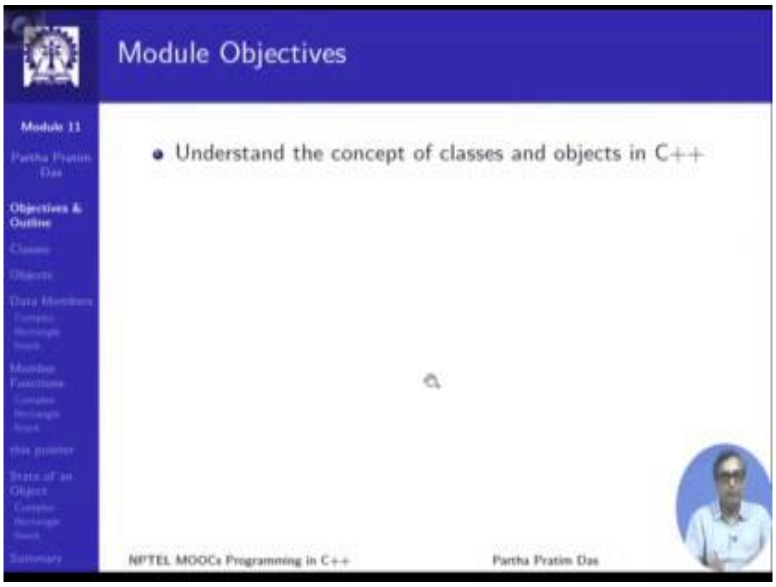
Programming in C++

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

Lecture – 19
Classes and Objects

Welcome to module 11 of Programming in C++. We have already taken a loop into the procedural extensions of C++ over modules 5, 6 through 10. And we have looked at different better C features. Now from this module onwards, we will start discussing the core object oriented features of C++, the concept of classes and objects.

(Refer Slide Time: 00:59)



Module Objectives

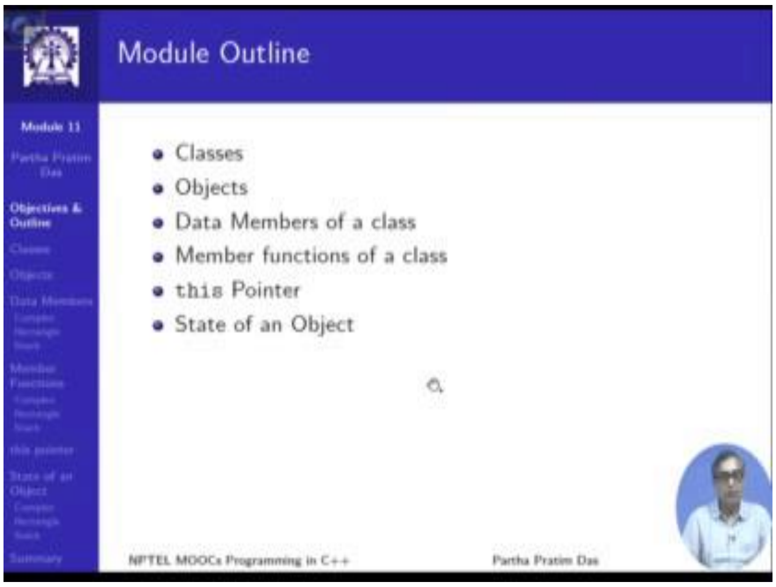
- Understand the concept of classes and objects in C++

Module 11
Partha Pratim Das
Objectives & Outline
Classes
Objects
Data Members
Complex
Rectangle
Stack
Member Functions
Complex
Rectangle
Stack
this pointer
State of an Object
Complex
Rectangle
Stack
Summary

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, the objective of this module is to understand the concepts of class and objects in C++.

(Refer Slide Time: 01:07)



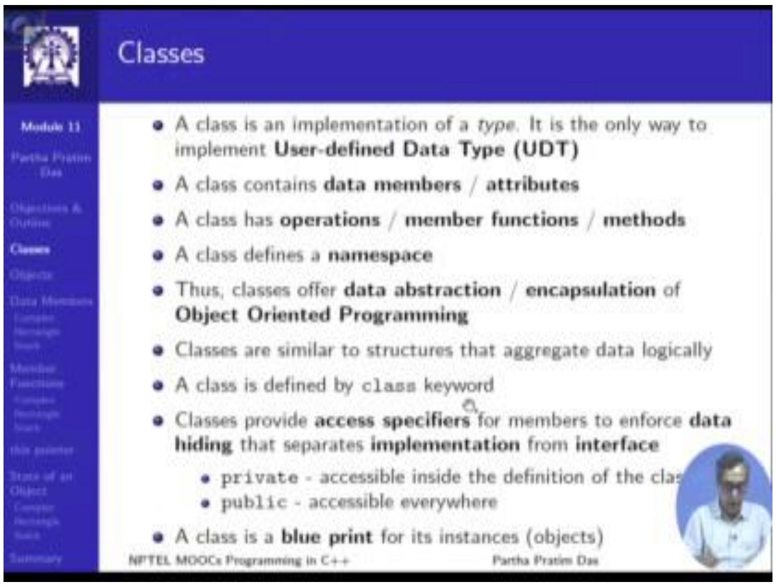
Module Outline

- Classes
- Objects
- Data Members of a class
- Member functions of a class
- `this` Pointer
- State of an Object

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, these are the items that we will walk through.

(Refer Slide Time: 01:12)



Classes

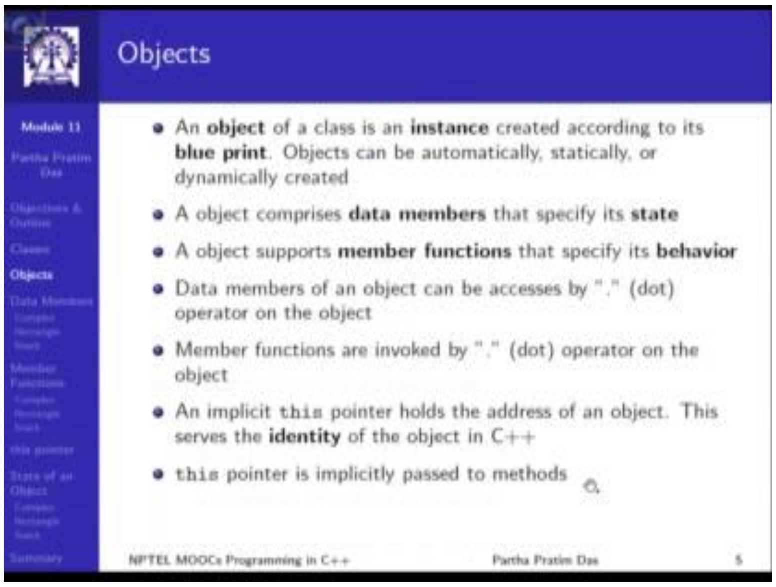
- A class is an implementation of a *type*. It is the only way to implement **User-defined Data Type (UDT)**
- A class contains **data members** / **attributes**
- A class has **operations** / **member functions** / **methods**
- A class defines a **namespace**
- Thus, classes offer **data abstraction** / **encapsulation** of **Object Oriented Programming**
- Classes are similar to structures that aggregate data logically
- A class is defined by `class` keyword
- Classes provide **access specifiers** for members to enforce **data hiding** that separates **implementation** from **interface**
  - `private` - accessible inside the definition of the clas
  - `public` - accessible everywhere
- A class is a **blue print** for its instances (objects)

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, let me first give you a very basic overview of what is a class, and what is an object. We will slowly demonstrate, illustrate these through example, so that each and every point will become clear. As we say is a class is an implementation of a type. A statement which will look or sound somewhat new to you totally because so far; so far as C was concerned, we either had built in types or we had types derived from the built in type like arrays like structure or like pointers. But, now we will be in a position to implement a user defined data type all by ourselves and that will be one of the major lessons that we will try to take from the current module and the next couple of modules.

As we will see a class will contain data members or attributes, a class will have operations member functions or methods; these are just alternate names of the same thing. We will see a class defines a name space that is once I define a class name, it becomes a surrounding property for all the data members and methods names that it contains. And in that way, a class will offer the data abstraction or the so called encapsulation of object oriented programming.

(Refer Slide Time: 03:24)



Module 11

Partha Pratim Das

Objectives & Outline
Classes
Objects
Data Members
Complex
Rectangle
Stack
Member Functions
Complex
Rectangle
Stack
this pointer
State of an Object
Complex
Rectangle
Stack
Summary

Objects

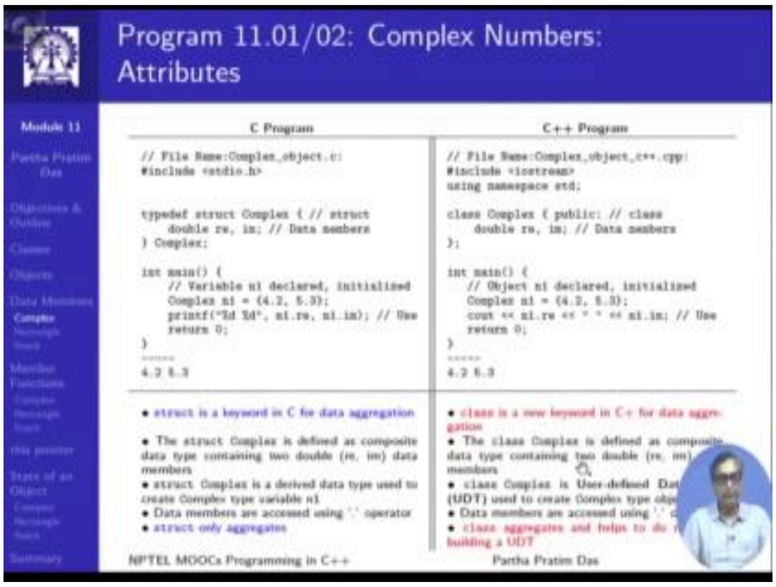
- An **object** of a class is an **instance** created according to its **blue print**. Objects can be automatically, statically, or dynamically created
- A object comprises **data members** that specify its **state**
- A object supports **member functions** that specify its **behavior**
- Data members of an object can be accesses by "." (dot) operator on the object
- Member functions are invoked by "." (dot) operator on the object
- An implicit `this` pointer holds the address of an object. This serves the **identity** of the object in C++
- `this` pointer is implicitly passed to methods

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

In terms of parallel with C programming, you can say that class is similar to structures that also aggregate data logically, but we will show how classes become different. To define class, C++ introduces a new keyword by the name class, and classes have different access specified; and finally, a class is a blue print and it can be instantiated for objects. So, objects are basically instances of a class; so given one class, I can have multiple instances of that class a class tells us the blue print, tells us the format in which the data and the method should be organised for an object

And every object will have a separate identity; it will have its own values of the data members that will specify the state in which the object will reside. It will support member functions, which will define the behaviour that it can offer; given an object will be able to use the dot operator, the one that we had seen in terms of accessing components of structures will be able to use the same operated to access data members as well as methods of the object. And in addition an object will have a special pointers know as a this pointer; and this pointer will get implicitly passed to each and every method. This is just a very brief overview of what classes and objects are. I am sure at this very beginning it may not be making a whole lot of sense to you all. So, we will start with an example, and slowly illustrate each of these points over the next couple of slides.

(Refer Slide Time: 04:48)



Program 11.01/02: Complex Numbers: Attributes

Module 11

Partha Pratim Das

Objectives & Outline

Classes

Objects

Data Members

Complex

Rectangle

Stack

Member Functions

Complex

Rectangle

Stack

this pointer

State of an Object

Complex

Rectangle

Stack

Summary

C Program

```
// File Name:Complex_object.c:
#include <stdio.h>

typedef struct Complex { // struct
    double re, im; // Data members
} Complex;

int main() {
    // Variable n1 declared, initialized
    Complex n1 = {4.2, 5.3};
    printf("%d %d", n1.re, n1.im); // Use
    return 0;
}
-----
4.2 5.3
```

- struct is a keyword in C for data aggregation
- The struct Complex is defined as composite data type containing two double (re, im) data members
- struct Complex is a derived data type used to create Complex type variable n1
- Data members are accessed using '.' operator
- struct only aggregates

C++ Program

```
// File Name:Complex_object_c++.cpp:
#include <iostream>
using namespace std;

class Complex { public: // class
    double re, im; // Data members
};

int main() {
    // Object n1 declared, initialized
    Complex n1 = {4.2, 5.3};
    cout << n1.re << " " << n1.im; // Use
    return 0;
}
-----
4.2 5.3
```

- class is a new keyword in C++ for data aggregation
- The class Complex is defined as composite data type containing two double (re, im) members
- class Complex is User-defined Da (UDT) used to create Complex type obj
- Data members are accessed using '.' o
- class aggregates and helps to do building a UDT

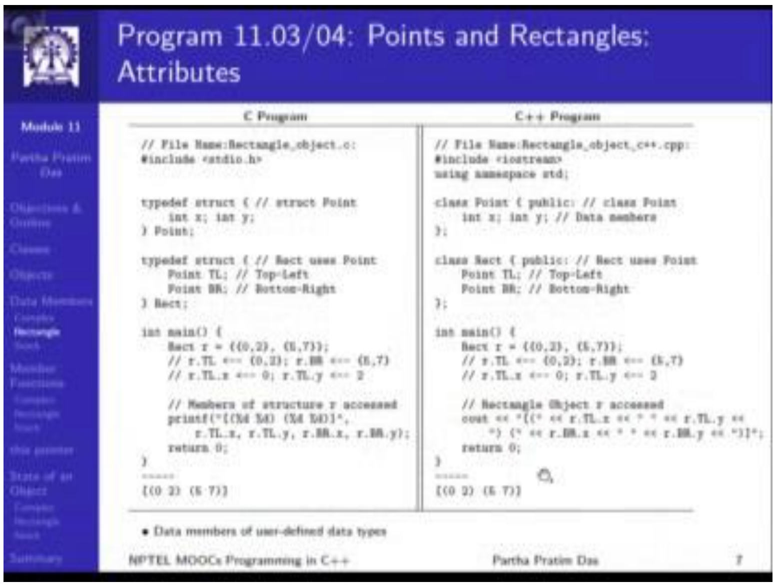
NPTEL MOOCs Programming in C++ Partha Pratim Das

So, let us consider a simple definition of a complex number. We had a taken this example earlier also, where we can actually use a structure which has two data members, two members re and im designating the two components of a complex number and that is defined as a structure and we typedef as earliest that to be the name complex. So, once we had done that then we can define variables of that structure, and put some initial values to this component of the structure. So, re will become 4.2, and im will become 5.3. So, if we now print the two components of this complex number in one, it will print out as 4.2 and 5.3. This is what you already know; this is what is available in C.

If I want to write this very similarly in C++, I will change and write this as class complex. So, we were doing a struct complex and then we are doing a aliasing with the typedef, now I simply write a class complex and put the member definitions within that. And rest of the code can be written in a very similar manner only difference being here we are using printf here we are using C out as we have seen in case of the C++ we use the streaming operators for doing this.

Now, let us see what difference it has actually made. So, struct is a keyword in C, class is a new keyword in C++. Rest of what we have done are very similar between the two approaches of defining and aggregation of two components of the real component and the imaginary component of a complex number, either through a struct in C or through a class in C++. So, the only contrast that we will slowly bring out now is while struct will allow us to only aggregate put the two components together refer to them together as we are referring to n 1 as a as a pair of two double numbers 4.2 and 5.3 designating a complex the class also does the same thing. But class we will see will do a lot more things than what struct can do.

(Refer Slide Time: 07:42)



Program 11.03/04: Points and Rectangles: Attributes

C Program

```
// File Name:Rectangle_object.c:
#include <stdio.h>

typedef struct { // struct Point
    int x; int y;
} Point;

typedef struct { // Rect uses Point
    Point TL; // Top-Left
    Point BR; // Bottom-Right
} Rect;

int main() {
    Rect r = {{0,2}, {5,7}};
    // r.TL <-- {0,2}; r.BR <-- {5,7}
    // r.TL.x <-- 0; r.TL.y <-- 2

    // Members of structure r accessed
    printf("[(%d %d) (%d %d)]",
        r.TL.x, r.TL.y, r.BR.x, r.BR.y);
    return 0;
}
-----
[(0 2) (5 7)]
```

C++ Program

```
// File Name:Rectangle_object_c++.cpp:
#include <iostream>
using namespace std;

class Point { public: // class Point
    int x; int y; // Data members
};

class Rect { public: // Rect uses Point
    Point TL; // Top-Left
    Point BR; // Bottom-Right
};

int main() {
    Rect r = {{0,2}, {5,7}};
    // r.TL <-- {0,2}; r.BR <-- {5,7}
    // r.TL.x <-- 0; r.TL.y <-- 2

    // Rectangle Object r accessed
    cout << "[(" << r.TL.x << " " << r.TL.y <<
        ") (" << r.BR.x << " " << r.BR.y << ")]";
    return 0;
}
-----
[(0 2) (5 7)]
```

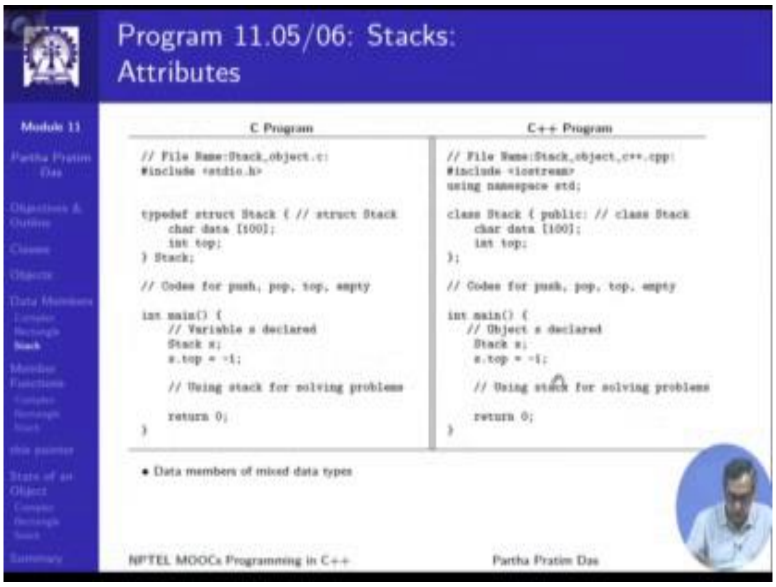
- Data members of user-defined data types

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

So, let us take a little different and bigger example. Here we are trying to define a rectangle and this is a special kind of rectangle which will says is isocratic rectangle in the sense that this rectangle is has its size parallel to the axis x and y-axis. So, if I just specify with two corners diagonally opposite corner, the top left and the bottom right corners of the rectangle, then the rectangle is fully specified. For doing this, first we define a structure which gives us points as an aggregation of two coordinates x and y and then we take two points, and let them designate that two diagonally opposite points of a rectangle. And once this has been done then I can define a rectangle here, rectangle r. You can see the notation the 0, 2 here first within the first pair of curly braces mean the point top left, 5, 7 mean the point bottom right. And both of them together mean the whole rectangle.

So, in the comments, you can see that actually if I specify 0, 2 as the first point, then I am actually specifying r being the name of the rectangle r dot t l is a top left point dot x is x coordinate of that which gets the value 0. Similarly, r dot t l dot y, which is a y component of the top left point, gets the value 2. So, through this initialization all these values are being set and then I can print them out. Similarly, I can write the whole thing using class. The difference is being exactly as it was before. I can initialize it in the similar way and I can print it out using C out. So, this is just showing another example of using the class here the data member in rect - the rectangle have actually not of any basic type, but they are user defined data types, they are class objects themselves, they are instances of the point class that I have already defined earlier.

(Refer Slide Time: 10:19)



Program 11.05/06: Stacks: Attributes

C Program

```
// File Name:Stack_object.c:
#include <stdio.h>


typedef struct Stack { // struct Stack
    char data [100];
    int top;
} Stack;

// Codes for push, pop, top, empty

int main() {
    // Variable s declared
    Stack s;
    s.top = -1;

    // Using stack for solving problems

    return 0;
}
```

C++ Program

```
// File Name:Stack_object_c++.cpp:
#include <iostream>
using namespace std;

class Stack { public: // class Stack
    char data [100];
    int top;
};

// Codes for push, pop, top, empty

int main() {
   // Object s declared
    Stack s;
    s.top = -1;

    // Using stack for solving problems

    return 0;
}
```

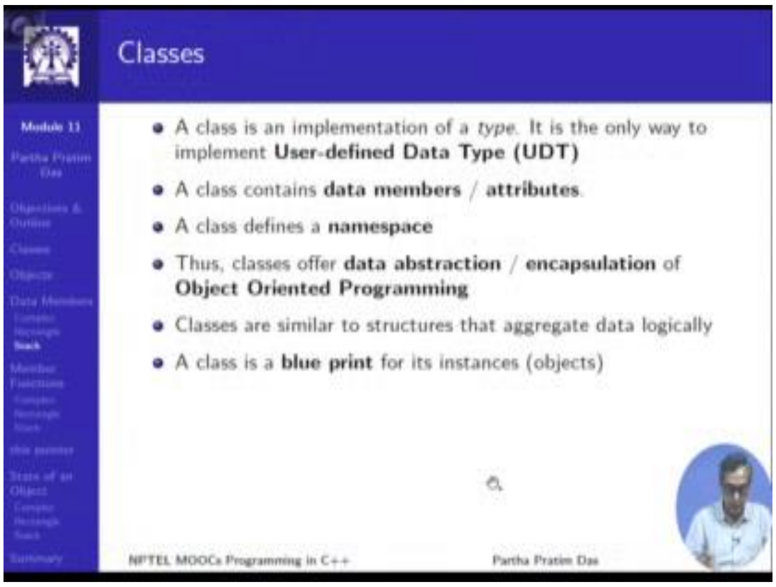
- Data members of mixed data types

NPTEL MOOCs Programming in C++ Partha Pratim Das

Let us take third example of a stack, which is something we have been discussing early too. So, in the stack, we have a mix combination of mixed data types we have a character array to keep the elements of the stack it is the stack of characters we have a top marker which is an index on the stack which shows where the top actually exists. So, I can define a stack variable here or here; and obviously, before I start using the stack, I need to make sure that the stack is empty which is designated by the top index being minus 1. So, I initialize the top index to minus 1, and then I can use the stack either whether it is define through struct or is define through class by using different algorithms for solving different problems.

So, these are different examples, where you can see that we show that the basic components of a class that is a class as a class keyword, it has a name which is the name and identifier, it has a number of data members. Each data member is defined as in struct as a variable declaration style and gives in the name of the class, we can declare variables of the class types these are known as instances. So, this is an instance or this is what is called an object. So, s is an object of the class stack. And once we have that then with that object, we can use the data member, the data member here is top using the dot member notation. So, we first show that a class is an aggregate which can put together one or more data members, and allows us to instantiate the objects of that class or define variables of that type using the C++ mechanism.

(Refer Slide Time: 12:41)

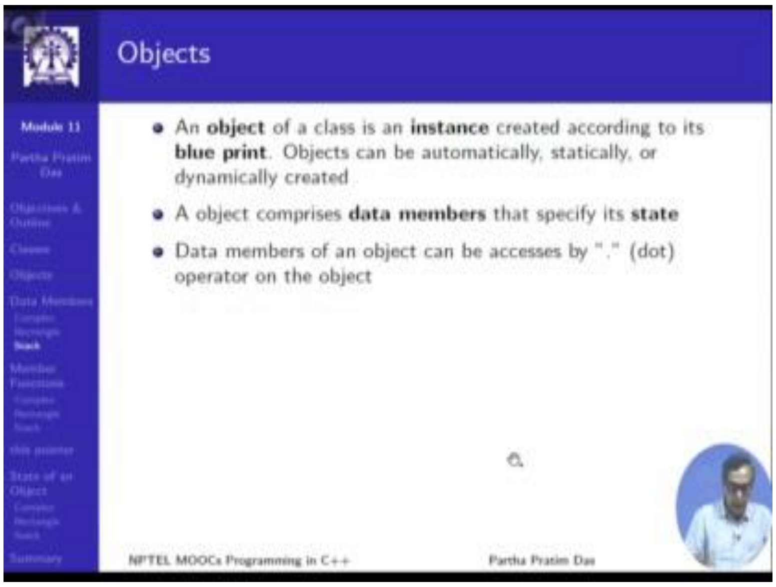


Classes

- A class is an implementation of a *type*. It is the only way to implement **User-defined Data Type (UDT)**
- A class contains **data members** / **attributes**.
- A class defines a **namespace**
- Thus, classes offer **data abstraction** / **encapsulation** of **Object Oriented Programming**
- Classes are similar to structures that aggregate data logically
- A class is a **blue print** for its instances (objects)

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, just to recap, we have seen that class is the implementation of a type we will see this more. We have seen three attempts to do three types a complex type a rectangle and point type and a stack type. We have shown that the class will contain data members, it defines a name space that is when I say that my I am defining a complex, my all data members of complex actually have a name, which is qualified by the complex the class name itself. And it is aggregating the data logically.



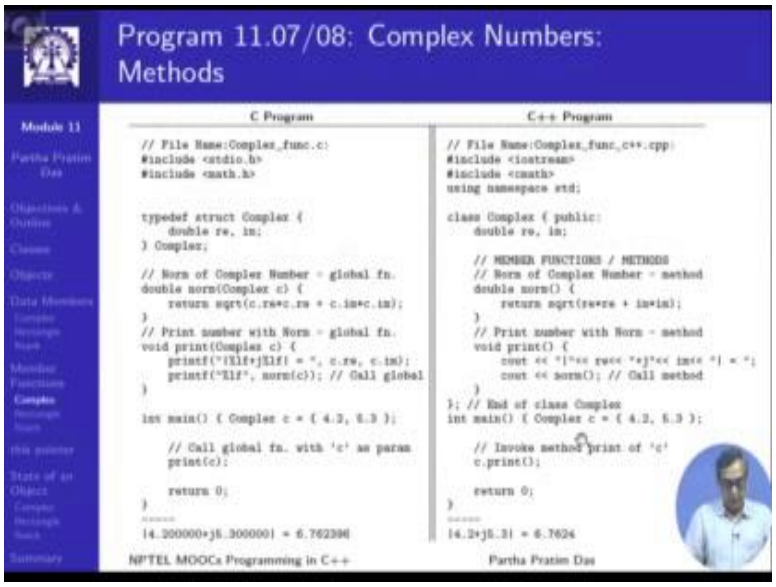
Objects

- An **object** of a class is an **instance** created according to its **blue print**. Objects can be automatically, statically, or dynamically created
- A object comprises **data members** that specify its **state**
- Data members of an object can be accesses by "." (dot) operator on the object

NPTEL MOOCs Programming in C++ Partha Pratim Das

In terms of objects instances, we have shown that for every type of class every three classes that we have defined we can define in different instances of those or objects of those and data members are accessed through the dot operations.

(Refer Slide Time: 13:43)



Program 11.07/08: Complex Numbers: Methods

C Program

```
// File Name:Complex_func.c:
#include <stdio.h>
#include <math.h>


typedef struct Complex {
    double re, im;
} Complex;

// Norm of Complex Number - global fn.
double norm(Complex c) {
    return sqrt(c.re*c.re + c.im*c.im);
}
// Print number with Norm - global fn.
void print(Complex c) {
    printf("|%lf+j%lf| = ", c.re, c.im);
    printf("%lf", norm(c)); // Call global
}

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

    // Call global fn. with 'c' as param
    print(c);

    return 0;
}
-----
|4.200000+j5.300000| = 6.762396
```

C++ Program

```
// File Name:Complex_func_c++.cpp:
#include <iostream>
#include <cmath>
using namespace std;

class Complex { public:
    double re, im;

    // MEMBER FUNCTIONS / METHODS
    // Norm of Complex Number - method
    double norm() {
        return sqrt(re*re + im*im);
    }
    // Print number with Norm - method
    void print() {
        cout << "|"<< re<< "+j"<< im<< "| = ";
        cout << norm(); // Call method
    }
}; // End of class Complex
int main() { Complex c = { 4.2, 5.3 };

    // Invoke method print of 'c'
    c.print();

    return 0;
}
-----
|4.2+j5.3| = 6.7624
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, so far what we have seen is something which is which is strongly in parallel to what we could do as structures. And now we are going to make the first major distinction or the first major step to move away from what we could do in structure in terms of the class definition. So, please follow this very carefully. We are back to the complex examples. So, this part is common you have the data members here in terms of the structure definition we have the data members the same way in terms class definition. Now, if I have such a complex number in C, I am looking at the left side of the slide, then I can define a number of function say I define a function norm which can takes such a complex number C as in here and define its norm. You all know how the norm is computed this sum of the square of the real and the imaginary parts. And then they take a square root of that sum, you get the norm of the complex number or we can write another function to print the complex number in the real plus j imaginary component kind of notation print it no value and so on.

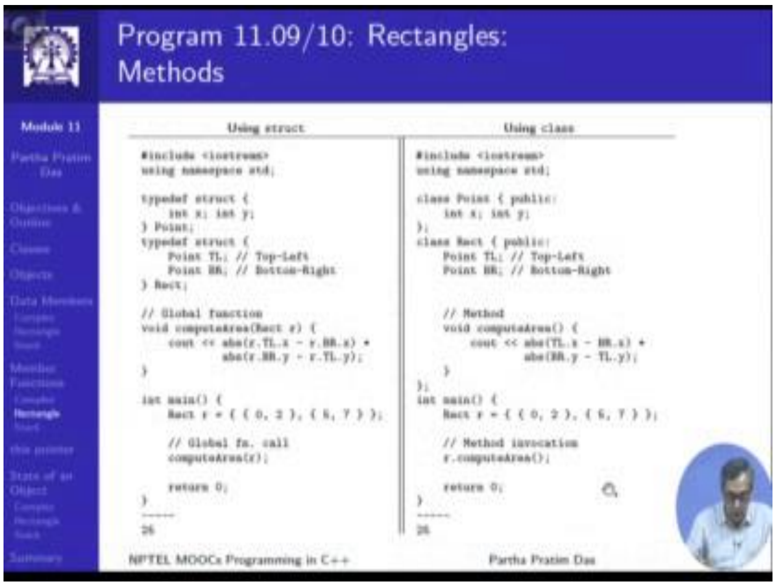
So, these functions can be written along with the structure complex type that I have already defined in C. And these are all as we know are C functions or more commonly global functions, and then I can use to print and if I do that the complex number will be printed out. So, if you just want to take a look as to how it what it prints, this is what it prints. So, given the complex number is 4.2, 5.3, it prints that 4.2 plus j 5.3 the norm of that is 6.7624.

Now, lets us look at the C++ side carefully. Here I am also defining the norm function, but with the difference. In the struct, case of struct, the struct definition is separate, my function definition is separate, but here my class definition, this is my whole of class definition of complex, and my function is a part of the class definition. And such functions will be called quite naturally will be called member functions like re is a member of the class complex, we call it is a data member, similarly double norm this function is also a member of the class complex. And it is called a member function or a method. You can also see that the other function print is also being contained within the definition of the class and print is yet another member function.

So, this is something member function is a completely new concepts for C++ no parallel of this exist in C. And with this member function, now we can say that my object instances that is given the class complex C is an instance of this class, now my object instances, say this instance C can use the method in this notation. Earlier, you had seen the notation in terms of data member only that is C dot re is something we had seen which means I am referring to the re data member of the complex number C. But now I am writing C dot print which means that for the object C for the instance C I am making use or I am invoking the method print. That is method print whatever it is suppose to do, it will do assuming that these data members have the values that the C object has and accordingly it will work, it will print out. Similarly, if I write C dot norm and invoke, this will invoke the other member function norm, and the norm will also behave with the real and imaginary components coming from the value of C.

This is the completely new concepts and this is what the member functions would do. So, the we are contrasting that as in C, if we define something as struct, every operation we need to do with it, we need to be done through some global functions that anybody and everybody can see and use. Whereas in C++, the class itself would have could define a number of member functions or method which the object can invoke as and when necessary to do some operations, and this is what is known as the behaviour of the object as we will see slowly.

(Refer Slide Time: 19:09)



Program 11.09/10: Rectangles: Methods

Module 11

Partha Pratim Das

Objectives & Outline
Classes
Objects
Data Members
Complex
Rectangle
Stack
Member Functions
Complex
Rectangle
Stack
this pointer
State of an Object
Complex
Rectangle
Stack
Summary

Using struct

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

typedef struct {
    int x; int y;
} Point;
typedef struct {
    Point TL; // Top-Left
    Point BR; // Bottom-Right
} Rect;

// Global function
void computeArea(Rect r) {
    cout << abs(r.TL.x - r.BR.x) *
            abs(r.BR.y - r.TL.y);
}

int main() {
    Rect r = { { 0, 2 }, { 5, 7 } };

    // Global fn. call
    computeArea(r);

    return 0;
}
-----
25
```

Using class

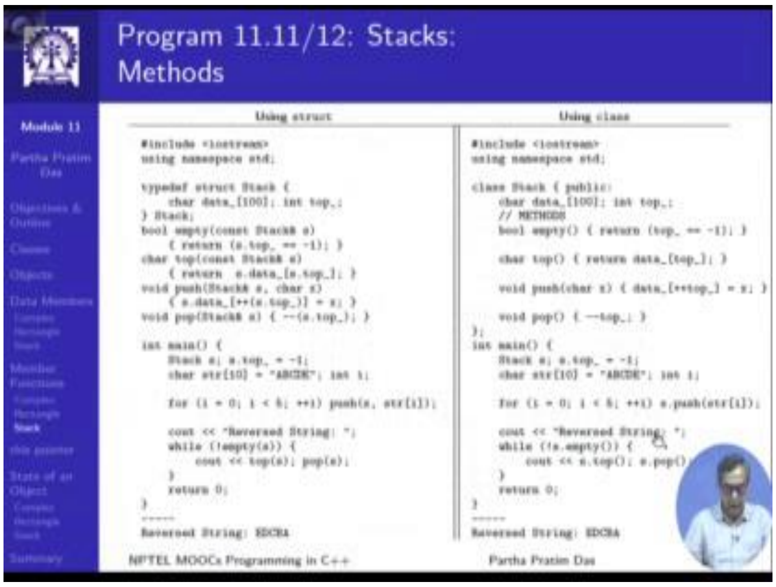
```
#include <iostream>
using namespace std;

class Point { public:
    int x; int y;
};
class Rect { public:
    Point TL; // Top-Left
    Point BR; // Bottom-Right


    // Method
    void computeArea() {
        cout << abs(TL.x - BR.x) *
                abs(BR.y - TL.y);
    }
};
int main() {
    Rect r = { { 0, 2 }, { 5, 7 } };

    // Method invocation
    r.computeArea();

    return 0;
}
-----
25
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, let us look at some more examples. Let us again bring back the rectangle point part, this part you have already seen, this part you have already seen. Here we are writing a function for C using struct this is a global function, which computes the area of the rectangle. The formula for computing the area is straightforward, I will not get deeper into that, but this is a global function compute area, which takes a rectangle as a parameter and computes the area and this is how it is called. In contrast, in the C++ class, this is what my rectangle class, this is what my rectangle class is my method my compute area method is a member functions that is a part of the class.

So, when I have a rectangle r object, I invoke the method or member function as in the same using the same dot notation, and it means that the compute area will work, this will work assuming that r as instantiated here is the object for which it is working. So, when it refers to t l dot x, it actually refers to the t l dot x of the r object, which has invoked this particular method. This is more as in C, we say this is function call, we continue to say it is a function call in C++ as well, but these when you use the member functions of different objects, you often would say that you are invoking the method of an object. A method or a member function, which is available as a part of its class definition, so that is the core idea of methods in C++.

(Refer Slide Time: 21:16)



Program 11.11/12: Stacks: Methods

Module 11

Partha Pratim Das

Objectives & Outline

Classes

Objects

Data Members

Complex

Rectangle

Stack

Member Functions

Complex

Rectangle

Stack

this pointer

State of an Object

Complex

Rectangle

Stack

Summary

Using struct

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

typedef struct Stack {
    char data_[100]; int top_;
} Stack;
bool empty(const Stack& s)
    { return (s.top_ == -1); }
char top(const Stack& s)
    { return  s.data_[s.top_]; }
void push(Stack& s, char x)
    { s.data_[++(s.top_)] = x; }
void pop(Stack& s) { --(s.top_); }

int main() {
    Stack s; s.top_ = -1;
    char str[10] = "ABCDE"; int i;

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

    cout << "Reversed String: ";
    while (!empty(s)) {
        cout << top(s); pop(s);
    }
    return 0;
}
-----
Reversed String: EDCBA
```

Using class

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

class Stack { public:
    char data_[100]; int top_;
    // METHODS
    bool empty() { return (top_ == -1); }

    char top() { return data_[top_]; }

    void push(char x) { data_[++top_] = x; }

    void pop() { --top_; }
};
int main() {
    Stack s; s.top_ = -1;
    char str[10] = "ABCDE"; int i;

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

    cout << "Reversed String: ";
    while (!s.empty()) {
        cout << s.top(); s.pop();
    }
    return 0;
}
-----
Reversed String: EDCBA
```

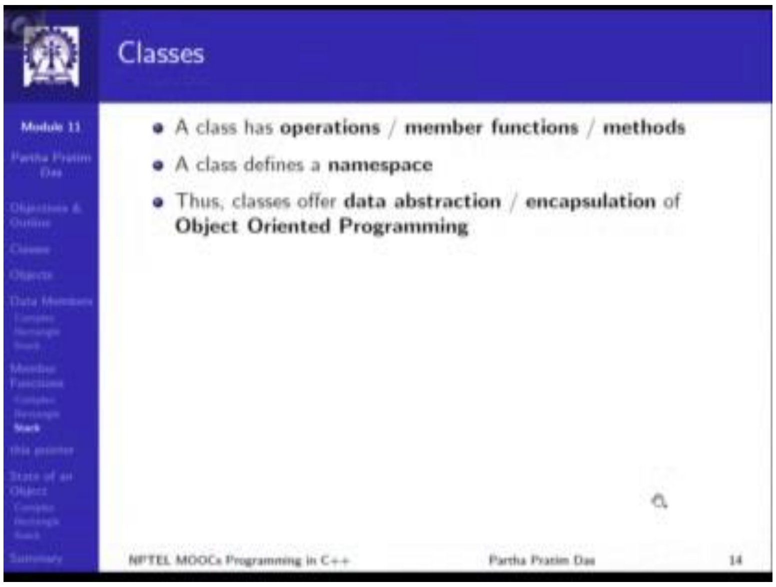
NPTEL MOOCs Programming in C++ Partha Pratim Das

You could take a look. We will not go very detailed into this you could take your time when you study this more, this presentation more. You can see that this is a complete example of a stack which, these has the data. These are the four operations of stack empty top push and pop in C given as global functions. We use the instance of a stack here, initialize it is a top maker use it and for a given string. We push the each and every character of that string one after the other into the stack, and then we keep on popping them till the stack gets empty. As you know this is the standard algorithm to reverse a string. So, this code will reverse the string using this global function.

And now we show that we can do the same thing as we have been showing using class complex, as we have shown using class rectangle. We show that in case of class stack, I could have the data, and I could make all of these stack operations as we need to do them into member functions and method a part of the class. And as soon as we make them as part of the class, then we use them just look at the difference. If you do push here you need to say what is the stack and then what are you pushing here you say the stack is yes. So, you are invoking the push method for the stack object and so you are just saying what you pushing here. Similarly, for checking empty you pass the stack s to the global function empty, here empty is a member function. So, you take for the stack s you simply take the method invoke the method empty which will us this method to find out if the stack s has a top which is equal to minus 1.

Similarly, top s is invoked here as s dot top; pop s is invoked here as s dot top. So, you can see that it is in contrast to using global functions, now this method invocation or member functions are allowing us to put together the data as well as the operations as well as the methods or member all that I need to do with the data put to give them together into one unified bundle. And that is what gives a more proper, more accurate in encapsulation or aggregation as object oriented programming would often ask for.

(Refer Slide Time: 24:03)



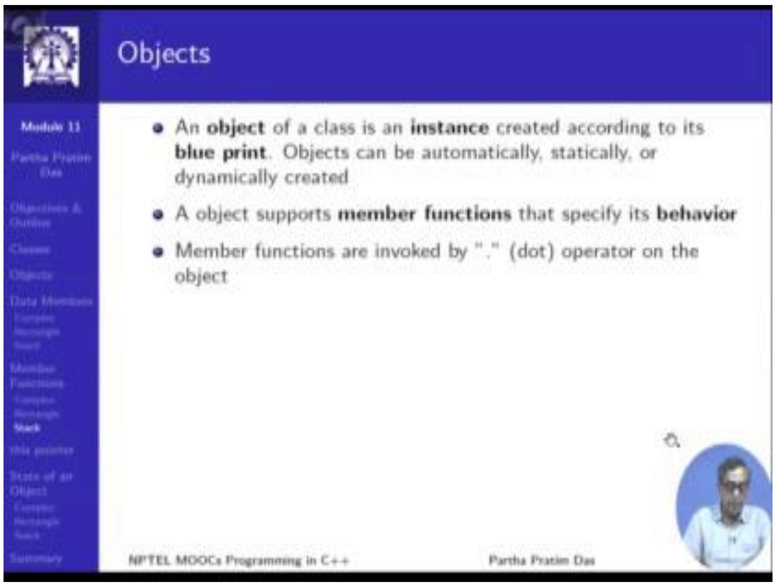
Classes

- A class has **operations** / **member functions** / **methods**
- A class defines a **namespace**
- Thus, classes offer **data abstraction** / **encapsulation** of **Object Oriented Programming**

NPTEL MOOCs Programming in C++ Partha Pratim Das 14

So, with this, what we achieve is we achieve that the class has operations and member functions methods and it offers a data abstraction or encapsulation as in oop will; obviously, have more to understand this.

(Refer Slide Time: 24:20)



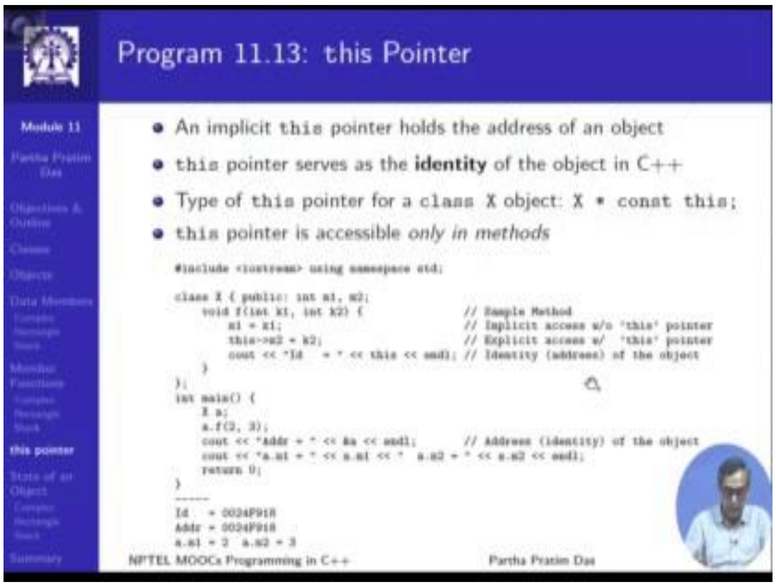
Objects

- An **object** of a class is an **instance** created according to its **blue print**. Objects can be automatically, statically, or dynamically created
- A object supports **member functions** that specify its **behavior**
- Member functions are invoked by "." (dot) operator on the object

NPTEL MOOCs Programming in C++ Partha Pratim Das

And in terms of objects we know that member functions are defining its behaviour; as we have just seen that in stack the member functions like push pop empty top defines all the LIFO behaviour that the stack needs to do. The advantage being that when we use C and we are using global functions, the global functions do not know about the stack structure, the stack structure has no idea about what global functions, I might pass it into. But when I do a class and put them together the data member and the methods that work with the data member are completely closely tied together, so you deal with them necessarily together and we will see particularly after access specification how they become completely tightly bound to give us a more complete data type in C++.

(Refer Slide Time: 25:08)


Program 11.13: this Pointer

Module 11

Partha Pratim Das

Objectives & Outline

Classes

Objects

Data Members
- Complex
- Rectangle
- Stack

Member Functions
- Complex
- Rectangle
- Stack

this pointer

State of an Object
- Complex
- Rectangle
- Stack

Summary

- An implicit `this` pointer holds the address of an object
- `this` pointer serves as the **identity** of the object in C++
- Type of `this` pointer for a `class X` object: `X * const this;`
- `this` pointer is accessible *only in methods*

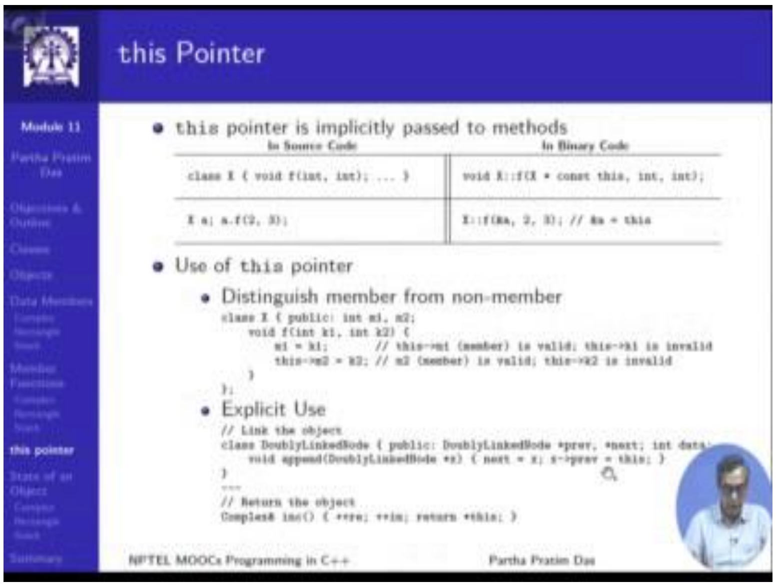
```
#include <iostream> using namespace std;

class X { public: int m1, m2;
    void f(int k1, int k2) {                    // Sample Method
        m1 = k1;                                // Implicit access w/o 'this' pointer
        this->m2 = k2;                          // Explicit access w/  'this' pointer
        cout << "Id   = " << this << endl;      // Identity (address) of the object
    }
};
int main() {
    X a;
    a.f(2, 3);
    cout << "Addr = " << &a << endl;            // Address (identity) of the object
    cout << "a.m1 = " << a.m1 << "  a.m2 = " << a.m2 << endl;
    return 0;
}
-----
Id   = 0024F918
Addr = 0024F918
a.m1 = 2  a.m2 = 3
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, the next concepts that we need to understand introduces here is a concepts of this pointers. This pointer this is actually a keyword in C++, and this is a name. And it is an implicit pointer that holds an address of an object. So, if I am talking about an object, the object can refer to itself its own identity, its own address within the methods of the object can be referred to as this. And this is this pointer has an interesting signature. So, you have already seen the const pointer, concepts earlier, so you can easily read this signature of how this pointer is implicitly defined. So, for a class x, the this pointer of its object will be x star const this, which x star tells it the this is a pointer to a class type of a object and const after this star of the pointer type tells us that this is a constant pointer that is you cannot change the value of this pointer, which is what make sense because we are saying that this is the address of an object. So, it is accessible in terms of different methods.

So, here I just show a couple of examples. This does not do anything fruitfully meaningful, but this is just for illustration that; x is a class, it has two members. And f is one function, which takes two parameters and sets them to two data members of the object. And we can refer to the data member by either directly as m 1 or I can refer to it by this pointer m 2, because the implicitly what it means that if I am talking about a object then I have a pointer which is a pointer to this object. So, when I am in f, I have as this, I have the value of this pointer through which I can refer to m 1 and m 2 in this object. So, if you go through this code, you will able to see that this pointer is actually carrying the address. So, in the main code, we have printed the address; and in the function f, we have printed the value of this pointer and you can see that they are identical. So, this pointer actually carries the address of the object.

(Refer Slide Time: 27:49)



this Pointer

Module 11

Partha Pratim Das

Objectives & Outline

Classes

Objects

Data Members
- Complex
- Rectangle
- Stack

Member Functions
- Complex
- Rectangle
- Stack

this pointer

State of an Object
- Complex
- Rectangle
- Stack

Summary

- `this` pointer is implicitly passed to methods

In Source Code	In Binary Code
`class X { void f(int, int); ... }`	`void X::f(X * const this, int, int);`
`X a; a.f(2, 3);`	`X::f(&a, 2, 3); // &a = this`

- Use of `this` pointer
  - Distinguish member from non-member

```
class X { public: int m1, m2;
    void f(int k1, int k2) {
        m1 = k1;       // this->m1 (member) is valid; this->k1 is invalid
        this->m2 = k2; // m2 (member) is valid; this->k2 is invalid
    }
};
```

  - Explicit Use

```
// Link the object
class DoublyLinkedNode { public: DoublyLinkedNode *prev, *next; int data;
    void append(DoublyLinkedNode *x) { next = x; x->prev = this; }
}
---
// Return the object
Complex& inc() { ++re; ++im; return *this; }
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, in here, I just it is usually optional to use this pointer when you are accessing different data members or methods of a class, but you can use them to distinguish data member from other variables like k 1, k 2 here. But there are some instances, some situations where it becomes very necessary and we have just put in two examples here. For examples, if you have a doubly linked list and you want to put in a node which inserts a node after a given node, which will forward link to the next node and backward link to the earlier node, you will need to use the address of the node that you are putting in. Or if you are returning an object, then you will need to refer to the object that you are returning itself, we will see more examples of these later on. So, I will just like you to try this out, but we will detailed it further when we do further examples on this.

So, with this we have learnt what is the concept of a class and its corresponding object? We have also learnt about data members of a class, and of the object corresponding object. We have learnt about the methods, which can be defined for a class and invoked for an object. And we have seen that every object has an identity, which can be captured in this pointer of the methods of that object, and that carries an address which is the address of the object. At this point, I would also like to just touch up on that in C++, the objects do not have any other separate identity. So, this pointer or the address of the object is taken to be the address everywhere, which is different from what some of the other object oriented languages do.