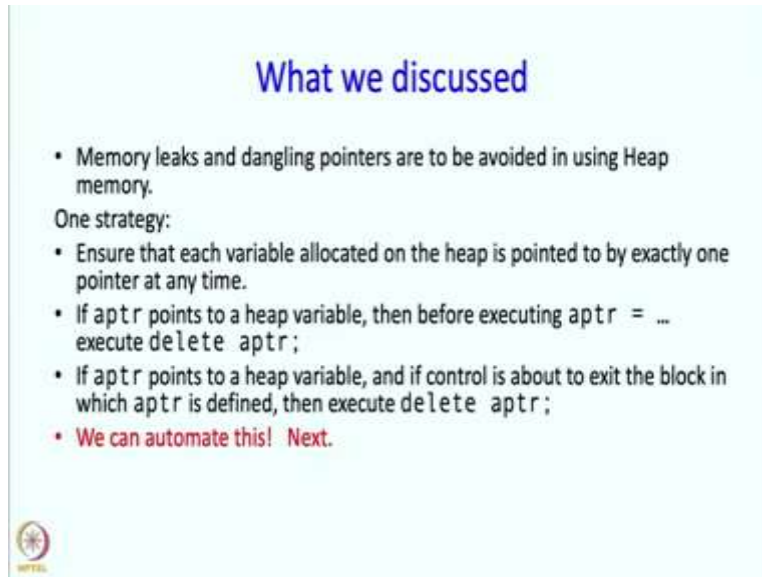


**An Introduction to Programming Through C++**  
**Professor Abhiram G. Ranade**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Bombay**  
**Lecture-22 Part-04**  
**Representing variable length entities**  
**Automating memory management**

(Refer Slide Time: 00:21)




**What we discussed**

- Memory leaks and dangling pointers are to be avoided in using Heap memory.

One strategy:

- Ensure that each variable allocated on the heap is pointed to by exactly one pointer at any time.
- If `aptr` points to a heap variable, then before executing `aptr = ...` execute `delete aptr;`
- If `aptr` points to a heap variable, and if control is about to exit the block in which `aptr` is defined, then execute `delete aptr;`
- **We can automate this! Next.**




Welcome back. In the previous segment we discussed a strategy for preventing memory leaks and dangling pointers. In this segment we are going to see how this strategy can be automated. So, we are going to do this by actually implementing the automation process. So, towards that we are going to define a class called String, which will be a class for representing character strings.

(Refer Slide Time: 00:47)

## A class for representing character strings

We would like to build a `String` class.

- We should be able to
  - store character strings of arbitrary length,
  - pass `Strings` to functions,
  - concatenate them,
  - copy them,
  - index into them to get individual characters
- Without worrying about allocating memory, memory leaks, dangling references




What we want to do is, we want to be able to store character strings of arbitrary length into objects of this class and we should be able to pass strings to function, we should be able to concatenate them, we should be able to copy them and we should be able to index to get the individual characters of the string. And all this we should be able to do without worrying about allocating memory, memory leaks, dangling references and things like that.

(Refer Slide Time: 01:16)

## What freedom do we have?

A sample program	How C++ executes it
<pre>int main(){   String a, b;   a = b = "pqr";   cout &lt;&lt; a[1] &lt;&lt; endl;   {     String b = a + a;     // concatenation     b.print();   }   F(a); }</pre>	<ul style="list-style-type: none"><li>• Constructor called on a, b.</li><li>• <code>a.operator=[b.operator]("pqr");</code></li><li>• <code>a[1] : a.operator()(1)</code></li> <li>• <code>a+a : a.operator+(a)</code></li> <li>• <code>~b</code> : destructor called on b</li><li>• a passed to F as argument a copied using "copy constructor"</li></ul>



All right, so what exactly are we wanting and what freedom do we have? So, let me sketch that. So, let me begin with a sample program, so here is a sample program. So, in this we are

assuming that this string class has already been implemented and so we are saying that look we want to create instances of that String class and then we want to assign values to that String class. Then here we want to index into this a. So, we want the first element, so if a is pqr, then this is the zeroth character, this is the first character, so they should print q.

Then there is a block over here, so because there is a block I can define another b inside. So, this b should be set equal to a+a. So, it should be set equal to a+a and what is a+a? Well a plus a we would like it if it means the concatenation of the left operand and the right operand. In this case these two a's are the same operands. So, it should be concatenation of this, so at the end of this, this b should be pqrpqr.

Then we should be able to print this b. So, there should be a member function print and then afterwards after we exit from this block, we should be able pass our object to a function. So, this is an argument to the function and if this is a pass, if this is a call by value then this value should get copied to the parameter of this function f. Alright so now this is our program.

So, let us see, how C++ executes it. So, how does C++ view the various statements and what is sort of that the normal way of executing something like this. So, first this string a, b. So, the way the string a, b works is that C++ cause the constructors on a and b. So, we should be writing the constructors and therefore we should be able to do whatever we want when the statement executes. So, this is relatively simple.

Now, the next statement is `a=b="pqr"`. So, how do you interpret this? Well these are operators equals to are operators, so this operation equal to, if there are several this is right justified. So, first this operation happens, so the way that happens is that this becomes the receiver. There is an operator equal to function and that is called with this. So that is what this is and then the result of this has to be assigned to this a, and so a is acted upon the operator equal to, but with the result of all of these things.

So, C++ will do this. C++ is going to allow us the flexibility of defining these functions `operator=`. So, the `operator=` functions should make copies, may be allocate memory. Whatever is needed can be done inside those operator equal to functions. And we have to do that and we have to also make sure that while doing that there are no memory leaks and there are no dangling references.

So, proceeding along in this program we have this `a[1]` if you remember we said that `a[1]` is also an operator expression. The operator `b` in this square bracket, so this C++ reads as a operated upon by the member function operator square brackets with the argument being `1`. So, again whatever we want to happen as far as evaluation of this a square bracket `1` is concerned we should put in a operator square bracket.

Then there is `a+a` the new part there is of course the assignment but that is leave it alone for the minute, this is the new part and `a plus a` is again as far as C++ is concerned a dot operator `a` with argument being `a`. So, this `a` is the receiver, this `a` is the argument.

So, if we want this whole thing to produce concatenation then we have to define this operator plus member function suitably. Then `b dot print` is there but `b dot print` is fairly standard we just have to have a member function, so this should not be too difficult, we just want to print it. But now there is this exit we are exiting the block.

So, at this point we want `b` to be destroyed. So, as we have said earlier that all the variables which are created inside a block will be destroyed at the end of the block. For this C++ also has a special member function that special member function is called the destructor and that is written by the symbol `~`. So, `~b` is the call that is made and that call is the destructor call.

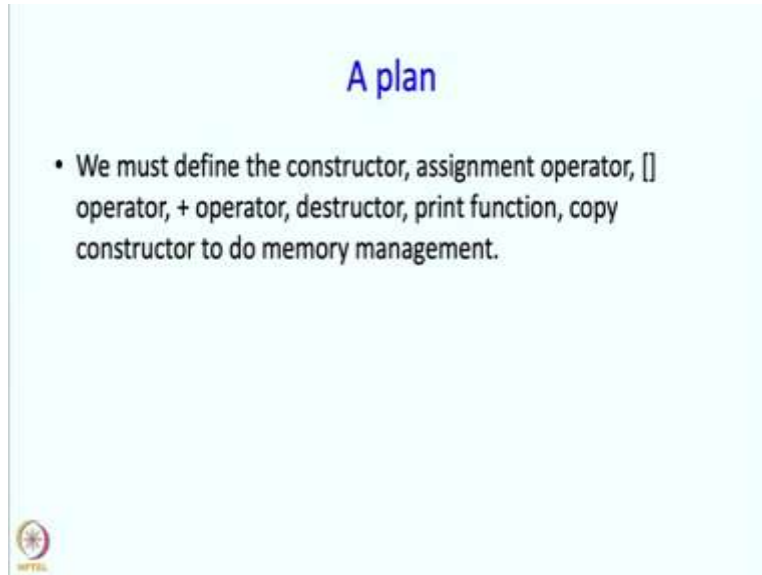
So, again if you want something to happen over here then that can be put in the destructor call and let us take a quick look at this would we want something to happen over here? Well here `b` has been assigned a value. So, presumably we allocated some memory. So, when we exit that memory should get deleted and that deletion we are going to put, the core for the deletion, we are going to put in the destructor of `b`.

So, any memory that was associated, any heap memory that was associated with `b` should be deallocated when the destructor gets called. If we do not do that and if we just destroy the variable `b` then we will have a memory leak.

Finally, we come to this call `f(a)`. So, `f(a)` is let us say it is a call by value in which case this `a` has to be copied over to the parameter of the function `f`. So, `a` is passed as argument and copy of `a` has to be made. And this copying C++ does using something called a copy constructor. Well that parameter is being constructed and it is being assigned a value the parameter is a variable of the

same type as a. And so that parameter is being constructed and its constructed with a copy of a and therefore this kind of this is called a copy constructor.

(Refer Slide Time: 09:03)




So, we will see this in a minute. Alright so the overall plan is as follows we will define the constructor, assignment operator, square bracket operator, plus operator, destructor, print function, copy constructor to do their own work which is say the indexing operator must get the appropriate character. But while doing all of these things, or say the plus operator should concatenate the strings. But while doing these things if some memory management is needed those corresponding functions should do that memory management as well. And thereby we should be preventing memory leaks and dangling references. So, that is the plan.

(Refer Slide Time: 09:54)


## Basic ideas

- We will store the string itself on the heap, while maintain a pointer ptr to it inside our class.
- The string will be terminated using the null character '\0'.



## What freedom do we have?

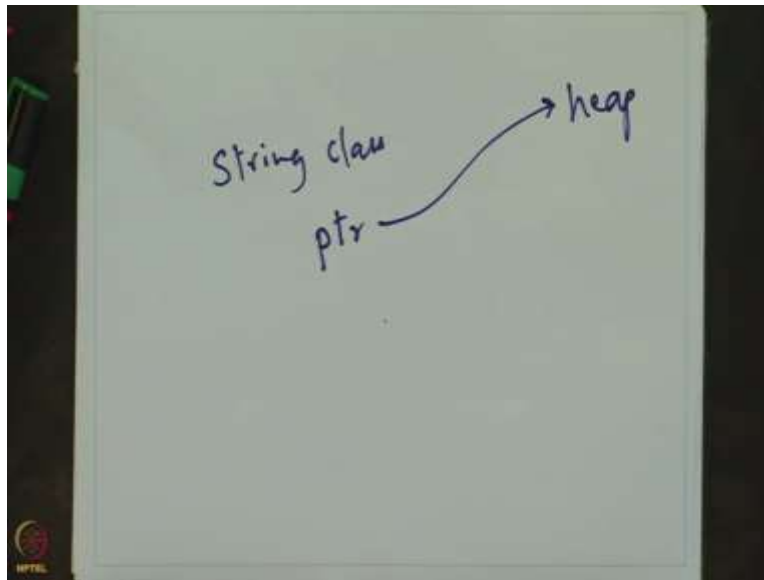
A sample program	How C++ executes it
<pre>int main(){ String a, b; a = b = "pqr"; cout &lt;&lt; a[1] &lt;&lt; endl; { String b = a + a; // concatenation b.print(); } F(a); }</pre>	<ul style="list-style-type: none"><li>• Constructor called on a, b.</li><li>• a.operator=(b.operator="pqr");</li><li>• a[1] : a.operator()(1)</li><li>• a+a : a.operator+(a)</li><li>• ~b : destructor called on b</li><li>• a passed to F as argument a copied using "copy constructor"</li></ul>



Here is one more basic idea that we will need in order to do this implementation. So, we will store the string itself. The string that is going to go and sit inside this assign to this variable that string itself will be stored on the heap while we maintain a pointer ptr to it inside our class.

So, the idea is that if we store pqr in b, what we are going to do is inside the object b, we will have a pointer to the heap and pqr itself will get stored on the heap and whenever we store strings, we will terminate them with the null character. And this is always done so that we do not have to keep the length of the strings around, the sentinel null will be enough.

(Refer Slide Time: 10:57)



### Basic ideas

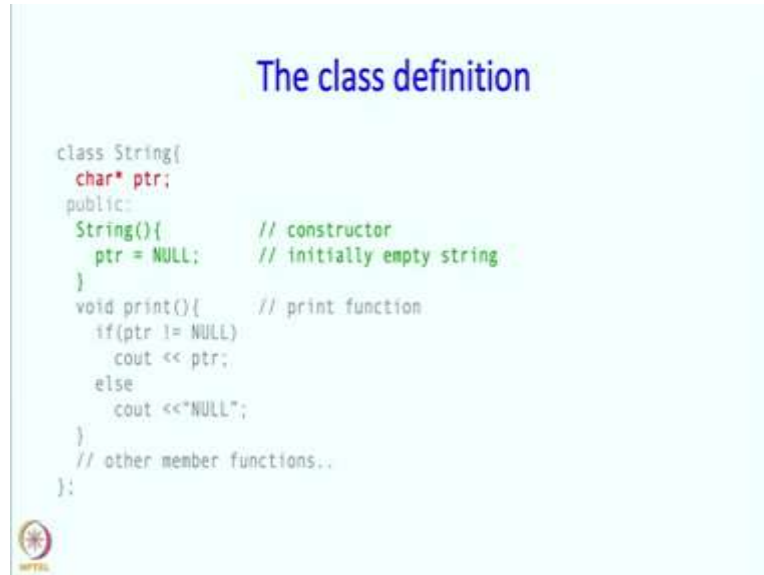
- We will store the string itself on the heap, while maintain a pointer ptr to it inside our class.
- The string will be terminated using the null character '\0'.
- When no string is stored in our class, we will set ptr to NULL.
- NULL (=0) : standard convention, means pointer is invalid.
- To avoid dangling references and memory leaks, we will ensure that
  - Each ptr will point to a distinct char array on the heap.
  - Before we store into ptr, we will delete the variable it points to.
  - When any ptr is about to go out of scope, we will delete it.

So, our object that we design for a string class is going to contain one member ptr and this is going to be pointing to the heap if there is something in this. If there is nothing in this, then this will be set to null. Null is also zero but never mind that, we are always going to use null so that we know that we are talking about pointers.

And this capital null means the pointer really is invalid. So, to avoid dangling references and memory leaks, we will ensure that each ptr will point to a distinct char array on the heap. Before we store into ptr, we will delete the variable it points to. And when any ptr is about to go out of

scope, we will delete the memory that it points to. So, other designs also possible, one of them is given in Appendix G of the book. But we will just discuss this simple design.

(Refer Slide Time: 12:00)



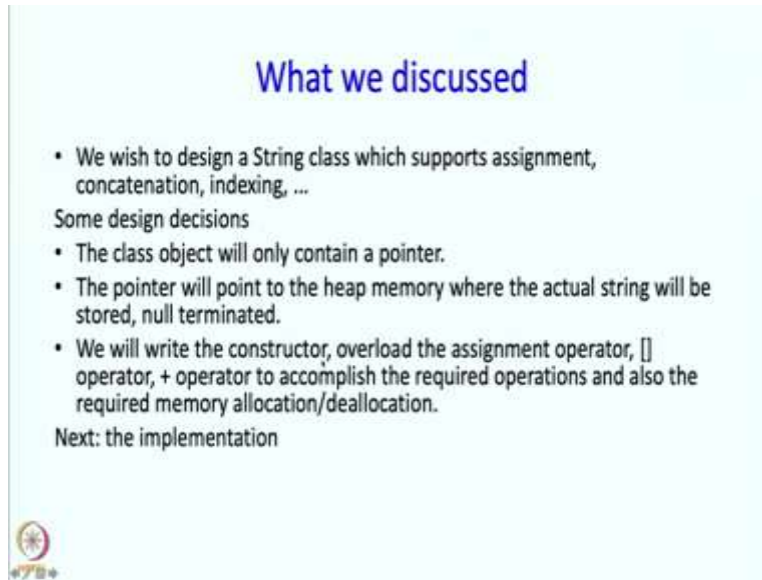
So, here is the definition of the class string. So, it contains this ptr and it is a pointer to char and then it has these public methods or public member functions. So, there is a constructor may be more than one constructor. So, the very basic constructor is going to just set the ptr to be null that just says that if I just write string a as I had written earlier, string a semicolon then the ptr member of a will be null indicating that a is empty.

Then we wanted a print function, we wanted to be able to say print, a.print, so that we can immediately define. So, all we have to do is, we have to print the characters that ptr points to. So, let us be a little bit fancy and let us check if ptr is not null in which case we just print everything starting from ptr till the null character.

Otherwise we will print out a message "NULL". So, there could be, you could have done other things, you could have put out full sentence whatever it is. So this is we have defined one constructor and we have defined one very simple member function but there are lots of other things to be define. So, that is what we will do next.



(Refer Slide Time: 13:42)




### What we discussed

- We wish to design a String class which supports assignment, concatenation, indexing, ...

Some design decisions

- The class object will only contain a pointer.
- The pointer will point to the heap memory where the actual string will be stored, null terminated.
- We will write the constructor, overload the assignment operator, [] operator, + operator to accomplish the required operations and also the required memory allocation/deallocation.

Next: the implementation



Alright, so what we have we discussed, we said that we wish to design a string class which supports assignment, concatenation, indexing and all of that. And then we took some decisions regarding how to design it. So, we said that there will be only one data member in the class which is a pointer. The pointer will point to the heap memory, to the position in the heap memory where the actual string will be stored and that string will be stored there null terminated.

We will write all the other member functions, lots of other member functions that are required say constructors, we will overload the assignment operator, the square bracket operator and all such things to accomplish the required operations and also the required memory allocation and deallocation. So, in the next segment we are going to talk about the implementation but before that we will take a short break.