Constructor and Destructor

Introduction

```
#include<iostream>
using namespace std;
class test
{
int x;
public:
void set(){
x=10;}
void display(){
cout<<x;
}
};
```

```
int main(){
test ob;
ob.display();
}
```

OUTPUT

10

Properties

- Declared as public
- Automatically invoked
- No return types
- Cannot be inherited
- Can have default arguments
- Cannot be declared as virtual

Example

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

class test{
int x;
public:
test(){
x=10;
}
void display(){
cout<<x;
}
};
```

```
int main(){
test ob;
ob.display();
}
```

OUTPUT

```
10
```

Parameterized Constructor

```
#include<iostream>
using namespace std;
class test
{
int x;
public:
test(int a){
x=a;
}
void display(){
cout<<x;
}};
```

```
int main(){
test ob(2);
ob.display();
}
```

Implicit Call
OUTPUT

2

Parameterized Constructor

```
#include<iostream>
using namespace std;
class test
{
int x;
public:
test(int a){
x=a;
}
void display(){
cout<<x;
}};
```

```
int main(){
test ob=test(2);
ob.display();
}
```

Explicit Call
OUTPUt

2

Multiple Constructors

```
#include<iostream>
using namespace std;
class test
{
public:
test()
{
cout<<"Hi";
}
test(int a){
cout<<a;
}
```

```
test(int a,int b)
{
cout<<a<<"\t"<<b;
}
};

int main(){
test ob= test(2);
test ob1=test(2,3);
test ob3=test();
}
```

OUTPUT

```
2
2    3
Hi
```

Multiple Constructors

```
#include<iostream>
using namespace std;
class test
{
public:
test(int a){
cout<<a;
}
test(int a,int b){
cout<<a<<"\t"<<b;
}};
```

```
int main(){
test ob= test(2);
test ob1=test(2,3);
test ob3=test();
}
```

OUTPUT

Error

Multiple Constructors

```
#include<iostream>
using namespace std;
class test{
public:
test(){
cout<<"Hi";
}
test(int a){
cout<<a;
}
test(int a,int b){
cout<<a<<"\t"<<b
}
```

```
test(float x, float y){
cout<<x<<"\t"<<y;
```

OUTPUT

```
test.cpp:26:22: error: call of overloaded 'test(double, double)' is ambiguous
test ob2=test(2.3,3.4);                                          ^
test.cpp:17:1: note: candidate: test::test(float, float)
test(float x,float y)                                   ^~~~

test.cpp:13:1: note: candidate: test::test(int, int)
test(int a,int b)
```

Example

```
#include<iostream>
using namespace std;
class test
{
public:
test(int a,int b=4){
cout<<a<<"\t"<<b;
}
};
```

```
int main(){
test ob= test(2);
}
```

OUTPUT

```
2   4
```

Example

```
#include<iostream>
using namespace std;
class test
{
public:
test(int a,int b=4, int c=10){
cout<<a<<"\t"<<b<<"\t"<<c;
}
};
```

```
int main(){
test ob= test(2);
}
```

OUTPUT

```
2   4   10
```

Example

```
#include<iostream>
using namespace std;
class test
{
public:
test(int b=4, int c=10,int a){
cout<<a<<"\t"<<b<<"\t"<<c;
}
};
```

```
int main(){
test ob= test(2);
}
```

OUTPUT

Error

Example

```
#include<iostream>
using namespace std;
class test{
int x;
public:
test(int a){
x=a;
cout<<x;
}
test(test & ob){
x=ob.x;
cout<<x;
}};
```

```
int main(){
test ob1(2);
test ob2(ob1);
}
```

OUTPUT

```
2     2
```

Example

```
#include<iostream>
using namespace std;
class test{
int x;
public:
test(int a){
x=a;
}
test(test & ob){
x=ob.x;
}
void display(){
cout<<x;
}};
```

```
int main(){
test ob1(2);
ob1.display();
test ob2(ob1);
ob2.display();
}
```

OUTPUT

```
2        2
```

Example

```
#include<iostream>
using namespace std;
class test{
int x;
public:
test(int a){
x=a;
}
void display(){
cout<<x;
}};
```

```
int main(){
test ob1(2);
ob1.display();
test ob2(ob1);
ob2.display();
}
```

OUTPUT

```
2        2
```

Example

```
#include<iostream>
using namespace std;
class test{
int x;
public:
test(int a){
x=a;
}
void display(){
cout<<x;
}};
```

```
int main(){
int x;
cin>>x;
test ob1(x);
ob1.display();
test ob2(ob1);
ob2.display();
}
```

OUTPUT

```
       23
23           23
```

Example

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

int nob=0;

class test{
int x;
public:
test(int a){
x=a;
nob++;
cout<<"\n"<<nob;
}
```

```
~test(){
nob--;
cout<<"\n"<<nob;
}};

int main(){
test ob1(5);
{
test ob2(10);
}
test ob3(20);
}
```

OUTPUT

```
1
2
1
2
1
0
```

```
#include<iostream>
using namespace std;
class test{
static int i;
public:    void getdata()
{       cout<<i;    }
void setdata()
{       i++;    }
};
```

```
int test::i;

int main(){
test ob,ob1;
ob.setdata();
ob1.setdata();
ob.getdata();
ob1.getdata();
}
```

```
OUTPUT
2         2
```

Static Data Member

- A static member variable is initialized to 0 when the first object is created.
  - No other initialization is permitted.
- Only one copy of the data member is created and is shared by all the objects.
- It is visible only within the class, but its lifetime is the entire program.
- Type and scope of each static member variable must be defined outside the class definition as the static data members are stored separately rather than as a part of an object.
- Also referred to as class variables.

Code

```
#include<iostream>
using namespace std;
class test{
static int i;
public:
void getdata()    {        cout<<i;    }
void setdata()    {        i++;    }
static void disp()    {
cout<<"Display invoked";    }
};
```

```
int test::i;
int main(){
test::disp();
test ob,ob1;
ob.setdata();
ob1.setdata();
ob.getdata();
 ob1.getdata();
}
```

OUTPUT

```
Display invoked
2            2
```

Code

```
#include<iostream>
using namespace std;
class test{
static int i;
public:
void getdata()    {        cout<<i;    }    void
setdata()    {        i++;    }
static void disp()    {        cout<<"Display
invoked";        setdata();        getdata();
}
};
```

```
int test::i;

int main()
{
test::disp();
}
```

OUTPUT

Error

Code

```
#include<iostream>
using namespace std;
class test{
static int i;
public:
void getdata()    {        cout<<i;    }    void
setdata()    {        i++;    }
static void disp()    {
test ob;
cout<<"Display invoked";
ob.setdata();        ob.getdata();
}};
```

```
int test::i;

int main()
{
test::disp();
}
```

```
OUTPUT
Display invoked 1
```

Static Member Functions

- A static member function can have access to only other static member function or variables declared in the same class.
- A static member function can be called using the class name instead of the object name.