Programming in C++

Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

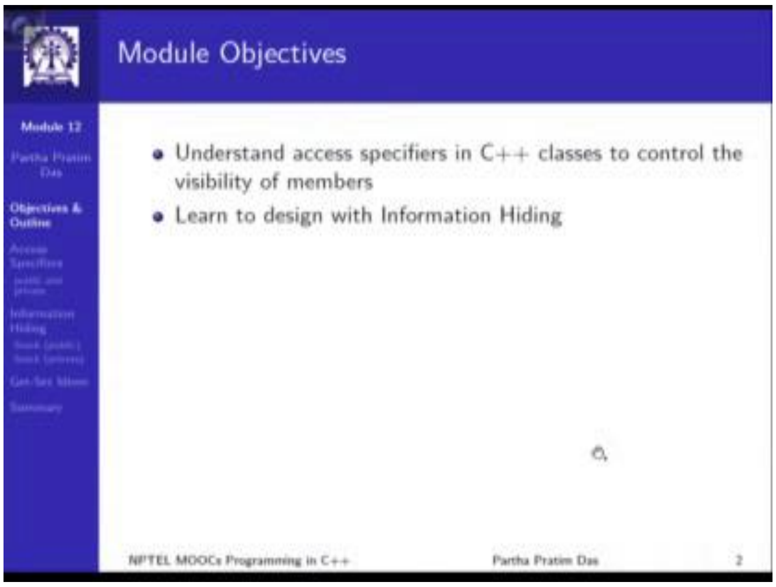
Lecture - 21
Access Specifiers

Welcome to Module 12 of programming in C++. In module 11, we have been introduced the basic concept of classes and objects. We have understood that using the class keyword in a style, which is very similar to the way we write struct in C. We can define aggregations of one or more data members.

We have also learned that along with the data members, the class also allows us to write functions. Which can what with those data members these functions, these special functions are called member functions or methods. And they could be invoked with a dot operator much in the same way that the data members are accessed. We have also seen that an object as an instance of a class every object as a unique identity, which is the address of that object and that address itself can be accessed within any member function of the class given the instance of the object has a special pointer known as this pointer.

So, we will continue on this and today we will focus on access specifiers or more specifically we would take a closer look into how class in C++ can be used to define better object encapsulation in object oriented programming.

(Refer Slide Time: 02:05)



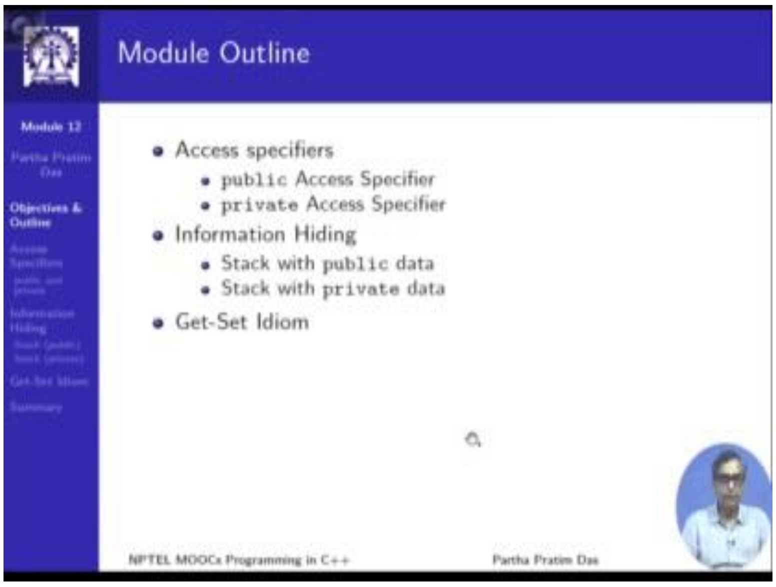
Module Objectives

- Understand access specifiers in C++ classes to control the visibility of members
- Learn to design with Information Hiding

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

So, our objectives are to understand access specifiers and to learn to design with information hiding.

(Refer Slide Time: 02:15)



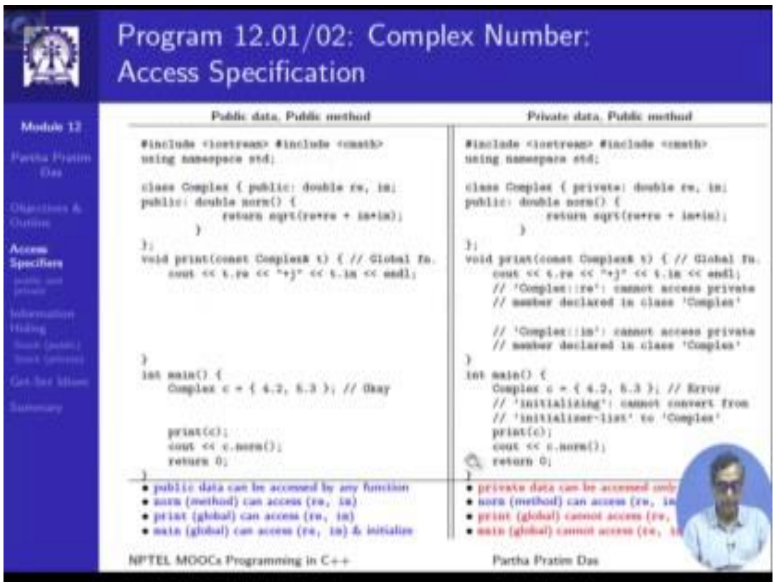
Module Outline

- Access specifiers
  - `public` Access Specifier
  - `private` Access Specifier
- Information Hiding
  - Stack with `public` data
  - Stack with `private` data
- Get-Set Idiom

NPTEL MOOCs Programming in C++ Partha Pratim Das

There are two kinds of access specifier, and we will show how they can be used to hide information and we will end with a specific get-set idiom.

(Refer Slide Time: 02:26)



Program 12.01/02: Complex Number: Access Specification

Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

Public data, Public method

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

class Complex { public: double re, im;
public: double norm() {
            return sqrt(re*re + im*im);
        }
};
void print(const Complex& t) { // Global fn.
    cout << t.re << "+j" << t.im << endl;




}
int main() {
    Complex c = { 4.2, 5.3 }; // Okay


    print(c);
    cout << c.norm();
    return 0;
}
```

- public data can be accessed by any function
- norm (method) can access (re, im)
- print (global) can access (re, im)
- main (global) can access (re, im) & initialize

Private data, Public method

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

class Complex { private: double re, im;
public: double norm() {
            return sqrt(re*re + im*im);
        }
};
void print(const Complex& t) { // Global fn.
    cout << t.re << "+j" << t.im << endl;
    // 'Complex::re': cannot access private
    // member declared in class 'Complex'

    // 'Complex::im': cannot access private
    // member declared in class 'Complex'
}
int main() {
    Complex c = { 4.2, 5.3 }; // Error
    // 'initializing': cannot convert from
    // 'initializer-list' to 'Complex'
    print(c);
    cout << c.norm();
    return 0;
}
```

- private data can be accessed only
- norm (method) can access (re, im
- print (global) cannot access (re,
- main (global) cannot access (re, im

NPTEL MOOCs Programming in C++ Partha Pratim Das

Start with the complex number example. We have seen this example before and now we are not looking at the C versions of the code anymore. We are looking at C++ class only show the left column and the right column both, show the C++ class complex; they have the same set of data members as we had already seen. But there is one difference if you see in the way they are specified. Here before the data members, there is a keyword public on the left hand side and there is a keyword private on the right hand side. These keywords are known as Access Specifier keywords and there are two of them for now to understand - a Public specifier and a Private specifier.

Earlier in module 11, every example, we saw was using public specifier. Which means that if you now look into the function that is written right after this the print function, which takes the instance of a complex object as a constant reference, it can print the components of that object; so, what it as to do for printing; given the object t, which it gets as a parameter, it as to refer to t dot re. That it as to refer to the specific data member of the class and it as allowed to do so. This will come as a little surprised that I am saying that is allowed to do so, because if instead of writing complex class if you would have written struct complex in c you would have always done only this because that is the only way C does it.

So, there is no surprise on this part if I say my data member is public then any method any function that I have globally defined, I can once I have the object of this class I can always access the data members and look what is there inside. So, using that we can define initialize an object and we can print the object.

Now, it may also be noted that this class complex also as a method norm and that method is also specify to be public. So, which means that if I have in main if I have an object c I can invoke this method norm and I will be allowed to do that and I can either use a global function print to print the values or I can use a method norm to compute the norm of that complex number.

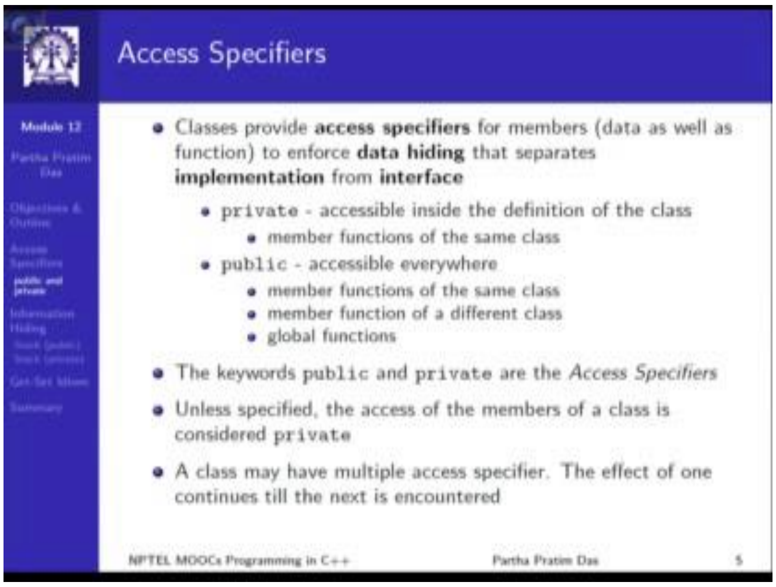
Now, let us look at the right hand side, here if you just compare the method norm is still called public, but the data members are now set to be private. Now look at the same norm function to print the complex number is a same code, but now if you try to compile it you find this kind of what is shown as commands or actually not part of the code.

If you just try to compile this with the C++ compiler these are the kind of compiler error messages that you will get. It says, for example, I will read out here complex colon colon re cannot access private member declared in class complex. Which mean that if I have a global function as in here and have an object, which needs to take or access the private data member of the class, the private data member I of the class then I am not allowed to do so. That is what is, but in this case I was allowed to do so, here I am not allowed to do so. That is the difference between be having a public access specifier or a private access specifier.

If you have a private access specifier then you are not allowed to access that member from outside the class, from a global function or some of the external functions. But if you look at the member function norm, member function norm is also trying to access the same component re; is trying to access the same component r e, but the compiler does not complain about it, because the norm is a member function of class. So, it is considered that norm will have the same kind of access rights, visibility rights as anyone else in that class, but a global function like print which is outside the class cannot have the private access kind of the private access is a privileged to the members of the class only.

So, if we summarize, just check the comments below. So, when the access of the data members is public, then data can be access by any function. When the access specifier of the data members is private, it can accessed only by the methods of that class and any global function or the main function is will not be able to access and manipulate and initialize those data members. This is want is known as a basic concept of access specification or feasible or it is also called feasibility restriction. Because as if something which is public, which you can sit outside the class in a global function or in the member function of some other class you can sit there and be able to see what is happening in all public members. But in terms of the private members you do not have the privilege of doing that.

(Refer Slide Time: 09:36)



Access Specifiers

Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

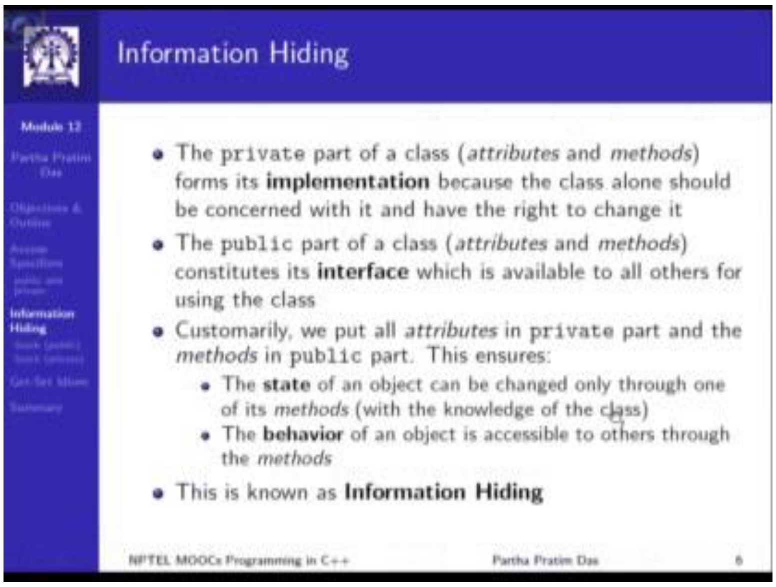
- Classes provide **access specifiers** for members (data as well as function) to enforce **data hiding** that separates **implementation** from **interface**
  - `private` - accessible inside the definition of the class
    - member functions of the same class
  - `public` - accessible everywhere
    - member functions of the same class
    - member function of a different class
    - global functions
- The keywords `public` and `private` are the *Access Specifiers*
- Unless specified, the access of the members of a class is considered `private`
- A class may have multiple access specifier. The effect of one continues till the next is encountered

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

Access specifiers, classes provide access specifiers for members and the access specifiers are available for data as well as for function we have shown example for data only for private, but they are available for both and there used to enforced data hiding. So, just to formally put, if some member is private it is accessible inside the definition of the class or which means that it is accessible from the member functions, accessible from the member functions of the class alone and not from any other many member function for some other class or from global functions. But if it is a public access then that member is accessible from naturally the member functions of the same class member functions of other classes' global functions and so on.

Public and private are the two keywords for access specification. By default if you have not, if you have just set class complex or class rectangle and we have not specified any access, then by default the access of the members of a class is considered to be private. And otherwise, we have to write private colon or not public colon to say what the access specification is and I can write it multiple times, every time I write access specification, that specification will continue to apply till a next specification is met. It could be again a public specification could be followed by another public specification or by a private specification and so on and as many of them as you want can be put.

(Refer Slide Time: 11:29)



Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

Information Hiding

- The `private` part of a class (*attributes* and *methods*) forms its **implementation** because the class alone should be concerned with it and have the right to change it
- The `public` part of a class (*attributes* and *methods*) constitutes its **interface** which is available to all others for using the class
- Customarily, we put all *attributes* in `private` part and the *methods* in `public` part. This ensures:
  - The **state** of an object can be changed only through one of its *methods* (with the knowledge of the class)
  - The **behavior** of an object is accessible to others through the *methods*
- This is known as **Information Hiding**

NPTEL MOOCs Programming in C++ Partha Pratim Das 6

Now, with this very tiny concept of access restriction or visibility restriction, we create what is known as the major infrastructure or major paradigm of object-oriented thinking object-oriented programming, which is known as information hiding. To be say, the private part of a class, private part of a class that is the attributes and method forms it is implementation. Because if it is private then the class alone can change it, class alone has a rights to change it and if it is an implementation the class alone should be concerned with it, others should not be concerned with what is there in the implementation.

Whereas the public part of a class attributes and methods both constitutes in interface which others can see others can access is available to anyone, who may want to use any instance of this class. So, private very strongly relates to implementation public strongly relates to interface.

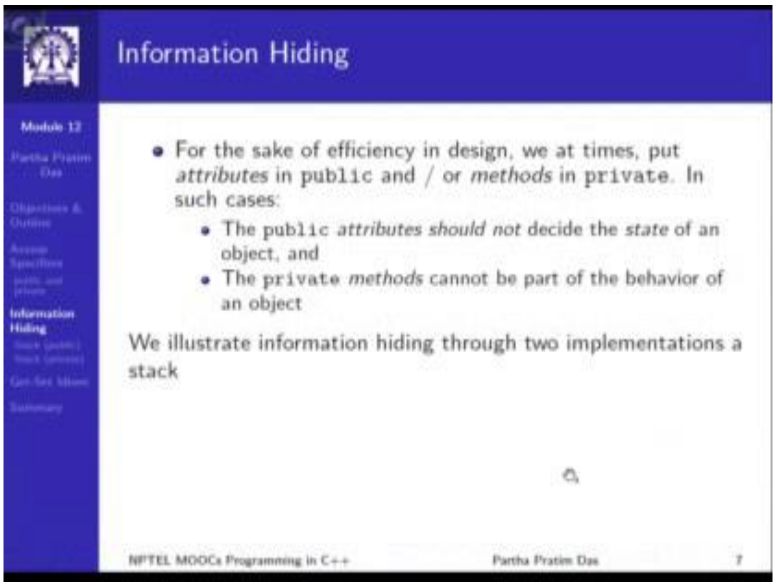
Now it is customarily, we will see this customarily, but this is a very strong design guideline that will consistently follow customarily; we will put all attributes that is data members in private please try to remember this and we put all this methods in public. So, if we put all data members in private what does it mean? What does the combination of values in the data members decide the combination of values in the data member we have seen in the last module decide the state of the object. So, the state of the object remains private, is a part of the implementation and it can be changed only through one of it is methods.

Because it is all data members are private if all attributes are private they cannot be changed addict the only way you can change them is to use some member function we changes them, which means that the object will always get to know that you have changed them because a function has been called. Whereas the behavior of the object, which is basically the collection of the methods that it supports, is accessible through other methods and it is accessible to others because that is the way the behavior is what you are offering to the external world.

So, if we talk little bit concrete in terms of example, that we have already seen, we have seen the example of a stack having an array of characters and a top marker. This character array in the top marker should be should have a visibility of private, that is they define the state of the object, state of the stack. And nobody should be able to change the top marker or write something arbitrarily in any location of the array, because the stack needs to maintain the last in first out property.

Whereas, the behavior of the stack will be defined by the methods that it support, specifically the empty top push and pop, which together allows us to define the LIFO behavior of the whole data structure. And this whole paradigm of separating out implementation from the interface separating or protecting the state behind the private visibility and exposing the behavior with the public visibility is known as information hiding in object oriented programming, alternate names follows like encapsulation, state base design and so on. But information hiding and encapsulation are the common names that we will try to follow here.

(Refer Slide Time: 15:44)



Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)
Stack (private)

Get-Set Idiom

Summary

Information Hiding

- For the sake of efficiency in design, we at times, put *attributes* in `public` and / or *methods* in `private`. In such cases:
  - The `public` *attributes should not* decide the *state* of an object, and
  - The `private` *methods* cannot be part of the behavior of an object

We illustrate information hiding through two implementations a stack

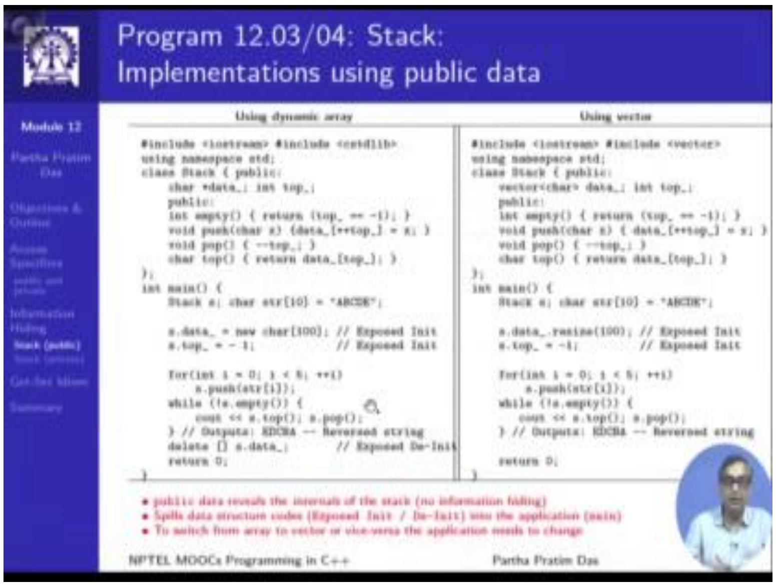
NPTEL MOOCs Programming in C++ Partha Pratim Das 7

Now obviously, when we learnt programming, when we learnt software design; a design can never be a something, which is very strongly cast in concrete. So, there are always some scopes for doing exceptions. And so for the sake of the efficiency we will show later on, that for the sake of efficiency at times we put some attributes in public, which goes against as we understand by now against the principle of information hiding or we can put some methods in private, which again goes against information hiding; because a method which is in private is not a part of the interface it as to be a part of the implementation.

So, the public attributes should be such that they do not decide the primarily the state of the object and the private methods cannot be the part of the behavior. So, just keep this in mind follow the rule principle of information hiding that is attributes are private and methods are public for the behavior and the state, but we will come across such exceptions, where we will show why this kind of changes are required in some designs.

Now, let us pick up two implementations of a stack to specifically discuss and illustrate this principle of information hiding.

(Refer Slide Time: 17:14)



Program 12.03/04: Stack: Implementations using public data

Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

Using dynamic array

```
#include <iostream> #include <cstdlib>
using namespace std;
class Stack { public:
    char *data_; int top_;
    public:
    int empty() { return (top_ == -1); }
    void push(char x) {data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";

    s.data_ = new char[100]; // Exposed Init
    s.top_ = - 1;            // Exposed Init

    for(int i = 0; i < 5; ++i)
        s.push(str[i]);
    while (!s.empty()) {
        cout << s.top(); s.pop();
    } // Outputs: EDCBA -- Reversed string
    delete [] s.data_;       // Exposed De-Init
    return 0;
}
```

Using vector

```
#include <iostream> #include <vector>
using namespace std;
class Stack { public:
    vector<char> data_; int top_;
    public:
    int empty() { return (top_ == -1); }
    void push(char x) { data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";

    s.data_.resize(100); // Exposed Init
    s.top_ = -1;         // Exposed Init

    for(int i = 0; i < 5; ++i)
        s.push(str[i]);
    while (!s.empty()) {
        cout << s.top(); s.pop();
    } // Outputs: EDCBA -- Reversed string

    return 0;
}
```

- public data reveals the internals of the stack (no information hiding)
- Spills data structure codes (Exposed Init / De-Init) into the application (main)
- To switch from array to vector or vice-versa the application needs to change

NPTEL MOOCs Programming in C++ Partha Pratim Das

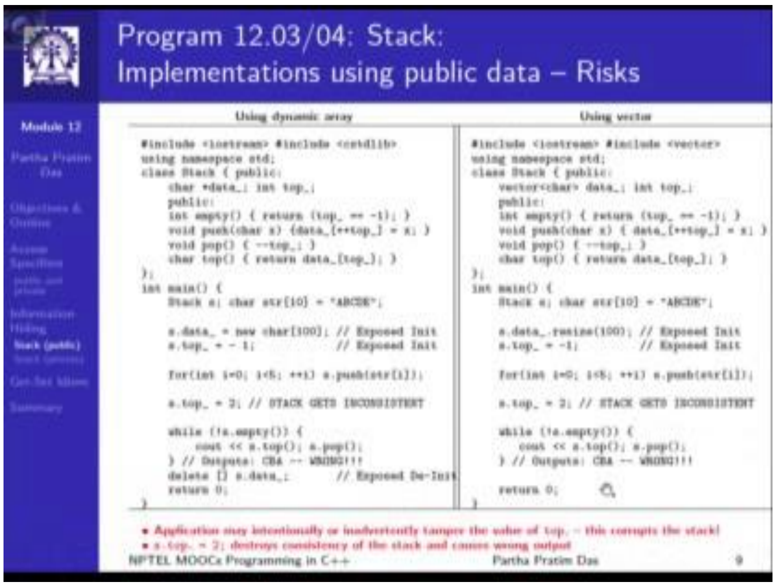
Here is a stack; here you are already very well familiar with the notion of the stack and how it is written. So, here you can see that we have the data of the stack as public access here, also we have the data of stack as public access the members methods are all public access. So, it is not strictly following the principle of information hiding it is exposed the internal data. So, since it is exposed the internal data and why does it need to expose the internal data, because if I have the stack defined like this I certainly need to do two things on it I need to allocate and appropriate sized array in the data underscore member of the stack class, so that the elements can be put in that array and I also need to initialize the top index to minus 1.

So, these are points where the internal state of the object is exposed and that is using the public access specification we can actually make use of that and then given that we can use the reversed string function realize the reversed string functionality by making use of the different stack functions.

Similarly, here we show another example here the data type that we use to key use as a container is a vector you will recall, we had discussed this in the early modules, but we talked about array vector duality and we showed how vectors can be very effective in doing this. But even if we use vectors then also we need to possibly do an initialization to give it is initialize size and of course, the index marker needs to be initialized the top marker needs to be initialized. So, if we use the public data that is if we do not enforce information hiding then you can very well see that between these two codes, which both of each are actually trying to take a string and reverse it.

There are a lot of statements that we need to put, which is basically trying to initialize the stack according to how it is implemented and so on. But the basic question is if I have a string and I want to use a stack of characters to reverse it why should I be bothered about, whether the stack is dynamically allocated using a character array or it uses as a vector or uses a automatically allocated static corrected array and so on. Why should I be at all concerned with it, why should the application at all need to know all these exposed initialization and de-initialization parts of the court? So, let us move on and see what other things can happen with the public data.

(Refer Slide Time: 20:22)



Program 12.03/04: Stack: Implementations using public data – Risks

Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

Using dynamic array

```
#include <iostream> #include <cstdlib>
using namespace std;
class Stack { public:
    char *data_; int top_;
    public:
    int empty() { return (top_ == -1); }
    void push(char x) {data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";

    s.data_ = new char[100]; // Exposed Init
    s.top_ = - 1;            // Exposed Init

    for(int i=0; i<5; ++i) s.push(str[i]);

    s.top_ = 2; // STACK GETS INCONSISTENT

    while (!s.empty()) {
        cout << s.top(); s.pop();
    } // Outputs: CBA -- WRONG!!!
    delete [] s.data_;       // Exposed De-Init
    return 0;
}
```

Using vector

```
#include <iostream> #include <vector>
using namespace std;
class Stack { public:
    vector<char> data_; int top_;
    public:
    int empty() { return (top_ == -1); }
    void push(char x) { data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";

    s.data_.resize(100); // Exposed Init
    s.top_ = -1;         // Exposed Init

    for(int i=0; i<5; ++i) s.push(str[i]);

    s.top_ = 2; // STACK GETS INCONSISTENT

    while (!s.empty()) {
        cout << s.top(); s.pop();
    } // Outputs: CBA -- WRONG!!!

    return 0;
}
```

- Application may intentionally or inadvertently tamper the value of top_ – this corrupts the stack!
- s.top_ = 2; destroys consistency of the stack and causes wrong output

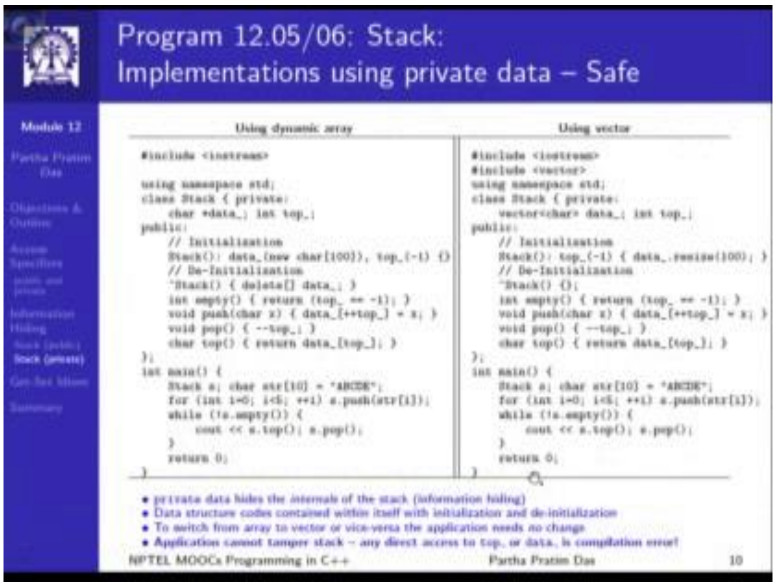
NPTEL MOOCs Programming in C++ Partha Pratim Das 9

The same example, but just think of just. I would like to just draw your attention to two lines one on the left here and one on the right here. Certainly, we will ask why should somebody do it what it is doing it is taking the value of a s dot top and assigning two into that in both cases now. If you do that what it means your top marker as been corrupted your top marker has being changed, which means the stack has become inconsistent, if you just now carefully looking to this code this is a string containing 5 characters.

So, this for loop runs for 5 times you have done 5 pushes. So, the top marker as got incremented 5 times for standing from minus 1 is gone through 0, 1, 2, 3, 4 and now we are just forcing it to a value 2, which means that the top most two elements, which are actually existing in the stack logically does not exist with the stack any more.

So, if you now complete this code then it prints C B A instead of it this should have printed E D C B A. But what has happened in the stack after it had done all the inserts A B C D E 1 2 3 4, I certainly forced the s top of the stack to two, which means that these two elements which are logically actual in the stack simply disappeared. So, the risk of exposing the data how could I do this, how could I change this, I do not know how the stack is managed this, but this was possible because I could access the top component of this class, the top data member of this class. So, exposing the implementation using public data as severe risks and that risk is what we are trying to avoid in terms of designing such classes.

(Refer Slide Time: 22:43)



Program 12.05/06: Stack: Implementations using private data – Safe

Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

Using dynamic array

```
#include <iostream>

using namespace std;
class Stack { private:
    char *data_; int top_;
public:
    // Initialization
    Stack(): data_(new char[100]), top_(-1) {}
    // De-Initialization
    ~Stack() { delete[] data_; }
    int empty() { return (top_ == -1); }
    void push(char x) { data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";
    for (int i=0; i<5; ++i) s.push(str[i]);
    while (!s.empty()) {
        cout << s.top(); s.pop();
    }
    return 0;
}
```

Using vector

```
#include <iostream>
#include <vector>
using namespace std;
class Stack { private:
    vector<char> data_; int top_;
public:
    // Initialization
    Stack(): top_(-1) { data_.resize(100); }
    // De-Initialization
    ~Stack() {};
    int empty() { return (top_ == -1); }
    void push(char x) { data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";
    for (int i=0; i<5; ++i) s.push(str[i]);
    while (!s.empty()) {
        cout << s.top(); s.pop();
    }
    return 0;
}
```

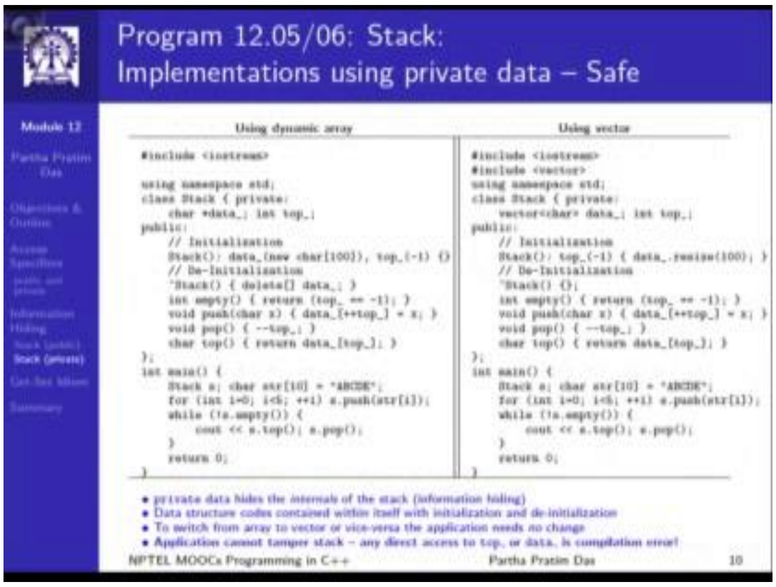
- private data hides the internals of the stack (information hiding)
- Data structure codes contained within itself with initialization and de-initialization
- To switch from array to vector or vice-versa the application needs no change
- Application cannot tamper stack – any direct access to top_ or data_ is compilation error!

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

Now we present the same design exactly the same code with a very minor change that is we now have the data put in as private access. So, now, we are following the principle of information hiding. What is the consequence? One consequence is the whole of the stack whether you do it by this or you do it by this is completely internal to the stack class. Which mean that certainly nothing regarding them can be done from the application, because the application will have no right to access the data underscore or top underscore members.

So, which means that the code the application code the two main functions now get absolutely identical, we can just refer you can just refer one slide backward, you can just refer one side backward we were here or if we go another slide backward we are here. And if you now compare the two main functions you can see that all those exposed in it, exposed d in it kind of lines are different between the two main functions. But both of them are trying to do the same string reverse, but they are just that they are using a stack whose internal container is different.

(Refer Slide Time: 24:15)



Program 12.05/06: Stack: Implementations using private data – Safe

Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

Using dynamic array

```
#include <iostream>

using namespace std;
class Stack { private:
    char *data_; int top_;
public:
    // Initialization
    Stack(): data_(new char[100]), top_(-1) {}
    // De-Initialization
    ~Stack() { delete[] data_; }
    int empty() { return (top_ == -1); }
    void push(char x) { data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";
    for (int i=0; i<5; ++i) s.push(str[i]);
    while (!s.empty()) {
        cout << s.top(); s.pop();
    }
    return 0;
}
```

Using vector

```
#include <iostream>
#include <vector>
using namespace std;
class Stack { private:
    vector<char> data_; int top_;
public:
    // Initialization
    Stack(): top_(-1) { data_.resize(100); }
    // De-Initialization
    ~Stack() {};
    int empty() { return (top_ == -1); }
    void push(char x) { data_[++top_] = x; }
    void pop() { --top_; }
    char top() { return data_[top_]; }
};
int main() {
    Stack s; char str[10] = "ABCDE";
    for (int i=0; i<5; ++i) s.push(str[i]);
    while (!s.empty()) {
        cout << s.top(); s.pop();
    }
    return 0;
}
```

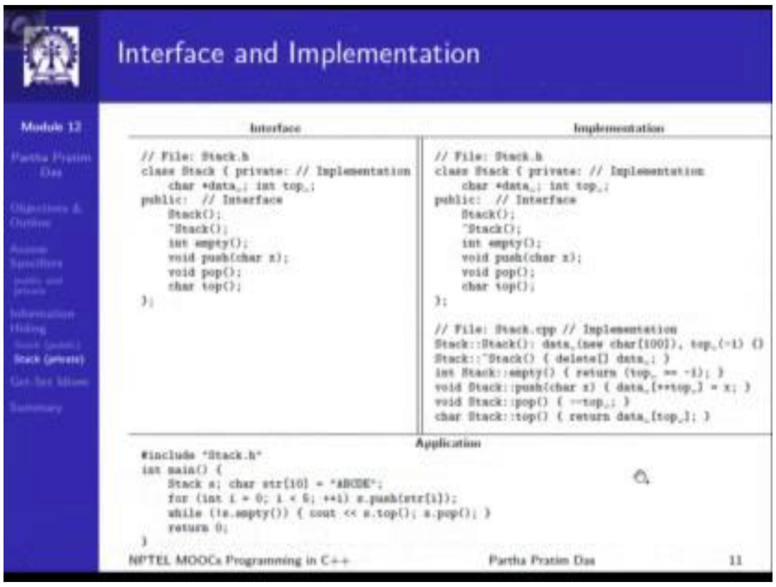
- private data hides the internals of the stack (information hiding)
- Data structure codes contained within itself with initialization and de-initialization
- To switch from array to vector or vice-versa the application needs *no change*
- Application cannot tamper stack – any direct access to top_ or data_ is compilation error!

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

But suppose now we have made, these private and as we have made them private, then certainly we cannot access them and therefore, the application functions become identical no changes can be made. There is no risk of assigning 2 to s dot top as we did last time because if we try to somewhere here we were doing it in the last example. If we try to now write s dot top underscore assign 2 here, then simply I will have a compilation error, the my compiler will tell me that s is of stack type, where top is a private data member. So, from outside the class stack main is outside of the class stack it is a global function with respect to class stack I am not allowed to access s dot top and therefore, such risks will not exits.

So, certainly with this principle of information hiding I can have a much better design, a much better implementation, where the private data the internals are completely hidden data structure is contained all within itself. And if I want to switch from one implementation to the other, from left to right or from right to left my application code will not need to make any change what show ever at all. So, that is the basic principle that we will try to follow in terms of this design.

(Refer Slide Time: 25:45)



Interface and Implementation

Module 12

Partha Pratim Das

Objectives & Outline

Access Specifiers

public and private

Information Hiding

Stack (public)

Stack (private)

Get-Set Idiom

Summary

Interface

```
// File: Stack.h
class Stack { private: // Implementation
    char *data_; int top_;
public:  // Interface
    Stack();
    ~Stack();
    int empty();
    void push(char x);
    void pop();
    char top();
};
```

Implementation

```
// File: Stack.h
class Stack { private: // Implementation
    char *data_; int top_;
public:  // Interface
    Stack();
    ~Stack();
    int empty();
    void push(char x);
    void pop();
    char top();
};

// File: Stack.cpp // Implementation
Stack::Stack(): data_(new char[100]), top_(-1) {}
Stack::~Stack() { delete[] data_; }
int Stack::empty() { return (top_ == -1); }
void Stack::push(char x) { data_[++top_] = x; }
void Stack::pop() { --top_; }
char Stack::top() { return data_[top_]; }
```

Application

```
#include "Stack.h"
int main() {
    Stack s; char str[10] = "ABCDE";
    for (int i = 0; i < 5; ++i) s.push(str[i]);
    while (!s.empty()) { cout << s.top(); s.pop(); }
    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

Now given this let me just briefly discuss, how should the basic interface and implementation be organized. So, here if we look at this is stack class and we will expect that this comes in a header file; where header file is which defines, what the different classes are there in their members and so on. And if you look into this, this part is private, which is part of the implementation and this part is interface, which everybody needs to know. So, if you looking to the implementation again you just have another copy of this and this will be required in the implementation as well as in the application below.

Implementation will need it, implementation means which ever will write the body of this functions the definition of this functions these codes, which certainly go in a separate dot cpp file, that is certainly needs to know what is the name of the class it needs to know, what are different methods and so on.

So, this header has to go here, but to be able to use this header you do not need to know whether it is a data member is allocated dynamically or it is automatically allocated and so on. All that you need to know are that there are different data members like in here which can be used in the push pop empty all this kind of methods.

So, to summarize we will have one header file, which has the class that primarily should show the interface there should be one implementation file for the class that as the codes of all the methods and should include this. So, you can see this is what I am calling stack dot h this is, what I called is stack dot cpp. And then I separate application file whatever the name of the that application file is, which should include the same header file, make sure that it is the same class definition that is used between the implementation and the application this is where you write the application code.

So, this is the basic structure of any good object oriented C++ design that we will try to follow, which perfectly allows us to perform information hiding. You will of course, raise some objection to the fact that the application does get to see that it cannot access the private data members, but that application still gets to see that what private data members I have, so is that knowledge necessary for the application. We will answer that much later when we have studied some bit more of C++ mechanisms, particularly you have studied about inheritance. We will be able to see that actually that two is not necessary and interface can truly just be the public methods that class support. Everything that is private could be defined separately also.

But for now let us just take it that implementation and interface will be separated out a combined in a header file that defines the class data members and the class methods as signature. A class implementation file, which has all the implementation codes of the methods, and the application can include the class header file will include the class header file use the methods and would be able to work out on the application.