

**An Introduction to Programming through C++**  
**Professor Abhiram G. Ranade**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Bombay**  
**Lecture No. 21 Part- 1**  
**Classes**  
**Constructors**

Welcome back!

(Refer Slide Time: 0:23)



**What we discussed**

- We would like to build safe and convenient “software components”
- Safe = less likelihood of errors
  - Think through how the component is going to be used and provide a set of corresponding member functions
  - Disallow direct access to members – usually error prone
- Convenient = make it easy to think about parts of the program
  - Member functions provide a clear interface
  - Users of the component do not worry about how the functionality is implemented
- These are “guidelines” not absolute rules.
  - Occasionally disobey guidelines, but have very good reasons for doing so.

Next: Constructors



In the previous segment we talked about software components. In this segment we are going to talk about constructors.


(Refer Slide Time: 0:31)

### Motivational example: the Queue struct in taxi dispatch

```
const int N=100;
struct Queue{
    int elements[N],
        nwaiting,front;
    void initialize();
    bool insert(int v){
        --
    }
    bool remove(int &v){
        --
    }
};
```

```
int main(){
    Queue q;
    q.initialize();
    ...
}
```

- A programmer may forget to call initialize!
- The designer can ensure that q.nWaiting and q.front will become 0 even so!  
-- Next.



So, as motivation for this, let us examine the Queue struct in taxi dispatch. So, this is the definition of the queue, the queue struct and if you remember, there were these, there was an array of elements storing the waiting drivers IDs, then a few variables, then an initialize member function and insert and remove member functions. The main program looks something like this. So, we started off by declaring, by defining the queue creating the queue and that we called queue.initialize(). And then there was a fairly long, there was a loop which I have not, not put over here I want to look at these 2 statements.


There is a simple problem over here, which is that a programmer may forget to call initialize, he will forget. What do you do? It turns out that the designer can ensure that nwaiting and queue front will become 0 which is what, which was the job of initialize if you remember, even if the programmer forgets to write this. How that happens, we are going to see in a minute. But that is the, that is the idea that somehow when we create the queue, we will execute something automatically, which will cause nwaiting and front to become 0.

(Refer Slide Time: 2:09)

### Constructor example

- In C++, the programmer may define a special member function called a **constructor**
- The constructor is called whenever an instance of the struct is created.
- A constructor has the same name as the struct, and no return type.
- The code inside the constructor can perform initializations of members.

```
struct Queue{
    int elements[N], front,
        nWaiting;
    Queue() { // constructor
        nWaiting = front = 0;
    }
    ...
};
int main(){
    Queue q;
    ...
}
```



So this, this special code that we execute at the time of the construction of an object is called the Constructor Code. So, basically the constructor is called, whenever an instance of a struct or a variable or a variable of a struct type is created. Now, in inside the body of the structure definition, in the structure type definition, a constructor has the same name as the struct and it does not have a return type. So, we will see that. So for a struct queue these were the members, and here is the constructor. So, queue there could be parameters, but this particular constructor does not have a parameter.


And then we are going to give the code that is to be executed. That is it, ok. So, this is what the constructor is and in main the moment I define queue, I define q as an object of type, structure type queue, this code is automatically going to be called. So, so the variables will be created and then this code will be called. So, the code inside the constructor is expected to perform initialization of the members, but of course it is any code, so you can make into whatever you want. But that is why, but its purpose really is, it was intended, the notion of constructor was created so as to enable initializations. So it is really the designer who is keeping track and saying that, “Oh! Look, when you create you need to initialize, and therefore, I will give you a way of making that happen automatically.” And then the user just writes this and the initialization happens. So the constructor is called automatically.

(Refer Slide Time: 4:34)

### Constructors in general

```
struct X{  
-  
  X(parameters){  
-  
  }  
};  
  
int main(){  
  X x(arguments);  
}
```

- Constructor can take arguments.
- The creation of the object x in main can be thought of as happening in two steps.
  - Memory is allocated for x in main.
  - The constructor is called on x with the given arguments.
- You can have many constructors, provided they have different signatures.



So, in general, we can have constructors in any struct. So, here is a struct x which is defined and here is the constructor, so this is the body of the constructor. And in main I can have, I am declaring little x to be an object of type capital x and I am supplying these arguments and this will cause this constructor to be called. So, the constructor can take arguments and the creation of the object x in main can be thought of as happening in 2 steps.


First, the memory is allocated for x, in the activation frame of main and then the constructor is called on x with the given arguments. So this code executes, but with respect to these arguments. And, and if you have members referenced over here they are considered to be the members of the constructed structure. You can have many constructors provided they have different signatures. And we are going to see example of this as well.

(Refer Slide Time: 5:43)

### Another example: Constructors for V3

```
struct V3{
    double x,y,z;
    V3(){
        x = y = z = 0;
    }
    V3(double a){
        x = y = z = a;
    }
};
int main();
    V3 v1(5), v2;
}
```

- When defining v1, an argument is given. So the constructor taking a single argument is called. Thus each component of v1 is set to 5.
- When defining v2, no argument is given. So the constructor taking no arguments gets called. Thus each component of v2 is set to 0.

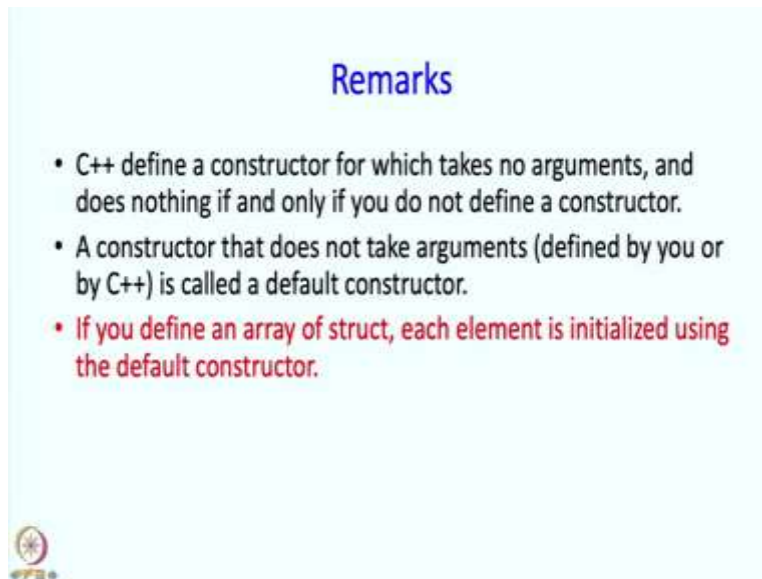


So here is an example of a constructor for V3, so this first constructor does not take any arguments and it just sets all the members to 0. This constructor takes 3 arguments, constructor takes a single argument and it sets all the members to that single argument. Are these useful? Well if they are not useful you can define your own, but I am showing this just to show you what the possibilities are.

So, how does the main program work? Well the main program works when defining V1 here and argument has been given, so the constructor which takes a single argument, so since the single argument is given, the constructor taking the single argument is called, so this is the constructor is takes a single argument, so thus, it means that v1 dot x, v1 dot y, v1 dot z are set to the value 5, set to value a which in this case is 5.


When defining v2, no arguments have been given, we note that we are not putting parenthesis either. So, this is kind of a non- uniformity but that is what it is, we are not putting parenthesis. So, we are defining, we are just giving the name of the variable which is being defined, and since, no parenthesis are given, no arguments are given, we look in the definition of v3 and see if there is a constructor without arguments so this is the one. So, this constructor will get called v2.x, v2.y, v2.z will be set to 0. So, that is how constructors works, so these constructors work.

(Refer Slide Time: 8:05)



### Remarks

- C++ define a constructor for which takes no arguments, and does nothing if and only if you do not define a constructor.
- A constructor that does not take arguments (defined by you or by C++) is called a default constructor.
- If you define an array of struct, each element is initialized using the default constructor.



So, now you might have wondered we have been creating all these structures without even knowing about constructors, so what is going on then? Well it turns out that C++ defines a constructor for which, which takes no arguments and does nothing, if and only if you do not define a constructor. So in the earliest examples we did not define any constructor, but you could think of this way that C++ supplied a constructor which is really doing nothing. So you could say that there is always a constructor either something provided by C++ or something that you provide. Now, there is a technical term which is very commonly used which is the notion of a default constructor. A constructor that does not take arguments and this constructor might be defined by you or by C++ is called the default constructor. If you define an array of struct, each element is initialized using the default constructor. So, if you want to construct an array of, of a certain kind of struct, then you had better have a default constructor.

So, now look at this line, if you define a constructor, then C++ does not define the constructor. So, if you define a 2 argument constructor, then C++ will not even define a 0 argument constructor. But if you want an array of struct, then that is not a satisfactory state. So if you, if you define the 2 argument constructor and you want to define arrays then you had better also define a 0 argument constructor and it could do nothing but that is fine, that is what C++ would define on its own anyway.

(Refer Slide Time: 9:42)

## More remarks

- If a member is itself a struct, its constructor is called first.

- Example:

```
struct point {double x, y};  
struct disk { point center; double radius; }
```

While constructing disk, center will be constructed using constructor of point.



One more remark, so if a member itself is a struct, then its constructor is called first and even this is the default constructor, however that constructor is used. So, for example, struct point double x, y, I have struct disk point center, so when constructing disk center will be constructed according to the constructor of point.

(Refer Slide Time: 10:14)

## What we discussed

- Constructors can be defined so that variables get initialized the moment they are created, automatically
- There can be many constructors with different signatures.

Next: Operator Overloading



So, what have we discussed in this segment? So we have said that the constructor can be defined so that variables get initialized in the, initialized the moment they are created automatically.

There can be many constructors with different signatures. Next, we are going to talk about something called operator overloading, but before that we will take a quick break.