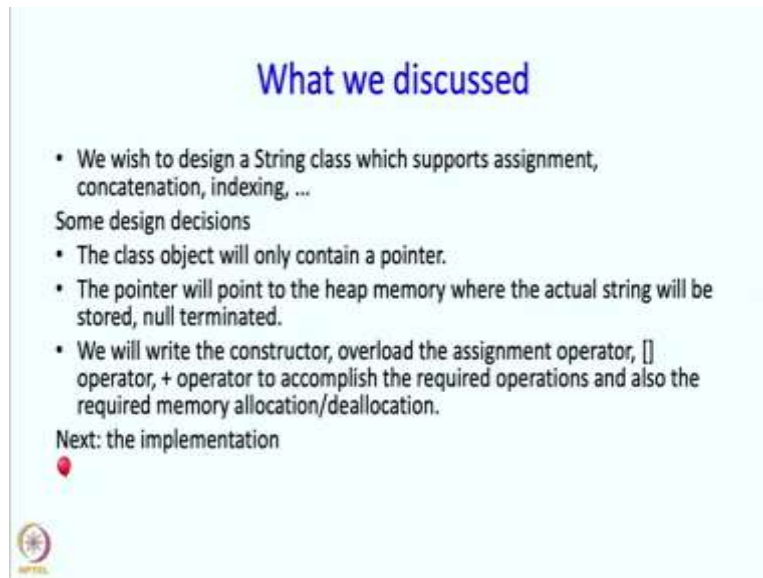


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture No. 22 Part – 5
Representing variable length entities
Implementing a class with automated memory management 1

Welcome back.

(Refer Slide Time: 0:25)




What we discussed

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

Some design decisions

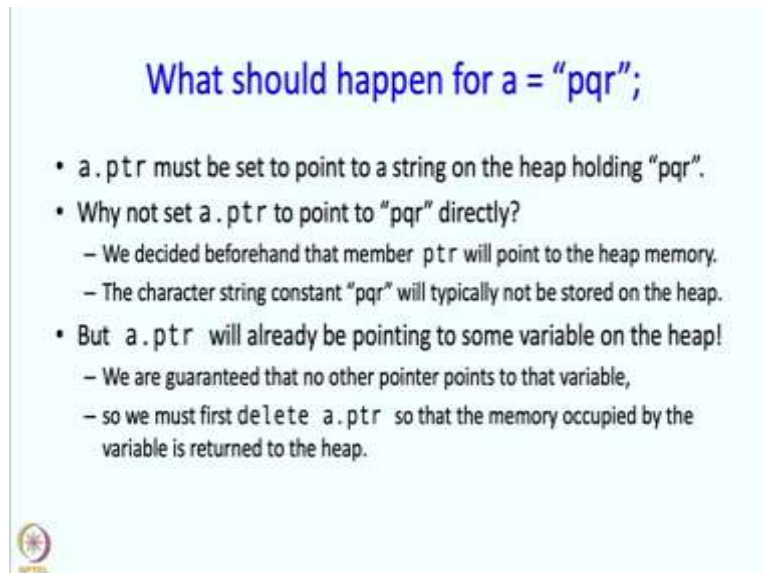
- The class object will only contain a pointer.
- The pointer will point to the heap memory where the actual string will be stored, null terminated.
- We will write the constructor, overload the assignment operator, [] operator, + operator to accomplish the required operations and also the required memory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

Next: the implementation




In the last segment we made a high level plan for designing our string class. Now, we are going to jump into the implementation of it.

(Refer Slide Time: 0:27)



What should happen for `a = "pqr";`

- `a.ptr` must be set to point to a string on the heap holding "pqr".
- Why not set `a.ptr` to point to "pqr" directly?
 - We decided beforehand that member `ptr` will point to the heap memory.
 - The character string constant "pqr" will typically not be stored on the heap.
- But `a.ptr` will already be pointing to some variable on the heap!
 - We are guaranteed that no other pointer points to that variable,
 - so we must first `delete a.ptr` so that the memory occupied by the variable is returned to the heap.



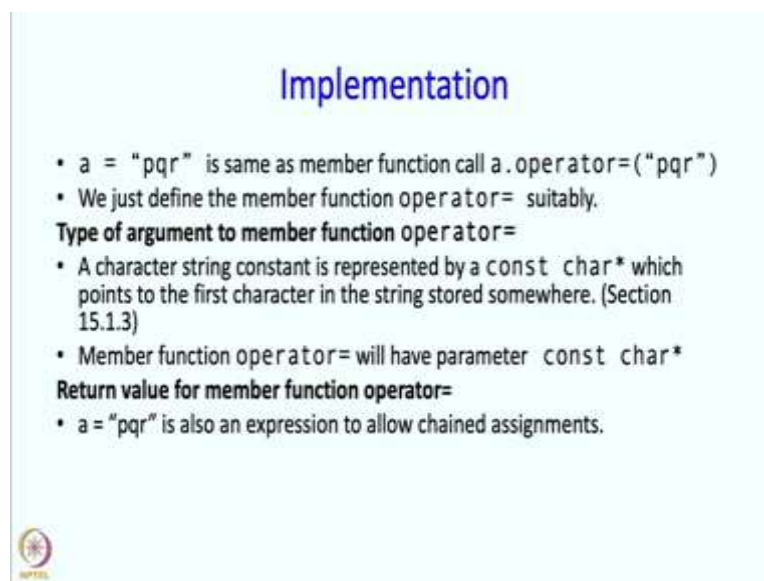
So, let us start off of it. `A=pqr`. So this is an assignment statement. We want the character string `pqr` to be stored in our string variable `a`. How should that happen? Clearly, as we said `a.ptr` should be pointing to this string `pqr`. And our rule was that `a.ptr` should point to things on the heap. Now, since, we said that we cannot immediately just directly set `a.ptr` to `"pqr"`, after all we know `pqr` we know that `pqr` is a `const char ptr`. So, we cannot just write `a.ptr` equal to what that pointer is because we decided beforehand that the member `ptr` will point only to the heap.

Why are we doing that? Well, if we know that is only going to point to the heap memory then we can delete it. If we do not know that, if there is a possibility that sometimes maybe it points to the heap the memory, sometimes it points to activation frame. Then we cannot delete it. And therefore, in order to be able to delete it confidently, whatever data it points to should always be on the heap. And therefore, we must first copy the character string constant to the heap.

All right! So, that is the first consideration that we must copy `pqr` to the heap because it in general it may not be on the heap. But then there is also another consideration, `a.ptr`, we want the set to point to this `pqr` but it is already pointing to some variable in the heap. That is what our convention is, well unless it is null but in general, it could be pointing to some variable on the heap. So what do we do?

We are making `a.ptr` point to something else so this variable that it points to is going to be useless and therefore, we should be deleting it. Why is that? Because we are guaranteed that no other pointer points to that variable. So, therefore, we can just delete, delete `a.ptr`.

(Refer Slide Time: 3:01)



Implementation


- `a = "pqr"` is same as member function call `a.operator=("pqr")`
- We just define the member function `operator=` suitably.

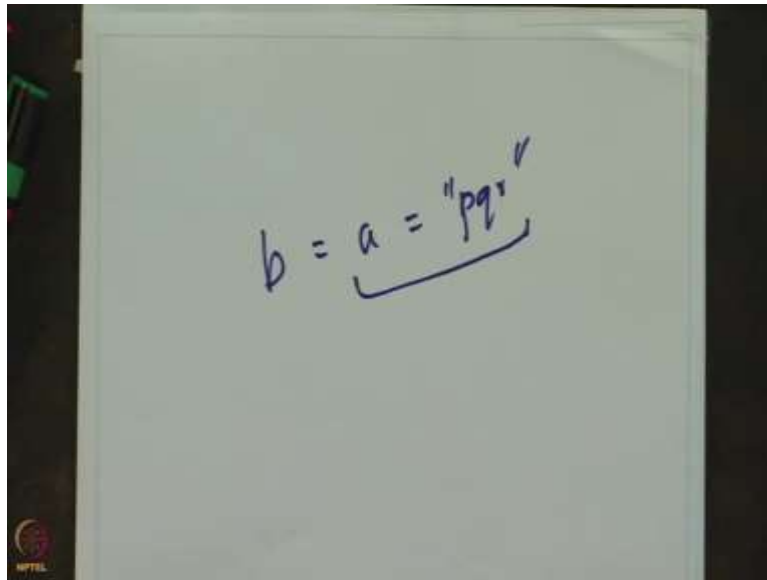
Type of argument to member function operator=

- A character string constant is represented by a `const char*` which points to the first character in the string stored somewhere. (Section 15.1.3)
- Member function `operator=` will have parameter `const char*`

Return value for member function operator=

- `a = "pqr"` is also an expression to allow chained assignments.





So that is all there is to it. So, the way the implementation works `a="pqr"`, for `a="pqr"` we should do what I just said, but the way to do it is that, this statement, for this statement C++ makes a call to `a.operator=`. To operator equal to with `a` being the receiver and `pqr` being the argument and so whatever I just said has to be put inside this function. So, we have to overload this function. So, what are the types of argument?

What is the type of the argument to this member function `operator=`? Well, so this is a character string constant. And we know that this is represented by a variable of type `const char *` and whose address is the starting address of this array. So, the member function `operator=` will have a parameter `const char *` so we can just, so this call will be a valid call, then what value should it return? So, this is a little tricky.

Now, `a="pqr"` wants us to assign `pqr` into `a`. But if you remember we can chain assignments, so what does that mean? We can write something like `b=a="pqr"`. So, if this assignment operator is a part of this expression then this assignment operator is supposed to produce a value. And that value should be returned by the member function `operator=`. So what is the value that it is that this is supposed to produce? Well it is supposed to produce a reference to this `a` itself. So, from which we can pick up the value. So, the return value should be the left hand side. So, it should be this `a` itself and the return type must be a reference to string because `a` has type strings, so we should be returning a reference to that string.

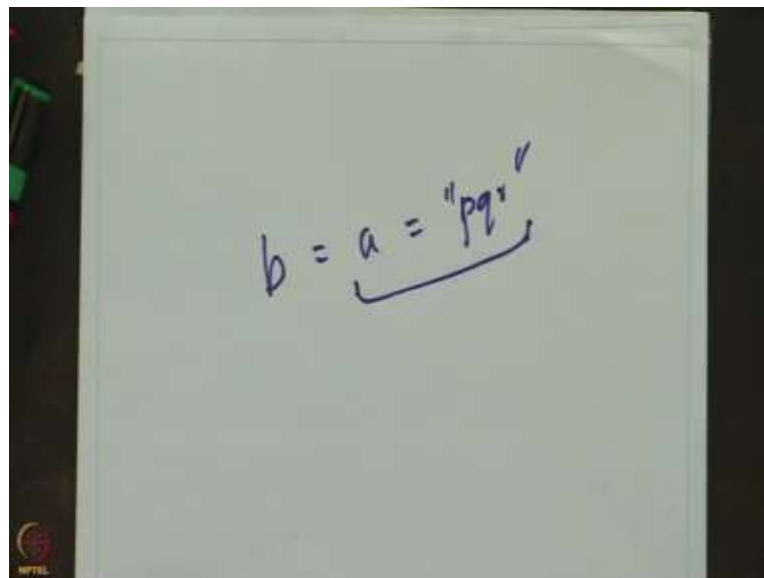
(Refer Slide Time: 5:31)

The code

```
String& operator=(const char* rhs){
    // release the memory that ptr already points to.
    delete[] ptr;

    // make a copy of rhs on the heap
    // allocate length(rhs) + 1 byte to store '\0'
    // Assume length function (Section 15.1.4)
    ptr = new char[length(rhs)+1];

    // actually copy. Function scopy from 15.1.4
    scopy(ptr, rhs);
    return *this;
}
```



So, here is the code. So, we have, we are defining a member function operator=, in the string. This will go into the String class and it will take, so it will be operating on, it will be operating on a string, a string will be the receiver and it will be receiving an argument a character string constant. So, some a variable of type const char * and let us call that variable RHS. So, first of all we said that, if we are implementing this so we are changing a.ptr.

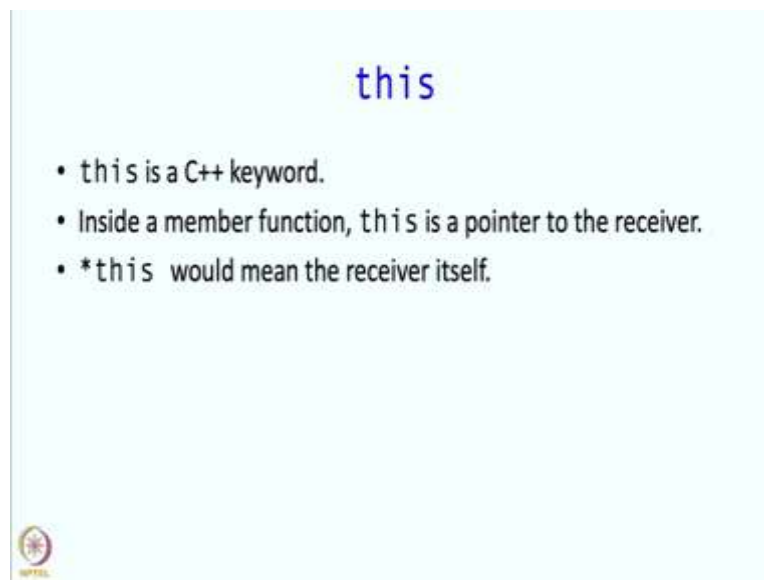
So, a is the receiver, we are changing a dot ptr, so whatever a dot ptr is pointing to should, should be deleted but since is the receiver then this is just ptr. Remember that if a member appears here without any qualification then it is just the pointer member of that receiver. So, we are going to delete that ptr. So, effectively what has happened now is, that we were assigning, we were executing this statement.

The memory that `a` was pointing to, `a.ptr` was pointing to has now been returned back to the heap. Now, we said that we have to make a copy of this `pqr` on the heap. So, how do we do that? Well, we are going to allocate an array of size `length` of this `rhs`, so for `pqr` it will be just three plus, one byte to store this, the null character. So, how do we do that? So in section 15.1.4 we had this `length` function, which we talked, which you mentioned earlier.

And we just call that, so `length` of `rhs` will tell us how long `rhs` is, we add 1 to that and we allocate an array on the heap. So, `new` does that and its address is put into `pqr`. So, this is the array that has been allocated. We have not yet moved the data into it. So, this whatever that string constant `rhs` is, it only its length has been used up. The data actually has not been used up. So the next step is to actually do the call.

So, again we had this `scopy` function which is very simple, which actually just copies every element of this into this until we reach the null and the null is also copied. So, there is a simple loop but any you can get that function from this section. And finally we are going to return this receiver itself, so remember `a=pqr` got translated into `a.operator=("pqr")`. So, this is the receiver and we want to return the receiver. And here is how we return the receiver. So, this is a special word in C++.

(Refer Slide Time: 8:57)



So, let me just explain that to you. So, this is a C++ keyword and inside a member function, this is a pointer to the receiver. So, star `this` would mean the receiver itself.


(Refer Slide Time: 9:14)

The code

```
String& operator=(const char* rhs){
    // release the memory that ptr already points to.
    delete[] ptr;


    // make a copy of rhs on the heap
    // allocate length(rhs) + 1 byte to store '\0'
    // Assume length function (Section 15.1.4)
    ptr = new char[length(rhs)+1];

    // actually copy. Function scopy from 15.1.4
    scopy(ptr, rhs);
    return *this;
}
```



So, here that is exactly what we want. We want to return the receiver itself. Why are we not returning a pointer? Because this says that we want to return a reference. This says we want to return a reference, and the references, references are to variables, so here we should not be having pointers but we should be having references, so then reference can be picked up. So, we should write star this over here.

(Refer Slide Time: 9:40)

- ### this
- this is a C++ keyword.
 - Inside a member function, this is a pointer to the receiver.
 - *this would mean the receiver itself.
 - The value of an assignment var = expr; is var.
- 


So, this is a key word and this would mean the receiver itself, and the value of an assignment var equal to expression is var, so the assignment must return var as a result and hence, the return value is *this.

(Refer Slide Time: 9:54)

Assigning a String to another String

- We want to allow code such as

```
String a, b;  
a = "pqr";  
b = a;
```
- The statement `b = a;` will cause a call `b.operator=(a)` to be made.
- So we need a member function `operator=` which takes a `String` as argument



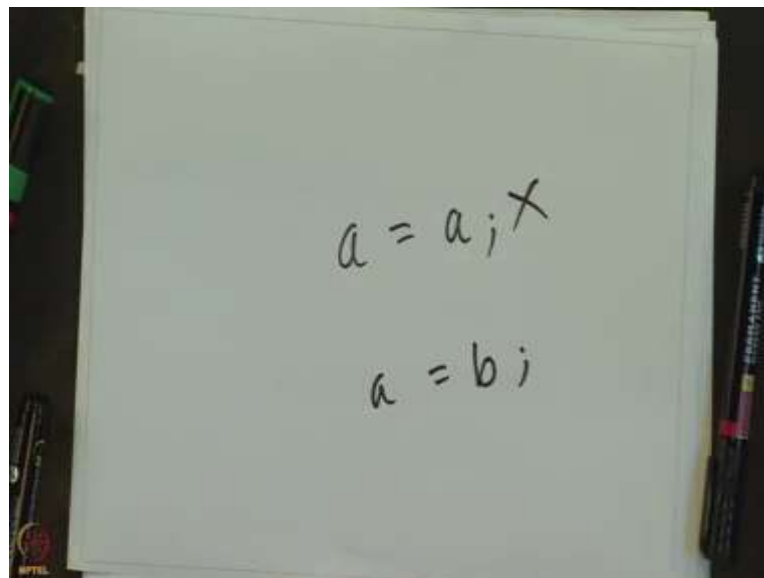

So, that was assigning a character string constant to one of our string variable. But we would also like to be able to assign one of our string variables to another of our string variables. So, we want to allow code such that, `a=b`, `a="pqr"` and `b=a`. Alright? So, how do we do this? So, the statement `b = a` is going to call, is going to be looked upon by C++ as the statement. So, it is the object `b` and we are making a member function call on it, so `b` is the receiver and argument is `a`.

So, this is another assignment and it is very similar to the assignment that we made earlier, except that now the argument is not `char *`, but it is a string variable. And if you want to get the actual string that we want copied, we have to go through its pointer, so we will do that. We want a member function `operator=`, which takes string as an argument. We already have a member function which takes `char const char *` as an argument but we can have another. So, we can overload, we can have more functions so long as the signatures are different.

(Refer Slide Time: 11:27)

The code

```
String& operator=(const String &rhs){  
  
    if(this == &rhs) return *this;  
    // If it is a self assignment, we do nothing.  
  
    delete ptr; // Release the memory that ptr already points to.  
  
    // Make a copy of rhs.ptr on the heap  
    // Allocate length(rhs.ptr) + 1 byte to store '\0'  
    ptr = new char[length(rhs.ptr)+1]; // length: Section 15.1.4  
  
    scopy(ptr, rhs.ptr); // Actually copy. Function scopy from 15.1.4  
    return *this;  
}
```



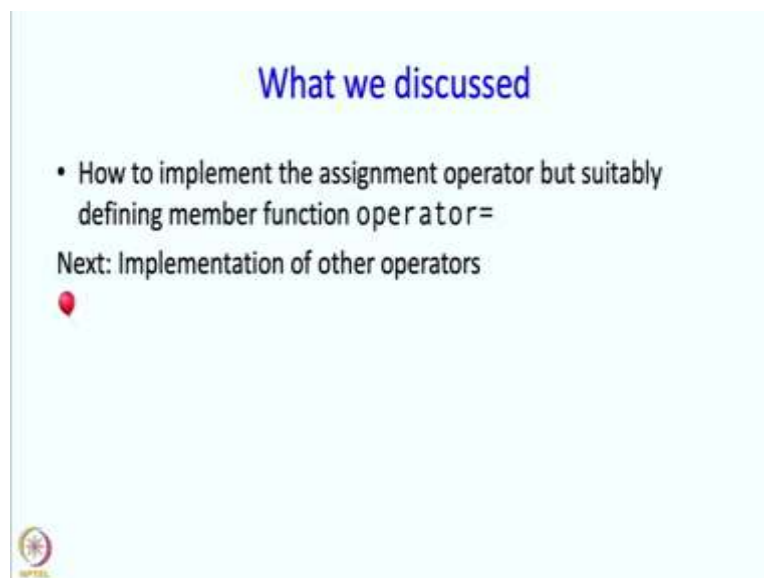
So, this time we have string reference returning, being returned, name of the member of function is operator equal to in the class string and the argument this time is not const char * but const string &rhs, so I could have had rhs over here but I do not really want unless is recopy and therefore, I am going to just get a reference to it. So, we have to be a little careful in all of this. And in principle, our programmer is allowed to write something a=a. So, a=a is really nonsense, but it is not disallowed and therefore, we had better deal with it.

So, the best way to deal with things like this is just make a special case, so what we are going to say is, if this if the receiver is the same, if the address of the receiver is the same as the address of the right hand side, then that means we are in this situation. So, in that case we have nothing to do but we can just return this object itself. And, of course, reference to that gets returned, but we are returning this object.

So, suppose, for now rest of the code that the receiver is different from this right hand side, so we are in a situation not like this, but we are in a situation like whatever $a=b$, where a and b are different strings. In that case, as argued earlier we are going to change ptr and if ptr points to something and ptr will be pointing to something, we have to release that memory, so for that we have to issue `delete ptr` and `delete ptr` is fine even if ptr is null by the way. So, that is not an issue. But if ptr is not NULL then we have to return that memory back to the heap and that is what this statement is going to end up doing then, as before we are going to make a copy of the string in the rhs on the heap. So for this, as before we will allocate length which is `rhs.ptr, length(rhs.ptr)+1`, that 1 is again to store that null ptr. And this is done quite easily by a very similar statement as before, length of `rhs.ptr+1`. So, that much memory we are going to allocate on the heap. And we are going to copy, we are going to copy the memory that rhs wants to the string that rhs is, rhs contains is being now copied to the receiver as well. So, ptr which is the receivers pointer the address that it points to, will now contain, whatever is contained in `rhs.ptr`.

So, for this we again use the `scopy` function, so it just essentially transfers characters from, copies characters from here to here at corresponding displacements and finally, again as before we are going to return this, the variable that this point stores. That is it.

(Refer Slide Time: 15:08)



So, what have we discussed? We have discussed how to implement assignment operator, so I should by suitably defining member function `operator=` and we define two member functions, `operator=`. One which takes as argument `const char star` and another which takes as argument another string object. In the next segment we will be talking about the implementation of the other operators but before that we will take a quick break.