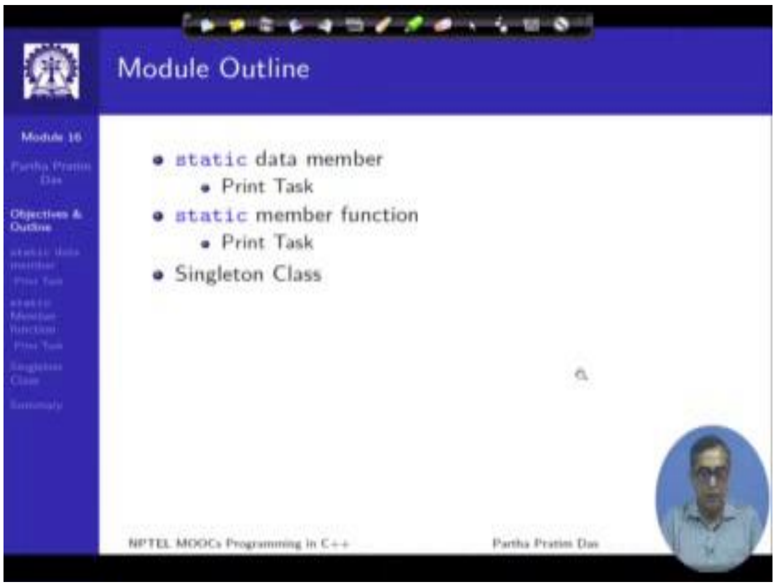
**Programming in C++**
**Prof. Partha Pratim Das**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Kharagpur**

**Lecture - 31**
**Static Members**

Welcome to Module 16 of Programming in C++. In this module, we will talk about static members; specifically we will try to understand the static data members and static member function.

(Refer Slide Time: 00:38)

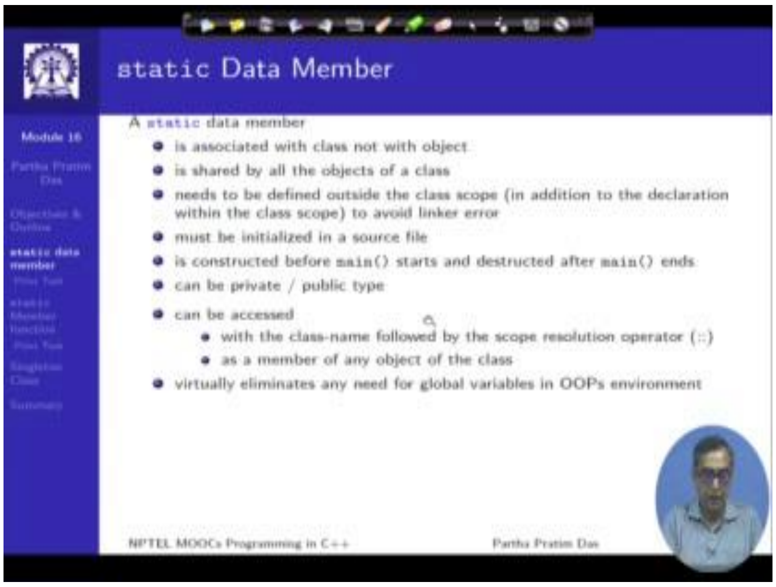


## Module Outline

- static data member
  - Print Task
- static member function
  - Print Task
- Singleton Class

The outline will comprise these two kinds of members, and we will have a discussion about singleton classes. The outline will be visible on the left of your every slide.

(Refer Slide Time: 00:54)



## static Data Member

A static data member

- is associated with class not with object
- is shared by all the objects of a class
- needs to be defined outside the class scope (in addition to the declaration within the class scope) to avoid linker error
- must be initialized in a source file
- is constructed before main() starts and destructed after main() ends
- can be private / public type
- can be accessed
  - with the class-name followed by the scope resolution operator (::)
  - as a member of any object of the class
- virtually eliminates any need for global variables in OOPs environment

NPTEL MOOCs Programming in C++ Partha Pratim Das

A static data member, we have learnt about data members. So, a data member is a component of a class structure. So, whenever a class is instantiated in terms of an object, the object will have the data members, and those data members we know can be accessed from the pointer of the object or from within the member functions of the object.

In terms of the static data member, the things are very different. Static data members is are not associated with the object; like the normal data members are associated with the object, but the static data members are in contrast are associated with the class of which the array member. So, the critical property of static data member is that no matter how many instances of a class is created there will be only one instance of the static data member of a class. So, it is shared by all objects of the class.

In fact, the interesting fact that we will see over the different examples is that a static data member may exist, will exist even when no object has been created for the class. So, the purpose of the static member is primarily to maintain information to maintain data that is specific to the class, but not specific to each and every instance.

And we will see there are several programming situations, there are several problems for which having such a static data member would really help. For those of few who have some familiarity with java, the static data member is a class variable in java, in contrast to the instance variable which is the ordinary non-static data member in C++.

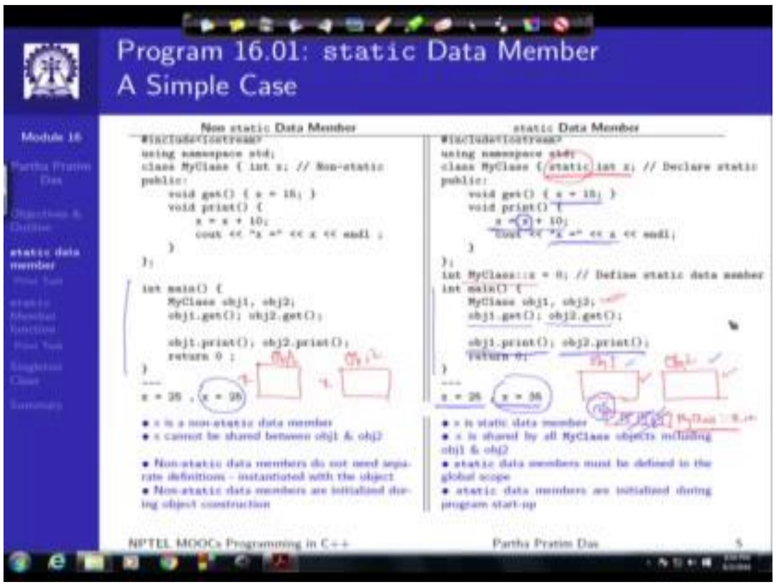
For the static data member, we will see some additional constraints; we will see that the static data member is declared within the class like the ordinary data members. But it needs to be defined outside the class that is the memory binding for the static data member has to be done further outside the class typically in a source file, which can be initialized. And the interesting fact is this static member of course, is any data member could be an object. So, static data member is also an object.

The static data member gets constructed before the main function starts; this is something which probably you have not seen before in earlier cases. We always thought that when we start executing ,the main is when objects is start getting constructed, but the static members can get created or will get created before the main starts and symmetrically they will get destructed after the main ends after the main returns. So, we will see this through the example, static data members can be public or private any of the access specifier is possible.

Now in terms of accessing a static data member we will need to use the class name because as I have already mentioned here that it is associated with a class and not with the object. So, the typical object dot kind of notation is not the best way to access the static data members. They will be accessed from the class name using the colon-colon or scope resolution operator.

And in terms of usage, the static data members are nothing other than they behave exactly like the global variables. The only difference being or the significant difference being that the global variables exist in the global scope, but static data members are within the scope of the class in which the static data member belongs. So, they virtually eliminate the requirement of having any global data in the program. So, with this introduction, let me take you through a simple example and then these points will become more and more clears.

(Refer Slide Time: 05:26)



## Program 16.01: static Data Member A Simple Case

Non static Data Member

```
#include<iostream>
using namespace std;
class MyClass { int x; // Non-static
public:
    void get() { x = 15; }
    void print() {
        x = x + 10;
        cout << "x =" << x << endl ;
    }
};

int main() {
    MyClass obj1, obj2;
    obj1.get(); obj2.get();

    obj1.print(); obj2.print();
    return 0 ;
}
---
x = 25 , x = 25
```

- x is a non-static data member
- x cannot be shared between obj1 & obj2

- Non-static data members do not need separate definitions - instantiated with the object
- Non-static data members are initialized during object construction

static Data Member

```
#include<iostream>
using namespace std;
class MyClass { static int x; // Declare static
public:
    void get() { x = 15; }
    void print() {
        x = x + 10;
        cout << "x =" << x << endl;
    }
};
int MyClass::x = 0; // Define static data member
int main() {
    MyClass obj1, obj2;
    obj1.get(); obj2.get();

    obj1.print(); obj2.print();
    return 0;
}
---
x = 25 , x = 35
```

- x is static data member
- x is shared by all MyClass objects including obj1 & obj2
- static data members must be defined in the global scope
- static data members are initialized during program start-up

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

This is an example of a static data member; it is very simple case. So, all that I have is a class, it is some this class does not do anything very meaningful, it is called by my class and it has ordinary data member here int x. So, on the left column, what you see, does not use static member it is what you have been introduced to earlier in terms of the data members and member function. So, x is a data member, there is a get member function which initializes x with 15. And there is a print member function, which takes the current value of x, increments it by or adds 10 to that, and then prints that value. So, these are simple thing we are trying to do.

So, now if we look into the application code here, we are constructing two such objects object 1 and object 2, as you can see that my class does not have any constructor. So, we know the free default constructor will be used; and by that constructor, x will not be assigned in a specific value. So, we do invoke get on object 1. So, if we invoke get on object 1 in here, then certainly for object 1, so if I can draw in this is my object 1, this is my object 2, then I have x in both. And as I execute get for object 1, this will become 15; as I execute get for object 2, this will become 15. And then if I do print, invoke print for object one, so it will print will take this value 15 as in here add 10 to it. So, this will become 25, and therefore, it will print 25. It prints x is equal to 25.

Similarly, print is again invoked for object 2, it again takes this value which is 15 adds 10 to it, because I have called print, the value becomes 25 and this is what is printed. So, this behavior if you look at then this behavior is already very clear to you. So, you will know that I am doing exactly the same thing with both the objects object 1 and object 2, and therefore, both of them print the same value 25 for x.

Now, look at the right column, where we are trying to use the static data member. So, how do I use a static member, earlier I was saying this is int x, now I am saying it is static int x. So, I have prefixed the keyword static in front of the data member declaration. So, this is a non-static declaration, this now becomes a static declaration. Now with this the difference that that happens is if I redraw what I had on this side with a non-static data I had obj 1, obj 2, data member x, data member x, here when I will construct the objects at this line, I will have obj 1, I will have obj 2, but none of these objects obj 1 and obj 2 has x as a data member, because x is a static data member and it is not associated with any specific object.

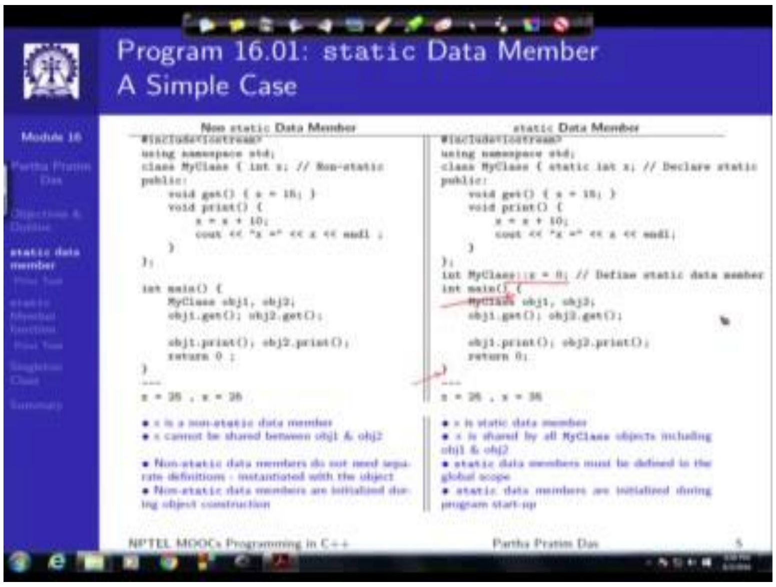
So, there is another object whose name is MyClass colon-colon x, my class colon-colon x, which is the static data member x of this class. So, you can see that the moment I define this to be static my data member is not a part of the object neither a part of object 1 nor a part of object 2, but it becomes a separate location, it becomes a separate object. Now the only thing is a name of that object is not x, it is MyClass colon-colon x that it is qualified by the class name and that is how I access it.

So, let us see when I tried to use it using the same application that I had here this there is no difference between this application and this application in main. The objects have got constructed as we have seen. I invoked get, so x is assigned 15. So, what does this mean what is x, x here is the x of MyClass, so it is this variable. So, this becomes 15. This static value becomes 15. I again execute get for invoke I get for obj 2, the same x is assigned 15, because x is static it is common between two objects. So, x is set to 15 twice.

Now I invoke print for obj 1, so what will happen the value of x is 15 and 10 is added to it, so the value of x will now become 25, because 10 has got added, and that value is what will get printed. So, when I do obj 1 dot print x is printed as 25. Till this point, there is no difference in the behavior from the non-static case.

But consider the next one when the print is invoked for obj 2. When print is invoked for obj 2, the value of x now is 25, it is not fifteen anymore. So, this will, 10 will get added to this, and now it will become 35 and x now becomes 35. And so, when I print according to this print method then the value that I get printed is 35, because the static object is shared by the both objects. So, when I have invoked print twice, once through object 1, and once through object 2, the value 10 has been added to x twice. And therefore, earlier it was 25; in the second case now with the static data member, it becomes 35, so that is the basic behavior of static data members that we will have to learn.

(Refer Slide Time: 12:50)



## Program 16.01: static Data Member A Simple Case

Module 16

Partha Pratim Das

Objectives & Outline

static data member

Print Task

static Member function

Print Task

Singleton Class

Summary

Non static Data Member

```
#include<iostream>
using namespace std;
class MyClass { int x; // Non-static
public:
    void get() { x = 15; }
    void print() {
        x = x + 10;
        cout << "x =" << x << endl ;
    }
};

int main() {
    MyClass obj1, obj2;
    obj1.get(); obj2.get();

    obj1.print(); obj2.print();
    return 0 ;
}
---
x = 25 , x = 25
```

- x is a non-static data member
- x cannot be shared between obj1 & obj2
- Non-static data members do not need separate definitions : instantiated with the object
- Non-static data members are initialized during object construction

static Data Member

```
#include<iostream>
using namespace std;
class MyClass { static int x; // Declare static
public:
    void get() { x = 15; }
    void print() {
        x = x + 10;
        cout << "x =" << x << endl;
    }
};
int MyClass::x = 0; // Define static data member
int main() {
    MyClass obj1, obj2;
    obj1.get(); obj2.get();

    obj1.print(); obj2.print();
    return 0;
}
---
x = 25 , x = 35
```

- x is static data member
- x is shared by all MyClass objects including obj1 & obj2
- static data members must be defined in the global scope
- static data members are initialized during program start-up

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

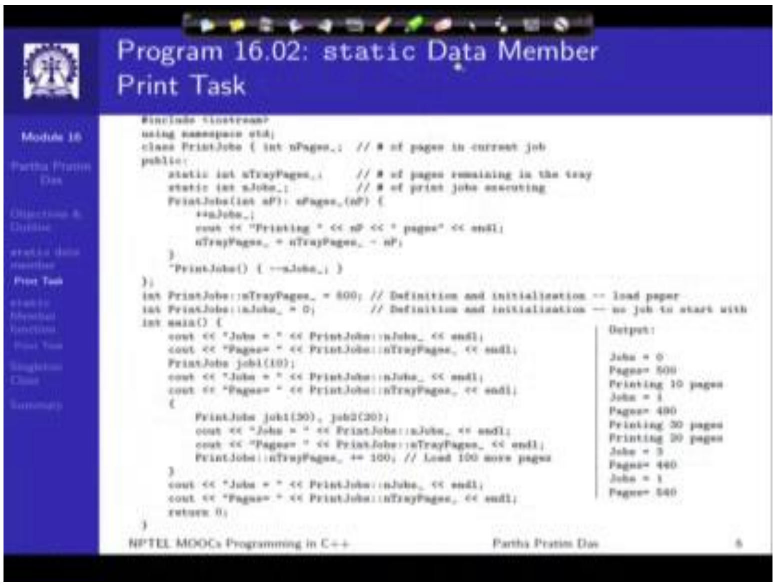
So, we have noted that x is a static data member it is shared by the two objects we have seen how they are shared. And this next point is very important, let me explain this point. If you compare the two programs other than the static keyword here, there is one additional line that has been included in this. We say that this when saying that this data member is static, this is nearly a declaration, which means that it is defining it is saying that the name of this variable is x, it is a class variable. It will have one instance for the class it is of type int, but it does not associate any memory with this variable. When it is non-static, I do not care about this, because if it is non-static I know at some point some object will get instantiated like obj 1; and wherever obj 1 gets instantiated, x will be a part of it. So, x will get its memory through the instantiation of obj 1, but in case of static this is just a declaration, because when obj 1 will get instantiated the class variable x is not a part of this object.

So, I need to create the memory or bind the memory for this data member - static data member separately and this is what is known as a definition through which I create that. So, that definition says that it is done in the in the global scope, I say that this is the name of the variable MyClass colon-colon x, class name colon-colon variable name. The type is int. So, I say the type is int and I put the initialization which I have put here as 0. So, this is to define the static data member. So, if you write a program which has static data member, and if you do not do this, do not put this then the compiler will start crying saying that this particular static data member MyClass colon-colon x has not been defined, and therefore will not get a memory.

So, this is something special that you will have to remember for the static data member that they will require a definition separately and they will need to be initialized when the program starts working.

So, when does this get executed, when does this x gets the value 0, it gets the value 0 before you start on the beginning of main. So, as we start executing the program, first all static data members of all classes will get constructed and initialized, and only then main will start working. Similarly when main ends only after that all the static data members of all classes will start getting destructed in the reverse order in which they were constructed. Of course, in this particular example, you do not see that construction, destruction process, because in this case the static data member is an object of the built in type, where we know there is no explicit constructor or destructor available.

(Refer Slide Time: 16:22)



## Program 16.02: static Data Member Print Task

```
#include <iostream>
using namespace std;
class PrintJobs { int nPages_;  // # of pages in current job
public:
    static int nTrayPages_;     // # of pages remaining in the tray
    static int nJobs_;          // # of print jobs executing
    PrintJobs(int nP): nPages_(nP) {
        ++nJobs_;
        cout << "Printing " << nP << " pages" << endl;
        nTrayPages_ = nTrayPages_ - nP;
    }
    ~PrintJobs() { --nJobs_; }
};
int PrintJobs::nTrayPages_ = 500; // Definition and initialization -- load paper
int PrintJobs::nJobs_ = 0;        // Definition and initialization -- no job to start with
int main() {
    cout << "Jobs = " << PrintJobs::nJobs_ << endl;
    cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
    PrintJobs job1(10);
    cout << "Jobs = " << PrintJobs::nJobs_ << endl;
    cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
    {
        PrintJobs job1(30), job2(20);
        cout << "Jobs = " << PrintJobs::nJobs_ << endl;
        cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
        PrintJobs::nTrayPages_ += 100; // Load 100 more pages
    }
    cout << "Jobs = " << PrintJobs::nJobs_ << endl;
    cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
    return 0;
}
```

```
Output:

Jobs = 0
Pages= 500
Printing 10 pages
Jobs = 1
Pages= 490
Printing 30 pages
Printing 20 pages
Jobs = 3
Pages= 440
Jobs = 1
Pages= 540
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 6

So, with this let us move on, and look at a little bigger example, a more realistic example of using the static data member. So, here what we are trying to do we have we are creating a class print jobs, which has every printing job has a number of pages to print. And when I construct the print job object, then I know what are the number of pages that need to be printed in this job. Now what I want to track through this is I want to track that certainly I am assuming that all these print jobs actually fire the printing on a single printer. So, the tasks print jobs actually task go there. So, I want to track two information; as to, how many different print jobs are currently working, so I call that n jobs.

Now naturally how many jobs are currently ready for printing does not depend on the specific job, it depends on the total number of print job objects that have been constructed. So, it is a property which is at the class level, which is a static property and therefore, I call it a static int n jobs. Similarly, I conceive that as if my printer has a printing tray where a certain number of pages have been loaded, so I want to track how many pages are remaining in that tray. So, I create n tray pages data member which I also make static, because it is not specific to a particular job, but it is specific to the total print job class as to how many pages are remaining in the tray.

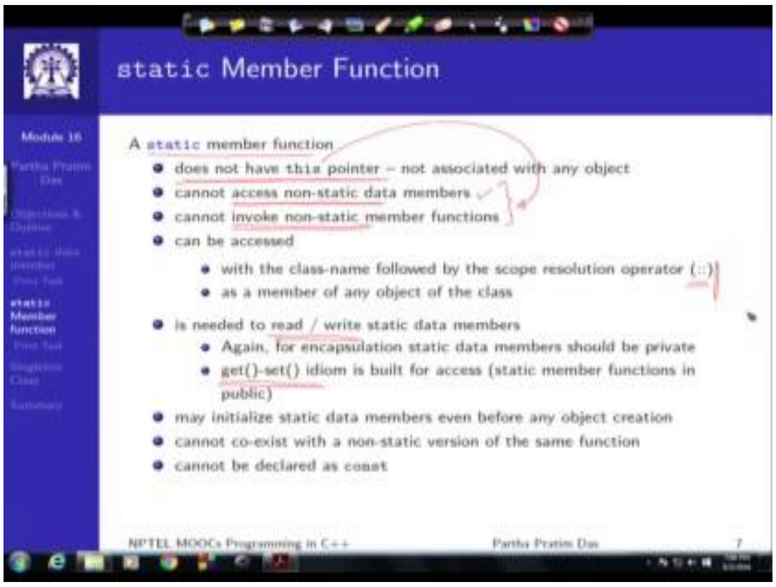
With that, now naturally here so these are the two static data members these are initialized constructed and initialize at this point and then I do a whole lot of tasks. I initially print the value of n jobs, which must be 0 here as the output, then I prints the number of pages that exist in the tray which is initialized with 500. So, the output should be 500, and then I construct a print job object. So, my number of jobs should get incremented by 1, and my number of pages remaining must get decremented by the number of pages that this job is supposed to print. So, it should get decremented by 10. So, after constructing this object, when I print the jobs and pages that are currently available, so I get jobs is now become 1 incremented from 0, and the number of pages remaining is 490 decremented by 10 from the 500 value.

Then I get into this particular scope, where I construct two more objects and then I see what is the value of the jobs, and the number of pages jobs will become 3, because two more print jobs are on, and the number of pages go down by 30 and 20 further. So, it becomes 30 less 460, 20 less from that, 440, so that is the remaining number of pages.

And then I increment the number of pages in the tray by 100, as if I am loading 100 pages, so my number of pages should become 440. And then I reach at this point and you can pretty well understand that since the scope is going out, this object job 1 and this object job 2 that were created within this scope will get destructed. So, if they get destructed then the number of jobs will come down from 3 to 1. So, when I print here I have one job, and when I print the number of pages remaining, I have loaded 100 more pages, so it becomes 540 in number, so that is how it behaves.

So, in this program using the static data members I can track some information which is not specific to every job, but it is specific to the whole collection of print jobs that I have that is the number of print jobs that are currently working; which is basically counting the number of objects of a class that is currently present in the system, and a global resource which is number of pages in the tray that I am manipulating.

(Refer Slide Time: 21:18)



## static Member Function

A static member function

- does not have this pointer – not associated with any object
- cannot access non-static data members
- cannot invoke non-static member functions
- can be accessed
  - with the class-name followed by the scope resolution operator (::)
  - as a member of any object of the class
- is needed to read / write static data members
  - Again, for encapsulation static data members should be private
  - get()-set() idiom is built for access (static member functions in public)
- may initialize static data members even before any object creation
- cannot co-exist with a non-static version of the same function
- cannot be declared as const

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

So, this is the typical use of the static data member. Now naturally as you can see that here the static data members are in the public visibility space, so as my application is changing it anyone can actually come and change these values by simply assigning a new value or changing on in terms of the incrementation or decrementation. So, what I should try to do next is to try to make them private, so that they cannot be changed directly.

Now certainly if I make them private then there comes a question of how do I manipulate it? I cannot change it anymore, because if I make it private then naturally I will not be able to change that value of the number of jobs or the number of pages in the tray. So, I need some function like, I have member functions, which can change data members, I need similar functions which can change the static data member values. And that brings us to the notion of static member functions.

Now, the critical thing about static member functions is just like functions. They are written with a static keyword in the front, but the difference is the key difference with other member function is they do not have a static member function, does not have at this pointer. Because, like the static data member is static member function is also not associated with any object, it is just associated with the class since it is not associated with an object, it does not have the objects address of the this pointer.

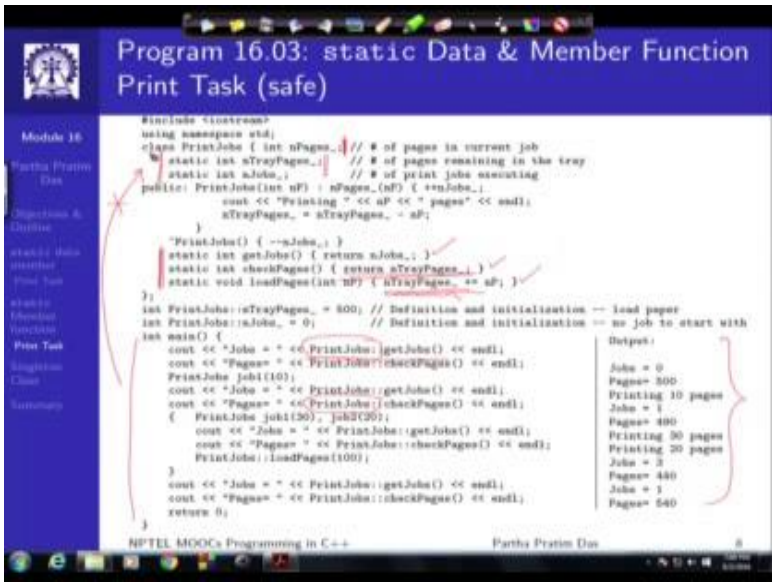
So, the consequence of that is a next statement is since it does not have at this pointer, it cannot know the address of the data members of the object, because it is not referring to any objects. So, which means it cannot access non-static data members of the class. It cannot invoke the non-static member functions of the class because all of this both of this need this pointer. So, not at being at this pointer has a consequence that the static member function cannot access anyone of them.

So, how do you access a static member function, you like the static data member, you access the invoke the static member function using the class notation that is class name colon-colon the scope resolution operator class name colon-colon, the member function name. Now, I have already started saying in terms of why we need these, because I wanted to read and write static data members.

Whereas, earlier in the example, we showed that the static data members are public which has encapsulation issues? So, if I encapsulate the static data members as private then I will be able to manipulate them, read and write them using the static member functions and static member functions since they are class specific they can access the class specific static data members and can change them together. So which means that static member functions along with static data members will again allow me to enforce a similar kind of encapsulation though in the class level not at the object level. And I can create similar kind of get set idiom with the static data member static member function and the static data members that we have.

It can few points to be noted that static data member does not have cannot co exist with a non-static member function. A static member function cannot co exist with a non-static member function of the same name that is you cannot have one member function of a class which is static, and by the same name have another which is non static, this is not allowed. And certainly the static data members cannot be const which is a consequence of not having this pointer, because we know that if a member function is const, and then basically the type of this pointer changes. This pointer becomes a pointer to a constant object the static member function has no object to refer to therefore there is no sense of calling it a constant.

(Refer Slide Time: 25:21)



## Program 16.03: static Data & Member Function Print Task (safe)

```
#include <iostream>
using namespace std;
class PrintJobs { int nPages_; // # of pages in current job
    static int nTrayPages_;    // # of pages remaining in the tray
    static int nJobs_;         // # of print jobs executing
public: PrintJobs(int nP) : nPages_(nP) { ++nJobs_;
            cout << "Printing " << nP << " pages" << endl;
            nTrayPages_ = nTrayPages_ - nP;
        }
    ~PrintJobs() { --nJobs_; }
    static int getJobs() { return nJobs_; }
    static int checkPages() { return nTrayPages_; }
    static void loadPages(int nP) { nTrayPages_ += nP; }
};
int PrintJobs::nTrayPages_ = 500; // Definition and initialization -- load paper
int PrintJobs::nJobs_ = 0;        // Definition and initialization -- no job to start with
int main() {
    cout << "Jobs = " << PrintJobs::getJobs() << endl;
    cout << "Pages= " << PrintJobs::checkPages() << endl;
    PrintJobs job1(10);
    cout << "Jobs = " << PrintJobs::getJobs() << endl;
    cout << "Pages= " << PrintJobs::checkPages() << endl;
    {   PrintJobs job1(30), job2(20);
        cout << "Jobs = " << PrintJobs::getJobs() << endl;
        cout << "Pages= " << PrintJobs::checkPages() << endl;
        PrintJobs::loadPages(100);
    }
    cout << "Jobs = " << PrintJobs::getJobs() << endl;
    cout << "Pages= " << PrintJobs::checkPages() << endl;
    return 0;
}
```

```
Output:

Jobs = 0
Pages= 500
Printing 10 pages
Jobs = 1
Pages= 490
Printing 30 pages
Printing 20 pages
Jobs = 3
Pages= 440
Jobs = 1
Pages= 540
```

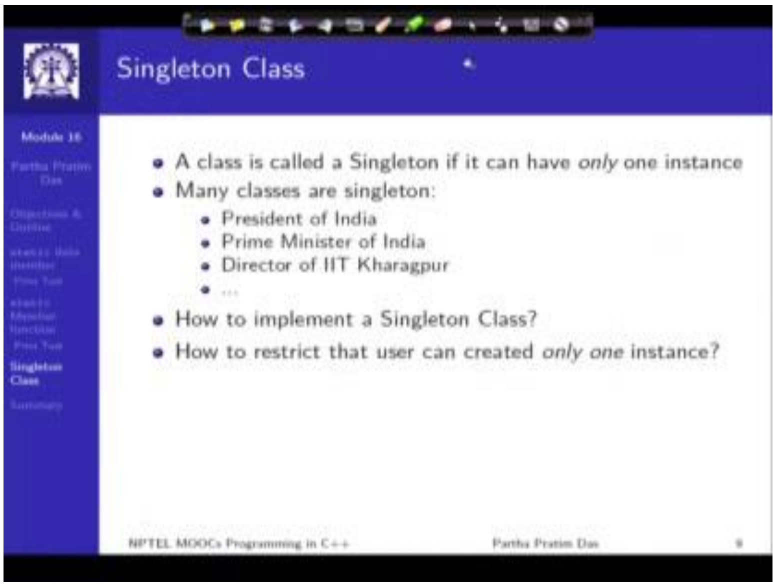
NPTEL MOOCs Programming in C++ Partha Pratim Das 8

With this, now let us go back and improve our print task example that we saw. So, what we have done here, we have moved the static members into the private part of the class declaration. So, now from main, you cannot directly change them it is not possible to change them directly, because they are private now. So, we have introduced a set of static member functions which can be used to read or write them, for example, getJobs reads, how many jobs are currently there; check pages read, how many pages are remaining in the tray; load pages, loads new pages on this and so on.

So, we rewrite this is the same application producing the same output, but instead of accessing these static data members directly, we now use the static member functions like when I want to know how many jobs are there I do PrintJobs colon-colon getJobs. Again note the notation this is a class name colon colon the static member function name. When I want to check the number of pages I do PrintJobs colon-colon checkPages and it will give me the number of pages remaining, I am sorry it will give me the number of pages remaining from here by invoking this function.

Certainly please note that none of these static member functions have this pointer. So, therefore they cannot access the non-static data like none of them can access the n pages data that may exist. They will only have to work with the static data that exist for the class.

(Refer Slide Time: 27:15)



## Singleton Class

- A class is called a Singleton if it can have *only* one instance
- Many classes are singleton:
  - President of India
  - Prime Minister of India
  - Director of IIT Kharagpur
  - ...
- How to implement a Singleton Class?
- How to restrict that user can created *only one* instance?

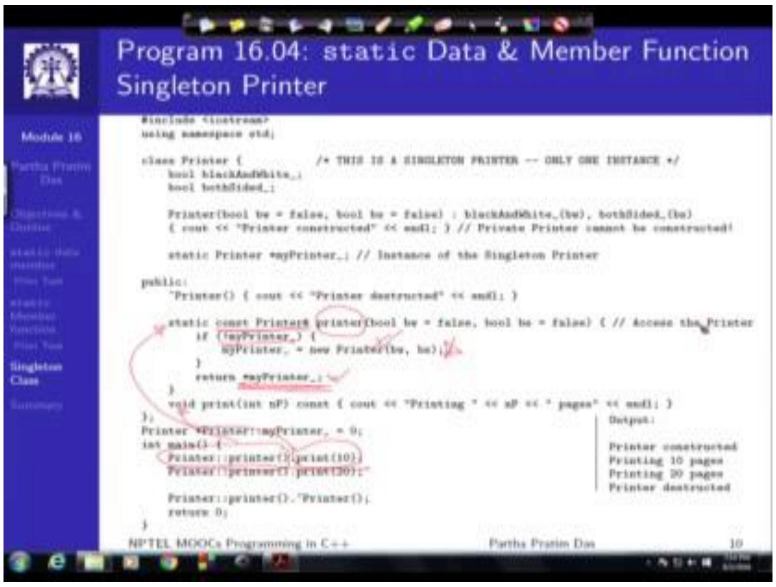
NPTEL MOOCs Programming in C++ Partha Pratim Das

You can go through this example at a later point you could line by line go through this example, and convince yourself that here we have the same functionality as the previous case, but we have been able to improve on the encapsulation. Now before we close, I would like to just quickly show you a very typical use of static members in terms of realizing, what we say is a singleton class.

A singleton class is a kind of design pattern which say that a class is called singleton, if you can have only one instance of that class at a time - only one instance at a time. Initially, it sounds little awkward absurd as to why do I need such classes. But if you think and look around, you will find number of classes have that kind of a behavior like president of India, there is only one president of India, there is only one prime minister of India, there is only one director of IIT, Kharagpur and so on, so several classes have only one instances.

Now if I want a class to be like that then how do I implement such class? Naturally it is easy to design a class which cannot have any instance. All that you need to is to make the constructor private. If you make the constructor private then nobody can call it, and therefore, you cannot have any instance of that class. But how do you make sure that you can construct only one object, but not more than one object, so that is something which can be done very easily using the static data member and the static member function.

(Refer Slide Time: 28:26)



## Program 16.04: static Data & Member Function Singleton Printer

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

class Printer {            /* THIS IS A SINGLETON PRINTER -- ONLY ONE INSTANCE */
    bool blackAndWhite_;
    bool bothSided_;

    Printer(bool bw = false, bool bs = false) : blackAndWhite_(bw), bothSided_(bs)
    { cout << "Printer constructed" << endl; } // Private Printer cannot be constructed!

    static Printer *myPrinter_; // Instance of the Singleton Printer

public:
    ~Printer() { cout << "Printer destructed" << endl; }

    static const Printer& printer(bool bw = false, bool bs = false) { // Access the Printer
        if (!myPrinter_) {
            myPrinter_ = new Printer(bw, bs);
        }
        return *myPrinter_;
    }
    void print(int nP) const { cout << "Printing " << nP << " pages" << endl; }
};
Printer *Printer::myPrinter_ = 0;
int main() {
    Printer::printer().print(10);
    Printer::printer().print(20);

    Printer::printer().~Printer();
    return 0;
}
```

```
Output:

Printer constructed
Printing 10 pages
Printing 20 pages
Printer destructed
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

So, what you do is printer is one class. So, I am trying to show that how printer can be made singleton that is the situation is in the organization possibly I have only one printer. So, I have to make sure that more than one printer should never get instantiated. So, in the printer class, these are the non-static attributes these are just for completeness; do not worry about that, but here is the constructor which is made private. See this is in the private part. So, nobody directly can construct any object of the printer class type done.

The destructor of course, is kept in the public; because once the constructor is private you cannot construct it. So, destructor can still remain in the. Now what you do you create you introduce a static data member, which is a pointer to a printer, and also put that in private, so that nobody can directly. So, your idea is that this static data member which is printer colon-colon myprinter this one will point to the only printer object that I can have.

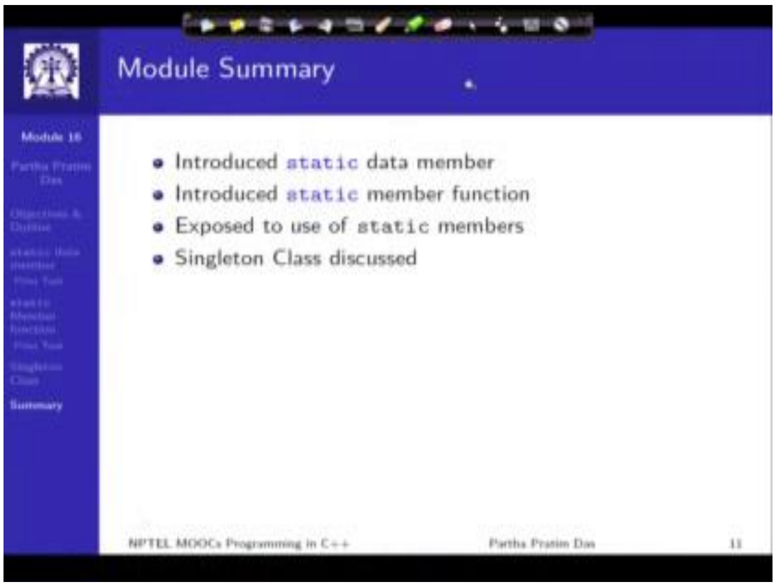
Now, the question of course, is how do I construct this only printer object. So, for that, in the public, you introduce a static member function. This member function is say called the printer. Now what it does, whenever you need a printer, whenever you need the printer object, you do not try to access the object directly, because you do not know where the object is.

Instead what you do, you invoke this particular static member function, what is static member function does, it initially checks, if this printer pointer is null or not null. What will happen at the very beginning here I have defined and initialized. So, at the very beginning this pointer will be null, there is no printer. If this is null you construct a printer and put the pointer here. So, you have constructed the printer and put it here.

Now mind you this particular static member function is able to construct the printer, because it is a member function. And therefore, it can access the private members. So, it invokes that gets a new printer, and returns you that printer. Subsequently, whenever you will call this particular static member function, you will find this to be non-zero. So, you will bypass this construction process, and you will always return the printer that you have constructed first.

In this process, only the first call will do a new on this printer object, and I will give you a printer object; from that point onwards, every time we will get back the same object. So, if you look into the use then this is you want to do the print job, so you say printer colon colon printer this part means this member function. So, this member function will then print, so it returns the printer mind you this is a return by reference. So, it returns the object printer, so on that object printer you invoke print which is this non-static member function and the printing will happen.

(Refer Slide Time: 32:31)


## Module Summary

- Introduced static data member
- Introduced static member function
- Exposed to use of static members
- Singleton Class discussed

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

Similarly, you can again do a print of 20. So, you can see this is the printer is got constructed, 10 pages are printed, 20 pages are printed and so on. And when you are done naturally you have created this object, so the responsibility is with you to destroy the object and therefore, you can call the destructor and destroy that object.

So, this is a simple way that singleton can be very safely implemented if you use static data members and static member functions. I have shown it with a printer class, but this can be done with any other class, which needs to be singleton. So, in summary, we have introduced the static data members and static member function, and we have shown that they can be used for various purposes of counting objects for maintaining any data at the class level and specifically for creating singleton objects.