            if (a[x].code<t.code)
            {
                t=a[x];
                p=x;
            }
        a[p]=a[k];
        a[k]=t;
    }
}
```

D) Write a C++ function to binary search for a **pname** in a sorted array of **product** sorted on **pname**. Array name, number of elements in the array and the **pname** to be searched are passed as parameters to the function. If search is successful then display **code**, **pname** and **price**. If search is not successful then display an error message

```
void binarysearch(product a[ ], int n, char pname[ ])
{
    int lb=0, ub=n-1, found=0, mid;
    while (lb<=ub && found==0)
    {
        mid=(lb+ub)/2;
        if (strcmpi(a[mid].pname, pname)<0)
            lb=mid+1;
        else
        if (strcmpi(a[mid].pname, pname)>0)
            ub=mid-1;
        else
```

```
                found=1;
        }
        if (found==1)
            cout <<a[mid].code<<","
                 <<a[mid].pname<<","
                 <<a[mid].price<<endl;
        else
            cout<<pname<<" not found in the array\n";
    }
```

e) Suppose **arr1**, **arr2** and **arr3** are three arrays of integers with **n1**, **n2**, and **n1**+**n2** elements respectively. Array **arr1** is sorted in descending order, array **arr2** is sorted in ascending order and array **arr3** to appear in ascending order. Use the function declaration given below for merging: void merge(int arr1[],int arr2[],int arr3[],int n1,int n2);

void merge(int arr1[ ], int arr2[ ], int arr3[ ], int n1, int n2)

```
{
    int x=n1-1, y=0, z=0;
    while (x>=0 && y<n2)
        if(arr1[x]<arr2[y])
         arr3[z++]=arr1[x--]
        else
        arr3[z++]=arr2[y++];
    while (x>=0)
        arr3[z++]=arr1[x--];
    while (y<n2)
        arr3[z++]=arr2[y++];
}
```

Some Useful Function based on Structure

struct Student{ int roll, char name[20];}; // this structure is used for all function below

```
void bubblesortroll(student arr[ ], int n)
{
  for(int x=1; x<n; x++)
      for(int k=0; k<n-x; k++)
          if (arr[k].roll>arr[k+1].roll)
          {
              student t=arr[k];
              arr[k]=arr[k+1];
              arr[k+1]=t;
          }
}
```

```
void bubblesortname(student arr[ ], int n)
  {
      for(int x=1; x<n; x++)
          for(int k=0; k<n-x; k++)
              if (strcmp(arr[k].name,
              arr[k+1].name)>0)
              {
                  student t=arr[k];
                  arr[k]=arr[k+1];
                  arr[k+1]=t;
              }
  }
```

```
void linearsearchroll(student arr[ ], int n, int
roll)
{   int x=0, found=0;
     while (x<n && found==0)
          if (roll ==arr[x].roll)
               found=1;
          else
               x++;
     if (found==1)
          cout<<roll<<" Found in the array\n";
     else
          cout<<roll<<" Does not Exist in the array\n";
}
```

```
void linearsearchname(student arr[ ], int n,
char name[ ])
   {
        int x=0, found=0;
        while (x<n && found==0)
             if (strcmp(name, arr[x].name)==0)
                  found=1;
             else
                  x++;
        if (found==1)
             cout<<name<<" Found in the array\n";
        else
             cout<<name<<" Does not Exist in the
             array\n";
   }
```

```
void selectionsortroll(student arr[ ], int n)
{
     for(int x=0; x<n-1; x++)
     {
          student min=arr[x];
          int pos=x;
          for(int k=x+1; k<n; k++)
               if (arr[k].roll<min.roll)
               {
                    min=arr[k];
                    pos=k;
               }
          arr[pos]=arr[x];
          arr[x]=min;
     }
}
```

```
void selectionsortname(student arr[ ], int n)
   {
        for(int x=0; x<n-1; x++)
        {
             student min=arr[x];
             int pos=x;
             for(int k=x+1; k<n; k++)
                  if (strcmp(arr[k].name,
                  min.name)<0)
                  {
                       min=arr[k];
                       pos=k;
                  }
             arr[pos]=arr[x];
             arr[x]=min;
        }
   }
```

```
void insertionsortroll(student arr[ ], int n)
{
     for(int x=1; x<n; x++)
     {
          student t=arr[x];
          int k=x-1;
          while(k>=0 && t.roll<arr[k].roll)
          {
               arr[k+1]=arr[k];
               k--;
          }
          arr[k+1]=t;
     }
}
```

```
void insertionsortname(student arr[ ], int n)
   {
        for(int x=1; x<n; x++)
        {
             student t=arr[x];
             int k=x-1;
             while(k>=0 && strcmp(t.name,
             arr[k].name)<0)
             {
                  arr[k+1]=arr[k];
                  k--;
             }
             arr[k+1]=t;
        }
   }
```

```
void binarysearchroll(student arr[ ], int n, int roll)
  {   int lb=0, ub=n-1;
      int found=0, mid;
      while (lb<=ub && found==0)
      {
          mid=(ub+lb)/2;
          if (roll<arr[mid].roll)
              ub=mid-1;
          else
          if (roll>arr[mid].roll)
              lb=mid+1;
          else
              found=1;
      }
      if (found==1)
          cout<<roll<<" Found in the array\n";
      else
          cout<<roll<<" Does not Exist in the
          array\n";
  }
```

```
void binarysearchname(student arr[ ], int n,
char name[ ])
 {
 int lb=0, ub=n-1, found=0, mid;
 while (lb<=ub && found==0)
 {
 mid=(ub+lb)/2;
 if (strcmp(name, arr[mid].name)<0)
 ub=mid-1;
 else
 if (strcmp(name, arr[mid].name)>0)
 lb=mid+1;
 else
 found=1;
 }
 if (found==1)
 cout<<name<<" Found in the array\n";
 else
 cout<<name<<" Does not Exist in the array\n";
 }
```

```
void arrinsert(student arr[ ], int& n, int pos,
student item)
{
    if (n==MAX)
        cout<<"Overflow\n";
    else
    {
        for (int x=n-1; x>=pos; x--)
            arr[x+1]=arr[x];
        arr[pos]=item;
        n++;
        cout<<item.roll<<','<<item.name<<','<<item.
        mark;
        cout<<" inserted in the array\n";
    }
}
```

```
void arrdelete(student arr[ ], int& n, int pos)
    {
        if (n==0)
            cout<<"Underflow\n";
        else
        {
            student item=arr[pos];
            for (int x=pos+1; x<n; x++)
                arr[x-1]=arr[x];
            n--;
            cout<<item.roll<<','<<item.name<<'
            ,'<<item.mark;
            cout<<" deleted from the array\n";
        }
    }
```

```
void mergeroll(student a[ ],student b[ ],student c[ ]
,int n1,int n2)
  {
      int i=0, j=0, k=0;
      while (i<n1 && j<n2)
          if (a[i].roll<b[j].roll)
              c[k++]=a[i++];
          else
              c[k++]=b[j++];
      while (i<n1)
          c[k++]=a[i++];
      while (j<n2)
          c[k++]=b[j++];
  }
```

```
void mergename(student a[ ],student b[ ]
student c[ ], int n1 ,int n2)
{
int i=0, j=0, k=0;
while (i<n1 && j<n2)
if (strcmp(a[i].name,b[j].name)<0)
c[k++]=a[i++];
else
c[k++]=b[j++];
while (i<n1)
c[k++]=a[i++];
while (j<n2)
c[k++]=b[j++];
}
```

Some Useful Integer Array / String Based function

```
int stringlength(char str[ ])
{
    int len=0;
    while (str[len])
        len++;
    return len;
}

void stringupper(char str[ ])
{
    for(int x=0; str[x]; x++)
        if (str[x]>='a' && str[x]<='z')
            str[x]=char(str[x]-32);
}

void stringlower(char str[ ])
{
    for(int x=0; str[x]; x++)
        if (str[x]>='A' && str[x]<='Z')
            str[x]=char(str[x]+32);
}

void stringtoggle(char str[ ])
{
    for(int x=0; str[x]; x++)
        if (str[x]>='A' && str[x]<='Z')
            str[x]=char(str[x]+32);
        else
        if (str[x]>='a' && str[x]<='z')
            str[x]=char(str[x]-32);
}
```

```
void stringjoin(char des[ ],char src [ ])
{
    int x=0;
    while (des[x])
        x++;
    int k=0;
    while (src[k])
        des[x++]=src[k++];
    des[x]='\0';
}

void stringcopy(char des[ ], char src[ ])
{
    int x=0;
    while (src[x])
    {
        des[x]=src[x];
        x++;
    }
    des[x]='\0';
}

void stringreverse(char str[ ])
{
    int len=0;
    while (str[len])
        len++;
    int l=0, r=len-1;
    while (l<r)
    {  char t=str[l];
        str[l]=str[r];
        str[r]=t;
        l++;
        r--;
    }
```

```
void bubblesort(int arr[ ], int n)
{
    for (int x=1; x<n; x++)
        for (int k=0; k<n-x; k++)
            if (arr[k]>arr[k+1])
            {
                int temp=arr[k];
                arr[k]=arr[k+1];
                arr[k+1]=temp;
            }
}
```

```
typedef char string[20];
void bubblesort(string arr[ ], int n)
{
    for (int x=1; x<n; x++)
        for (int k=0; k<n-x; k++)
            if (strcmp(arr[k], arr[k+1])>0)
            {
                string t;
                strcpy(t, arr[k]);
                strcpy(arr[k], arr[k+1]);
                strcpy(arr[k+1], t);
            }
}
```

```
void insertionsort(int arr[ ], int n)
{
    for(int k=1; k<n; k++)
    {
        int t=arr[k], x=k-1;
        while(x>=0 && t<arr[x])
        {
            arr[x+1]=arr[x];
            x--;
        }
        arr[x+1]=t;
    }
}
```

```
void selectionsort(int arr[ ], int n)
{
    for (int x=0; x<n-1; x++)
    {
        int min=arr[x], pos=x;
        for (int k=x+1; k<n; k++)
            if (arr[k]<min)
            {
                min=arr[k];
                pos=k;
            }
        arr[pos]=arr[x];
        arr[x]=min;
    }
}
```

```
void insertionStrsort(string arr[ ], int n)
{
    for (int x=1; x<n; x++)
    {
        string t;
        strcpy(t,arr[x]);
        int k=x-1;
        while (k>=0 && strcmp(t, arr[k])<0)
        {
            strcpy(arr[k+1], arr[k]);
            k--;
        }
        strcpy(arr[k+1], t);
    }
}
```

```
void selectionStrsort(string arr[ ], int n)
{
    for (int x=0; x<n-1; x++)
    {
        string min;
        strcpy(min,arr[x]);
        int pos=x;
        for (int k=x+1; k<n; k++)
            if (strcmp(min, arr[k])>0)
            {
                strcpy(min, arr[k]);
                pos=k;
            }
        strcpy(arr[pos], arr[x]);
        strcpy(arr[x], min);
    }
}
```

```
void merge(int a[ ], int b[ ], int c[ ], int n1, int n2)
{
    int i=0, j=0, k=0;
    while (i<n1 && j<n2)
        if (a[i]<b[j])
            c[k++]=a[i++];
        else
            c[k++]=b[j++];
    while (i<n1)
        c[k++]=a[i++];
    while (j<n2)
        c[k++]=b[j++];
}
```

```
void merge(string a[ ], string b[ ], string c[ ], int n1, int n2)
{
    int i=0, j=0, k=0;
    while (i<n1 && j<n2)
        if (strcmp(a[i],b[j])<0)
            strcpy(c[k++], a[i++]);
        else
            strcpy(c[k++], b[j++]);
    while (i<n1)
        strcpy(c[k++], a[i++]);
    while (j<n2)
        strcpy(c[k++], b[j++]);
}
```

```
void binarysearch(int arr[ ], int n, int item)
{
    int lb=0, ub=n-1, mid, found=0;
    while (lb<=ub && found==0)
    {
        mid=(lb+ub)/2;
        if (item<arr[mid])
            ub=mid-1;
        else
        if (item>arr[mid])
            lb=mid+1;
        else
            found=1;
    }
    if (found==1)
        cout<<item<<" found in the array\n";
    else
        cout<<item<<" not found in the array\n";
}
```

```
int binarysearch(string arr[ ], int n, string item)
{
    int lb=0, ub=n-1, mid, found=0;
    while (lb<=ub && found==0)
    {
        mid=(lb+ub)/2;
        if (strcmp(item, arr[mid])<0)
            ub=mid-1;
        else
        if (strcmp(item, arr[mid])>0)
            lb=mid+1;
        else
            found=1;
    }
    return found;
}
```

```
void arrayins(int arr[ ], int& n, int pos, int item)
{
    if (n==MAX)         //MAX is the size of the array
        cout<<"Overflow\n";
    else
    {
        for (int x=n; x>=pos; x--)
            arr[x+1]=arr[x];
        arr[pos]=item;
        n++;
    }
}
```

```
void arrayins(string arr[ ], int& n, int pos, string item)
{
    if (n==MAX)         //MAX is the size of the array
        cout<<"Overflow\n";
    else
    {
        for (int x=n-1; x>=pos; x--)
            strcpy(arr[x+1], arr[x]);
        strcpy(arr[pos], item);
        n++;
    }
}
```

```
void arraydel(int arr[ ], int& n, int pos)
{
    if (n==0)
        cout<<"Underflow\n";
    else
    {
        for (int x=pos+1; x<n; x++)
            arr[x-1]=arr[x];
        n--;
    }
}
```

```
void arraydel(string arr[ ], int& n, int pos)
{
    if (n==0)
        cout<<"Underflow\n";
    else
    {
        for (int x=pos+1; x<n; x++)
            strcpy(arr[x-1], arr[x]);
        n--;
    }
}
```

Function returning Structure Example

```
#include<iostream.h>
struct Time
{   int hh, mm;   };

void InputTime(Time& t)
{   cout<<"Hour  ? "; cin>>t.hh;
    cout<<"Minute? "; cin>>t.mm;
}
void ShowTime(Time t)
{   cout<<"Time="<<t.hh<<':'<<t.mm<<endl;  }

Time AddTime(Time t1, Time t2)
{   Time t;
    int m=t1.mm+t2.mm
    t.hh=t1.hh+t2.hh+m/60;
    t.mm=m%60;
    return t;
}
void main()
{   Time t1, t2;
    InputTime(t1);
    InputTime(t2);
    Time t3=AddTime(t1,t2);
    ShowTime(t1);
    ShowTime(t2);
    ShowTime(t3);
}
```

Return value of the function **AddTime**() is **Time** where **Time** is a structure. Statement **return t**, returns value stored in structure variable **t** to the calling function.

Running of the program **twice**, produces following output

```
Hour  ? 2
Minute? 30
Hour  ? 3
Minute? 10
Time=2:30
Time=3:10
Time=5:40
Hour  ? 4
Minute? 50
Hour  ? 3
Minute? 40
Time=4:50
Time=3:40
Time=8:30
```