



Destructor

The class Destructor:

A destructor is a special member function which is also having the same name as that of the class but prefixed with the tilde(~) symbol.

For example:

if **sample** is the name of the class then

~ sample()

is the destructor.



Characteristics of a destructor

- A destructor is invoked automatically by the compiler upon exit from the program and cleans up the memory that is no longer needed.
- A destructor does not return any value.
- A destructor does not accept arguments and therefore it cannot be overloaded.
- A destructor must be declared in **public** section of a class.

This program illustrates how an object is init and dest

```
class Sample {  
    private : int x;  
    public : Sample () {  
        x=0;  
        cout << "In constructor," << "x =" << x << endl;  
    }  
    void printx (void){  
        x= 25;  
        Cout << "In printx, x= " <<x<<endl;  
    }  
    ~ Sample ( ){  
        cout << "In destructor. Object dies" <<endl;  
    }  
};  
void main( ){  
    Sample s1;  
    s1. Printx( );  
}
```

What is the output ?
In constructor x=0
In printx x=25
In destructor. Object dies

Order of invocation of cons and destructors

```
class Sample {  
    private : int id;  
    public :Sample (int i) {  
        id=i;  
        cout << "constructor,object" << id << endl;  
    }  
    ~Sample ( ) {  
        cout << "object destroyed"<<id <<endl;  
    }  
};  
  
int main( )  
{  
    Sample s1(1),s2(2),s3(3);  
    return 0;  
}
```

Objects will be destroyed in Reverse order

What is the output?

In constructor,object1
In constructor,object2
In constructor,object3

object destroyed3
object destroyed2
object destroyed1

Array of Objects

We can define an array of any data type. It may be used to hold the data of type int, float, char etc..

It is possible to define an array struct.

Like array of structures, we can define an array of class objects.

Consider the following class definition.

```
class student
{
    private : int rollno;
             char name[20];
             float percentage;
    public : void getData (void);
            void printData (void);
}; // End of class student
```



Creating array of objects

```
int a[50];
```

Suppose we want to process the information of student objects

having this class type, then

```
student stud[50]
```

This creates an object called **stud** as an array of 50 students.

The array **stud** can be represented as

student stud[50]

Stud [0]

Rollno

Name

Percentage

Stud [1]

Rollno

Name

Percentage

.....

.....

Stud [49]

Rollno

Name

percentage

Accessing individual array objects

Here, `stud[0]` is the first student and `stud[49]` is the last

To access i^{th} object of array **`Stud[i]`**

If `getData()` and `PrintData` are functions then

```
stud[i].getData( );  
stud[i].printData( );
```

To iterate through array of objects

```
for(i=0;i<n;i++)  
{  
    s[i].printData()  
}
```




Array of objects.

Write a program to illustrate array of objects.

```
class Student
{
    Regno, name;
public:
    void ReadStudentDetails();
    void printStudentDetails;
};
```



```
void main()
```

```
{
```

```
    student stud[50];
```

```
    int n,i;
```

```
    cout<<"Enter number of students";
```

```
    cin>>n;
```

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

```
    {
```

```
        cout<<"Enter the info of student";
```

```
        stud[i]. ReadStudentDetails();
```

```
    }
```

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

```
    {
```

```
        stud[i]. printStudentDetails();
```

```
    }
```

```
}
```

Passing objects to functions

Add(x,y); //variables

Add(10,20); //constants

Add(x+10,y+4); //expressions

They can also be objects; means, function arguments can be objects.

Add(c1,c2);

Where c1 & c2 are objects of type complex

Add(t1,t2);

Where t1 & t2 are objects.

Write a program to show passing objects as arguments for adding complex numbers

```
class complex    // x+iy form
{
    private:    float x , y;
    public:
        void read() {
            cout<<"enter real and imag part";
            cin>>x>>y;
        }
        void print() {
            cout<<x<<"+i"<<y<<endl;
        }
        void add(complex c1,complex c2);
};
```

```
void complex::add(complex c1,complex c2)
```

```
{
```

```
    x=c1.x+c2.x;    //not c3.x
```

```
    y=c1.y+c2.y;    //not c3.y
```

```
}
```

```
int main()
```

```
{
```

```
    complex c1,c2,c3;
```

```
    c1.read();
```

```
    c2.read();
```

```
    c3.add(c1,c2);
```

```
    c3.print();
```

```
    return o;
```

```
}
```

C1 & c2 as argument to add()

When u say c1.read(), in which obj it is reading?

```
void read() {
```

```
    cout<<"enter real and imag part";
```

```
    cin>>x>>y;
```

```
}
```

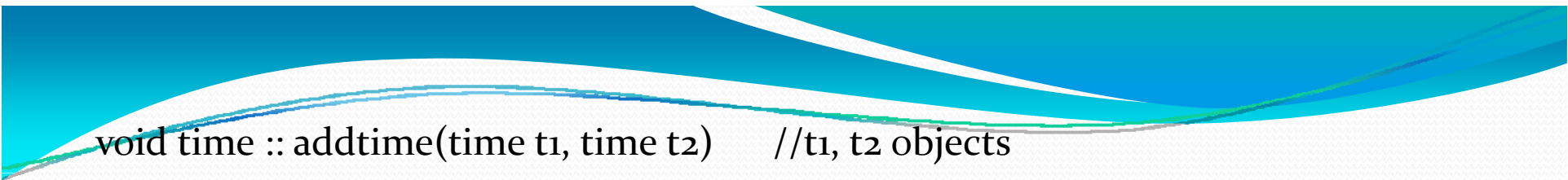
When u say c2.read(), in which obj it is reading?

When u say c3.add(c1,c2), in which obj it is adding?

Write a program to create a class time(hours,minutes),
include gettime() function to read time,
puttime() function to print time,
addtime() to add two time objects.

```
class time
{
    int hours;
    int minutes;
public:
    void gettime() {

        cin>>hours;
        cin>>minutes;
    }
    void puttime() {
        cout<<hours<<":";
        cout<<minutes<<endl;
    }
    void addtime(time, time); //declaration with objects as arguments
};
```



```
void time :: addtime(time t1, time t2)    //t1, t2 objects
```

```
{
```

```
    minutes = t1.minutes+t2.minutes;
```

```
    hours = minutes/60;
```

```
    minutes = minutes%60;
```

```
    hours = hours + t1.hours + t2.hours;
```

```
}
```

```
int main()
```

```
{
```

```
    time T1, T2, T3;
```

```
    T1.gettime(); //get T1
```

```
    T2.gettime(); //get T2
```

```
    T3.addtime(T1,T2);    //T3=T1+T2
```


```
    cout << "T1 = "; T1.puttime(); //display T1
```

```
    cout << "T2 = "; T2.puttime(); //display T2
```

```
    cout << "T3 = "; T3.puttime(); //display T3
```

```
    return 0;
```

```
}
```

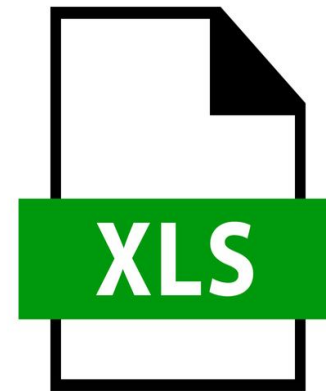



WAP to create a class student details with
id,name,p,c,m,b marks as members
Include functions to read ,print and findavg marks

Reading excel (CSV)files

```
FILE *fp=fopen("data.csv","r");
char line[500]; int i=0;
while(fgets(line,200,fp))
{
    sscanf(line,"%[^,],%[^,]",s[i].srn,s[i].name);
    i++;
}
```

```
while (!inFile.eof())
{
    getline ( fromFile, srn, ',' );
    getline ( fromFile, name, ',' );
}
```





```
class student{
    public:
        char srn[15],name[50];
        void disp();
};
student s[100];
void student::disp(){
    cout<<srn<<"\t"<<name<<endl;
}
int main()
{
    FILE *fp=fopen("data.csv","r");
    char line[500]; int i=0;
    while(fgets(line,200,fp)) {
        sscanf(line,"%[^,],%[^,]",s[i].srn,s[i].name);
        i++;
    }
    for(int k=0;k<i;k++)
        s[k].disp();
}
```



Access-specifiers

There are three member Access-specifiers , they are

Private : within the class(only members of that class/only by family members)

Public: within the class and outside the class(members & non members)

Protected: within the class and derived class(members and relatives)

So

Private members are not accessible to non members

Friends in C++



Class customer

{

void dp(),wd()
friend balEnq();

}

Friends in C++:

Definition : A function declared with the keyword friend & it is a non-member function still which has full access rights to the private members of the class.

The private members of a class cannot be accessed by the non-member functions. But there may be some situations where it is needed to access the private members of a class by the non-member functions. This can be achieved by changing the access specifier of private members to public. The attempt to change the access specifier of the private members to public, violates the underlying concept of data hiding and encapsulation. So, we can overcome this problem by declaring the non-member functions as friend functions(to the class). It is just like your friend(s) using your books, iron box, TV or computer, though they are solely used by you.

Remember the following points while using friend function.

- The keyword **friend** is used to declare friend functions.
- It is not in the scope of the class in which it has been declared So a friend function is called just like a normal function.
- A friend function can be declared either in the **private** or **public** section.



Remember the following points while using **friend function**.

- Usually, a friend function has the object as its arguments.
- A friend function cannot access the class members directly but using object (object.membername)
- The friend function definition does not use either the keyword friend or the scope operator.



Example:

Class customer

{

void dp(),wd()

friend balEnq();

}

Friend function

```
#include<iostream>

using namespace std;

class Customer
{
    int accno,bal;
    string name;
public:
    void Deposit();
    void Withdraw();
    void BalEnq();
    Customer(int no,string n, int b)
    {
        accno=no;
        name=n;
        bal=b;
    }
};

void Customer::Deposit()
{
    int amt;
    cout<<"Enter the amount to be deposited";
    cin>>amt;
    bal=bal+amt;
    cout<<"Present Bal is "<<bal<<endl;
}

void Customer::Withdraw()
{
    int amt;
    cout<<"Enter the amount to be withdrawn";
    cin>>amt;
    bal=bal-amt;
    cout<<"Present Bal is "<<bal<<endl;
}

void Customer::BalEnq()
{
    cout<<"Present Bal is "<<bal<<endl;
}

int main()
{
    Customer c(123,"Kumar",5000);
    c.Deposit();
    c.Withdraw();
    c.BalEnq();
    return 0;
}
```
