```
void print2Array(string x[],string y[],int sizeX)
 {
        for(int i=0;i<sizeX;i++)
           {cout<<"\nx["<<i<<"]="<<x[i];
           cout<<"   y["<<i<<"]="<<y[i];}
                cout<<endl; }
 void fillArray(string x[],int sizeX)
  { cout<<"\n enter the elements of array\n";
        for(int i=0;i<sizeX;i++)
           cin>>x[i];   }
```

```
int main()
{
        string st[size],st2[size];
        fillArray(st,size);
        cout<<st[0];
        swap_arr(st,st2,size);
        print2Array(st,st2,size);}
```

1.13 Pointers

Lecture Four

C++'s data types are classified into three categories: simple, structured, and pointers. here discusses the third data type called the pointer data type. We have already seen how variables are seen as memory cells that can be accessed using their identifiers. This way we did not have to care about the physical location of our data within memory, we simply used its identifier whenever we wanted to refer to our variable.

The memory of your computer can be imagined as a succession of memory cells, each one of the minimal size that computers manage (one byte). These single-byte memory cells are numbered in a consecutive way, so as, within any block of memory, every cell has the same number as the previous one plus one. This way, each cell can be easily located in the memory because it has a unique address and all the memory cells follow a successive pattern.

As soon as we declare a variable, the amount of memory needed is assigned for it at a specific location in memory (its memory address). We generally do not actively decide the exact location of the variable that is a task automatically performed by the operating system during runtime. However, in some cases, we may be interested in knowing the address where our variable is being stored during runtime. **Pointer variable:** A variable whose content is an address (that is, a memory address).

```
dataType *identifier;
```

As an example, consider the following statements:
int *p;
char *ch;

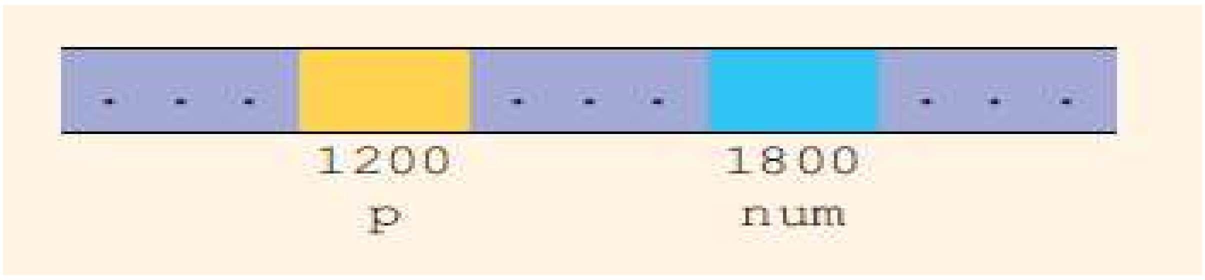
In these statements, both p and ch are pointer variables. The content of p points to a memory location of type int, and the content of ch points to a memory location of type char. Usually, p is called a pointer variable of type int, and ch is called a pointer variable of type char.
In C++, **the ampersand, &,** called the **address of the operator**, is a unary operator that returns the address of its operand. For example, given the statements:
int x;
int *p;
the statement:
p = &x;
assigns the address of x to p. That is, x and the value of p refers to the same memory location.

C++ also uses * as a unary operator. When used as a unary operator, *, commonly referred to as the **dereferencing operator** or indirection operator, refers to the object to which its operand (that is, the pointer) points. For example, given the statements:
int x = 25;
int *p;
p = &x; //store the address of x in p
the statement:
cout << *p << endl;
prints the value stored in the memory space pointed to by p, which is the value of x.
Also, the statement:
*p = 55;
stores 55 in the memory location pointed to by p—that is, in x.

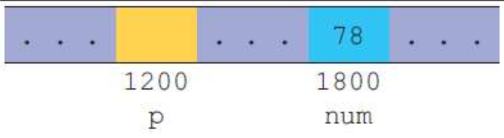
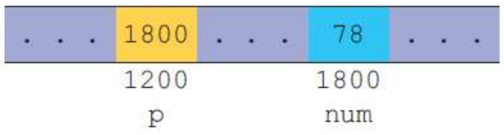
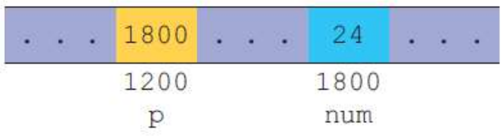
EX: Let us consider the following statements:

int *p;
int num;

In these statements, p is a pointer variable of type int, and num is a variable of type int. Let us assume that memory location 1200 is allocated for p, and memory location 1800 is allocated for num.



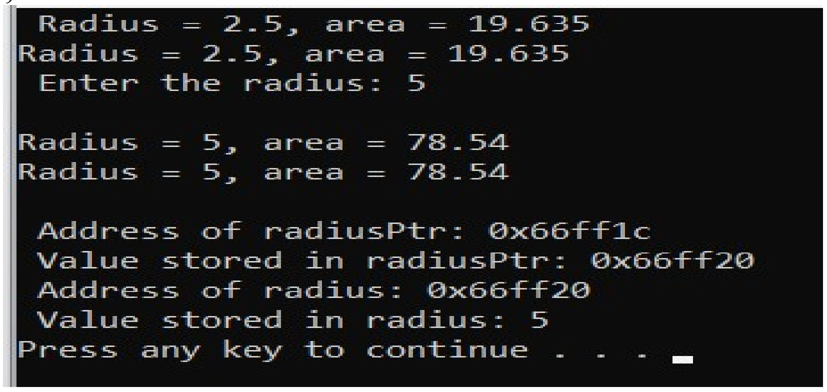
Consider the following statements:
1. num = 78;
2. p = &num;
3. *p = 24;

After Statement	Values of the Variables	Explanation
1		The statement `num = 78;` stores 78 into num.
2		The statement `p = &num;` stores the address of num, which is 1800, into p.
3		The statement `*p = 24;` stores 24 into the memory location to which p points. Because the value of p is 1800, statement 3 stores 24 into memory location 1800. Note that the value of num is also changed.

EX: The following program illustrates how pointer variables work:

```
#include <iostream>
#include <iomanip>
using namespace std;
const double PI = 3.1416;
int main()
```

```
{
double radius;
double *radiusPtr;
radius = 2.5;
radiusPtr = &radius;
cout << " Radius = " << radius
<< ", area = " << PI * radius * radius << endl;
//using pointers
cout << "Radius = " << *radiusPtr<< ", area = "<< PI * (*radiusPtr) * (*radiusPtr)
<< endl;

//read from pointer
cout << " Enter the radius: ";
cin >> *radiusPtr;
cout << endl;

cout << "Radius = " << radius << ", area = " << PI * radius * radius << endl;
cout << "Radius = " << *radiusPtr << ", area = " << PI * (*radiusPtr) * (*radiusPtr)
<< endl
cout << " Address of radiusPtr: "<< &radiusPtr << endl;
cout << " Value stored in radiusPtr: "<< radiusPtr << endl;
cout << " Address of radius: "<< &radius << endl;
cout << " Value stored in radius: "<< radius << endl;
return 0;
}
```

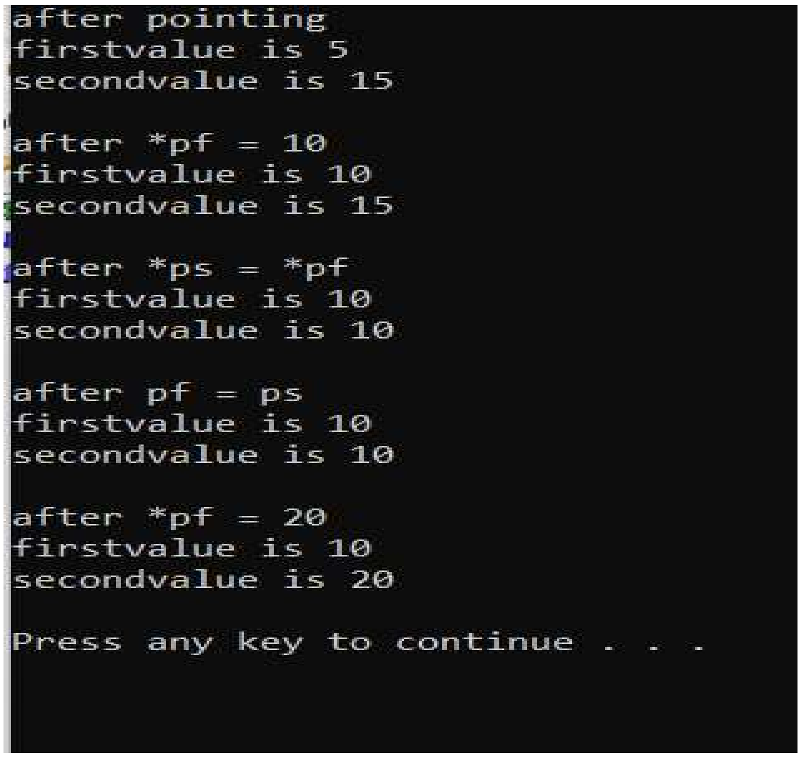


```
 Radius = 2.5, area = 19.635
Radius = 2.5, area = 19.635
 Enter the radius: 5

Radius = 5, area = 78.54
Radius = 5, area = 78.54

 Address of radiusPtr: 0x66ff1c
 Value stored in radiusPtr: 0x66ff20
 Address of radius: 0x66ff20
 Value stored in radius: 5
Press any key to continue . . .
```

EX: The following program illustrates how pointer variables work:

```
#include <iostream>
using namespace std;
int main ()
```

```
{
int firstvalue = 5, secondvalue = 15;
int * pf, * ps;

pf = &firstvalue; // pf = address of firstvalue
ps = &secondvalue; // ps = address of secondvalue
cout << "after pointing " <<endl;
cout<<"firstvalue is " << firstvalue << endl;
cout << "secondvalue is " << secondvalue << endl<< endl;

cout << "after *pf = 10 " <<endl;
*pf = 10; // value pointed by pf = 10
cout<<"firstvalue is " << firstvalue << endl;
cout << "secondvalue is " << secondvalue << endl<<endl;
*ps = *pf; // value pointed by ps = value pointed by
cout << "after *ps = *pf " <<endl;
cout<<"firstvalue is " << firstvalue << endl;
cout << "secondvalue is " << secondvalue << endl<<endl;

pf = ps; // pf = ps (value of pointer is copied)
cout << "after pf = ps " <<endl;
cout<<"firstvalue is " << firstvalue << endl;
cout << "secondvalue is " << secondvalue << endl<<endl;

*pf = 20; // value pointed by pf = 20
cout << "after *pf = 20 " <<endl;
cout<<"firstvalue is " << firstvalue << endl;
cout << "secondvalue is " << secondvalue << endl<<endl;

return 0; }
```



```
after pointing
firstvalue is 5
secondvalue is 15

after *pf = 10
firstvalue is 10
secondvalue is 15

after *ps = *pf
firstvalue is 10
secondvalue is 10

after pf = ps
firstvalue is 10
secondvalue is 10

after *pf = 20
firstvalue is 10
secondvalue is 20

Press any key to continue . . .
```

Pointers and arrays

The concept of array is very much bound to the one of pointer. In fact, the identifier of an array is equivalent to the address of its first element, as a pointer is equivalent to the address of the first element that it points to, so in fact they are the same concept. For example, supposing these two declarations:

```
int numbers [20];
int * p;
```

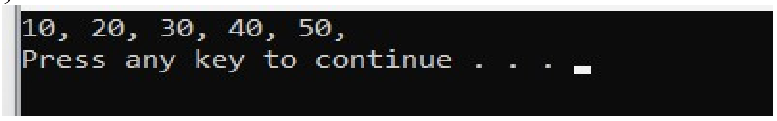
The following assignment operation would be valid:

```
p = numbers;
```

After that, p and numbers would be equivalent and would have the same properties. The only difference is that we could change the value of pointer p by another one, whereas numbers will always point to the first of the 20 elements of type int with which it was defined. Therefore, unlike p, which is an ordinary pointer, numbers is an array, and an array can be considered a constant pointer. Therefore, the following allocation would not be valid:

numbers = p;
EX:

```
#include <iostream>
using namespace std;
int main ()
{
        int numbers[5];
        int * p;
        p = numbers;
        *p = 10;
        p++; *p = 20;
        p = &numbers[2];
        *p = 30;
        p = numbers + 3;
        *p = 40;
        p = numbers;
        *(p+4) = 50;
        for (int n=0; n<5; n++)
        cout << numbers[n] << ", ";
        cout <<endl;
        return 0;
}
```



```
10, 20, 30, 40, 50,
Press any key to continue . . .
```

Pointer arithmetics

To conduct arithmetical operations on pointers is a little different than to conduct them on regular integer data types. To begin with, only addition and subtraction operations are allowed to be conducted with them, the others make no sense in the world of pointers.

Suppose that we define three pointers in this compiler:

char *mychar;

short *myshort;

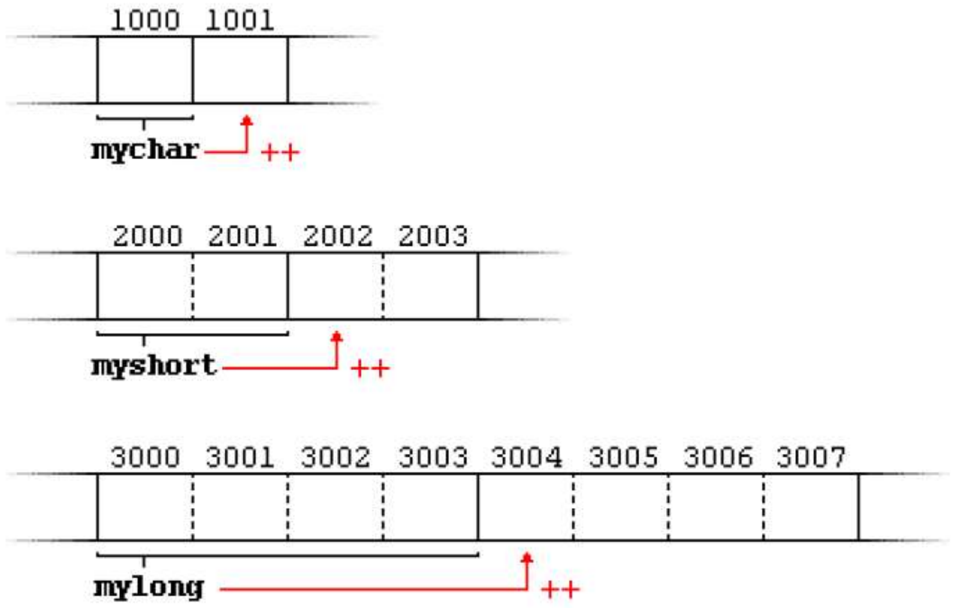
long *mylong;

and that we know that they point to memory locations 1000, 2000 and 3000 respectively.So if we write:

mychar++;

myshort++;

mylong++;



1.14 Data structures

We have already learned how groups of sequential data can be used in C++. But this is somewhat restrictive, since in many occasions what we want to store are not mere sequences of elements all of the same data type, but sets of different elements with different data types.

Data structures

A data structure is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different lengths. Data structures are declared in C++ using the following syntax:

```
struct structure_name
{
        member_type1 member_name1;
```

member_type2 member_name2;

member_type3 member_name3;

.

} object_names;

EX:

struct product {
int weight;
float price;
} apple, banana, melon;
	struct product {
	int weight;
	float price;
	} ;
	product apple;
	product banana, melon;

It is important to clearly differentiate between what is the **structure type name**, and what is an **object (variable)** that has this structure type. We can instantiate many objects (i.e. variables, like apple, banana and melon) from a single structure type (product).

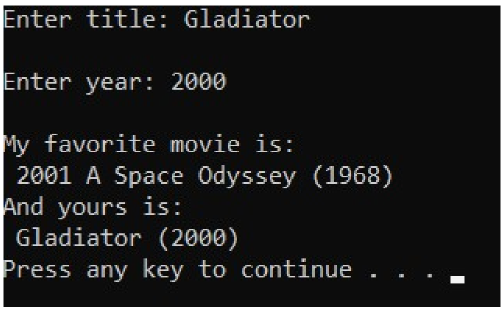
EX: C++ Program to assign data to members of a structure variable and display it.

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

struct Person
{
   string name;
   int age;
   float salary;
};

int main()
{
   Person p1;

   cout << "Enter Full name: ";
   cin>>p1.name;
```

```
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}
```

EX: example about structures pass as parameter of function

```
#include <iostream>
#include <string>

using namespace std;
struct movies_t {
string title;
int year;
} mine, yours;
void printmovie (movies_t movie)
{
cout << movie.title;
cout << " (" << movie.year << ")\n";
}
int main ()
{
string mystr;
mine.title = "2001 A Space Odyssey";
mine.year = 1968;
cout << "Enter title: ";
cin>>yours.title;
cout << "\nEnter year: ";
cin>>yours.year;

cout << "\nMy favorite movie is:\n ";
printmovie (mine);
cout << "And yours is:\n ";
```



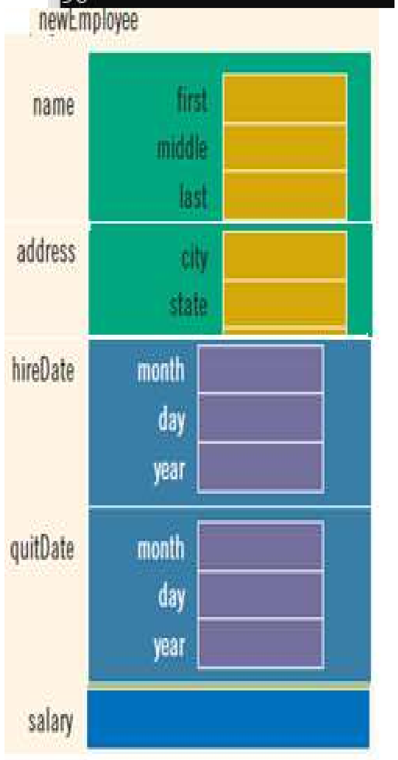
```
Enter title: Gladiator

Enter year: 2000

My favorite movie is:
 2001 A Space Odyssey (1968)
And yours is:
 Gladiator (2000)
Press any key to continue . . .
```

```
printmovie (yours);
return 0;
}
```

EX: example about Arrays in structs

```
#include <iostream>
using namespace std;
const int ARRAY_SIZE = 5;
struct listType
{
int listElem[ARRAY_SIZE]; //array containing the list
int listLength; //length of the list
};

int main()
{
   listType intList;

   intList.listLength = 0;
        intList.listElem[0] = 12;
        intList.listLength++;

        intList.listElem[1] = 37;
        intList.listLength++;
        for (int i=0;i<=5;i++)
          cin>>intList.listElem[i];
       for (int i=0;i<=5;i++)
          cout<<"\nintList.listElem["<<i<<"]=
"<<intList.listElem[i]      ;
}
```

```
60
90
90
```

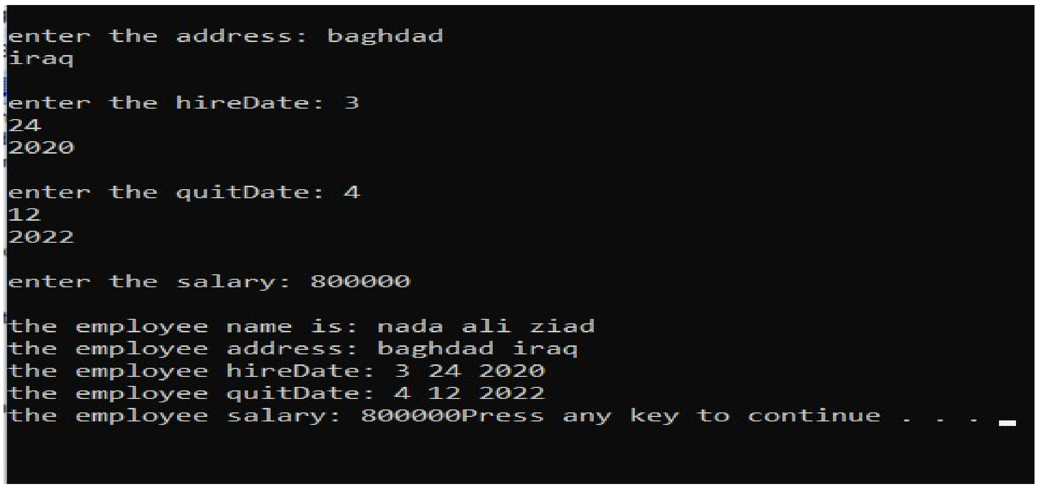


EX:structs within a struct

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

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

```
            string last;
};
struct addressType
{
            string city;
            string state;

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

struct employeeType
{
            nameType name;
            string empID;
            addressType address;
            dateType hireDate;
            dateType quitDate;
            double salary;
};
void employ_print(employeeType em)
     {

            cout<<"\nthe employee name is: ";
               cout<<em.name.first<<" "<<em.name.middle<<" "<<em.name.last;
        cout<<"\nthe employee address: ";
               cout<<em.address.city<<" "<<em.address.state;
                     cout<<"\nthe employee hireDate: ";
                     cout<<em.hireDate.month<<" "<<em.hireDate.day<<" "<<em.hireDate.year;
                     cout<<"\nthe employee quitDate: ";
                     cout<<em.quitDate.month<<" "<<em.quitDate.day<<" "<<em.quitDate.year;
                     cout<<"\nthe employee salary: ";
                     cout<<em.salary;
     }
int main()
{
   employeeType em1;

   em1.name.first = "nada";
            em1.name.middle = "ali";
            em1.name.last="ziad";
            cout<<"\nenter the address: ";
```

```
cin>>em1.address.city>>em1.address.state;
cout<<"\nenter the hireDate: ";
cin>>em1.hireDate.month>>em1.hireDate.day>>em1.hireDate.year;
cout<<"\nenter the quitDate: ";
cin>>em1.quitDate.month>>em1.quitDate.day>>em1.quitDate.year;
cout<<"\nenter the salary: ";
cin>>em1.salary;
employ_print(em1);}
```



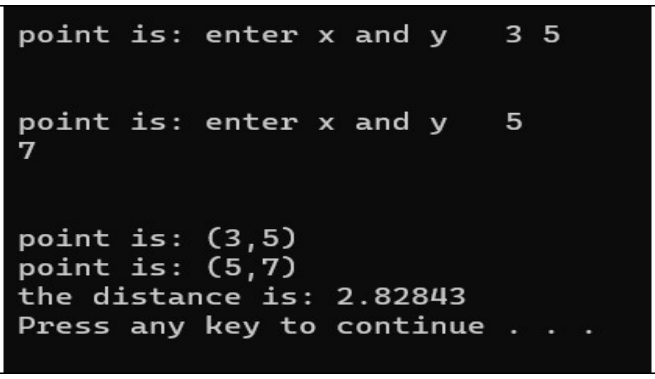
```
enter the address: baghdad
iraq

enter the hireDate: 3
24
2020

enter the quitDate: 4
12
2022

enter the salary: 800000

the employee name is: nada ali ziad
the employee address: baghdad iraq
the employee hireDate: 3 24 2020
the employee quitDate: 4 12 2022
the employee salary: 800000Press any key to continue . . . _
```

Examples of structures:

EX: program to find the distance between Two Points while representing the point as a struct.

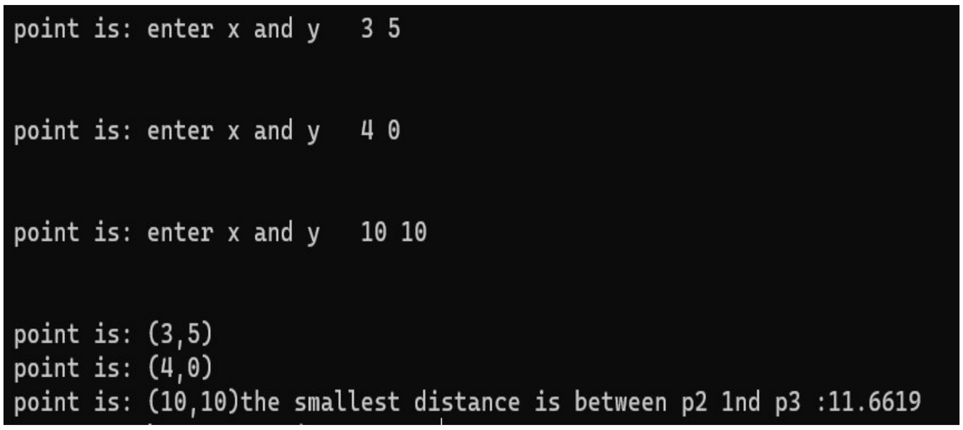
#include <iostream>
#include<cmath>
using namespace std;
struct point
{ float x;
float y;};
void point_print(point po) {
cout<<"\npoint is: ";
cout<<"("<<po.x<<","<<po.y<<")";}
point point_read()
{ point po;
	int main()
	{ point p1,p2;
	p1=point_read();
	p2=point_read();
	point_print(p1);
	point_print(p2);
	double dis;
	dis=sqrt(pow(p1.x-p2.x,2)+pow(p1.y-p2.y,2));
	cout<<"the distance is: "<<dis<<endl; }

```
        cout<<"\npoint is: ";
        cout<<"enter x and y   ";
           cin>>po.x>>po.y;
           cout<<endl;
          return po; }
```



```
point is: enter x and y   3 5


point is: enter x and y   5
7


point is: (3,5)
point is: (5,7)
the distance is: 2.82843
Press any key to continue . . .
```

EX: program to find the smallest distance between two points among three points while representing the point as a struct.



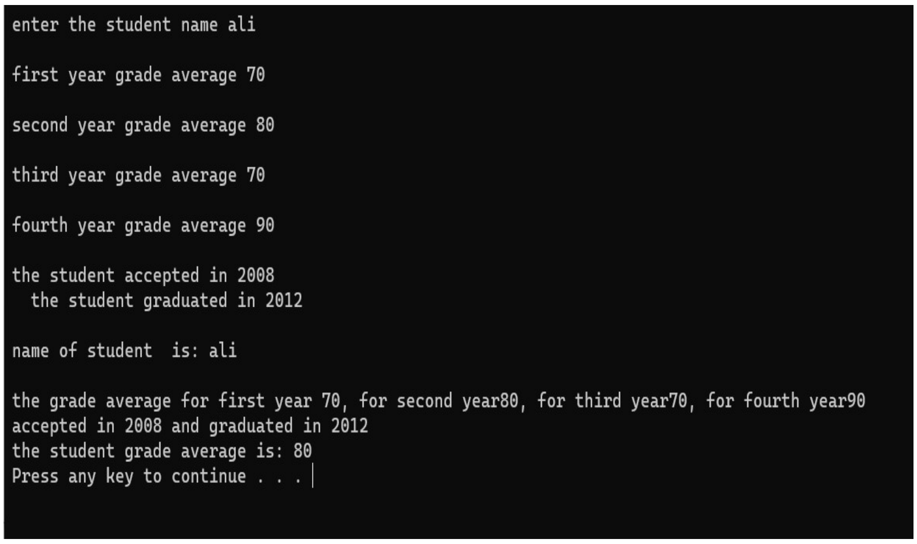
```
point is: enter x and y   3 5


point is: enter x and y   4 0


point is: enter x and y   10 10


point is: (3,5)
point is: (4,0)
point is: (10,10)the smallest distance is between p2 1nd p3 :11.6619
```

```
#include <iostream>
#include<cmath>
using namespace std;
struct point
{       float x;
        float y;};
void point_print(point po)  {   cout<<"\npoint is: ";
cout<<"("<<po.x<<","<<po.y<<")";}
point point_read()
   {      point po;
        cout<<"\npoint is: ";
        cout<<"enter x and y   ";
           cin>>po.x>>po.y;
           cout<<endl;
          return po; }
void poin_print(point po)
 {   cout<<"\npoint is: ";
  cout<<"("<<po.x<<","<<po.y<<")";}
double point_dis(point p1,point p2)
   {  double d;
        d=sqrt(pow(p1.x-p2.x,2)+pow(p1.y-p2.y,2));
          return d; }
```

```
int main()
{   point p1,p2,p3;
   p1=point_read();
   p2=point_read();
   p3=point_read();
   point_print(p1);
   point_print(p2);
   point_print(p3);
   if(point_dis(p1,p2)>=point_dis(p1,p3)&&point_dis(p1,p2)>=point_dis(p2,p3))
      cout<<"the smallest distance is between p1 and p2    :"<<point_dis(p1,p2)<<endl;
   else if(point_dis(p1,p3)>=point_dis(p1,p2)&&point_dis(p1,p3)>=point_dis(p2,p3))
     cout<<"the smallest distance is between p1 and p3 :"<<point_dis(p1,p3)<<endl;
   else
     cout<<"the smallest distance is between p2 and p3 :"<<point_dis(p2,p3)<<endl;}
```

EX: program to illustrate the information of the graduated student, his grade average to each year, and the acceptance year with the graduating year.

```
#include <iostream>
#include<string>
using namespace std;
struct student
{       string name;
        int GPA [4];
        int acc_y;

        int gra_y;};
        student
student_read()
    { student st;
          cout<<"enter the
student name ";
          cin>>st.name;
```



```
enter the student name ali

first year grade average 70

second year grade average 80

third year grade average 70

fourth year grade average 90

the student accepted in 2008
  the student graduated in 2012

name of student  is: ali

the grade average for first year 70, for second year80, for third year70, for fourth year90
accepted in 2008 and graduated in 2012
the student grade average is: 80
Press any key to continue . . .
```

```
            cout<<endl;
            cout<<"first year grade average ";
            cin>>st.GPA[0];
            cout<<endl;
            cout<<"second year grade average ";
            cin>>st.GPA[1];
            cout<<endl;
             cout<<"third year grade average ";
            cin>>st.GPA[2];
            cout<<endl;
            cout<<"fourth year grade average ";
            cin>>st.GPA[3];
            cout<<endl;
            cout<<"the student accepted in ";
             cin>>st.acc_y;
             cout<<"  the student graduated in ";
             cin>>st.gra_y;
             return st;
             }
int ave(int ar[])
{
         return(ar[0]*0.10+ar[1]*0.20+ar[2]*0.30+ar[3]*0.40);
}
void student_print(student st)  {
   cout<<"\nname of student  is: "<<st.name<<endl;
```

```
  cout<<"\nthe grade average for first year "<<st.GPA[0];
  cout<<", for second year"<<st.GPA[1]<<", for third year"<<st.GPA[2];
  cout<<", for fourth year"<<st.GPA[3]<<endl;
  cout<<"accepted in "<<st.acc_y<<" and graduated in "<<st.gra_y<<endl;
  cout<<"the student grade average is: "<<ave(st.GPA)<<endl;
 }

int main()
{
        student st1;
        st1=student_read();
   student_print(st1);
        return 1;

}
```