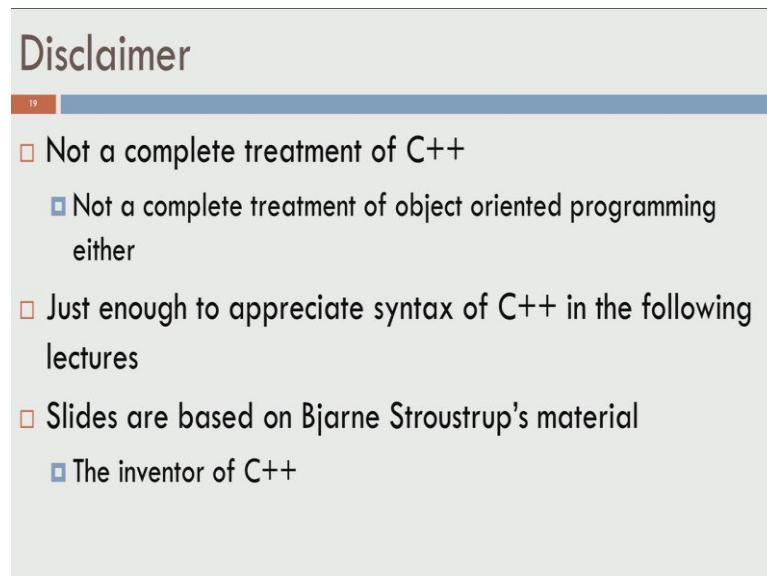


Programming Data Structures and Algorithms
Prof. Shankar Balachandran
Department of Computer Science
Indian Institute of Technology, Madras

Module 12B
Lecture - 41
Brief introduction to C plus plus

Hello, welcome to this module on C plus plus. I promised earlier that we will see a quick introduction to C plus plus and this material is done in such a way that, it will give you just enough material to understand what is happening in the rest of the course. So, I will start with a disclaimer.

(Refer Slide Time: 00:31)



The slide is titled "Disclaimer" and contains a list of four points. The first point is "Not a complete treatment of C++", which has a sub-point "Not a complete treatment of object oriented programming either". The second point is "Just enough to appreciate syntax of C++ in the following lectures". The third point is "Slides are based on Bjarne Stroustrup's material", which has a sub-point "The inventor of C++".

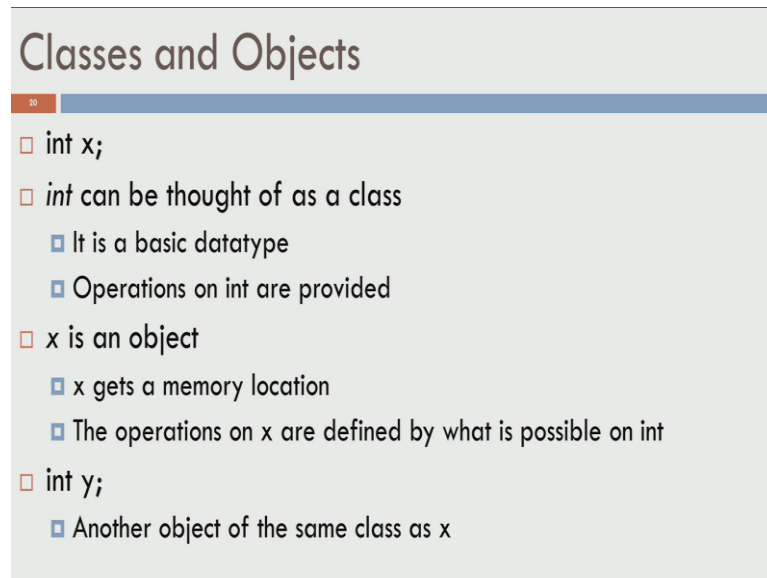
- Not a complete treatment of C++
 - ▣ Not a complete treatment of object oriented programming either
- Just enough to appreciate syntax of C++ in the following lectures
- Slides are based on Bjarne Stroustrup's material
 - ▣ The inventor of C++

This module is not a complete treatment on C plus plus. C plus plus is a vast ocean and this next 15 or 20 minutes is not going to give enough justice to C plus plus. So, it is not complete exhaustive treatment of C plus plus. Also, do not expect a lot of object oriented programming skills, to be learnt in this 15 minute module, right. There is going to be just enough material to appreciate the syntax of C plus plus and by no means, it is complete. So, the basic material is based on Stroustrup's slides. So, he teaches this course on C plus plus and he is actually the inventor of C plus plus. So, in some sense this is from the horse's mouth.

So, let us look at the notion of classes and objects. So, if you are in slightly senior year in your class, in the second year or third year may be have done some C plus plus

programming. So, the notion of classes and objects are probably familiar to you, but I am not going to assume anything like that. So, I am going to assume that you do not know what classes and objects are. Let us start with this basic declaration called `int x`.

(Refer Slide Time: 01:45)



Classes and Objects

- `int x;`
- `int` can be thought of as a class
 - ▣ It is a basic datatype
 - ▣ Operations on `int` are provided
- `x` is an object
 - ▣ `x` gets a memory location
 - ▣ The operations on `x` are defined by what is possible on `int`
- `int y;`
 - ▣ Another object of the same class as `x`

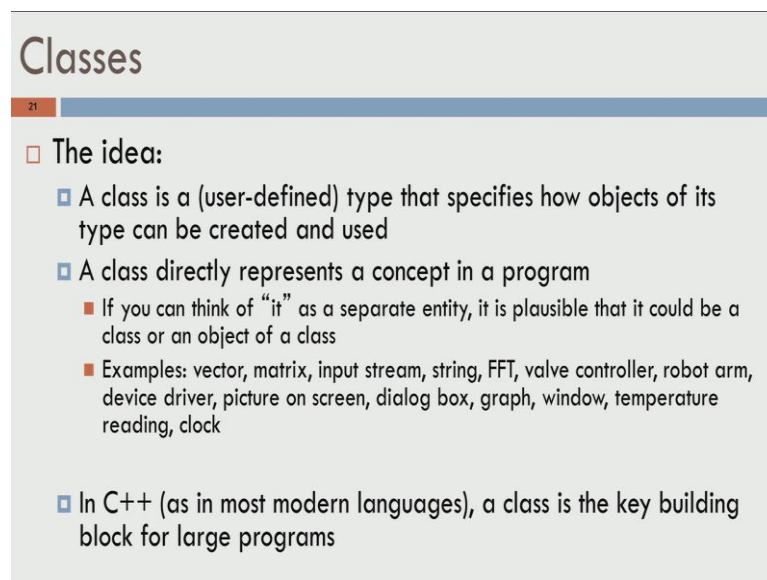
We can think of `int` as a class. So, it is a basic data type and the moment you say something is `int` or an integer, you know that, there are certain operations that you can do on it, and there are certain properties of integers. So, for example, integers can have only values like 1, 2, 3, 0, -1, -2, -3 and so, on. You cannot have a valid value 1.6, right. So, we know some of these properties about `int`. So, any time when you have something called a class, you know that there are certain types. So, there are certain properties of a class and there are only certain operations that you can do on the data type. So, for the basic data types, the language itself tells you that these are the basic set of values that you can take, and these are the valid operations that you can do, right. So, that is the notion of our class..

Here if you look at `x`, `x` is actually an object, right. So, what we mean by an object is, it is not something abstract when I say integer, right. When I say `int`, I am describing only the properties that `int` integers have and the operations. Only when I say `int x`, we get a location allocated to it and on this location, we can do various operations. So, the operations of whatever on `x`, whatever operations are allowed, only those operations that are allowed on integers. So, `x` is an object of the type integer. So, the class name is `int` and the object name is `x`. I can have more objects of the same type. So, for example, when I say `int y`, `y` is another object of the same type integer. So, at some level you can

think of types as classes and actual variables are objects. So, here y is another object of the data type integer. So, this is all for basic data types. So, for basic data types, you really do not need classes and objects this distinction is not really necessary because the language itself gives you all of that. So, the notion of operations that you do on integers and so, on also is already given by the language, whereas, if you have a user defined data type, then it becomes useful and necessary to define, what is allowed and what is not allowed, and to define what operations can be done.

So, let us look at the basic idea of classes. A class is actually a user defined data type that specifies how objects of its type can be created and used, how do you create an object of a certain class, and how to use it is described in the class. So, it directly represents some kind of a concept in a program.

(Refer Slide Time: 04:36)



Classes

- **The idea:**
 - A class is a (user-defined) type that specifies how objects of its type can be created and used
 - A class directly represents a concept in a program
 - If you can think of “it” as a separate entity, it is plausible that it could be a class or an object of a class
 - Examples: vector, matrix, input stream, string, FFT, valve controller, robot arm, device driver, picture on screen, dialog box, graph, window, temperature reading, clock
- In C++ (as in most modern languages), a class is the key building block for large programs

So, whenever you think of it. So, this notion of it as an entity, it is plausible that it is a class or it is an object. So, let see things like this. So, I have the notion of a vector, a matrix, a string and so, on. So, these are probably classes, whereas, I have a matrix which contains all my student’s names and let say records their marks and so, on. Then, that is a specific object. I may have a matrix which contains all these integers and I want to do multiplication of integers and so, on. That is also a matrix, but it is a matrix of integers. So, the object offers particular class. So, it is a physical entity or it is an abstract notion, and C plus plus gives you this basic notion of defining classes. You can define what the basic data structure is or the data type is, and you can also have instances of that particular data type and classes form basic building block for building alley large scale

program. So, let us take a small example here.

(Refer Slide Time: 05:55)

Members and member access

22

- One way of looking at a class;

```
class X { // this class' name is X
//data members (they store information)
//function members (they do things, using the information)
};
```
- Example

```
class X {
public:
int m; // data member
int mf (int v) { int old = m; m=v; return old; } // function member
};

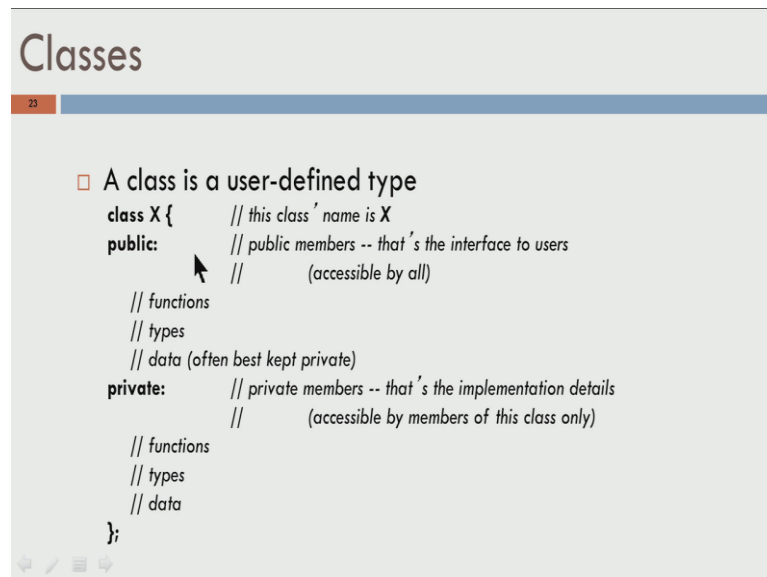
X var; // var is a variable of type X
var.m = 7; // access var's data member m
int x = var.mf(9); // call var's member function mf()
```

We have class x followed by left and right flower braces and a semicolon, and we are going to put in data members and function members inside this. So, the data members are supposed to store the information and the function members are supposed to tell you how to manipulate the data members, right. So, this resembles how you did it for structures. So, there also we had something called struct type x and followed by braces, we had members inside. Only difference is that we not only have data members here, we also have function members. So, let us see a small example here, Class x has int m which is a data member and int mf of int v which is a function or a method. So, we have a basic data type inside it and we have a method inside it. So, that is a class. So, x is a class and it has a variable m or the member m and it has a function mf. So, this function mf is supposed to take an integer v, and it takes the value of v, puts it in m, but the old value that was there before the function is called is supposed to be returned. So, that is the functions member. So, this is something called get and set.

So, you are getting the previous value which is stored, but you are also setting it to the new value that you supply. So, v is the new value that you supply, and the old value of m is supposed to be retrieved. So, let say that is the class description and we will get to this notion of public in a little while. So, let see how this class can be used. We have x var. So, remember x is a data type. So, instead of class x var which we did for structures, right wherever we had structures, we said struct x and so, on. For classes you do not have to do that in C plus plus. X var. So, var is an object of the type x or the class x. Now, if

you want to access the member `m` in it, we can say `var dot m`. So, this is just like the structure, right. We saw this before, right. `var dot m is 7` will change the data member to 7. Interestingly you can do something like this. We can do `var dot mf` of 9, right. What this will do is, we will take the value 9 and supply that to this class. So, if the previous value that contains 7, you will get this small case `x` to be the value 7, but you set it to 9. So, here if you go and print `var dot m` after this line, this would have the value 9. So, basically the nice thing about this class is that we have the member's access with dot `m`; the methods are also accessed with dot. So, both the methods and the data members are accessed using dot operator.

(Refer Slide Time: 09:19)



Classes

23

- A class is a user-defined type

```
class X {           // this class' name is X
public:            // public members -- that's the interface to users
                  // (accessible by all)
    // functions
    // types
    // data (often best kept private)
private:         // private members -- that's the implementation details
                // (accessible by members of this class only)
    // functions
    // types
    // data
};
```

So, now let us go and look at what is this notion of public versus private. So, again if you are exposed to C plus plus, you probably know this already, but in C plus plus you can have members and functions that are of two types, public or private. So, if it is public, these members can be accessed by anyone outside whereas, if it is private, then both. If a member is private, only the class's methods can access it. If a function is private, right, only this function cannot be called from outside. So, private members are supposed to have implementation details that are hidden from the outside world, and public methods and members are directly accessible from the outside world. We will see specific examples in a little while.

(Refer Slide Time: 10:13)

Struct and class

24

- Class members are private by default:

```
class X {  
    int mf();  
    // ...  
};
```
- Means

```
class X {  
private:  
    int mf();  
    // ...  
};
```
- So

```
X x;           // variable x of type X  
int y = x.mf(); // error: mf is private (i.e., inaccessible)
```

So, what are the difference between struct and a class? So, let see class x. Let us say I put int mf. By default all members are private in a class. So, if I did something like this class x int mf, it means that mf is a method that you cannot call from outside. So, let say I did class x, object x and y is x dot mf. This would be invalid because mf is a private method. It cannot be called from the external world, whereas, if this had been a structure, right struct x int m, by default that means, it is actually equivalent to our class x with m being a public variable and structs are primarily used for data structures, where the members can take any value whereas, you use classes whenever you want something hidden from the programmer. So, the programmer should get clean interfaces like insert, delete and so, on, and whatever manipulation is happening internally need not be exposed to the program.

(Refer Slide Time: 11:30)

The slide is titled "Structs" and shows a C code example for a `Date` struct. The code includes comments and function definitions. On the right side, there is a table representing the state of the `my_birthday` struct:

my_birthday: y	Date:
	1950
m	12
d	30

```
// simple Date
//   guarantee initialization with constructor
//   provide some notational convenience
struct Date {
    int y,m,d; // year, month, day
    Date(int y, int m, int d); // constructor: check for valid date and initialize
    void add_day(int n); // increase the Date by n days
};

// ...
Date my_birthday; // error: my_birthday not initialized
Date my_birthday(12, 30, 1950); // oops! Runtime error
Date my_day(1950, 12, 30); // ok
my_birthday.add_day(2); // January 1, 1951
my_birthday.m = 14; // ouch! (now my_day is a bad date)
```

So, let see a small example which explains this in a little more detail. Let us say I have a struct call date which has year, month and day and I have date my birthday. So, let say it is tracking a birthday. If you do this, there is nothing which will stop you from writing something like this. My birthday dot y is 12, my birthday dot m is 30, my birthday dot d is 1950. So, if you look at the right side, they are all integers and the left side data types are all integers. So, you are setting y to 12, m to 30 and d to 1950. So, as three it is separate integers used, assign three separate values which is all, but there is a problem. If we go and deal with this as a date, there is a small problem. You are looking at year 12, maybe it is 12 AD, but the month is 30 and the day is 1950. This does not make sense..

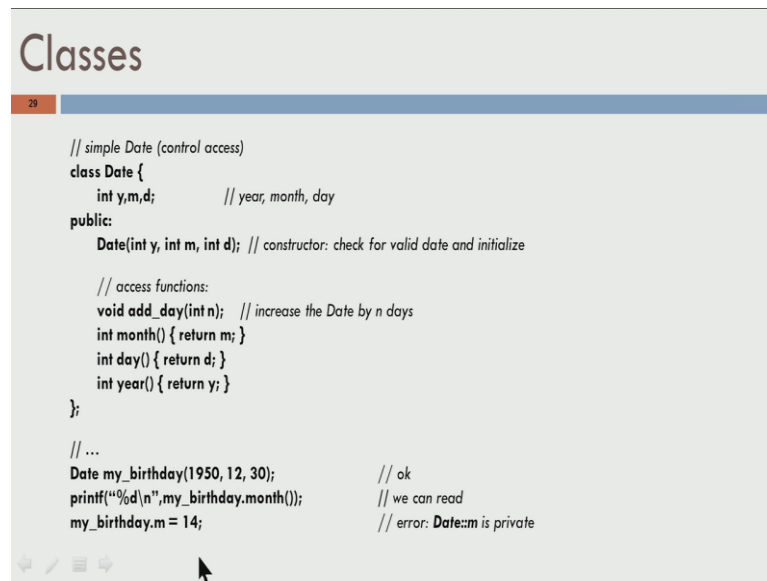
So, maybe it was a mistake. It should have been year 1950 and day must have been 30 and probably the month must have been 12, where we are talking about 30th December1950, right. So, maybe it was a mistake, but if you have a structure from the outside, by mistake if you do this, there is nothing which will stop you from doing it whereas, if it is a class, then I can put some check and protect invalid values from not being taken. So, let us look at some other additional things, right. Let us say I want to initialize the day. I have Y, M and D which are passed as three integers and I want to set up this DD which is pointed to the date data type. So, I want DD's, Y, M and D to be the values Y, M and D that are passed on. So, maybe I want a function like this, initialize the members of this structure to certain values or I may want to add a certain number of days to the type.

So, let say the current date is 30th December 1950, and I want to add 2 days to it and

move to January 1, 1951. Let's say I want to do that. I should be able to do it. Add a few days and how a mechanism by which I not only recognize that 2 days from 30th December is not 32nd December. In fact, it is not even in December, it is in January and it is not even in January of this year, it is actually moving to the next year. So, if I add 2 days to 30th December, 1950, internally I want this to automatically move to January 1st, 1951. In fact, I may want to handle cases like leap year and so, on also appropriately. All these details I do not want to expose it to the programmer. So, this becomes very cumbersome. All I want to give to the programmer is add day as an interface. With structures it is not easily doable because we can always come and manipulate these values without going through these methods. You may go and improve increment d directly without realizing that the triple D, M, Y is not valid. You may do that, but if you are forced to go through add day as function always, then add day can be implemented in a much cleaner way and all the errors can be checked and the function can be implemented. So, that it is always correct. So, let us look at this, a small example. I have int Y, M, D. I always want some valid value for the dates. So, I want what is called a constructor. So, any time this object call date is going to be created, it has to be supplied valid values, otherwise it would be incorrect and I want to be supporting add day. So, even though it is given with struct date, this is not valid syntax. So, we would want class date here, and class date will have three members Y, M and D and it will have two methods date which will take the three initial values and add days which add then certain number of days to the current date.

So, the moment you do this, if you do date my birthday, this would be incorrect because it is not initialized. If you do date my birthday of 12, 30, 1950, the year is 12, the month is 30 and day is 1950. Add day can be written in such a way that this can be recognized as an error and this can be indicated. However, if you do date my day of 1950, 12, 30, this would be recognized as a valid day and later if I do my birthday dot add day of 2, the birthday moves from 30th December 1950 to 1st January 1951. So, this is clean. However, if I do my birthday dot m is 14, this would be invalid because you are accessing the member m directly and this is not correct.

(Refer Slide Time: 17:01)



```
Classes
29

// simple Date (control access)
class Date {
    int y,m,d;           // year, month, day
public:
    Date(int y, int m, int d); // constructor: check for valid date and initialize

    // access functions:
    void add_day(int n); // increase the Date by n days
    int month() { return m; }
    int day() { return d; }
    int year() { return y; }
};

// ...
Date my_birthday(1950, 12, 30); // ok
printf("%d\n",my_birthday.month()); // we can read
my_birthday.m = 14; // error: Date::m is private
```

So, if you do classes, the way to do that is you have Y, M and D as private and you have all the methods as public. So, date is public. Add day will actually change the contents of Y, M and D and we have three methods called month, day and year and these three methods actually return the current month, the current day and current year. So, these three methods give you only read access to the data. Date is supposed to give right access. You can change the contents of Y, M and D and add day can also change the contents of Y, M and D. So, date my birthday of 1950, 12, 30 will change Y, M and D to be these three values and I can do printf percentage d my birthday dot month and my birthday dot month is a method. If I go and look at that method, it returns m and what is the value of m, it is supposed to be 12. So, it will print 12 here. However, if I do my birthday dot m is 14; the compiler will catch it and dot it. Even when you compile it, the compiler will catch it and say that m is a private attribute. It cannot be manipulated directly. So, if you want to manipulate m, you can either, go and create a new variable and access through it, or you can only add days to it. You cannot manipulate it directly.

(Refer Slide Time: 18:31)

The slide is titled "Classes" and shows C++ code for a `Date` class. The code includes a class definition with public methods and private members, and a definition of the class. To the right of the code is a table representing a `Date` object with columns for year, month, and day.

```
// simple Date (some people prefer implementation details last)
class Date {
public:
    Date(int yy, int mm, int dd); // constructor: check for valid date and initialize
    void add_day(int n); // increase the Date by n days
    int month();
    // ...
private:
    int y,m,d; // year, month, day
};

Date::Date(int yy, int mm, int dd) // definition; note :: "member of"
:y(yy), m(mm), d(dd) { /* ... */ }; // note: member initializers

void Date::add_day(int n) { /* ... */ }; // definition
```

	Date:
my_birthday: y	1950
m	12
d	30

So, the notion of a valid date is a very important special case, and you want to do this and you want to try and design our data type. So, that there is always some guarantee that the underlined data is valid. So, for example, may be I want to design a class that is supposed to represent a point in the first quadrant, right. First quadrant is anything including 0, 0 or origin, right. So, the first quadrant is anything including 0, 0 and if I move to the right or I move up, that would be the first quadrant in a plane. I want to be able to ensure that at no point of time, this point gets out of the first quadrant. I do not want to do manipulations or I do not want to allow manipulations to go out of the first quadrant. If I want to do things like that, then I will do operations and always check whether the data is still in the first quadrant or not, and report an error if it gets out of the first quadrant for things like this. So, to check validity, it is always useful to do this. To allow classes to be defined and the classes can also go and check for the validity, right..

So, let see how this whole thing would look like. So, I was talking about this function called date. This date has a special function which carries the same name as the class's name. So, we can see that blue and red one, they have the same name, right. The constructors always take the same name as the class. In this case, the constructor is the method which takes three integers yy, mm and dd and what it does is, it assigns your local variable y to yy and the local variable m is assigned to the value mm, and the local variable d is assigned the value dd. So, if you create an object of the type date and if you put date of 12, 30, 1950, internally you can write code within the constructor which will go and check that and say that it is invalid, right. However, if it is valid, it initializes it

and this check that you do inside the core may just leave it as it is.

(Refer Slide Time: 20:52)

```
Classes
32
// simple Date (some people prefer implementation details last)
class Date {
public:
    Date(int yy, int mm, int dd); // constructor: check for valid date and
                                // initialize
    void add_day(int n);         // increase the Date by n days
    int month();
    // ...
private:
    int y,m,d; // year, month, day
};

int Date::season() { /* ... */ } // error: no member called season
```

So, if I end up define, let say I design a function call `int date::season`. This is supposed to be written 0, 1, 2 or 3 depending on whether it is winter, spring, summer and autumn. Let say that is what we want to do if you do `int date::season`; it means you are looking at a function called `season` which is a method inside this class called `date`. However, the class called `date` does not have a method called `season`. The compiler will catch it and tell you that it is an error.

(Refer Slide Time: 21:25)

- ```
Classes
33
```
- Why bother with the public/private distinction?
  - Why not make everything public?
    - ▣ To provide a clean interface
      - Data and messy functions can be made private
    - ▣ To maintain an invariant
      - Only a fixed set of functions can access the data
    - ▣ To ease debugging
      - Only a fixed set of functions can access the data
      - (known as the “round up the usual suspects” technique)
    - ▣ To allow a change of representation
      - You need only to change a fixed set of functions
      - You don't really know who is using a public member

So, this notion of public and private is a useful distinction to make. We do not make

everything public by default because by keeping things private, we can provide a very clean interface and we can also maintain things which are supposed to be invariance. What is the validity of the data inside, we can maintain the invariance. This can also help in debugging programs because if you manipulated something and if something went wrong, it could have happened only through functions that can manipulate the variables. It could not have happened through something else which is outside. So, you can go and round up the usual suspects essentially and say that if manipulations happened, it happened only through these methods and there is something in these methods which is incorrect. You will also see that it allows you to change the internal representation. The notion of a class allows you to change the internal representation. This is something that you will see in a lot of detail in a later class when we talk about the notion of adt and data structures, right.

So, as an adt for a list, I could use a link list internally or it could use an array internally. The notion of a list is just a sequence of elements. I could have used an array internally or I could have used a link list internally. This kind of implementation detail can be hidden from somebody who just wants to use a list and this is possible if you use classes.

(Refer Slide Time: 22:58)

## Classes

- What makes a good interface?
  - Minimal
    - As small as possible
  - Complete
    - And no smaller
  - Type safe
    - Beware of confusing argument orders

So, many times we go and look at what makes a good interface. So, we define classes in such a way that an interface is minimal. It should be as small as possible and at the same time, it has to be complete. So, it has to be small, it has to be small enough to do all the basic things, but not any smaller than that. It has to be the smallest and elegant set of things that you want to expose to the internal world, to the external world I mean and it

has to be safe. So, you do not want to have the arguments passed in a different manner and ensure that it is type safe.

(Refer Slide Time: 23:36)

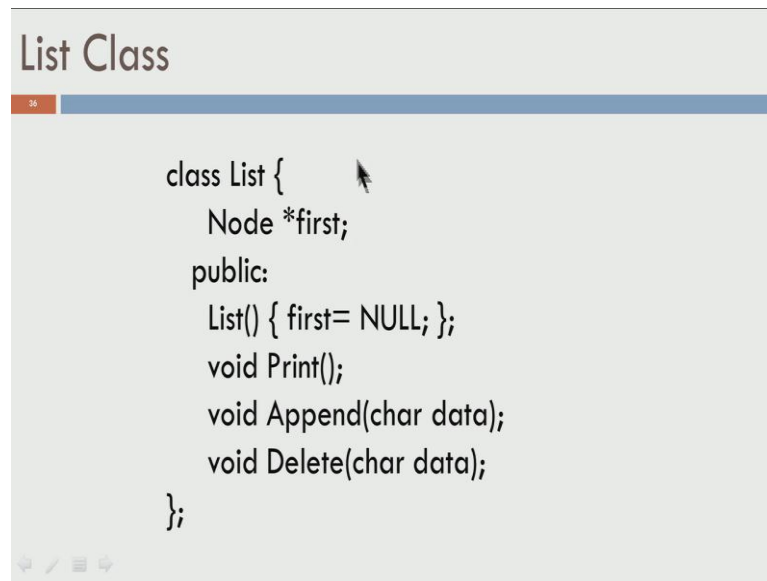
### Let us define a linked list the C++ way

- Not the complete code
- Just to give you a feel for how to do it

```
class Node {
 int data;
 Node* next;
public:
 Node() {}
 void SetData(int aData) { data = aData; };
 void SetNext(Node* aNext) { next = aNext; };
 char Data() { return data; };
 Node* Next() { return next; };
};
```

So, let see how to define the link list in a C plus plus way. This is what I was promising earlier, right. So, to define link list in a C plus plus way, you would do something like this. We define a class called node. This is very similar to a structure, right. So, we had struct node data and next, here we have class node where the private members are data and next. We also provide public methods. Let us look at the public methods. You have set data and set next which will let data and next to be changed, and we have data and next which will do the get. So, these two methods give you put access to node, right. So, you are allowed to change with these two and you are allowed to read using these two. There is another method called node which is the constructor for class node. This is not doing anything. So, it is not changing anything at all. So, it is not setting of anything. So, this is the basic class and we are going to use this class inside the link list.

(Refer Slide Time: 24:43)

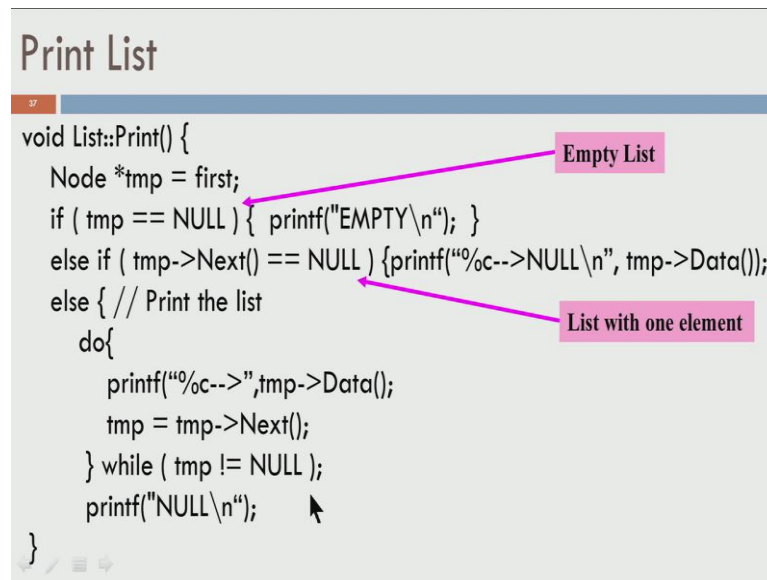


```
class List {
 Node *first;
public:
 List() { first= NULL; };
 void Print();
 void Append(char data);
 void Delete(char data);
};
```

So, what is the link list has? The link list has a node called the first node, right. So, the link list has a first node and it supports various operations like print, append, delete and so, on and it has a constructor called list. If I want an initially empty list, all it supposed to do is ensure that there is no valid node that is pointed to from it. So, first equals null and that is it. So, now, you can see that the notion of a link list becomes much cleaner, right. So, if we want to manipulate anything in the list, you have to go through these functions here; append, delete and print. You cannot go and access first directly because by default first is private. You cannot access first directly. You can only go through the interface functions namely print, append and delete.

(Refer Slide Time: 25:38)

```
Print List
37
void List::Print() {
 Node *tmp = first;
 if (tmp == NULL) { printf("EMPTY\n"); }
 else if (tmp->Next() == NULL) {printf("%c-->NULL\n", tmp->Data());}
 else { // Print the list
 do{
 printf("%c-->",tmp->Data());
 tmp = tmp->Next();
 } while (tmp != NULL);
 printf("NULL\n");
 }
}
```



The diagram shows two callout boxes with arrows pointing to the code. The first callout, labeled 'Empty List', points to the line `if ( tmp == NULL ) { printf("EMPTY\n"); }`. The second callout, labeled 'List with one element', points to the line `else if ( tmp->Next() == NULL ) {printf("%c-->NULL\n", tmp->Data());}`.

Let see how a print method would look like. So, the implementation of print would look like this. So, node star temp equals first. If temp itself is null which means it is an empty list, you may want to print empty on the screen. If temp, if there is only one node in the list, then the first nodes next will be null. We actually do this not using the next field, but we call the method called next temps next. If that returns null, then print the character which is contained in temp data and then, followed by arrow null. Otherwise we run a do while loop which prints one character at a time and the way do we do that is, we do not access the data element or the next pointer of the nodes directly. We call the methods temp data and temp next to that and we keep doing this till we hit the end of the list.

So, this basic loop takes care of reading one element at a time from the nodes and printing them based on whether there are only zero elements or one element or more than one element. So, this is clean because any one who calls print does not have to worry whether it is an empty list or does it contain one element or more. So, the print itself takes care of the implementation. The detail is hidden from the user. So, the user can just call print and be done with it.

(Refer Slide Time: 27:19)

```
void List::Append(char data) {
 Node* newNode = new Node();
 newNode->SetData(data);
 newNode->SetNext(NULL);
 Node *tmp = first;
 if (tmp != NULL) {
 while (tmp->Next() != NULL) { tmp = tmp->Next(); }
 tmp->SetNext(newNode);
 }
 else
 first= newNode;
}
```

Let see append to the list. So, for appending to the list, we are trying to add an element to the end of the list. So, we create a new node, we set it to the data that is passed. So, append is supposed to take a data and it sets the data to the last and sets the next pointed to null. So, you create a new object of the type node and you start with a temporary variable here which points to first. If the list is empty, then you just make first equals new node because there is nothing else to do. You created a new node and the first pointer will point to that. However, if it is not an empty list, then we keep moving the pointer till we hit the end of the list at that point we insert it. So, this kind of a set up where we traverse the list and. So, on is completely hidden from the outside world, right. So, let me explain what I mean by hidden from the outside world by showing how the program will look like.



(Refer Slide Time: 28:21)

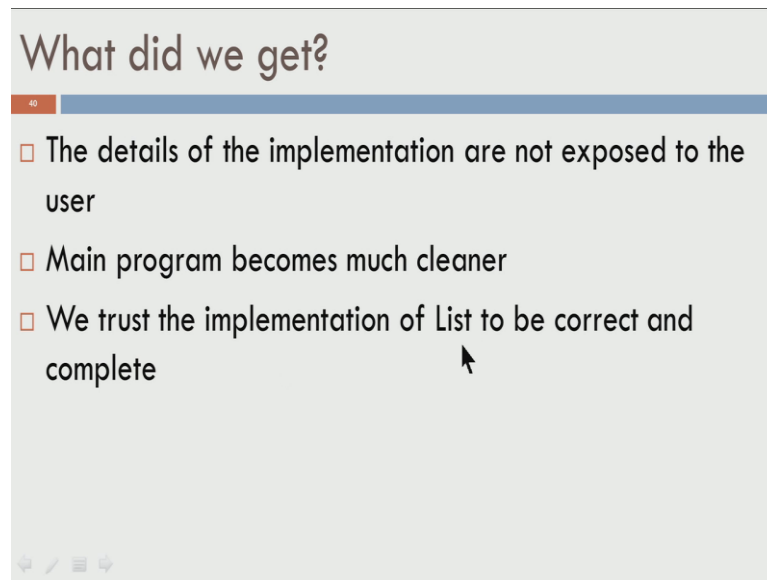
```
Main Program
39
int main() {
 List list;
 // Append nodes to the list
 list.Append('a'); list.Print();
 list.Append('b'); list.Append('c'); list.Print();
 list.Append('d');
 list.Delete('b'); list.Print();
}
```

Let me write a small program which shows how clean this notion of classes makes the whole program. So, I want to create a new link list and I want to add elements a, b, c and d and may be at a later point of time, I want to even delete the element b in the list. So, here I declare a variable call list or an object list which is of the data type list and this appends a, b, c and d is going to add these characters a, b and c and d to the list. Let see how the syntax is. So, list is an object. When I say object dot append, this object is of the data type list. So, it is going to find out if there is a method by the name append; and what does it take. Append takes that data type character. So, we are passing a character and list dot append will add a to the list. At this point of time, this list is empty. We do not have to worry about that.

We just said list dot append of a, and the first time append is called, inside here you create a node. This would be not true. You would just change the first point out in new node and maybe we can print it. Even as soon as you create list with one node, you can print it and then, we do list dot append of b. At this point the list already contains a, and now the method will look at this is not the first node. So, it will go through the list, find out that the null is pointed from a. At that point, it will be added. So, a will now point to b. If you do this, b will point to c and when you print, you start from the first pointer, print a, print b, print c and. So, on and go to the end. Now, you can append d. So, the list will have a, b, c, d and finally, if you call list dot delete of b, if there is a function implemented by the name delete, we assume that programmer has already taken care of that. The classes designer is taking care of print append and delete as a user of the class.

So, when I said user and programmer and. So, on till now I am talking about user of the class, right. So, the user of the class, as a user I want the link list. I do not want to worry about how the list is implemented, how append is implemented; delete is implemented and. So, on. I leave it to the designer of the class and this delete of b if there is a function defined by the name delete, I will assume that gets done and list dot print I will expect it to print a, c and d.

(Refer Slide Time: 31:22)



40

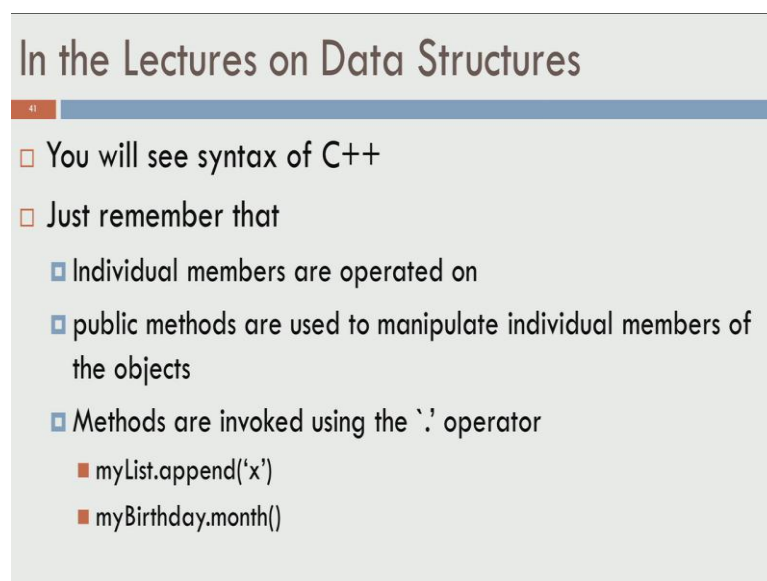
## What did we get?

- The details of the implementation are not exposed to the user
- Main program becomes much cleaner
- We trust the implementation of List to be correct and complete

40

So, what did we get? The details of the implementation of append, delete and creation of the list and. So, on are not exposed to the user of the class. The main program became much cleaner. See this is much cleaner than making function calls to append and checking whether this is the first and second and. So, on. We did not do any of that. This becomes much cleaner here, and we trust the implementation of the class list to be correct and complete. This is something that we get from the implementation of a class. This is the reason why we use C plus plus in the data structure lectures that are following.

(Refer Slide Time: 32:00)



41

## In the Lectures on Data Structures

- You will see syntax of C++
- Just remember that
  - ▣ Individual members are operated on
  - ▣ public methods are used to manipulate individual members of the objects
  - ▣ Methods are invoked using the `.` operator
    - myList.append('x')
    - myBirthday.month()

41

So, in the lectures on data structures, you will see syntax of C plus plus. So, do not be bogged on by the syntax of C plus plus. Just remember that you have individual members that are going to get operated on or manipulated. You will either change the values of the members, or you are going to read the values of the members. You will see that there are public methods that are used to manipulate the individual members, and you may also see private methods which are only called by public methods, and you will see that the methods are actually invoked using the dot operator. For example, the link list append would do my list dot append of x or you have my birthday dot month and so, on. So, do not be surprised by functions being called with the dot operator. So, all you need is the basic syntax of C plus plus which shows that there are classes which has internal members and internal methods. They may have public members and public methods, and the members and methods are going to be used just like you do it for structures using the dot operator..

So, beyond this you probably do not need much of C plus plus and as I mentioned earlier, we are not going to do anything more than that in this lecture. So, I said this is not a completely justified introduction to C plus plus. I have done just enough. So, that you can appreciate the slides that are following or the lectures that are following on data structures. So, if you need C plus plus, this is actually a completely new course that you have to do, where you start with the syntax and semantics of C plus plus and also, understand how to do object oriented design. Since, this course is objective, the basic programming and data structures and algorithms; we will not touch up on C plus plus in any further detail.

So, thank you and this brings me to the end of this lecture.