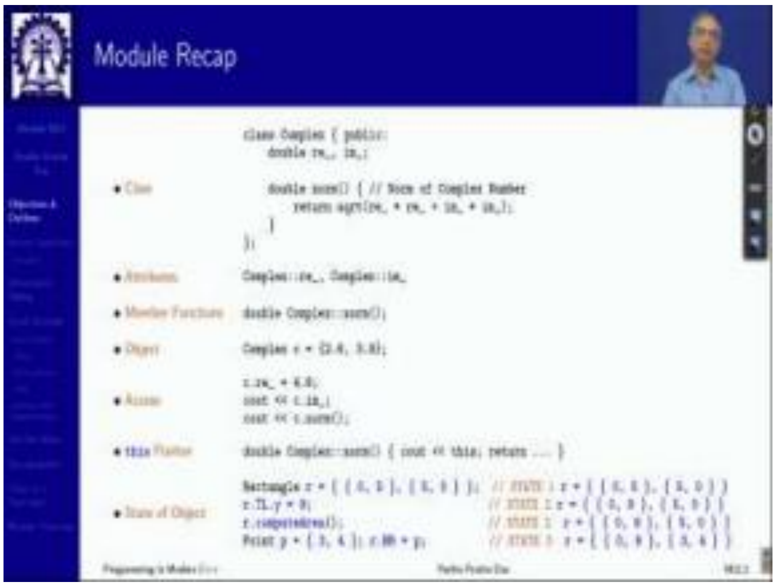
Programming in Modern C++
Professor Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 12
Access Specifiers

Welcome to Programming in Modern C++, we are in week 3, and we will now discuss module 12.

(Refer Slide Time: 00:35)



Module Recap

- Class

```
class Complex { public:
    double re_, im_;

    double norm() { // Norm of Complex Number
        return sqrt(re_ * re_ + im_ * im_);
    }
};
```

- Attributes `Complex::re_, Complex::im_`
- Member Functions `double Complex::norm();`
- Object `Complex c = {2.6, 3.9};`
- Access

```
c.re_ = 4.6;
cout << c.im_;
cout << c.norm();
```

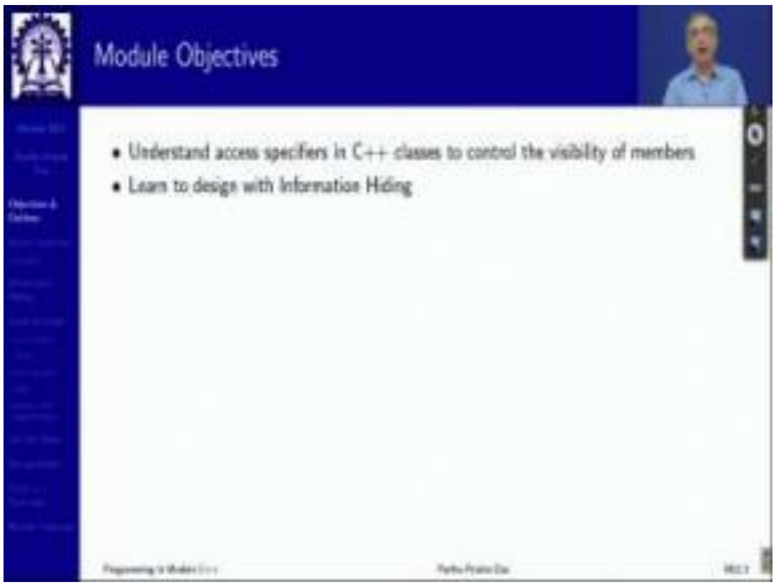
- this Pointer `double Complex::norm() { cout << this; return ... }`
- State of Object

```
Rectangle r = { { 2, 5 }, { 6, 3 } };  // STATE 1 : r = { { 2, 5 }, { 6, 3 } }
r.TL.y = 9;                            // STATE 2 : r = { { 2, 9 }, { 6, 3 } }
r.computeArea();                       // STATE 2 : r = { { 2, 9 }, { 6, 3 } }
Point p = { 3, 4 }; r.BR = p;          // STATE 3 : r = { { 2, 9 }, { 3, 4 } }
```

Programming in Modern C++ Partha Pratim Das M12.2

In the last module, we have introduced all important concepts of class and objects, the foundation of object-oriented programming in C++, we talked about classes, data members, member functions, objects, how to access data members and member functions, we introduced the notion of this pointer and importance of it and we talked about the state of object.

(Refer Slide Time: 01:07)

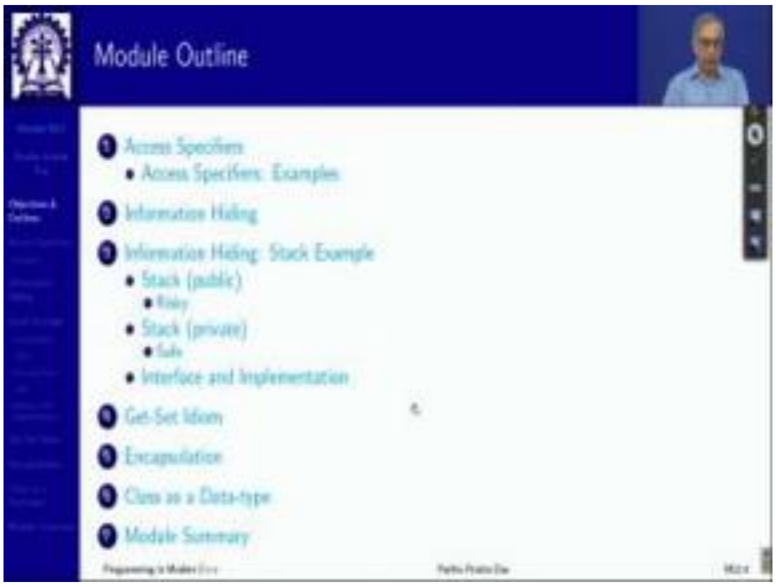


Module Objectives

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

Now, going forward from this module and subsequent to we will go deep into some of these aspects and discuss them specifically. So, in the current module, we will talk about the visibility of members or access specifiers, public and private access specifiers. And learn how these can be used judiciously in designing good classes, which provide the right kind of information hiding.

(Refer Slide Time: 01:37)



Module Outline

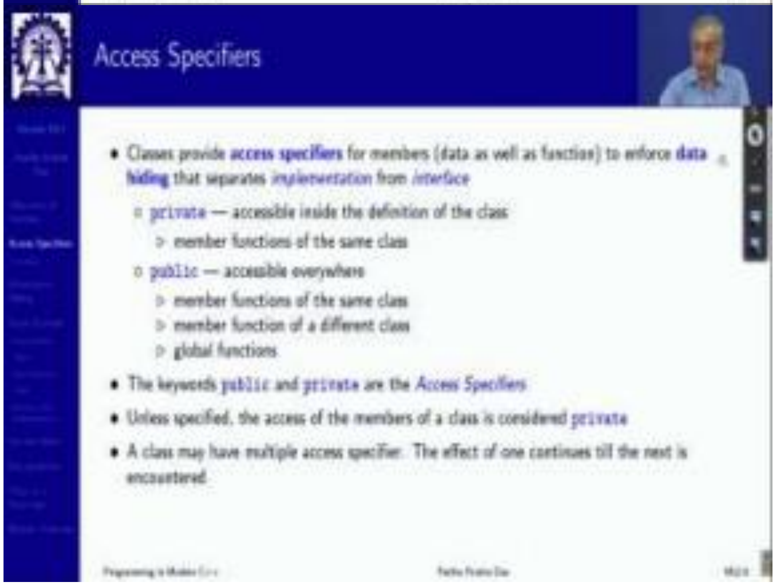
1. Access Specifiers
   - Access Specifiers: Examples
2. Information Hiding
3. Information Hiding: Stack Example
   - Stack (public)
     - Risky
   - Stack (private)
     - Safe
   - Interface and Implementation
4. Get-Set Idiom
5. Encapsulation
6. Class as a Data-type
7. Module Summary

So, this is the outline, will get to see it on left always.
(Refer Slide Time: 01:42)

Access Specifiers

Access Specifiers

Programming in Modern C++ Partha Pratim Das



Access Specifiers

- 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

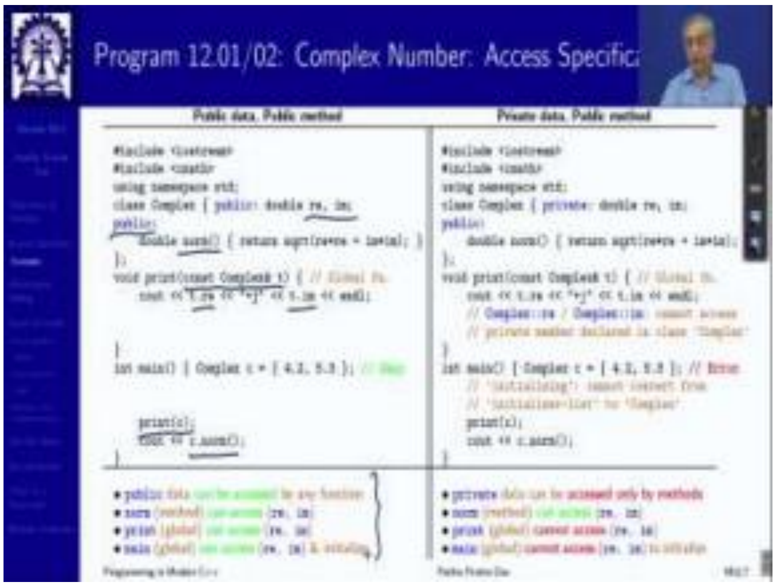
Programming in Modern C++ Partha Pratim Das

So, access specifiers. So, we have two kinds of access specifiers private and public. So, far you have seen only the public access specifier. So, if you specify a data member or a member function with public then you can access it from anywhere, you can access it from the member functions of the same class, you can access it from member functions of different classes, you can access it from any global function as well.

Whereas, if you specify a data member or a member function as private, then as the name suggests, you can only access it within the definition of the class or to put it differently you can access it only from the member functions of the same class and no one else, it is like our bedroom and or our house and the road, road is public, but our houses private, there is access restrictions only the members of the house can enter others cannot.
So, this is something which will be used to enforce data hiding to separate implementation from interface, we will see what does that mean, public and private both are key words and

to be used before the name of the data member or member function.

(Refer Slide Time: 03:12)



Program 12.01/02: Complex Number: Access Specifica

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();
}
```

- 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 / 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();
}
```

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

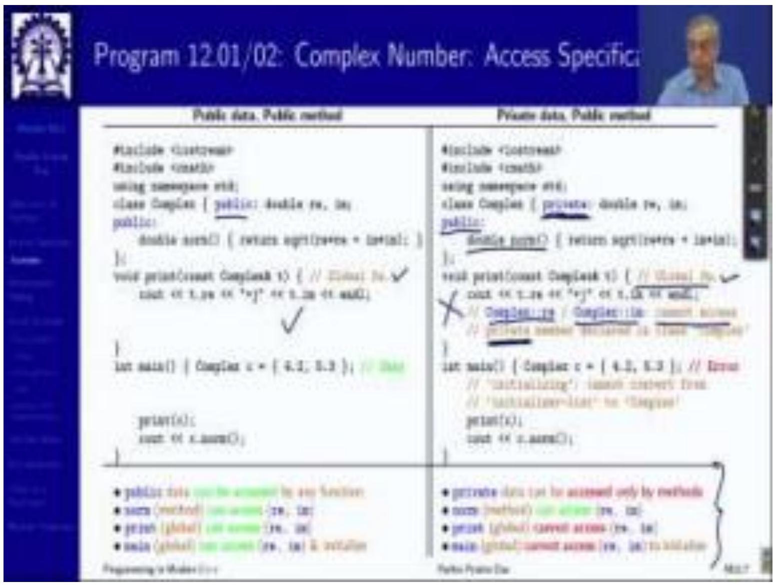
Programming in Modern C++ Partha Pratim Das M12.7

So, the example will clarify things very clearly. So, on the left, we are again we will continue with the Complex example often, Complex, Rectangle and Stack kind of examples. So, here I say that it is public. So, data members can be accessed anywhere. I say that this member function norm() is public, the member function print() is public as well. And I can do the operations.

And just to show you I have shown that print a some other any arbitrary complex number, I am sorry actually print I have put here as a global function you can please I stand corrected that this is the scope of the class. So, norm is a member function whereas, print is a global function. So, print has to take the object t and it can print the members directly because they are public.

So, from outside you can use it. From main I can call print() and get the print. I can also directly call c.norm, c.norm because norm is a member function, but this member function also is public. So, I can, main can directly, main is a global function outside of the class, main can also directly call this. So, this is what is summarized here.

(Refer Slide Time: 05:05)



Program 12.01/02: Complex Number: Access Specifica

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();
}
```

- 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 / 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();
}
```

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

Programming in Modern C++ Partha Pratim Das M12.7

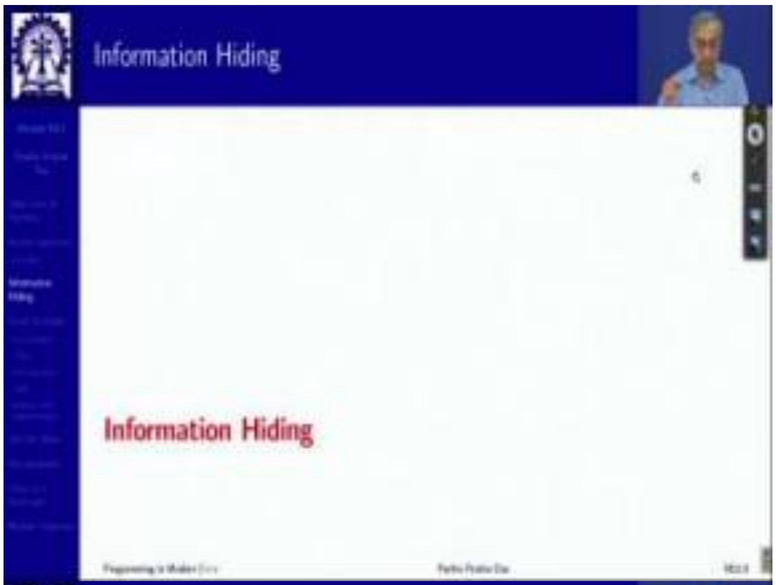
Now, let us change that style with proper access specification. So, what I do is I make the data private, and to say that I will like to protect my data, I do not want to expose my data to any change, because if somebody changes my data member, then that state of the object will change without me knowing it. So, I want to create a situation where every change of state must be known to the object.

And that is possible only if I make the data members private, so that only the member functions of the class can change it and no one else can. I still keep the norm public because people need to use this object and get the norm. So, I do the define the norm() function here. Now, like in this case, I am also I am again trying to define the global function the same code as I do this.

Now, if I tried to compile, I will not be able to compile, it will say that t.re and t.im which are data members complex::re and complex::im, they cannot be accessed, because they are private members in the class. So, while this will work, because this is public, this will not.

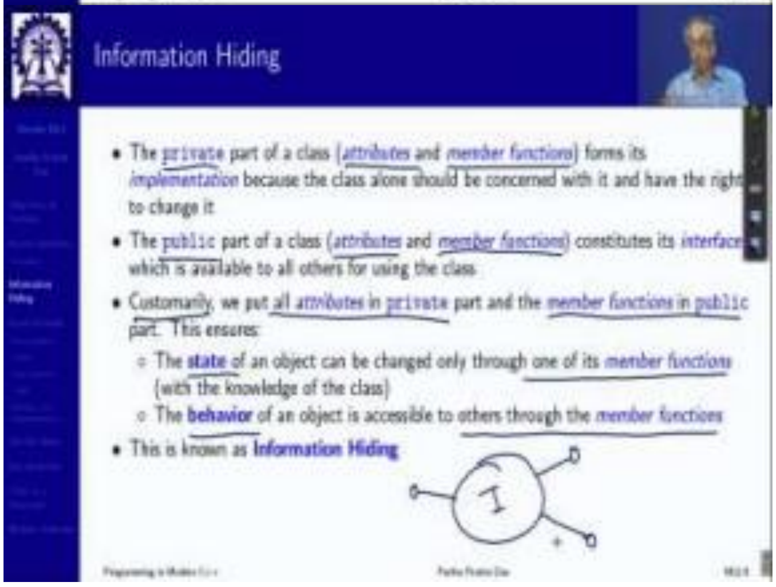
So, this is kind of the protection I am getting by using the private that I have now restricted that, no, nobody cannot just walk in and access my members, though it was not changing the value, it was just trying to read the value, but I have not allowed it by making it private. So, that is the difference between the private and the public. The summary of points are given here.

(Refer Slide Time: 07:11)



Information Hiding

Information Hiding



Information Hiding

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

So, let us see how does this help in information hiding. Information hiding is a key concept of object orientation that you do not want to expose the state of an object, you just want to expose the interface that is the set of, a set of methods by which the object can be used or you also say that these are the services that the object provide to the external world.

So, when I say private members of a class, then whether they are data members or member function, they are necessarily how I have implemented it. And I have a right to change it anytime, as long as I do not change the interface. And what is the interface that is a public part, the public attributes if I choose to keep some data members as public and public member function constitute the interface which is known available to anyone outside this class and can be used to get the service from the class.

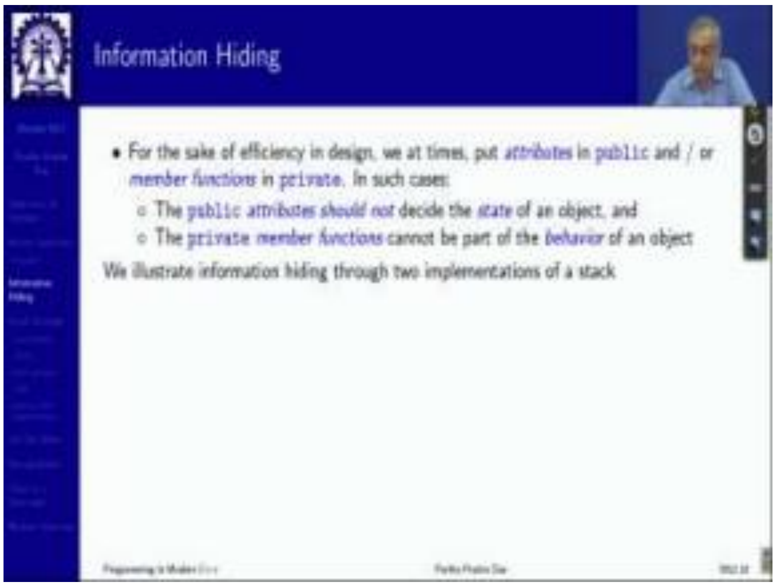
It is customary, not mandatory, but it is customary that all attributes are in private. So, that the state is always protected and member functions are in public. But often you will have some member functions which are also private, because I want to have some utilities that my other member functions use and I do not want to give those utilities as a service to the external world.

Maybe because I just want to have it as an IP or maybe because I might want to change them in future, I might want to change their names and so on. So, I do not want those to be accessed. So, member functions can be public or private in the design. The public part is the interface and the private part are part of the implementation.

And attributes data members preferably should all be private, I mean, there are very, very rare situations where you need a data member really to be public. So, the state of an object can only be changed through one of its member functions. So, when that member function is called it is called on the object.

So, if it can be changed only by member functions then every time a change is done, the object will get to know because member function on the object has been invoked. So, that is the basic advantage. And the behaviour, behaviour is a service that you are giving to others, like the stack is giving the service to push, pop, check for emptiness and get the top marker and so on. So, that behaviour is accessible to others through the member functions, the public member functions.

And this mechanism of separating out implementation, which is here from the interface, these are the different member function interfaces, is the basic notion of information hiding. I mean I am not going into the deeper theoretical aspects of object oriented concepts on hiding and all that, but I am trying to keep it closest to the language so that we can learn the features and learn to use it in a proper way so that we can hide the information separate out the implementation from the interface as we need.
(Refer Slide Time: 11:20)



Information Hiding

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

We illustrate information hiding through two implementations of a stack

So, for the sake of efficiency in design, we at times put attributes in public or member functions in private and or I have already mentioned this. So, when you put an attribute in public, you have to make sure that in your design, that value of that member function does not decide the state of the object, because you have otherwise your information hiding properties lost.

But you could have, we will subsequently see examples, we could have some data members, which you want to have maybe give it as a convenience to use. For example, a simple thing many libraries provide in their class design is they provide a public data member where the user can keep users own data, user can create a packet of data and put a pointer to that.

So that whenever the user is using that object, user will also get its related data. Now, you have nothing to do with that related data, you never considered that pointer to be a part of your object state, but it helps the user to carry on its computation or carry on its information from one point of access to the object to the next point have access to the object.

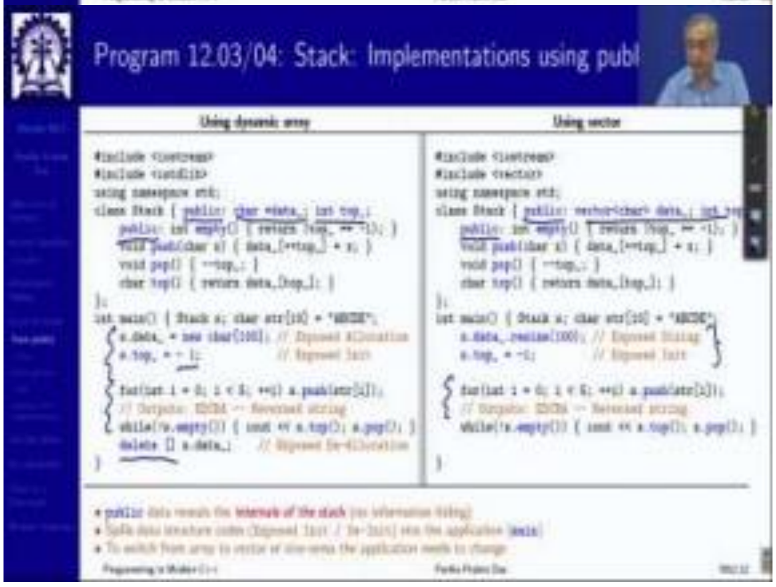
Because it does not need to programmatically remember that in this object, I have this associated data, in that other object I have that associated data because you are having that pointer within the object itself. So, that kind of situations we will need it, but remember, that it should never form a part of the state.
And in the same manner, naturally, it is a compiler restriction by itself. Private member functions can never be part of the behaviour because they cannot be accessed from outside. They are just for your own convenience.

(Refer Slide Time: 13:24)

Information Hiding: Stack Example

Information Hiding: Stack Example

Programming in Modern C++ Partha Pratim Das [illegible]



Program 12.03/04: Stack: Implementations using publ

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 Allocation
    s.top_ = - 1;            // Exposed Init

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

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 Sizing
    s.top_ = -1;         // Exposed Init

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

- 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

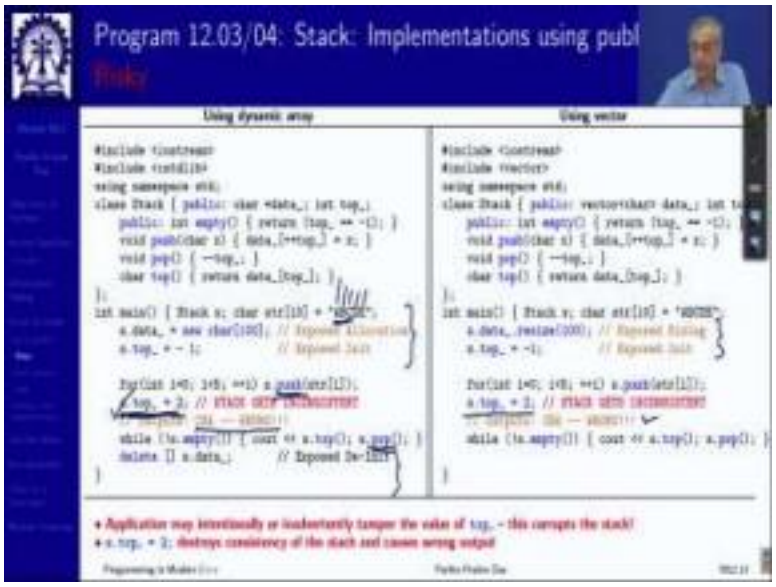
Programming in Modern C++ Partha Pratim Das [illegible]

So, let us look at Stack at length in terms of information hiding. So, usually, I mean initially let us say that everything is public. So, this is public, the data members, two types of implementations using a dynamic array or the vector we have seen this before, when we are comparing with the C style and all members member functions are also public. So, when you do that, then you have an exposed allocation to the stack.

So, the user is doing that allocation user is initializing it. Similarly, in terms of vector, you have an exposed sizing of the vector and expose the initialization of the top. Then you do your basic program requirement, which is here we have assumed it is reversing a string. And then when you are done, you are releasing the memory here because you are dynamically allocated it.

In case of vector there is nothing like that will exist because it is an automatic object it will destroy itself. It will release the memory itself. So, we will talk about those things later. So, the public data now reveals the internals of the stack all the time. So, it is a deep risk, because the programmer who is using this stack class could intentionally or inadvertently change any value in the array, it can change top and I mean the whole concept of stack could disappear altogether.

(Refer Slide Time: 15:26)



Program 12.03/04: Stack: Implementations using publ

Risky

Using dynamic array	Using vector

```
#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 Allocation
    s.top_ = - 1;            // Exposed Init

    for(int i=0; i<5; ++i) s.push(str[i]);
    s.top_ = 2; // STACK GETS INCONSISTENT
    // Outputs: CBA -- WRONG!!!
    while (!s.empty()) { cout << s.top(); s.pop(); }
    delete [] s.data_;      // Exposed De-Init
}
```

```
#include <iostream>
#include <vector>
using namespace std;
class Stack { public: vector<char> data_; int t
    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 Sizing
    s.top_ = -1;         // Exposed Init

    for(int i=0; i<5; ++i) s.push(str[i]);
    s.top_ = 2; // STACK GETS INCONSISTENT
    // Outputs: CBA -- WRONG!!!
    while (!s.empty()) { cout << s.top(); s.pop(); }
}
```

- 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

Programming in Modern C++ | Partha Pratim Das | M12.13

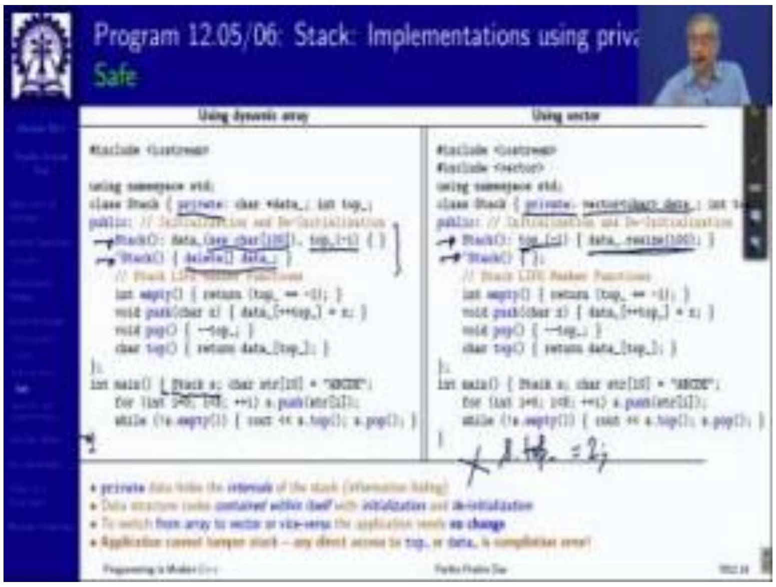
So, here is the risk. So, you can note that I have put in red that this is risky to do. So, do not do this role. So, this is all that we had, this allocation is the risk exposed in the initialization is a risk and the same thing here. Now, what the user has done possibly inadvertently, possibly because he did not know or intentionally, he has put 2 to stack top.

After pushing everything, after pushing ABCD, so after pushing ABCD what will happen, this goes to 0, 1, 2, 3, 4. So, the value of top should be 5. And now, it says it is 2, which means that it has now what reduced to just the string AB. So, you get an output which is very, very I mean it is true. So, it gets up to ABC and you get this similar thing has been done here. Now, this kind of risk is of paramount importance.

In this case it is a small example might look pathological but in when your program starts getting bigger, if this kind of change of state can be done by the user directly without the knowledge of the stack. So, here what has happened is state has changed, when you have changed, when you change it through push or through pop the object knows.

And therefore, it adjusts top accordingly, when you do that is you are changing the state one by putting an element or considering that an element is not there, but what it is also doing is it is adjusting the top accordingly. Here by exposing it, you have let the user change the state and the user does not know the entire logic and therefore, the user has changed it to an inconsistent state. So, that is a big, big risk, we can never allow this to happen.

(Refer Slide Time: 17:48)



Program 12.05/06: Stack: Implementations using priva
Safe

Using dynamic array	Using vector

```
#include <iostream>

using namespace std;
class Stack { private: char *data_; int top_;
public: // Initialization and De-Initialization
    Stack(): data_(new char[100]), top_(-1) { }
    ~Stack() { delete[] data_; }
    // Stack LIFO Member Functions
    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(); }
}
```

```
#include <iostream>
#include <vector>
using namespace std;
class Stack { private: vector<char> data_; int t
public: // Initialization and De-Initialization
    Stack(): top_(-1) { data_.resize(100); }
    ~Stack() { };
    // Stack LIFO Member Functions
    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(); }
}
    s.top_ = 2;
```

- 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!

Programming in Modern C++ Partha Pratim Das [illegible]

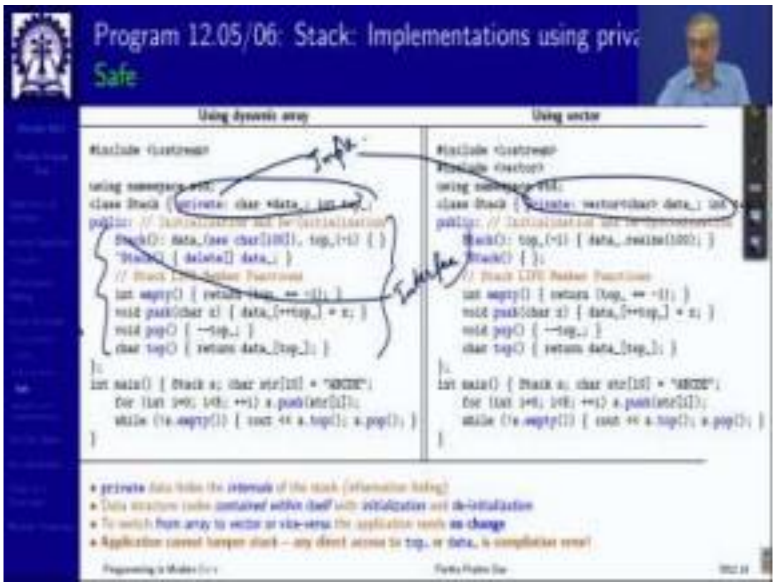
And mind you in C this is always possible, because everything is global. So, that is the risk that we are trying to get out of and come to a safe scenario of design. So, what we are doing here is I have defined answer, I have defined two special functions, we will talk about this functions, but just to give you the idea, this first function is called a constructor which initializes the data members because you are starting down.

So, at the very beginning when you say Stack s this constructor gets called, and it does the initialization job, you can see that it is initializing with allocation and it is initializing the top. The other function is called the destructor. So, when this particular object is no more available at this point, the scope is ending here.

So, after this point, s is no more available at this point, this deinitializing destructor function will be called to do the necessary release, it will happen in an automatic manner. You can see that in the case of vector it will do in the constructor, it will set tops and it will resize to the value that you actually want, in the destructor there is nothing to do. So, this is encapsulating the initialization and deinitializing.

But the most important thing here to note is the data members have been made private. So, it is no more possible that in main somebody will write s.top, not possible. This is now a compilation error. So, what does that mean, these two together mean?

(Refer Slide Time: 20:12)



Program 12.05/06: Stack: Implementations using priva
Safe

Using dynamic array

```
#include <iostream>

using namespace std;
class Stack { private: char *data_; int top_;
public: // Initialization and De-Initialization
    Stack(): data_(new char[100]), top_(-1) { }
    ~Stack() { delete[] data_; }
    // Stack LIFO Member Functions
    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(); }
}
```

Using vector

```
#include <iostream>
#include <vector>
using namespace std;
class Stack { private: vector<char> data_; int top_;
public: // Initialization and De-Initialization
    Stack(): top_(-1) { data_.resize(100); }
    ~Stack() { };
    // Stack LIFO Member Functions
    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(); }
}
```

- 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!

One is, since data members are private, and only the member functions are public, you can use the stack exactly the way you want you need. But you cannot intentionally or inadvertently change the state of the stack to make it inconsistent without the knowledge of these member functions. So that is the safe way to actually do the design. And this is just to recap this part private.

This part is what we will say is the implementation and this part is my interface. Implementation is private, interface is public. Anybody can use the interface, make use of the stack, solve problems, but they will not be allowed to touch my data members and change anything to make things go wrong. This is a huge safety through this notion of information hiding that C++ provides over C programming.
(Refer Slide Time: 21:42)

You could not have done this in C. So, to formally specify this is my interface. So, I am showing little jumping to show you another style, where in the class, which I will put in a header file, I put the data members, put them as private. And for the member functions, I have just put the prototype, just a signature, not the actual implementation, as I was actual code of the body, as I was doing.

Then I will have another file, which I call the source file or the implementation file, where I include this, so this gets to know the class, it knows that stack is a class. And then I actually write the code for each one of them. I told already, that class gives you a namespace. So, if I am calling a function empty() in the class stack, then outside that class, globally, its name is stack::empty. So, now, I have already defined the class, and I am writing this code outside of it, I am not writing in C to within the class.

So, I need to refer to every member as by the class name, so that it is fully qualified. Note that I do not need to do that here inside the body. Because, as I say, Stack::push, the compiler knows that this is the push() function of the Stack class. And it knows the Stack class. So, within push, when I write, top, it knows it is the top member of the class Stack. So, but I need to identify the function that I am writing the code for.
(Refer Slide Time: 23:54)

Now, so if I do that, then I will say that this is my interface. Because anybody who uses the stack needs to use this. And this is my implementation, because nobody needs to know how I am doing it, whether I am using a dynamically created array or I am using a vector or whatever I may be doing is my choice, I can change that any time also without changing the behaviour of the stack at all. So, this is the implementation which gives the state and this is the interface, which is the behaviour.

So, you can see that the application finally, which is in yet another file, I have the use that simply a Stack s, knowing the Stack s will be able to call the constructor, do the which we will actually call here, the construction process will get done, the object will get ready with the array dynamically created and the top initialized to -1, then as I do push, it will again knows the interface.

So, it is calling as if here, which actually calls this function which makes changes to the state to push the element to the Stack s keeps on doing that, similar things can be said about they, all these. And when it comes at the end of the scope, the compiler knows that s cannot be used anymore, it again calls the destructor which calls this cleans up because I know how to clean up.

For example, the risk of giving it to the user is I may have created it by new or by malloc. So, I have to accordingly release it. So, all those are taken care of in my implementation, which the user does not need to know. So, actually, I do not need to provide the source code for Stack.cpp to the user. I can just give the object file, Stack.o or Stack.obj.
I need to give this header which is interface and the user will be able to write the application. My implementation remains with me interface is with the application programmer who uses it and gets all the service, information is perfectly hidden. So, that is the basic design scenario which we will always have to keep in mind.

(Refer Slide Time: 26:51)

Now, the question is, which now data members in this case, case of stack, I never actually required to know the value of the data member, because all that I need is a data structure, but in many times like I keep the information about employees, students, their name, their date of birth, employee ID, salary all of that.

So, I would need to give access to the interface because everything is private, if everything is private, then nobody can read the data. So, I have to provide specific interface public member function to let the application program read or write or make changes to the data members at needed.

So, there can obviously be different situations, some data members is both read write, like the case of Complex you saw, you want to, you will need to change that whenever you are making an assignment to the complex you are adding to complex numbers putting that. Usually, this kind of data members you need them to be read only, like Date of Birth does not keep on changing or employee ID, roll number does not keep on changing, they are kind of they should not need a change.

Some are write only, sounds a little funny, but it is true, for example, password you can only write you cannot read your password because it is, when we give the password is encrypted and written somewhere and you cannot, you are just to write that this is the original password they had, you ever not expected to take it out in any way.

Then some may be invisible, that is you do not want the user to even know that these exists, like the top, the data in stack, you will want the user to use the stack button. So, there are, so both read write, only read, only write, none of them are the possibilities. And using what is known as the get set idiom is a very easy and in a common way to control them what you do is very simple.

If you want to read you give a method to read which returns that value, if you want to write give it a set method. So is there a get method and a set method. So, here you return the value here you set the value. Now it is your choice, any data member which you want to make read only, you do not give a set method or if we wanted to write only do not give a get on that. Something which you do not want to be accessed like in top and data you do not provide any methods on the data members to be accessed.
(Refer Slide Time: 29:53)

So, with access specification these kinds of design considerations can be put in so that you can very easily control exactly how much of the state should be known to the user and how much the user should be able to directly change and how much you will keep in control. So, these are obviously the different get set choices that you have.

(Refer Slide Time: 30:19)

So, here is for an Employee class, you can go through that and understand that on the name we have given read and write both address we said okay, once the address is registered, it cannot be changed, salary can be read only to know but there is a variable component in the salary possibly bonus which you do not even want to tell that it exists. So, it is, there is no get set method and that accordingly this design has been done, I would suggest that you go through this executed and get to know better.

(Refer Slide Time: 30:56)

So, this all together, the class has first provided an encapsulation which is an aggregation simply putting things together and with access specifiers and information hiding, you get proper encapsulation, you have encapsulated the data and maybe some methods within the private specifiers and you are exposed the interface which is to the public specifiers.

(Refer Slide Time: 31:31)

So, you get to achieve the basic objectives of data modeling through object-oriented design. So, with that, before I close, I will just try to point out and I will keep on enhancing this is how is with this, the class is becoming closer to being user define type. For example, these are some of the things you can do on a built-in type you can declare and you can do the same thing here.

You can initialize, you can do the same thing here. You can print, you can do that by component wise printing. Of course, that means that your data will have to be exposed. Or if you do not want to expose, then you can define a print method which does it, which is in the interface, but actual data members are not known.
Or as we will see that, we will be able to overload the output streaming operator to write a cout instruction, exactly the same way you write it for the built-in data types, we will learn that as well. So, it is the same. For example, we can add two integers.

Similarly, I can overload I can provide a add method or even better, I can overload the operator+ to do the add in the same way. So, we can see that in parallel, I mean, if we just replace int by complex everything still holds, we can make the data type in that way. So, that is the reason we often say that class is actually data type.

(Refer Slide Time: 33:16)

So, on top of the classes and objects and basic data member functions we had learned earlier. Now, we have learned the whole game of how to play around with that with the proper information hiding and encapsulation. Thank you very much for your attention. We would meet in the next module.