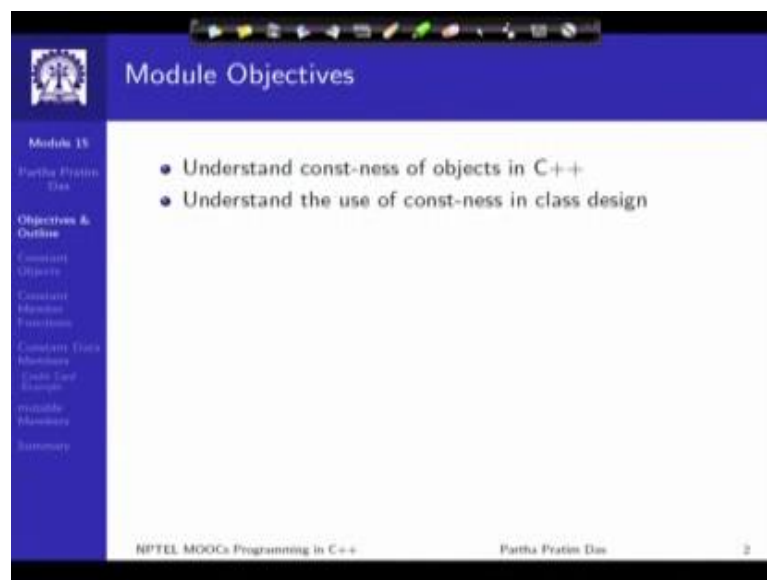


**Programming in C++**  
**Prof. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 29**  
**Const-ness**

Welcome to module 15 of Programming in C++. In this module, we will discuss about const-ness.

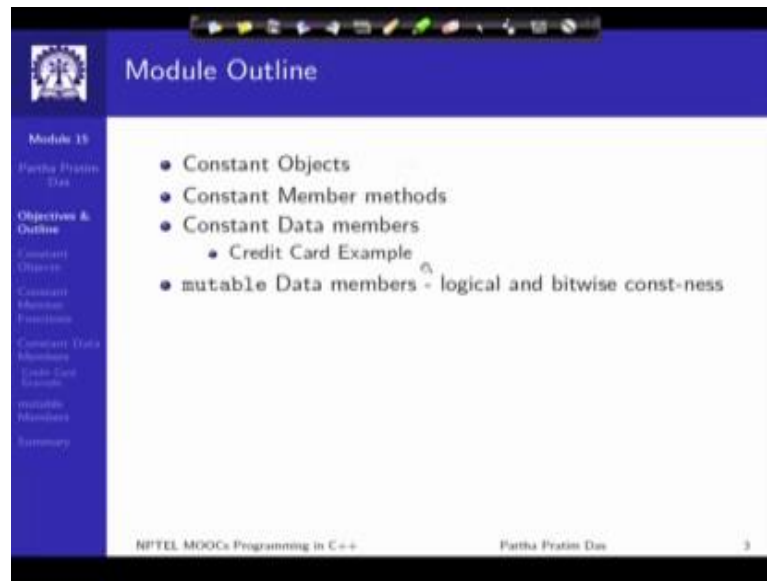
(Refer Slide Time: 00:36)



The image shows a slide from an NPTEL MOOC course. The slide has a blue header with the text 'Module Objectives' and a small logo on the left. Below the header is a white area containing two bullet points: '• Understand const-ness of objects in C++' and '• Understand the use of const-ness in class design'. On the left side of the slide, there is a vertical navigation menu with the following items: 'Module 15', 'Partha Pratim Das', 'Objectives & Outline', 'Constant Objects', 'Constant Member Functions', 'Constant Data Members', 'Const Cast Example', 'Mutable Members', and 'Summary'. At the bottom of the slide, there is a footer with the text 'NPTEL MOOCs Programming in C++', 'Partha Pratim Das', and the number '2'.

To understand the const-ness of C++, objects will be the main objectives for this module and we will also see how const-ness can be used in class design.

(Refer Slide Time: 00:51)



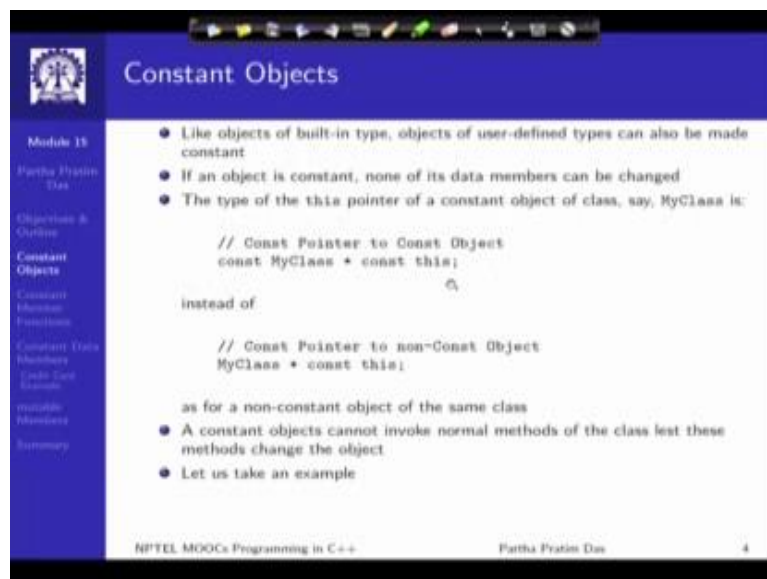
The slide shows a presentation interface with a blue header and a sidebar. The main content area lists the following items:

- Constant Objects
- Constant Member methods
- Constant Data members
  - Credit Card Example
- mutable Data members - logical and bitwise const-ness

At the bottom, it reads: NPTEL MOOCs Programming in C++ Partha Pratim Das 3

The outline is as given, it will follow on the left hand side. We will talk about constant objects; member functions; data members take an elaborate example and also discuss mutable data members.

(Refer Slide Time: 01:08)



The slide shows a presentation interface with a blue header and a sidebar. The main content area discusses constant objects and includes the following text:

- Like objects of built-in type, objects of user-defined types can also be made constant
- If an object is constant, none of its data members can be changed
- The type of the `this` pointer of a constant object of class, say, `MyClass` is:  

```
// Const Pointer to Const Object  
const MyClass * const this;
```

instead of

```
// Const Pointer to non-Const Object  
MyClass * const this;
```

as for a non-constant object of the same class

- A constant objects cannot invoke normal methods of the class lest these methods change the object
- Let us take an example

At the bottom, it reads: NPTEL MOOCs Programming in C++ Partha Pratim Das 4

First, let us introduce constant objects. We have seen const-ness earlier when we talked

about the procedural extension of C++, the better C++ part. We had seen that if we make an object constant or a data constant, a variable constant then that cannot be changed that has been set a value at the initialization time and after that, that value cannot be changed. So, this we had seen for built-in type data - int, double and so on; the pointer.

Now the same thing can be done with C++ objects. So, we note that like objects in a built-in type, the user define type objects can also be made constant. Now, naturally if an object is made constant then none of its data members can be changed, so these are main. An object has one or more data members. Once you make it constant none of them can be changed.

The primary consequence of making an object constant is that type of it is this pointer. We remember that this pointer refers to the address where this object reside, this pointer becomes a constant now becomes a pointer to a constant object. So, earlier when we introduced this pointer, we saw this is the type of this pointer, it is a constant pointer to the object of the class, but now it becomes a constant pointer to a constant object of the class. So, introduction of this const in the type of this pointer is a consequence of have declaring an object as constant.

Naturally, a constant object cannot invoke the normal methods of the class. This is understandable because we are saying that a constant object should not be allowed to change the data members of the object. Now, instead of directly changing if a method is invoked then that method can in turn change the data member. Therefore, to avoid that constant objects are not allowed to invoke the data member, we invoke the member functions. We will see in spite of this how constant objects can actually function.

(Refer Slide Time: 03:43)

**Program 15.01: Example: Non-Constant Objects**

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

class MyClass {
    int myPriMember_;
public:
    int myPubMember_;
    MyClass(int aPri, int aPub) : myPriMember_(aPri), myPubMember_(aPub) {}
    int getNumber() { return myPriMember_; }
    void setNumber(int i) { myPriMember_ = i; }
    void print() { cout << myPriMember_ << ", " << myPubMember_ << endl; }
};

int main() {
    MyClass myObj(0, 1); // Non-constant object

    cout << myObj.getNumber() << endl;
    myObj.setNumber(2);
    myObj.myPubMember_ = 3;
    myObj.print();

    return 0;
}
---
```

0  
2, 3

- It is okay to invoke methods for non-constant object `myObj`
- It is okay to make changes in non-constant object `myObj` by method `(setMember())`
- It is okay to make changes in non-constant object `myObj` directly `(myPubMember_)`

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

So, let us first take an example to understand this. So, here is an example of a non constant object. So, this non constant object has two members, one that is a private. So, this non constant object has a private member and a public member. So, we have introduced two member functions; get member and set member to read and write the private member and the public member naturally cannot be directly read or written. So, here is an application which is trying to read the private member, write the private member; read the public member and over all print the object. So, all of these will work fine you already understand this very well.

(Refer Slide Time: 04:43)

Module 15  
Partha Pratim Das  
Objectives & Outline  
Constant Objects  
Constant Member Functions  
Constant Data Members  
Code List  
Exercises  
Summary

### Program 15.02: Example: Constant Objects

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

class MyClass {
    int mPriNumber_;
public:
    int mPubNumber_;
    MyClass(int mPri, int mPub) : mPriNumber_(mPri), mPubNumber_(mPub) {}
    int getNumber() { return mPriNumber_; }
    void setNumber(int i) { mPriNumber_ = i; }
    void print() { cout << mPriNumber_ << ", " << mPubNumber_ << endl; }
};

int main() {
    const MyClass myConstObj(5, 0); // Constant object

    cout << myConstObj.getNumber() << endl; // Error 1
    myConstObj.setNumber(7); // Error 2
    myConstObj.mPubNumber_ = 8; // Error 3
    myConstObj.print(); // Error 4

    return 0;
}
```

- It is not allowed to invoke methods or make changes in constant object `myConstObj`.
- Error [1, 2 & 4] on method invocation typically is: cannot convert 'this' pointer from 'const MyClass' to 'MyClass &'.
- Error [3] on member update typically is: 'myConstObj': you cannot assign to a variable that is const.
- With `const`, this pointer is `const MyClass *` `const` while the methods expects `MyClass *`.
- Consequently, we cannot point the data member of the class (even without changing it).
- Fortunately, constant objects can invoke (select) methods if they are constant member functions.

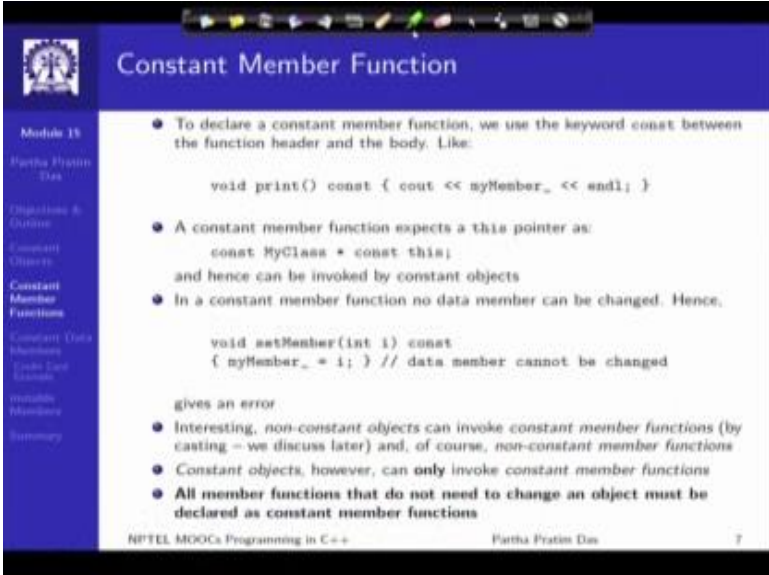
NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, we take a constant object. So, the same example the difference that we are making is now you have made the object constant. So, before the declaration of the object we have written `const`. So, by this the object `my const obj` is become a constant object, rest of the class is the same. It is just the use of the object which we have made constant, now with that you will notice that all these four use of the object gives compilation error. So, in the first case it is trying to invoke a get member to read the private member and print it. Now, this gives an error as you can see here, it says that cannot convert this pointer form `my const my class` to `my class ampersand`.

We will understand little bit more of what it really means, but to take the bottom line the compiler is saying that since the object is constant, you cannot use it to invoke a member function. The same error is a obtained in this case of set member, in case of print error 1, 2 and 4, all of them give the same error and if we try to directly write to the public member as we are trying to do here there is a public member. So, if we try to directly write to this public member then we get a different error, which says that you cannot assign to a data member when the object is constant. So, this ensures that if the object is been declared, is constant then we cannot access or change the data member's member functions of that particular class through that constant object.

Now, naturally on one side of this achieves the objective because it has been able to protect the data members, being constant it has to protect the data members that they cannot be changed, but now on other side it basically defeats the purpose because, for example, if I just wanted to read like in here, I just wanted to read the private member, I am not being able to even to do that or I just wanted to print the object. I cannot even do that because I cannot invoke any member function. So, we need another support in C++ to be able to do this that is for constant member functions, for constant objects we need to introduce constant member functions. We will see how this will be done.

(Refer Slide Time: 07:39)



The slide is titled "Constant Member Function" and is part of a presentation on C++ programming. It contains the following content:

- To declare a constant member function, we use the keyword `const` between the function header and the body. Like:  

```
void print() const { cout << myMember_ << endl; }
```
- A constant member function expects a `this` pointer as:  

```
const MyClass * const this;
```

and hence can be invoked by constant objects
- In a constant member function no data member can be changed. Hence,  

```
void setMember(int i) const  
{ myMember_ = i; } // data member cannot be changed
```

gives an error.
- Interesting, *non-constant* objects can invoke *constant member functions* (by casting – we discuss later) and, of course, *non-constant member functions*
- *Constant objects*, however, can **only** invoke *constant member functions*
- **All member functions that do not need to change an object must be declared as constant member functions**

At the bottom of the slide, it says "NPTEL MOOCs Programming in C++" and "Partha Pratim Das".

So, constant member functions are a new feature to support const-ness in C++. It is just like another member function, the only difference being that in the constant member function after the function prototype the header this is the header part and this is the body part of the function. In between these two, you write the keyword `const` and when you write this keyword `const`, the member function is called a constant member function.

Now, what it does, as we all know that when an objects wants to invoke the member function of the class then the address of that object get passed as a `this` pointer in that member function and we known the type of this pointer. Now, when you declare the constant member function then this pointer that is passed gets changed in its signature. It

now, is a pointer to a constant object because you are saying that the member function is a constant member function, so what compiler wants to emphasize is; only constant objects can invoke this member function.

So, we have already seen that the constant objects have this pointer feature of this type, which are constant pointers to constant objects and constant member functions need this pointer of the same type. So, now, it is possible that a constant object would be able to invoke this constant member function.

The consequence of that is what is given here, is if a member function is constant then no data member can be changed in that function. So, if we have the set member function and if we declare that to be a constant member function. So, that a constant object can invoke it then we will get a compilation error because in this function we are trying to change a data member by making an assignment to it. So, to sum up a constant member function can be used by constant objects only to read data members, only to access data members, but they cannot be used to write or change the data members and in any case non constant member functions cannot be invoked by the constant object.

So, with the combination of these two, we achieve the objective that a constant object can only maintain the data members that it already has maintains the same values, but it cannot allow to change the values of the data members.

Now, the next point here is also interesting that a non constant object can invoke a constant member function. This is simple because a constant member function guarantees that it will not make change to an object, if the object itself is non constant then we do not care whether it is changed or it is not changed. So, you are saying that it is a non constant object and a constant member function invoked on it says that I will not make any changes there is no contradiction. So, to sum up a non constant object can invoke constant member function as well as non constant member function, but a constant object can invoke only constant member function.

So, all member functions that do not need to change an object, those must be declared as constant member function to make sure that any constant object can also use them, can

also invoke those member functions.

(Refer Slide Time: 11:58)

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

class MyClass {
    int mPriNumber_;
public:
    int mPubNumber_;
    MyClass(int mPri, int mPub) : mPriNumber_(mPri), mPubNumber_(mPub) {}
    int getMember() const { return mPriNumber_; }
    void setMember(int i) { mPriNumber_ = i; }
    void print() const { cout << mPriNumber_ << ", " << mPubNumber_ << endl; }
};

int main() {
    MyClass myObj(0, 1); // Non-constant object
    const MyClass myConstObj(5, 0); // Constant object

    cout << myObj.getMember() << endl;
    myObj.setMember(2);
    myObj.mPubNumber_ = 3;
    myObj.print();

    cout << myConstObj.getMember() << endl;
    //myConstObj.setMember(7);
    //myConstObj.mPubNumber_ = 0;
    myConstObj.print();
    return 0;
}
```

Output

```
0
2, 3
5
5, 0
```

- Now myConstObj can invoke getMember() and print(), but cannot invoke setMember()
- Naturally myConstObj cannot update myPubMember.
- myObj can invoke all of getMember(), print(), and setMember()

NPTEL MOOCs Programming in C++ Partha Pratim Das

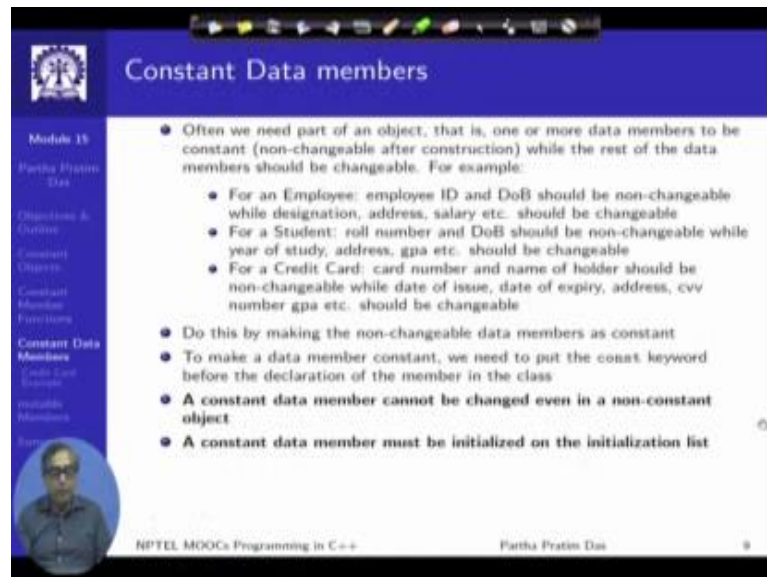
Let us see look at an example here. So, the same we go back to the same example now what we are doing we will look at the member functions and decide whether they should be constant or they should not be constant; get member is only reading a value so you make it constant, print is only reading and printing values of data members will make it constant, but this one the set member we cannot make a constant member function because set member intends to change the data member. If we have two objects now, one non constant and one constant, then as you can see that the non constant object can make all the changes as we have seen before.

Constant object now can invoke the get member function because it is a constant member function and read the value of my prime member data. It can print because print is a constant member function, but it cannot do any of these that is why I have made them, I have commented them it neither can invokes set member to change the value of the private member because set member is a non constant member function cannot be invoked by the constant object and of course, it cannot directly make assignments to the public member because it is a constant object. So, this we achieve the objective that we can control exactly, which a member functions can be invoked by the constant object and



protect all the values of the data members as they have been set at the time of construction.

(Refer Slide Time: 13:52)



The slide is titled "Constant Data members" and is part of a presentation by Partha Pratim Das. It contains the following bullet points:

- Often we need part of an object, that is, one or more data members to be constant (non-changeable after construction) while the rest of the data members should be changeable. For example:
  - For an Employee: employee ID and DoB should be non-changeable while designation, address, salary etc. should be changeable
  - For a Student: roll number and DoB should be non-changeable while year of study, address, gpa etc. should be changeable
  - For a Credit Card: card number and name of holder should be non-changeable while date of issue, date of expiry, address, cvv number gpa etc. should be changeable
- Do this by making the non-changeable data members as constant
- To make a data member constant, we need to put the const keyword before the declaration of the member in the class
- A constant data member cannot be changed even in a non-constant object
- A constant data member must be initialized on the initialization list

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, let us introduce a third type of const-ness that C++ supports. We have just talked about constant objects, where if I make an object constant then the whole of that object is constant. I just cannot make any change to that object, none of the data members can be changed, but often what we will need is, we will need to change some data members while some other data members should remain constant and I have mention several example, for example, here making an employee record and for that employee record you have an employee id, you have a date of birth.

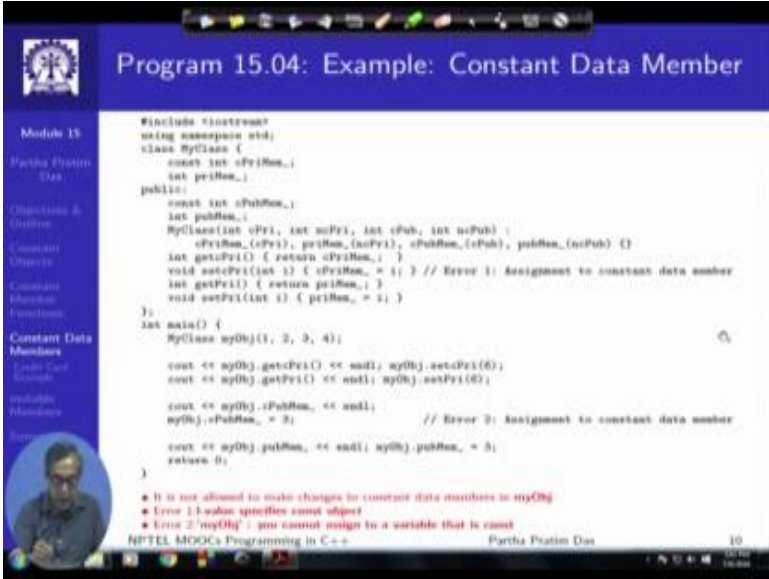
Certainly, once a record has been created we do not expect the employees id to be changed and of course, date of birth cannot change, but the same record same object will have designation, address, salary, all these they should be changeable because that is what the programming is all about.

So, several other examples if I am dealing with a student then again roll number, date of birth should be non changeable rest should be changeable, if we were talking about the credit card example we discussed earlier then the card number and the holder of the card

should be non changeable, whereas the cards do have a certain period of issue. So, once those are over, we would like to issue, re-issue the card again. So, the date of issue the date of expiry, the cvv all these will change even the holder can apply and change for an address. So, this should be changeable.

So, constant data members are a feature to support this kind of selective const-ness is a part of the object. So, we do this by making the data members constant by using a const keyword in front of its declaration. So, a constant data member cannot be changed even when the object is not constant. Of course, if an object is constant then nothing can be changed, but even if an object is non constant then the constant data member will not be able to change its value therefore, it must be set at the time of initialization of the object.

(Refer Slide Time: 16:10)



```
#include <iostream>
using namespace std;
class MyClass {
    const int cPriNum_;
    int priNum_;
public:
    const int cPubNum_;
    int pubNum_;
    MyClass(int cPri, int ncPri, int cPub, int ncPub) :
        cPriNum_(cPri), priNum_(ncPri), cPubNum_(cPub), pubNum_(ncPub) {}
    int getCPri() { return cPriNum_; }
    void setCPri(int i) { cPriNum_ = i; } // Error 1: Assignment to constant data member
    int getPri() { return priNum_; }
    void setPri(int i) { priNum_ = i; }
};

int main() {
    MyClass myObj(1, 2, 3, 4);

    cout << myObj.getCPri() << endl; myObj.setCPri(6);
    cout << myObj.getPri() << endl; myObj.setPri(6);

    cout << myObj.cPubNum_ << endl;
    myObj.cPubNum_ = 5; // Error 2: Assignment to constant data member
    cout << myObj.pubNum_ << endl; myObj.pubNum_ = 5;
    return 0;
}
```

• It is not allowed to make changes to constant data members in myObj  
• Error 1: value specified cannot object  
• Error 2: 'myObj.cPubNum\_' you cannot assign to a variable that is const

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

So, let's us look at this. Now, we have an example where we have two private members this is a non constant one and this is a constant one, we have made it constant by putting this const keyword. Similarly, I have shown two public data members, one of them is constant, and the other one is non constant. We have constructors which initialize all of them and we have the set get on all of these members.

Naturally, what will happen is if I am saying that my, this data member is constant I do

not expect it to change. So, when we try to write a set function on this data member which will assign a value to c prime MEM, then the compiler gives you an error because if this could compile without any error then an object would be able to invoke this and make changes to the data member. Be careful to understand that, in this case the member function does not need to be a const member function. This is a non const member function, but the reason you will get an error is because the member itself, the data member itself that you want to change itself has been declared to be constant. So, the set function cannot be invoked.

Similarly, if you look into this line where we are trying to assign a value to the public data member which has been declared as constant, we will get another error from the compiler because if you look into the declaration of the public data member, the public data member has been declared as constant. So, it should not be change it. So, this is how by using const-ness in front of the data members you can selectively make the data members constant, whereas we have been able to make changes, will be able to, for example, here we are making assignment to this particular data member, which is in the public member and that assignment does not give you an error, we are changing this which is also allowed because this is not a constant member.

(Refer Slide Time: 18:56)

**Credit Card Example**

Module 13  
Partha Pratim Das

Structure & Classes  
Constants  
Constant Objects  
Constant Member Functions  
Constant Data Members  
**Credit Card Example**  
Variable Members  
Summary

We now illustrate constant data members with a complete example of CreditCard class with the following supporting classes:

- String class
- Date class
- Name class
- Address class

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

So, now let us take a re look in our credit card example and see how constant data member can be used for a better design of the credit card class.

(Refer Slide Time: 19:12)

Module 15  
Partha Pratim Das

### Program 15.05: String Class: In header file with copy

```
#ifndef __STRING_H
#define __STRING_H
#include <iostream>
#include <string>
using namespace std;

class String { char *str_; size_t len_;
public:
    String(const char *s) : str_(strdup(s)), len_(strlen(s)) // ctor
    { cout << "String ctor: "; print(); cout << endl; }
    String(const String s) : str_(strdup(s.str_)), len_(strlen(s.str_)) // cctor
    { cout << "String cctor: "; print(); cout << endl; }
    String& operator=(const String s) {
        if (this != &s) {
            free(str_);
            str_ = strdup(s.str_);
            len_ = s.len_;
        }
        return *this;
    }
    ~String() { cout << "String dtor: "; print(); cout << endl; free(str_); } // dtor
    void print() const { cout << str_; }
};
#endif // __STRING_H
```

- Copy Constructor and Copy Assignment Operator added
- print() made a constant member function

NPTEL MOOCs Programming in C++ Partha Pratim Das 12

You have already seen all these class designs all these codes. So, I will not reiterate them, just note the small changes that have been made, for example, if you look into this string class then we have introduced in. Here, we have introduced the copy constructor and copy assignment operator, it was discussed in a module 14 and importantly in the print function, we have introduced a const because as you said, the print function is not expected to change any data member and we have just learnt that any member function that does not change the data members must be declared as a constant member function to protect that no constant object in advertently can change this. So, we will make all this print functions as const.

(Refer Slide Time: 20:13)

**Program 15.05: Date Class:**  
In header file with copy

```
#ifndef __DATE_H
#define __DATE_H
#include <iostream>
using namespace std;

char monthName[] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
char dayName[] = { "Monday", "Tuesday", "Wednesday", "Thursday",
                  "Friday", "Saturday", "Sunday" };

class Date {
    enum Month { Jan = 1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
    enum Day { Mon, Tue, Wed, Thu, Fri, Sat, Sun };
    U_INT date_; Month month_; U_INT year_;
public:
    Date(U_INT d, U_INT m, U_INT y) : date_(d), month_(MonthJan), year_(y)
    { cout << "Date ctor: "; print(); cout << endl; }
    Date(const Date& d) : date_(d.date_), month_(d.month_), year_(d.year_)
    { cout << "Date ctor: "; print(); cout << endl; }
    Date& operator=(const Date& d) { date_ = d.date_; month_ = d.month_; year_ = d.year_;
    return *this; }
    ~Date() { cout << "Date dtor: "; print(); cout << endl; }
    void print() const { cout << date_ << "/" << monthName[month_ - 1] << "/" << year_ << endl; }
    bool validate() const { /* Check validity */ return true; } // Not Implemented (RI)
    Day day() const { /* Compute day from date using time.h */ return Mon; } // NI
};
#endif // __DATE_H

• Copy Constructor and Copy Assignment Operator added
• print(), validate(), and day() made constant member functions
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 13

The date class; again in the date class we have provided the provisions for copy and we had made all these functions print, validate date, day, all these functions as const member functions as is protocol that we have agreed to.

(Refer Slide Time: 20:40)

**Program 15.05: Name Class:**  
In header file with copy

```
#ifndef __NAME_H
#define __NAME_H
#include <iostream>
using namespace std;
#include <string>

class Name {
    String firstName_, lastName_;
public:
    Name(const char* fn, const char* ln) : firstName_(fn), lastName_(ln)
    { cout << "Name ctor: "; print(); cout << endl; }
    Name(const Name& n) : firstName_(n.firstName_), lastName_(n.lastName_)
    { cout << "Name ctor: "; print(); cout << endl; }
    Name& operator=(const Name& n) {
        firstName_ = n.firstName_;
        lastName_ = n.lastName_;
        return *this;
    }
    ~Name() { cout << "Name dtor: "; print(); cout << endl; }
    void print() const { firstName_.print(); cout << " "; lastName_.print(); }
};
#endif // __NAME_H

• Copy Constructor and Copy Assignment Operator added
• print() made a constant member function
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 14

The name class again copies are added and the print function has been made into a

constant member function. We have seen this class also earlier it can be use to define the name of any person.

(Refer Slide Time: 21:05)

```
#ifndef __ADDRESS_H
#define __ADDRESS_H
#include <iostream>
using namespace std;
#include <string.h>

class Address {
    unsigned int houseNo,;
    String street, city, pin,;
public:
    Address(unsigned int hn, const char* st, const char* cn, const char* pin) :
        houseNo(hn), street(st), city(cn), pin(pin)
    { cout << "Address ctor: "; print(); cout << endl; }
    Address(const Address& a) :
        houseNo(a.houseNo), street(a.street), city(a.city), pin(a.pin)
    { cout << "Address ctor: "; print(); cout << endl; }
    Address operator=(const Address& a) {
        houseNo = a.houseNo, street = a.street, city = a.city, pin = a.pin,;
        return *this;
    }
    "Address() { cout << "Address ctor: "; print(); cout << endl; }
    void print() const {
        cout << houseNo, << " "; street, print(); cout << " ";
        city, print(); cout << " "; pin, print();
    }
};
#endif // __ADDRESS_H
```

- Copy Constructor and Copy Assignment Operator added
- print() made a constant member function

NPTEL MOOCs Programming in C++ Partha Pratim Das 15

The address class copy is added and I get the print function has been made a constant member function.

(Refer Slide Time: 21:14)

```
#ifndef __CREDIT_CARD_H
#define __CREDIT_CARD_H
#include <iostream>
using namespace std;
#include "Data.h"
#include "Name.h"
#include "Address.h"

class CreditCard { typedef unsigned int UINT; char *cardNumber,;
    Name holder,; Address addr,; Data issueDate, expiryDate,; UINT cvv,;
public:
    CreditCard(const char* cNumber, const char* fn, const char* ln,
        unsigned int hn, const char* st, const char* cn, const char* pin,
        UINT issueMonth, UINT issueYear, UINT expiryMonth, UINT expiryYear, UINT cvv) :
        holder(fn, ln), addr(hn, st, cn, pin), issueDate(1, issueMonth, issueYear),
        expiryDate(1, expiryMonth, expiryYear), cvv(cvv)
    { cardNumber = new char[strlen(cNumber) + 1]; strcpy(cardNumber, cNumber);
        cout << "CC ctor: "; print(); cout << endl; }
    "CreditCard() { cout << "CC ctor: "; print(); cout << endl; }

    void setHolder(const Name& h) { holder = h; } // Change holder name
    void setAddress(const Address& a) { addr = a; } // Change address
    void setIssueDate(const Data& d) { issueDate = d; } // Change issue date
    void setExpiryDate(const Data& d) { expiryDate = d; } // Change expiry date
    void setCVV(UINT v) { cvv = v; } // Change cvv number
    void print() const { cout << cardNumber, << " "; holder, print(); cout << " "; addr, print();
        cout << " "; issueDate, print(); cout << " "; expiryDate, print(); cout << " "; cout << cvv,;
    }
};
#endif // __CREDIT_CARD_H
```

- Set methods added
- print() made a constant member function

NPTEL MOOCs Programming in C++ Partha Pratim Das 15

Now, finally, we are with the credit cards class which makes use of this string date, name, address, all these classes and we have the same set of a data members and functionality. Now, what we do, we introduce a number of set methods earlier we have just been constructing the credit card object and printing it.

Now, we have introduced a number of set methods by which the different data members of the credit cards object can be changed that is I can set the name of a new holder, I can set the address, I can set the issue date and so on and of course, print has been made constant. So, this is the change that we have made and with this let us see what consequences happen.

(Refer Slide Time: 22:06)



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

#include "CreditCard.h"

int main() {
    CreditCard cc("6021711894640027", "Sherlock", "Holmes",
                221, "Baker Street", "London", "SW1 0KE", 7, 2014, 8, 2016, 011);
    cout << endl; cc.print(); cout << endl << endl;

    cc.setHolder(Name("David", "Cameron"));
    cc.setAddress(Address(10, "Downing Street", "London", "SW1A 3AA"));
    cc.setIssueDate(Date(1, 7, 2017));
    cc.setExpiryDate(Date(1, 8, 2019));
    cc.setCVV(123);
    cout << endl; cc.print(); cout << endl << endl;

    return 0;
}

// Construction of Data Members & Object
6021711894640027 Sherlock Holmes 221 Baker Street London SW1 0KE 1/Jul/2014 1/Jan/2016 011

// Construction & Destruction of temporary objects
6021711894640027 David Cameron 10 Downing Street London SW1A 3AA 1/Jul/2017 1/Jan/2019 123

// Destruction of Data Members & Object
• We could change address, issue date, expiry date, and cvv. This is fine
• We could change the name of the holder! This should not be allowed
NPTEL MOOCs Programming in C++ Partha Pratim Das 17
```

So, if you look into the credit card application. Now, in the application we have used the set members to make changes. So, all of these changes show up in terms of it was originally created, the object was created for Sherlock Holmes having certain address and some certain dates and now, it has been edited to have a different holder name with different address and so on.

So, we could change all of these different fields which are fine, but the disturbing part is we are able to change the name of the holder which according to a credit card issue

system should not be possible. A card issue to some one cannot be re issue to someone else. So, we have to stop this possibility of changing the name of the holder. So, this is where we will now use constant data member's tool.

(Refer Slide Time: 23:15)

**Program 15.06: Credit Card Class: Constant data members**

```
#ifndef __CREDIT_CARD_H
#define __CREDIT_CARD_H
// Include <iostream>, <string.h>, <date.h>, <time.h>, <address.h>
using namespace std;

class CreditCard { typedef unsigned int UINT;
    char *cardNumber_;
    const Name holder_; // Holder name cannot be changed after construction
    Address addr_;
    Date issueDate_, expiryDate_; UINT cvv_;
public:
    CreditCard(...); // ...
    ~CreditCard() { ... }

    void setHolder(const Name& h) { holder_ = h; } // Change holder name
    // error C2078: binary '*' : no operator found which takes a left-hand operand
    // of type 'const Name' (or there is no acceptable conversion)

    void setAddress(const Address& a) { addr_ = a; } // Change address
    void setIssueDate(const Date& d) { issueDate_ = d; } // Change issue date
    void setExpiryDate(const Date& d) { expiryDate_ = d; } // Change expiry date
    void setCVV(UINT v) { cvv_ = v; } // Change cvv number

    void print() { ... }
};
#endif // __CREDIT_CARD_H
```

- We prefix Name holder\_ with const. Now the holder name cannot be changed after construction
- In setHolder(), we get a compilation error for holder\_ = h; in an attempt to change holder\_.
- With const prefix holder\_ becomes constant - unchangeable

NPTEL MOOCs Programming in C++ Partha Pratim Das 18

So, what we do now, we have learnt how to do this. If we do not want the name of the holder to be changed, before the declaration of the name of the holder we put a const. So, the name of the holder now becomes a constant data member. So, in a credit object this cannot be edited which means that the set holder function that we had written the set holder function now becomes error because it is trying to assign to the holder which defines this constant. So, you get this some kind of compiler errors saying that you cannot have this kind of a function. If I cannot have this kind of a function then naturally I cannot change the name of the holder. So, we achieve our objective in that.



(Refer Slide Time: 24:09)



The slide displays the following C++ code for a cleaned-up CreditCard class:

```
#include <CREDIT_CARD.H>
#define __CREDIT_CARD_H
// Include <iostream>, <string.h>, <Date.h>, <Name.h>, <Address.h>
using namespace std;

class CreditCard { typedef unsigned int UINT;
    char cardNumber;
    const Name holder; // Holder name cannot be changed after construction
    Address addr;
    Date issueDate, expiryDate; UINT cvv;
public:
    CreditCard() { ... }
    CreditCard() { ... }

    void setAddress(const Address& a) { addr_ = a; } // Change address
    void setIssueDate(const Date& d) { issueDate_ = d; } // Change issue date
    void setExpiryDate(const Date& d) { expiryDate_ = d; } // Change expiry date
    void setCVV(UINT v) { cvv_ = v; } // Change cvv number

    void print() { ... }
};
#endif // __CREDIT_CARD_H
• Method setHolder() removed
```

So, this is just the same class cleaned up. We have now removed the set holder function because that function cannot be compiled and therefore, there is no way to change the set holder and we will use this too.

(Refer Slide Time: 24:25)



The slide displays the following C++ code for a revised application:

```
#include <iostream>
using namespace std;
#include "CreditCard.h"

int main() {
    CreditCard cc("6021711894640027", "Sherlock", "Holmes",
                221, "Baker Street", "London", "NW1 6XE", 7, 2014, 8, 2016, 811);
    cout << endl; cc.print(); cout << endl << endl;

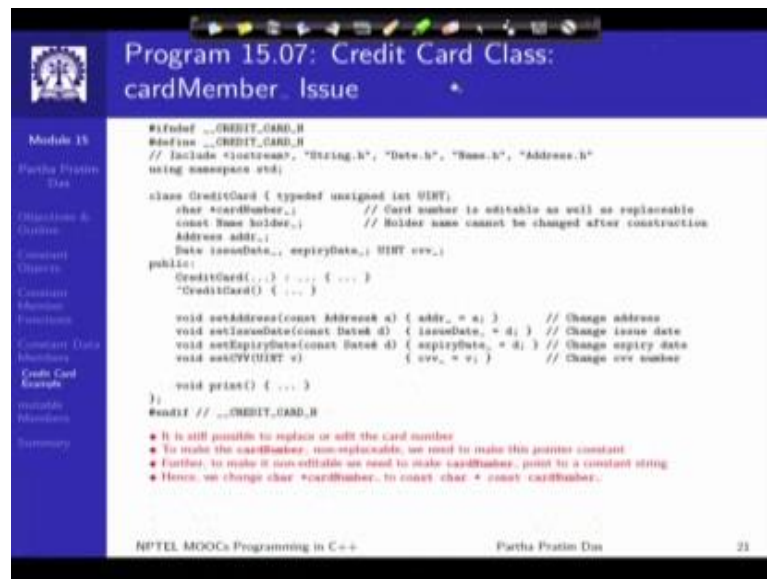
    // cc.setHolder(Name("David", "Cameron"));
    cc.setAddress(Address(10, "Downing Street", "London", "SW1A 2AA"));
    cc.setIssueDate(Date(1, 7, 2017));
    cc.setExpiryDate(Date(1, 8, 2019));
    cc.setCVV(127);
    cout << endl; cc.print(); cout << endl << endl;

    return 0;
}
// Construction of Data Members & Object
6021711894640027 Sherlock Holmes 221 Baker Street London NW1 6XE 1/Jul/2014 1/Jun/2016 811
// Construction & Destruction of temporary objects
6021711894640027 Sherlock Holmes 10 Downing Street London SW1A 2AA 1/Jul/2017 1/Jun/2019 127
// Destruction of Data Members & Object
• Now holder cannot be changed. So we are safe
• However, it is still possible to replace or edit the card number. This, too, should be disallowed
```

Again go through the application once more. Now, in the application, we have this is

commented out this cannot be done, but rest of the changes can still be done. They can still change the address, the issue date and all those. So, we are safe, but it is still possible that I can actually change the card number itself. I can edit the card number or replace the card number, put a different card number. So, how do we guard against that? How do we take care that this cannot be changed?

(Refer Slide Time: 25:02)



**Program 15.07: Credit Card Class: cardMember Issue**

```
#ifndef __CREDIT_CARD_H
#define __CREDIT_CARD_H
// Include <iostream>, <string.h>, <date.h>, <time.h>, <address.h>
using namespace std;

class CreditCard { typedef unsigned int UINT;
    char *cardNumber; // Card number is editable as well as replaceable
    const Time holder; // Holder name cannot be changed after construction
    Address addr;
    Date (issueDate, expiryDate, UINT svv);
public:
    CreditCard(..., ..., { ... }
    ~CreditCard() { ... }

    void setAddress(const Address& a) { addr = a; } // Change address
    void setIssueDate(const Date& d) { issueDate = d; } // Change issue date
    void setExpiryDate(const Date& d) { expiryDate = d; } // Change expiry date
    void setSVV(UINT s) { svv = s; } // Change svv number

    void print() { ... }
};
#endif // __CREDIT_CARD_H
```

- It is still possible to replace or edit the card number
- To make the cardNumber non-replaceable, we need to make this pointer constant
- Further, to make it non-editable we need to make cardNumber point to a constant string
- Hence, we change char \*cardNumber, to const char \* const cardNumber.

NPTEL MOOCs Programming in C++ Partha Pratim Das 21

For this we will have to see that a card number is a pointer variable. So, it is supposed to get allocated and it will point to a string. So, there are two ways that the card number can get affected, one is I can traverse this pointer and change the string, I can edit the object itself.

The other is I could reallocate a new object and put in the place of the card number there is I can just reallocate this. So, this is the situation like this is a card number, there is an allocation here of the card number some 4, 3, 8, 6 and so on. So, I can either edit this or I can change, add a new card number and so both of this have to be stopped. This should not be possible and we discuss this situation where we discussed about const-ness of pointers that here what we need we need the pointer as well as the pointed object to be both to be constant.

(Refer Slide Time: 26:15)

Program 15.07: Credit Card Class:  
cardMember\_Issue

```
#ifndef __CREDIT_CARD_H
#define __CREDIT_CARD_H
// Include <iostream>, <string.h>, <Date.h>, <Name.h>, <Address.h>
using namespace std;
class CreditCard {
    typedef unsigned int UINT;
    const char * const cardNumber_; // Card number cannot be changed after construction
    const Name holder_; // Holder name cannot be changed after construction
    Address addr_; // Data issueDate_, expiryDate_, UINT cvv_;
public:
    CreditCard(const char* cNumber, const char* fn, const char* ln,
              unsigned int bn, const char* cv, const char* pin,
              UINT issueMonth, UINT issueYear, UINT expiryMonth, UINT expiryYear,
              holder_(fn, ln), addr_(bn, bn, cv, pin), issueDate_(1, issueMonth, issueYear),
              expiryDate_(1, expiryMonth, expiryYear), cvv_(cvv)
    {
        cardNumber_ = new char[strlen(cNumber) + 1]; // ERROR: No assignment to const pointer
        strcpy(cardNumber_, cNumber); // ERROR: No copy to const C-string
        cout << "CC ctor: "; print(); cout << endl;
    }
    ~CreditCard() { cout << "CC dtor: "; print(); cout << endl; }
    // Set methods and print method skipped ...
};
#endif // __CREDIT_CARD_H
```

- cardNumber\_ is now a constant pointer to a constant string
- With this the allocation for the C-string fails in the body as constant pointer cannot be assigned
- Further, copy of C-string (strcpy) fails as copy of constant C-string is not allowed
- We need to move these codes to the initialization list

NPTEL MOOCs Programming in C++ Partha Pratim Das 22

So, this is what we simply do we make it a constant pointer and we make it point to constant string constant object and with this it should not be possible it would not be possible to change it, but now it face a different kind of problem. The problem is look at how this was initialized in the body of the constructor, you know at this point the object has been constructed, initialization is over and in this we have not initialized the card member because there is quite some code to write, to initialize that you have to allocate the string which we do here.

Before that we have to get the length of the string then we allocate the string and then we copy the given numbers strings in to this. So, we decided to do this in the body of the constructor, but now that this has become a constant pointer this assignment is not possible because the card member has already been initialized in the initialization list further since, the object itself is constant. This copies is not possible because the object also is constant, it is pointing to a constant object. So, if I do s t r c p y, if I copy the string then I am actually changing that string object. So, both of these will give an error. So, we will have to make sure, we will have to take care that we actually do all of these task in the initialization list itself.

(Refer Slide Time: 27:45)

```
#include <iostream>
using namespace std;
#include <string.h>
#include <Date.h>
#include <Name.h>
#include <Address.h>
class CreditCard {
    typedef unsigned int UINT;
    const char * const cardNumber_; // Card number cannot be changed after construction
    const Name holder_; // Holder name cannot be changed after construction
    Address addr_; Date issueDate_; expiryDate_; UINT cvv_;
public:
    CreditCard(const char* cNumber, const char* fn, const char* ln,
              unsigned int bn, const char* su, const char* cu, const char* pin,
              UINT issueMonth, UINT issueYear, UINT expiryMonth, UINT expiryYear, UINT cvv) :
        cardNumber_(strcpy(new char[strlen(cNumber)+1], cNumber)),
        holder_(fn, ln), addr_(bn, su, cu, pin), issueDate_(fn, issueMonth, issueYear),
        expiryDate_(1, expiryMonth, expiryYear), cvv_(cvv)
    { cout << "CC ctor" << endl; cout << endl; }
    ~CreditCard() { cout << "CC dtor" << endl; cout << endl; }
    void setAddress(const Address& a) { addr_ = a; } // Change address
    void setIssueDate(const Date& d) { issueDate_ = d; } // Change issue date
    void setExpiryDate(const Date& d) { expiryDate_ = d; } // Change expiry date
    void setCVV(UINT v) { cvv_ = v; } // Change cvv number
    void print() { cout<<cardNumber_<<" "; holder_.print(); cout<<" "; addr_.print();
        cout<<" "; issueDate_.print(); cout<<" "; expiryDate_.print(); cout<<" "; cout<<cvv_<<
    }
};
```

• Note the initialization of cardNumber\_. In initialization list  
• All constant data members must be initialized in initialization list

NPTEL MOOCs Programming in C++ Partha Pratim Das 23

We will not be able to do this in the body of the constructor. Now, if you look carefully, we have introduced the initialization of the card member in here. Please try to understand this, do not get scared by the little complexity of this expression, if you go one by one C number is basically the given card string. So, first we do find the length of it then we make an allocation for it. So, we get a pointer having adequate space where the card member card number can be put and then we do s t r c p y to copy C number into this location. So, it is like first we get the length. So, the length is 16, we add 1; so we get 17.

This 1 is for the null character then you make an allocation. So, we have an allocation of an array of 16 into some location and then we copy whatever my input string is into this using the s t r c p y. So, that my card member gets initialized to this string and it will have to be done here it will have to be done in the initialization list because both the pointer card MEM number pointer as well as the string that it will point to have become constant now, but with this we are able to protect and construct the object in a way, so that you cannot make any changes to the card number, once the credit card object has got created.

(Refer Slide Time: 29:49)



**mutable Data Members**

- While a *constant* data member is *not changeable* even in a *non-constant* object, a **mutable** data member is *changeable* in a *constant* object
- `mutable` is provided to model *Logical (Semantic) const-ness* against the default *Bit-wise (Syntactic) const-ness* of C++
- Note that:
  - `mutable` is applicable only to data members and not to variables
  - Reference data members cannot be declared `mutable`
  - Static data members cannot be declared `mutable`
  - `const` data members cannot be declared `mutable`
- If a data member is declared `mutable`, then it is legal to assign a value to it from a `const` member function.
- Let us see an example

NPTEL MOOCs Programming in C++ Partha Pratim Das 24

We have just seen how to create constant objects, constant member functions and constant data members and how to use that in the design.