

Unary operator overloading:

---

```
//Complex:
```

```
using namespace std;
```

```
#include<iostream>
```

```
struct Complex
```

```
{
```

```
    int real,imag;
```

```
    Complex()
```

```
    {
```

```
        //    cout<<"\n\ndefault constructor called\n";
```

```
        this->real=0;
```

```
        this->imag=0;
```

```
    }
```

```
    Complex(int r,int i)
```

```
    {
```

```
        //    cout<<"\n\nparameterised constructor called\n";
```

```
        this->real=r;
```

```
        this->imag=i;
```

```
    }
```

```
    void setReal(int r)          //setters(mutator)
```

```
    {
```

```
        this->real=r;
```

```
    }
```

```
    void setImg(int i)          //setters(mutator)
```

```
    {
```

```

        this->imag=i;
    }

    int getReal()          //getters(accessor)
    {
        return this->real;
    }

    int getImag()    //getters(accessor)
    {
        return this->imag;
    }

    void display()
    {
        cout<<"\ncomplex number: "<<this->real<<"+ "<<this->imag<<"i"<<"\n";
    }

    Complex operator++(int a)
    {
        Complex temp;

        temp.real=this->real++;
        temp.imag=this->imag++;

        return temp;
    }

    Complex operator++()
    {
        Complex temp;

        temp.real=++this->real;

```

```

        temp.imag=++this->imag;

        return temp;
    }

    Complex operator--(int a)
    {
        Complex temp;

        temp.real=this->real--;

        temp.imag=this->imag--;

        return temp;
    }

    Complex operator--()
    {
        Complex temp;

        temp.real=--this->real;

        temp.imag=--this->imag;

        return temp;
    }
};

int main()
{
    Complex c1(30,23);

    Complex c2,c3;

    cout<<"\nNumber is: ";

    c1.display();

    cout<<"\nAfter post increamentation\n";

```

```

        c2=c1++;

        c2.display();

        cout<<"\nNumber is ";

        c1.display();

        cout<<"\nAfter pre increamentation\n";

        c2=++c1;

        c2.display();

        cout<<"\nNumber is ";

        c1.display();

        c3=c1--;

        cout<<"\nAfter post decreamentation\n";

        c3.display();

        cout<<"\nNumber is ";

        c3=--c1;

        cout<<"\nAfter pre decreamentation\n";

        c3.display();

    }

```

---

```

//Distance:

using namespace std;

#include<iostream>

struct Distance

{

    int feet,inch;

    Distance()

```

```

{
    cout<<"\n\ndefault constructor called\n";
    this->feet=-1;
    this->inch=-1;
}

Distance(int f,int i)
{
    cout<<"\n\nparameterised constructor called\n";
    this->feet=f;
    this->inch=i;
}

void setFeet(int f)    //setter(mutator)
{
    this->feet=f;
}

void setInch(int i)    //setter(mutator)
{
    this->inch=i;
}

int getFeet()    //getter(accessor)
{
    return this->feet;
}

int getInch()    //getter(accessor)
{

```

```

        return this->inch;
    }

    void display()
    {
        cout<<"\ndistance is: "<<this->feet<<"feet and "<<this->inch<<"inches\n";
    }

```

Distance operator++(int p)

```

{
    Distance temp;

    temp.feet=this->feet++;
    temp.inch=this->inch++;

    return temp;
}

```

Distance operator++()

```

{
    Distance temp;

    temp.feet=++this->feet;
    temp.inch=++this->inch;

    return temp;
}

```

Distance operator--(int p)

```

{
    Distance temp;

    temp.feet=this->feet--;
    temp.inch=this->inch--;
}

```

```

        return temp;
    }

    Distance operator--()
    {
        Distance temp;

        temp.feet=--this->feet;

        temp.inch=--this->inch;

        return temp;
    }
};

int main()
{
    Distance d1,d2,d3;

    int feet,inch;

    d1.display();

    cout<<"\nenter distance in feet:\n";

    cin>>feet;

    cout<<"\nenter distance in inch:\n";

    cin>>inch;

    d1.setFeet(feet);

    d1.setInch(inch);

    cout<<"\nDistance is:";

    d1.display();

    d2=d1++;

    cout<<"\nDistance after post incrementation:";

```

```

    d2.display();

    cout<<"\nDistance is:";

    d1.display();

    cout<<"\nDistance after pre incrementation:";

    d2=++d1;

    d2.display();

    cout<<"\nDistance is:";

    d1.display();

    cout<<"\nDistance after post decrementation:";

    d3=d1--;

    d3.display();

    cout<<"\nDistance is:";

    d1.display();

    d3=--d1;

    cout<<"\nDistance after pre increamentation:";

    d3.display();

    return 0;

}

```

---

Structure using “**new**” keyword (dynamic memory allocation):

1. Student:

```
using namespace std;
```

```
#include<iostream>
```

```
#include<string.h>
```

```
int n;
```



```
struct Student
```

```
{
```

```
    int roll_no;
```

```
    char name[20];
```

```
    Student()
```

```
    {
```

```
        cout<<"\n\ndefault constructor called\n";
```

```
        this->roll_no=0;
```

```
        strcpy(this->name,"not_given");
```

```
    }
```

```
    Student(int r,char* n)
```

```
    {
```

```
        cout<<"\n\nparameterised constructor called\n";
```

```
        this->roll_no=r;
```

```
        strcpy(this->name,n);
```

```
    }
```

```
    void setRoll(int r)           //setters(mumtator)
```

```
    {
```

```
        this->roll_no=r;
```

```
    }
```

```
    void setName(const char* n)
```

```
    {
```

```
        strcpy(this->name,n);    //setters(mumtator)
```

```
    }
```

```
    int getRoll()                //getters(accessors)
```

```

    {
        return this->roll_no;
    }

    char* getName()        //getters(accessors)
    {
        return this->name;
    }

    void display()
    {
        cout<<"\nroll no "<<this->roll_no<<" is "<<this->name<<"\n";
    }
};

int search(Student*,int);

int main()
{
    Student *s;

    int roll,i,ans;

    char name[20];

    cout<<"\nEnter no of students: ";

    cin>>n;

    s=new Student[n];

    for(i=0;i<n;i++)
    {

        s[i].display();

        cout<<"\nEnter roll no of the student: ";
    }
}

```

```

        cin>>roll;

        cout<<"\nEnter name of the student: ";

        cin>>name;

        s[i].setRoll(roll);

        s[i].setName(name);

        s[i].display();

    }

    cout<<"\nEnter roll no to search: ";

    cin>>roll;

    ans=search(s,roll);

    if(ans== -1)

        cout<<"\nroll number not found\n";

    else

        s[ans].display();

    Student *s1;

    s1=new Student(42,"pragati");

    s1->display();

    return 0;

}

int search(Student* s,int r)

{

    int i,count=0;

    for(i=0;i<n;i++)

    {

        if(s[i].getRoll()==r)

```

```

        {
            count++;
            break;
        }
    }
    if(count!=0)
        return i;
    else
        return -1;
}

```

---

## 2. Employee:

```

using namespace std;

#include<iostream>

#include<string.h>

int n;

struct Employee
{
    int emp_id;
    char name[20];
    double salary;
    Employee()
    {
        cout<<"\n\ndefault constructor called\n";
        this->emp_id=0;
        strcpy(this->name,"not_given");
    }
}

```

```

        this->salary=0;
    }
Employee(int i,const char* n,double s)
{
    cout<<"\n\nparameterised called\n";
    this->emp_id=i;
    strcpy(this->name,n);
    this->salary=s;
}
void setId(int i) //setters(mutators)
{
    this->emp_id=i;
}
void setName(const char* n) //setters(mutators)
{
    strcpy(this->name,n);
}
void setSalary(double s)//setters(mutators)
{
    this->salary=s;
}
int getId() //getters(accessors)
{
    return this->emp_id;
}

```

```

char* getName()      //getters(accessors)
{
    return this->name;
}

double getSalary()   //getters(accessors)
{
    return this->salary;
}

void display()
{
    cout<<"\nemployees detail: \nid: "<<this->emp_id<<"\tname: "<<this->name<<"\tsalary: "<<this->salary<<"\n";
}

};

int search(Employee*,int);

int main()
{
    Employee *e;

    int id,i,ans;

    char name[20];

    double salary;

    cout<<"\nEnter no of employees: ";

    cin>>n;

    e=new Employee[n];

    for(i=0;i<n;i++)
    {

```

```

        e[i].display();

        cout<<"\nenter employee id:\n";

        cin>>id;

        cout<<"enter employee name: \n";

        cin>>name;

        cout<<"enter employee salary: \n";

        cin>>salary;

        e[i].setId(id);

        e[i].setName(name);

        e[i].setSalary(salary);

        e[i].display();

    }

    cout<<"\nEnter employee id to search: ";

    cin>>id;

    ans=search(e,id);

    if(ans==-1)

        cout<<"\nEmployee id not found\n";

    else

        cout<<"\nEmployee id found at "<<ans<<" location\n";

    e[ans].display();

    Employee *e1;

    e1=new Employee(42,"pragati",60000);

    e1->display();

    return 0;

}

```

```
int search(Employee *e,int id)
{
    int i,count=0;
    for(i=0;i<n;i++)
    {
        if(e[i].getId()==id)
        {
            count++;
            break;
        }
    }
    if(count!=0)
        return i;
    else
        return -1;
}
```

---

### 3. Sales Manager:

```
using namespace std;
```

```
#include<iostream>
```

```
#include<string.h>
```

```
int n;
```

```
struct SalesMan
```

```
{
```

```
    int id,target;
```

```
    char name[20];
```



```

double salary,intensive;

SalesMan()
{
    cout<<"\n\ndefault constructor called\n";

    this->id=0;

    strcpy(this->name,"not_given");

    this->salary=0;

    this->target=0;

    this->intensive=0;
}

SalesMan(int i,const char* n,double s,int t,int in)
{
    printf("\n\nparameterised constructor called\n");

    this->id=i;

    strcpy(this->name,n);

    this->salary=s;

    this->target=t;

    this->intensive=in;
}

void setId(int i) //setters(mutator)
{
    this->id=i;
}

void setName(const char* n) //setters(mutator)
{

```

```

        strcpy(this->name,n);
    }

    void setSalary(double s)        //setters(mutator)
    {
        this->salary=s;
    }

    void setTarget(int t)          //setters(mutator)
    {
        this->target=t;
    }

    void setIntense(double in)      //setters(mutator)
    {
        this->intensive=in;
    }

    int getId()                    //getters(accessor)
    {
        return this->id;
    }

    char* getName()                //getters(accessor)
    {
        return this->name;
    }

    double getSalary()             //getters(accessor)
    {
        return this->salary;
    }

```

```

    }

    int getTarget()          //getters(accessor)
    {
        return this->target;
    }

    double getIntense()      //getters(accessor)
    {
        return this->intensive;
    }

    void display()
    {
        cout<<"\nsales managers details:\nid: "<<this->id<<"\tname: "<<this->name<<"\tsalary:
"<<this->salary<<"\ttarget: "<<this->target<<"\tintensive: "<<this->intensive;

    }
};

int search(SalesMan*,int);

int main()
{
    SalesMan *m;

    int id,target,i,ans;

    char name[20];

    double salary,intensive;

    cout<<"\nEnter no of sales manager: ";

    cin>>n;

    m=new SalesMan[n];

    for(i=0;i<n;i++)

```

```

{
    m[i].display();

    cout<<"enter sale managers id:\n";

    cin>>id;

    cout<<"\nenter the name of sales manager:\n";

    cin>>name;

    cout<<"\nenter salary of sales manager:\n";

    cin>>salary;

    cout<<"\nenter target of sales manager:\n";

    cin>>target;

    cout<<"\nenter intensive for target completion:\n";

    cin>>intensive;

    m[i].setId(id);

    m[i].setName(name);

    m[i].setSalary(salary);

    m[i].setTarget(target);

    m[i].setIntense(intensive);

    m[i].display();

}

cout<<"\nEnter sales mans id to search: ";

cin>>id;

ans=search(m,id);

if(ans!=-1)

{

    cout<<"\nid found at "<<ans<<" location\n";

```

```
        m[ans].display();
    }
    else
        cout<<"\nId not found\n";

    SalesMan *m1;

    m1=new SalesMan(42,"pragati",60000,40,5000);

    m1->display();

    return 0;
}

int search(SalesMan* m,int id)
{
    int i,count=0;

    for(i=0;i<n;i++)
    {
        if(m[i].getId()==id)
        {
            count++;

            break;
        }
    }

    if(count!=0)

        return i;

    else

        return -1;
}
```

---

4. Admin:

```
using namespace std;
```

```
#include<iostream>
```

```
#include<string.h>
```

```
int n;
```

```
struct Admin
```

```
{
```

```
    int id;
```

```
    char name[20];
```

```
    double salary,allowance;
```

```
    Admin()
```

```
    {
```

```
        cout<<"\n\ndefault constructor called\n";
```

```
        this->id=0;
```

```
        strcpy(this->name,"not_given");
```

```
        this->salary=0;
```

```
        this->allowance=0;
```

```
    }
```

```
    Admin(int i,const char* n,double s,double a)
```

```
    {
```

```
        cout<<"\n\nparameterised constructor called\n";
```

```
        this->id=i;
```

```
        strcpy(this->name,n);
```

```
        this->salary=s;
```

```
        this->allowance=a;
```

```

}

void setId(int i) //setters(mutator)

{
    this->id=i;
}

void setName(const char* n)          //setters(mutator)

{
    strcpy(this->name,n);
}

void setSalary(double s)//setters(mutator)

{
    this->salary=s;
}

void setAllow(double a)          //setters(mutator)

{
    this->allowance=a;
}

int getId()          //getters(accessor)

{
    return this->id;
}

char* getName()          //getters(accessor)

{
    return this->name;
}

```

```

double getSalary()          //getters(accessor)
{
    return this->salary;
}

double getAllow()          //getters(accessor)
{
    return this->allowance;
}

void display()
{
    cout<<"\nadmins details:\nid: "<<this->id<<"\tname: "<<this->name<<"\tsalary: "<<this->salary<<"\tallowance: "<<this->allowance<<"\n";
}

};

int search(Admin*,int);

int main()
{
    Admin *a;

    int id,i,ans;

    char name[20];

    double salary,allowance;

    cout<<"\nEnter no of admin: ";

    cin>>n;

    for(i=0;i<n;i++)
    {
        a[i].display();
    }
}

```



```

        cout<<"enter admin id:\n";

        cin>>id;

        cout<<"\nenter name of the admin:\n";

        cin>>name;

        cout<<"\nenter salary of admin:\n";

        cin>>salary;

        cout<<"\nallowance for admin:\n";

        cin>>allowance;

        a[i].setId(id);

        a[i].setName(name);

        a[i].setSalary(salary);

        a[i].setAllow(allowance);

        cout<<"\nafter setting values\n";

        a[i].display();

    }

    cout<<"\nEnter admin id to search: ";

    cin>>id;

    ans=search(a,id);

    if(ans!=-1)

    {

        cout<<"\nAdmin id found at "<<ans<<" location\n";

        a[ans].display();

    }

    else

        cout<<"\nid not found\n";

```

```

        Admin *a1;

        a1=new Admin(42,"pragati",60000,5000);

        a1->display();

        return 0;

    }

int search(Admin* a,int id)
{
    int i,count=0;

    for(i=0;i<n;i++)
    {
        if(a[i].getId()==id)
        {
            count++;

            break;

        }
    }

    if(count!=0)

        return i;

    else

        return -1;

}

```

---

## 5. HR Manager:

```

using namespace std;

#include<iostream>

#include<string.h>

```

```
int n;

struct HrManager
{
    int id;

    char name[20];

    double salary,commission;

    HrManager()
    {
        cout<<"\n\ndefault constructor called\n";

        this->id=0;

        strcpy(this->name,"not_given");

        this->salary=0;

        this->commission=0;
    }

    HrManager(int i,const char* n,double s,double c)
    {
        cout<<"\n\nparameterised constructor called\n";

        this->id=i;

        strcpy(this->name,n);

        this->salary=s;

        this->commission=c;
    }

    void setId(int i) //setters(mutator)
    {
        this->id=i;
```

```
}

void setName(const char* n)          //setters(mutator)

{
    strcpy(this->name,n);
}

void setSalary(double s)//setters(mutator)

{
    this->salary=s;
}

void setComm(double c)              //setters(mutator)

{
    this->commission=c;
}

int getId()                        //getters(accessor)

{
    return this->id;
}

char* getName()                    //getters(accessor)

{
    return this->name;
}

double getSalary()                 //getters(accessor)

{
    return this->salary;
}
```

```

double getComm()    //getters(accessor)
{
    return this->commission;
}

void display()
{
    cout<<"\nHR Managers detail: \nid: "<<this->id<<"\tName: "<<this->name<<"\tSalary:
"<<this->salary<<"\tCommission: "<<this->commission<<"\n";
}
};

int search(HrManager*,int);

int main()
{
    HrManager *h;

    int id,i,ans;

    char name[20];

    double salary,commission;

    cout<<"\nEnter no of HR manager ";

    cin>>n;

    h=new HrManager[n];

    for(i=0;i<n;i++)
    {
        h[i].display();

        cout<<"\nEnter hr managers id:\n";

        cin>>id;

        cout<<"\nEnter name of hr manager:\n";

```

```

        cin>>name;

        cout<<"\nenter salary of hr manager:\n";

        cin>>salary;

        cout<<"\nenter commission for hr manager:\n";

        cin>>commission;

        h[i].setId(id);

        h[i].setName(name);

        h[i].setSalary(salary);

        h[i].setComm(commission);

        h[i].display();

    }

    cout<<"\nEnter HR managers id to search: ";

    cin>>id;

    ans=search(h,id);

    if(ans!=-1)

    {

        cout<<"\nid found at "<<ans<<" location\n";

        h[ans].display();

    }

    else

        cout<<"\nid not found\n";

    HrManager *h1;

    h1=new HrManager(42,"pragati",60000,5000);

    h1->display();

    return 0;

```

```

}

int search(HrManager* h,int id)
{
    int i,count=0;
    for(i=0;i<n;i++)
    {
        if(h[i].getId()==id)
        {
            count++;
            break;
        }
    }
    if(count!=0)
        return i;
    else
        return -1;
}

```

---

6. Date:

```

using namespace std;

#include<iostream>

int n;

struct Date
{
    int day,month,year;

    Date()

```

```

{
    cout<<"\n\ndefault constructor called\n";
    this->day=0;
    this->month=0;
    this->year=0;
}

Date(int d,int m,int y)
{
    cout<<"\n\nparameterised constructor called\n";
    this->day=d;
    this->month=m;
    this->year=y;
}

void setDay(int d)    //setter(mutator)
{
    this->day=d;
}

void setMonth(int m)  //setter(mutator)
{
    this->month=m;
}

void setYear(int y)    //setter(mutator)
{
    this->year=y;
}

```



```

int getDay()    //getters(accessor)
{
    return this->day;
}

int getMonth()    //getters(accessor)
{
    return this->month;
}

int getYear()    //getters(accessor)
{
    return this->year;
}

void display()
{
    cout<<"\n\ndate is: \n"<<this->day<<"/"<<this->month<<"/"<<this->year<<"\n";
}

};

int search(Date*,Date);

int main()
{
    Date *d;

    Date dt;

    int day,month,year;

    int i,ans;

    cout<<"\nEnter no of dates to store: ";

```

```

cin>>n;

for(i=0;i<n;i++)
{
    d[i].display();

    cout<<"\nenter date: ";

    cin>>day;

    cout<<"\nenter month: ";

    cin>>month;

    cout<<"\nenter year: ";

    cin>>year;

    d[i].setDay(day);

    d[i].setMonth(month);

    d[i].setYear(year);

    d[i].display();
}

cout<<"\nEnter date to search(dd/mm/yy): ";

cin>>day>>month>>year;

dt.setDay(day);

dt.setMonth(month);

dt.setYear(year);

ans=search(d,dt);

if(ans!=-1)
{
    cout<<"\nDate found at "<<ans<<" location\n";

    d[ans].display();
}

```

```

    }

    else

    cout<<"\nDate not found\n";

    Date *d1;

    d1=new Date(23,4,2001);

    d1->display();

    return 0;

}

int search(Date* d,Date dt)
{
    int i,count=0;

    for(i=0;i<n;i++)
    {

        if(d[i].getDay()==dt.getDay()&& d[i].getMonth()==dt.getMonth()&& d[i].getYear()==dt.getYear())
        {

            count++;

            break;

        }

    }

    if(count!=0)

    return i;

    else

    return -1;

}

```

---

7. Time:

```

using namespace std;

#include<iostream>

int n;

struct Time
{
    int hr,min,sec;

    Time()
    {
        cout<<"\n\ndefault constructor called\n";

        this->hr=-1;

        this->min=-1;

        this->sec=-1;
    }

    Time(int h,int m,int s)
    {
        cout<<"\n\nparameterised constructor called\n";

        this->hr=h;

        this->min=m;

        this->sec=s;
    }

    void setHour(int h)           //setter(mutator)
    {
        this->hr=h;
    }

    void setMin(int m)           //setter(mutator)

```

```

{
    this->min=m;
}

void setSec(int s)          //setter(mutator)
{
    this->sec=s;
}

int getHr()                //getter(accessor)
{
    return this->hr;
}

int getMin()              //getter(accessor)
{
    return this->min;
}

int getSec()              //getter(accessor)
{
    return this->sec;
}

void display()
{
    cout<<"\nTime is: "<<this->hr<<":"<<this->min<<":"<<this->sec;
}

};

int search(Time*,Time);

```

```
int main()
{
    Time *t;
    Time tm;
    int hr,min,sec;
    int r,q,i,ans;

    cout<<"\nEnter no of time slot to store: ";

    cin>>n;

    t=new Time[n];
    for(i=0;i<n;i++)
    {
        t[i].display();

        cout<<"\nEnter hours:\n";

        cin>>hr;

        cout<<"\nEnter minuits:\n";

        cin>>min;

        cout<<"\nEnter seconds:\n";

        cin>>sec;

        t[i].setSec(sec);

        t[i].setMin(min);

        t[i].setHour(hr);

        t[i].display();

    }

    cout<<"\nEnter time slot to search(hr/min/sec): ";

    cin>>hr>>min>>sec;
```

```

        tm.setHour(hr);

        tm.setMin(min);

        tm.setSec(sec);

        ans=search(t,tm);

        if(ans!=-1)

        {

                cout<<"\nTime slot found at "<<ans<<" location\n";

                t[ans].display();

        }

        Time *t1;

        t1=new Time(10,49,55);

        t1->display();

        return 0;

}

int search(Time* t,Time tm)

{

        int i,count=0;

        for(i=0;i<n;i++)

        {

                if(t[i].getHr()==tm.getHr()&&t[i].getMin()==tm.getMin()&&t[i].getSec()==tm.getSec())

                {

                        count++;

                        break;

                }

        }

}

```

```
        if(count!=0)

            return i;

        else

            return -1;

    }
```

---

#### 8. Distance:

```
using namespace std;

#include<iostream>

int n;

struct Distance

{

    int feet,inch;

    Distance()

    {

        cout<<"\n\ndefault constructor called\n";

        this->feet=-1;

        this->inch=-1;

    }

    Distance(int f,int i)

    {

        cout<<"\n\nparameterised constructor called\n";

        this->feet=f;

        this->inch=i;

    }

    void setFeet(int f)    //setter(mutator)
```



```

    {
        this->feet=f;
    }

    void setInch(int i)          //setter(mutator)
    {
        this->inch=i;
    }

    int getFeet()    //getter(accessor)
    {
        return this->feet;
    }

    int getInch()    //getter(accessor)
    {
        return this->inch;
    }

    void display()
    {
        cout<<"\ndistance is: "<<this->feet<<"feet and "<<this->inch<<"inches\n";
    }

};

int search(Distance*,Distance);

int main()
{
    Distance *d;

    Distance ds;

```

```

int feet,inch;

int i,ans;

cout<<"\nEnter no of distance to store: ";

cin>>n;

d=new Distance[n];

for(i=0;i<n;i++)
{
    d[i].display();

    cout<<"\nEnter distance in feet:\n";

    cin>>feet;

    cout<<"\nEnter distance in inch:\n";

    cin>>inch;

    d[i].setFeet(feet);

    d[i].setInch(inch);

    d[i].display();

}

cout<<"\nEnter distance to search (in feet and inch): ";

cin>>feet>>inch;

ds.setFeet(feet);

ds.setInch(inch);

ans=search(d,ds);

if(ans!=-1)
{
    cout<<"\nDistance found at "<<ans<<" location\n";

    d[ans].display();
}

```

```

    }

    Distance *d1;

    d1=new Distance(5,2);

    d1->display();

    return 0;
}

int search(Distance* d,Distance ds)
{
    int i,count=0;

    for(i=0;i<n;i++)
    {
        if(d[i].getFeet()==ds.getFeet()&& d[i].getInch()==ds.getInch())
        {
            count++;

            break;
        }
    }

    if(count!=0)

    return i;

    else

    return 0;
}

```

---

## 9. Complex:

```
using namespace std;
```

```
#include<iostream>
```

```

int n;

struct Complex
{
    int real,imag;

    Complex()
    {
        //      cout<<"\n\ndefault constructor called\n";

        this->real=0;

        this->imag=0;
    }

    Complex(int r,int i)
    {
        //      cout<<"\n\nparameterised constructor called\n";

        this->real=r;

        this->imag=i;
    }

    void setReal(int r)           //setters(mutator)
    {
        this->real=r;
    }

    void setImg(int i)           //setters(mutator)
    {
        this->imag=i;
    }

    int getReal()                //getters(accessor)

```

```

    {
        return this->real;
    }

    int getImag()    //getters(accessor)
    {
        return this->imag;
    }

    void display()
    {
        cout<<"\ncomplex number: %d+%di\n",this->real,this->imag;
    }
};

int search(Complex*,Complex);

int main()
{
    Complex *c;

    Complex cm;

    int real,imag;

    int i,ans;

    c=new Complex[n];

    for(i=0;i<n;i++)
    {
        c[i].display();

        cout<<"\nenter real part of complex number:\n";

        cin>>real;
    }
}

```

```

        cout<<"\nenter imaginary part of complex number:\n";

        cin>>imag;

        c[i].setReal(real);

        c[i].setImg(imag);

        c[i].display();

    }

    cout<<"\nEnter complex number to search(real and imaginary part): ";

    cin>>real>>imag;

    cm.setReal(real);

    cm.setImg(imag);

    ans=search(c,cm);

    if(ans!=-1)

    {

        cout<<"\nComplex number found at "<<ans<<" location\n";

        c[ans].display();

    }

    Complex *c1;

    c1=new Complex(30,23);

    c1->display();

    return 0;

}

int search(Complex* c,Complex cm)

{

    int i,count=0;

    for(i=0;i<n;i++)

```

```
{  
    if(c[i].getReal()==cm.getReal()&& c[i].getImag()==cm.getImag())  
    {  
        count++;  
        break;  
    }  
}  
if(count!=0)  
    return i;  
else  
    return -1;  
}
```

---

---

Short notes:

## # One liner macro:

└ macro is a adjustment which works like find & replace, by the value of macro.

└ Macro is defined by '#define' directive

└ Whenever macro name is encountered by the compiler, it replaces the name with definition (value of macro) of the macro.

eg. #include <stdio.h>

#define ~~DATA~~ LIMIT 5

void main()

{ printf("The value of limit is: %d", LIMIT);

}

## # Dangling pointers ~~error~~:

└ Most common bugs related to pointers & memory management is dangling/wild pointers. ~~Some~~

└ Sometimes programmer fails to initialize the pointer with valid address, then this type of initialized pointer is known as dangling pointer.

└ Dangling pointer occurs at the time of object destruction, when object is deleted or de-allocated.

└ When the variable goes out of the scope (out of the limitation of stack frame) then the pointer pointing to the variable becomes dangling pointer.

└ Even when we call ~~from~~ the function, pointer can become dangling pointer. How? - If we are storing some data in local variable of a function, & returning it's address to other function using pointer, But after returning the address, the stack frame of that function



vanished & segment when we try to access the data of that location that becomes unauthorised access. And segmentation fault ~~at~~ occurs, & pointer returned pointer becomes dangling pointer.