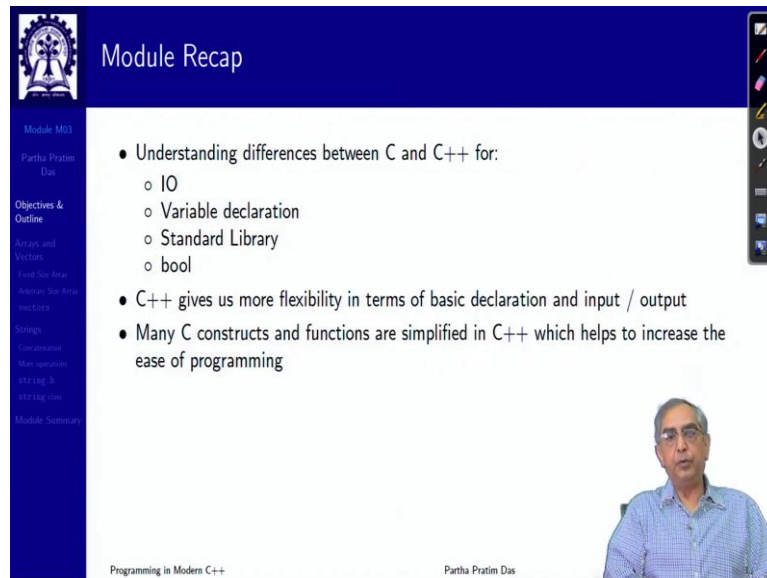


**Programming in Modern C++**  
**Professor. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture No. 06**  
**Arrays and Strings**

Welcome to Programming in Modern C++, we are in week 1, module 3.

(Refer Slide Time: 00:34)



The screenshot shows a presentation slide titled "Module Recap" with a dark blue header. On the left, there is a vertical navigation menu with the following items: "Module M03", "Partha Pratim Das", "Objectives & Outline", "Arrays and Vectors", "Fixed Size Arrays", "Arbitrary Size Arrays", "vectors", "Strings", "Containers", "More operators", "string & string view", and "Module Summary". The main content area is white and contains the following text:

- Understanding differences between C and C++ for:
  - IO
  - Variable declaration
  - Standard Library
  - bool
- C++ gives us more flexibility in terms of basic declaration and input / output
- Many C constructs and functions are simplified in C++ which helps to increase the ease of programming

In the bottom right corner of the slide, there is a small video feed of Professor Partha Pratim Das. At the bottom of the slide, the text "Programming in Modern C++" and "Partha Pratim Das" is visible.

In the last module, we started taking a look at how to write equivalent C++ programs for simple C programs. We took examples which involved IOs, variable declarations, standard library, and bool, and so on. And observed that most of the times, C++ gives us more flexibility in terms of the declaration, input output as well as typing and things start getting simplified to read as well as write.

(Refer Slide Time: 01:19)

The slide is titled "Module Objectives" and is part of a presentation on "Programming in Modern C++" by Partha Pratim Das. The slide lists three objectives:

- Understand array usage in C and C++
- Understand vector usage in C++
- Understand string functions in C and string type in C++

The slide also features a sidebar on the left with a navigation menu and a small video inset of the presenter in the bottom right corner.

So, we will continue that study in this module as well, looking into the next two important things all of us need to know in programming that is arrays and strings.

(Refer Slide Time: 01:36)

The slide is titled "Module Outline" and is part of a presentation on "Programming in Modern C++" by Partha Pratim Das. The slide lists three main sections:

- 1 Arrays & Vectors
  - Array Implementations for fixed size array
  - Array Implementations for arbitrary sized array
  - vectors in C++
- 2 C-Style Strings and string type in C++
  - Concatenation of strings
  - More string operations
  - string.h
  - string class
- 3 Module Summary

The slide also features a sidebar on the left with a navigation menu and a small video inset of the presenter in the bottom right corner.

So, this is a outline, which will be there on the left always.

(Refer Slide Time: 01:42)

Arrays and Vectors

Module M03  
Partha Pratim Das  
Objectives & Outline  
Arrays and Vectors  
Fixed Size Array  
Arbitrary Size Array  
vectors  
Strings  
Containers  
More operators  
string &  
string view  
Module Summary

Arrays and Vectors

Programming in Modern C++ Partha Pratim Das

Program 03.01: Fixed Size Array

Module M03  
Partha Pratim Das  
Objectives & Outline  
Arrays and Vectors  
Fixed Size Array  
Arbitrary Size Array  
vectors  
Strings  
Containers  
More operators  
string &  
string view  
Module Summary

C Program	C++ Program
<pre>// Array_Fixed_Size.c #include &lt;stdio.h&gt;  int main() {     short age[4];      age[0] = 23;     age[1] = 34;     age[2] = 65;     age[3] = 74;      printf("%d ", age[0]);     printf("%d ", age[1]);     printf("%d ", age[2]);     printf("%d ", age[3]);      return 0; }</pre>	<pre>// Array_Fixed_Size_c++.cpp #include &lt;iostream&gt;  int main() {     short age[4];      age[0] = 23;     age[1] = 34;     age[2] = 65;     age[3] = 74;      std::cout &lt;&lt; age[0] &lt;&lt; " ";     std::cout &lt;&lt; age[1] &lt;&lt; " ";     std::cout &lt;&lt; age[2] &lt;&lt; " ";     std::cout &lt;&lt; age[3] &lt;&lt; " ";      return 0; }</pre>
23 34 65 74	23 34 65 74

• No difference between arrays in C and C++

Programming in Modern C++ Partha Pratim Das

So, let us talk about arrays and well, vectors. So, first, arrays, so here is a program, which defines an array of length 4, that is four elements and of type short, so every number is short. And then it assigns values to different array locations and prints them one after the other. Now, this is a program, which is if write in C++, it remains the same. I mean, there is nothing, no difference of using fixed size array, between C and C++ is the same.

(Refer Slide Time: 02:29)

Hard-coded	Using manifest constant
<pre>// Array_Large_Size.c #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt;  int main() { int arr[100], sum = 0, i; printf("Enter no. of elements: "); int count; scanf("%d", &amp;count);  for(i = 0; i &lt; count; i++) { arr[i] = i; sum += arr[i]; } printf("Array Sum: %d", sum); }</pre>	<pre>// Array_Macro.c #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #define MAX 100  int main() { int arr[MAX], sum = 0, i; printf("Enter no. of elements: "); int count; scanf("%d", &amp;count);  for(i = 0; i &lt; count; i++) { arr[i] = i; sum += arr[i]; } printf("Array Sum: %d", sum); }</pre>
Enter no. of elements: 10 Array Sum: 45	Enter no. of elements: 10 Array Sum: 45
• Hard-coded size	• Size by manifest constant

Programming in Modern C++ Partha Pratim Das

Now, let us say this fixed size array could be a large one. And it is possible that it is fixed size, because we do not know what the maximum size should be, so we just make it large. So here is one way of writing it, where we make it, we use the constant size of the array right into the code. And then add the elements and so on, assign some element add them, that is just for illustration.

The other option is to put a manifest constant, defining the size and then use that manifest constant. This is what I mean here, both are C, it is not, you do not need C++ yet, they both are C. Now we often preferred this, because if we in future, need to change the size, either increase it or decrease it, then it becomes quite an exercise to deep inside the code and find out where this size has been defined.

Because it is not going to be as simple a program like what you see here, it could be a couple of 1000 lines that you have in the file. And finding out exactly this dimension, size dimension of this array has to change is a huge problem. Whereas, if we do a manifest constant, you can just simply change it here. And it also gives another advantage that often maybe number of different arrays may have their size related.

I am reading two vectors and adding them in a third vector, I have 3 arrays, and all of them must have the same maximum size. So, if I had code, I have to do the changes in three places, I might miss that out. But if I put the manifest constant and use it the change is only at one place and things solutions are simpler. So, this is just talking about how to keep things better in C itself.

(Refer Slide Time: 04:55)

Arbitrary Size Array

This can be implemented in C (C++) in the following ways:

- **Case 1:** Declaring a large array with size greater than the size given by users in all (most) of the cases
  - Hard-code the maximum size in code ✓
  - Declare a manifest constant for the maximum size ✓
- **Case 2:** Using `malloc (new[])` to dynamically allocate space at run-time for the array

Programming in Modern C++ Partha Pratim Das

And so, let us say now, we want an arbitrary sized array because we do not know how big it should be, I mean should I unnecessarily keep it 10,000, 100,000, how much I have no idea. So, the one option is to declare a size in array large enough, which is greater than all possible size that can have and that can be hard-code as a maximum size as we have seen or declared to it manifest constant.

Now, as we have learned in C that this is not often a preferred approach, if you do not really know what is going to be the size. So, what you can do you can use a dynamic allocation using malloc, there is a C equivalent of that called new which you will learn later on, but know that there is an equivalent of this, malloc itself will also work in C++. This will be help to dynamically allocate space at the runtime.

So, at the runtime you may be able to figure out that you need this much of space. So, you allocate that sized array and proceed, but this will still not allow you to change the size while once you have allocated it. Once you have allocated it is kind of frozen. So, that is a available approach that you have. So, with that let us see how things go.

(Refer Slide Time: 06:31)

C (array & constant)	C++ (vector & constant)
<pre>// Array_Macro_c.c #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt;  #define MAX 100  int main() { int arr[MAX]; printf("Enter no. of elements: "); int count, sum = 0, i; scanf("%d", &amp;count); for(i = 0; i &lt; count; i++) { arr[i] = i; sum += arr[i]; } printf("Array Sum: %d", sum); }</pre>	<pre>// Array_Macro_c++.cpp #include &lt;iostream&gt; #include &lt;vector&gt; using namespace std; #define MAX 100  int main() { vector&lt;int&gt; arr(MAX); // Define-time size cout &lt;&lt; "Enter the no. of elements: "; int count, sum = 0; cin &gt;&gt; count; for(int i = 0; i &lt; count; i++) { arr[i] = i; sum += arr[i]; } cout &lt;&lt; "Array Sum: " &lt;&lt; sum &lt;&lt; endl; }</pre>
Enter no. of elements: 10 Array Sum: 45	Enter no. of elements: 10 Array Sum: 45
<ul style="list-style-type: none"><li>• MAX is the declared size of array</li><li>• No header needed</li><li>• arr declared as int []</li></ul>	<ul style="list-style-type: none"><li>• MAX is the declared size of vector</li><li>• Header vector included</li><li>• arr declared as vector&lt;int&gt;</li></ul>

So, now in C you have this manifest constant and this. In C++ what I do is I include a new thing called vector, vector is a type defined in the library not in the language, but in the library therefore, I need to include the library component vector. Then I write this where arr is the name of the array.

Vector is what I want, vector is like a one-dimensional consecutive location, I want those locations to be of int type and at the time of declaration, at the time of definition, I want the size or the number of such ints as MAX. So, you can relate that int comes here, array name is here, the size is here. And instead of this built-in array, I do this vector stuff, a particular notation.

(Refer Slide Time: 07:59)

C (array & constant)	C++ (vector & constant)
<pre>// Array_Macro_c.c #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt;  #define MAX 100  int main() { int arr[MAX]; printf("Enter no. of elements: "); int count, sum = 0, i; scanf("%d", &amp;count); for(i = 0; i &lt; count; i++) { arr[i] = i; sum += arr[i]; } printf("Array Sum: %d", sum); }</pre>	<pre>// Array_Macro_c++.cpp #include &lt;iostream&gt; #include &lt;vector&gt; using namespace std; #define MAX 100  int main() { vector&lt;int&gt; arr(MAX); // Define-time size cout &lt;&lt; "Enter the no. of elements: "; int count, sum = 0; cin &gt;&gt; count; for(int i = 0; i &lt; count; i++) { arr[i] = i; sum += arr[i]; } cout &lt;&lt; "Array Sum: " &lt;&lt; sum &lt;&lt; endl; }</pre>
Enter no. of elements: 10 Array Sum: 45	Enter no. of elements: 10 Array Sum: 45
<ul style="list-style-type: none"><li>• MAX is the declared size of array</li><li>• No header needed</li><li>• arr declared as int []</li></ul>	<ul style="list-style-type: none"><li>• MAX is the declared size of vector</li><li>• Header vector included</li><li>• arr declared as vector&lt;int&gt;</li></ul>

So, if I do that, then once I have declared that, then I can use it exactly in the same way, there is no difference, I use it in the, I have not declared it with this array notation, but I can use it in the array notation. So, nothing else changes, is just the declaration that step and the same program works. So, MAX is the declared size of the array, MAX here is a declared size of the vector, there is no header included here the vector header is required, here you declare it as you know, and here you have to declare it as vector int.

So, kind of you can see that the type of the array and the type of the vector maps syntactically in this way. So, your question as to as to what great thing we have done. I mean, it is the same thing same index of notation. To see the advantages, we will now have to take the next example where we actually dynamically allocate an array.

(Refer Slide Time: 09:07)

C Program	C++ Program
<pre>// Array_Malloc.c #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt;  int main() { printf("Enter no. of elements "); int count, sum = 0, i; scanf("%d", &amp;count);  int *arr = (int*) malloc (sizeof(int)*count); for(i = 0; i &lt; count; i++) { arr[i] = i; sum += arr[i]; } printf("Array Sum:%d ", sum); }</pre>	<pre>// Array_Resize_c++.cpp #include &lt;iostream&gt; #include &lt;vector&gt; using namespace std;  int main() { cout &lt;&lt; "Enter the no. of elements: int count, sum=0; cin &gt;&gt; count;  vector&lt;int&gt; arr; // Default size arr.resize(count); // Set resize for(int i = 0; i &lt; arr.size(); i++) { arr[i] = i; sum += arr[i]; } cout &lt;&lt; "Array Sum: " &lt;&lt; sum &lt;&lt; endl; }</pre>
<pre>Enter no. of elements: 10 Array Sum: 45</pre>	<pre>Enter no. of elements: 10 Array Sum: 45</pre>
<ul style="list-style-type: none"> <li>• malloc allocates space using sizeof</li> </ul>	<ul style="list-style-type: none"> <li>• resize fixes vector size at run-time</li> </ul>

So, look at the C site included this to get malloc and this is what you know we have to do is we have to call malloc passing the size, the total size in bytes. So, where count is the number of elements that I get at the runtime, size of int is the size of every location. So, I multiply them to get the total number of bytes in the array. And the array actually is defined as a pointer to integer, because I could not have defined it as an array because I did not know the size.

But as you know due to pointer and array duality, I can use this in the array notation. Coming to C++, you do the same thing. It declares now you are not making it explicit dynamic allocation; you just declare the array. And unlike the previous time, we have not even provided a size MAX as we did earlier. Because if I do not provide, then it takes a default size, whatever the default size is.

Now, what I can do once I have got this count, which says how many elements I want, I can resize the array, I can resize the array, this is something which is very, very important that I can change the size. So, the default says, count here is 10, say suppose my default was 5, then the vector arr will be resized to fit 10 elements. It can be resized, suppose I started with a default of 1000 and my value is 10.

As I resize this 1000 will be reduced to 10 size vector and the remaining 990 elements will be released, because I do not need them, rest of the code remains the same. So, couple of advantages, one is for using malloc you have to do the size of and, its complicated syntax, a lot of things to be written. And then, you have to do a kind of a bypass by taking a pointer and then interpreting it is an array, here it is a straightforward, simple to write.

Second and most important thing is you can set the size by malloc only once because this is the time that you are allocating and getting space whereas here, you have first got there and then you are resizing it. So, it is possible that even going forward, if you decide to have more elements, or if you decide that, no, I do not need that much of it, you can again resize. So, at the runtime, you can change the size of the array as you need.

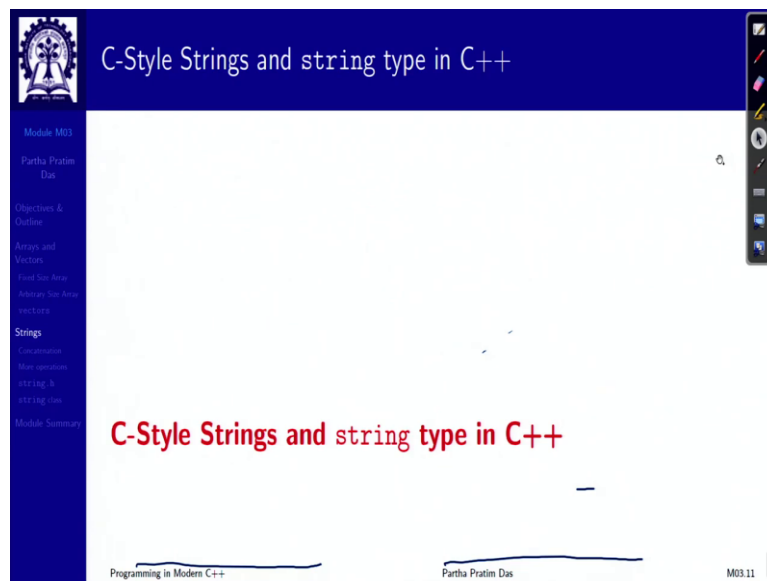
Those who have done dynamic arrays well, we will also say well, that is possible in C as well, I can do a realloc and so on. And but if you do that, yes, you can do that reallocation more or less, but then you will have to manage the entire thing of copying the original array into the relocated array and manage the pointers everything you have to do, here nothing you need to do. You say, it is a vector and resize. So, it resizes at runtime and makes your coding much easier.

So, whether it is a fixed size array, whether it is a array which is fixed size by manifest constants, or whether it could be potentially a dynamically managed array size, all of that can be handled by just this vector definition, which naturally makes arrays much easier or array like things much easier in C++. So very often, unless it is absolutely known that this array has this much of fixed size and so on, we will be using vector to make things much easier to deal with.

And mind you, the vector is as efficient as arrays are, there is hardly any overhead of using a vector over using traditional language defined array. So that is a big point to note, that is a big thing to learn that I can actually do all of these in terms of C++ and make any program because most programs will have arrays and we can make them simpler to read, write, and manage by using the vectors that are available in the standard library.

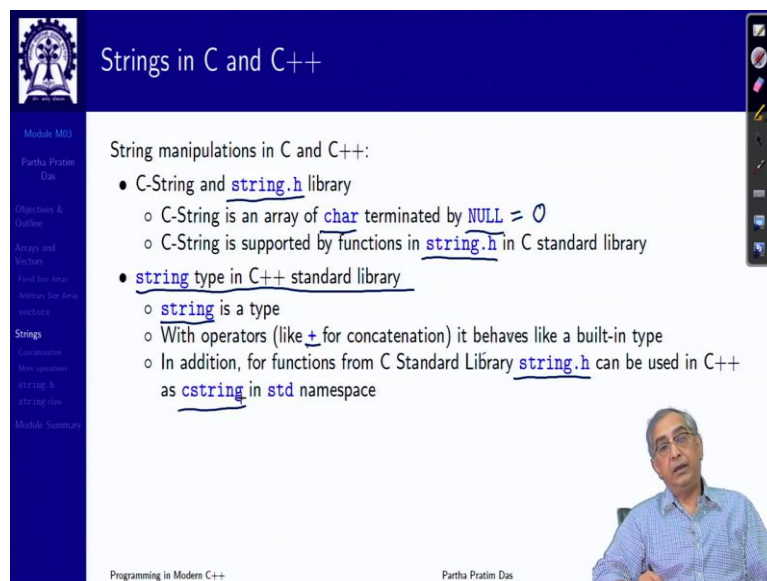


(Refer Slide Time: 14:50)



Next, move on to the next. Now move on to the next type which is string, string we all know are heavily, I am sorry.

(Refer Slide Time: 15:00)



So, strings will be manipulated in C and in C++ for in C as you know we have string.h, all of you know this, I am sure. And what is a C string, it is an array of character, array of character which is terminated by null which actually is defined to be 0. So, after 0 whatever is there will not be considered. And how does it become a string, it is not a string by type, it is just an array of characters. And by convention, you are saying that somewhere it will have a null character and that means, the end of this stream of characters as a string.

So, how does it become a string, it becomes a string by the way the different functions in string.h standard library component interprets it. So, you often do strlen to find out the length. What it does? It start from the index 0 of this array keeps on going till it finds the null character and as many characters are crossed over is returned as the value of the length. Similar things are done in strcpy, strcat, strdup, and so on.

So, the language does not have any support for string nor does the library has a support, as a string type as such, it just provides you a few functions and a convention of storing characters with a null termination which makes the whole story of string which is heavily, heavily used. Now, similar thing is provided in C++ in terms of a type which is still not in the language, but in the C++ standard library.

The standard library components string defines a string type, it is a type, and it is a type with very convenient operators like you can have + operator to concatenate two strings, its kind of equivalent of strcat, you have an assignment operator to do an strcpy and so on. In addition, in addition to that, you can use the C standard library string.h functions in C++ by including C string as in the std namespace. So, this is the overall story of strings between C and C++.

(Refer Slide Time: 17:52)

**Program 03.05: Concatenation of Strings**

C Program	C++ Program
<pre>// Add_strings.c #include &lt;stdio.h&gt; #include &lt;string.h&gt;  int main() { char str1[] = {'H','E','L','L','O',' ',' ','\0'}; char str2[] = "WORLD"; char str[20]; strcpy(str, str1); strcat(str, str2); printf("%s\n", str); }</pre>	<pre>// Add_strings_c++.cpp #include &lt;iostream&gt; #include &lt;string&gt; using namespace std;  int main(void) { string str1 = "HELLO"; string str2 = "WORLD"; string str = str1 + str2; cout &lt;&lt; str; }</pre>
HELLO WORLD	HELLO WORLD
<ul style="list-style-type: none"> <li>• Need header string.h ✓</li> <li>• C-String is an array of characters ✓</li> <li>• String concatenation done with strcat function ✓</li> <li>• Need a copy into str ✓</li> <li>• str must be large to fit the result ✓</li> </ul>	<ul style="list-style-type: none"> <li>• Need header string ✓</li> <li>• string is a data-type in C++ standard library ✓</li> <li>• Strings are concatenated like addition of int ✓</li> </ul>

Now, let us go with concatenation of strings that is. So, in C you have string.h here you have only string not C string, if I have hash include C string, then I will actually include the C standard library header which is strcat, strcpy, and all that, I do not want that, what I want is the C++ standard library string type, so I just include string.

Now, here I have initialized an array with these characters, initialized another array with these characters up just for the purpose of illustration I have shown two ways of initialization both of which are available in C++ as well. And then I want to, I declare an array which should be the concatenation of this HELLO followed by WORLD. So, to get there what do I do, I copy the string one into str which overrides everything in str and make it str1.

And then I do strcat, str2 which means after str1, remove the null terminator put the characters of str2 with the null terminator so that you get the whole string. Very simple algorithm, you all have written this, you know. In C++ this reduces to simply saying this str1 + str2 means concatenation. And then what you achieved in terms of strcpy and strcat is putting the values in that target array str, that is simply done by another operator assignment.

A string is a type here, so it supports + to mean concatenation, it supports assignment to mean overwriting a string and the name of the type of string. So, it is a huge advantage in terms of using this. So, if you compare there is a need for string.h in C, there is a need for string, again mind do not C string, C string is an array of characters, whereas string is a data type, which internally may keep a C string, we will talk about that later.

But so far as use is concerned, you can just treat it as a data type. String concatenation is done with strcat, need to copy into str, and str must be large enough to fit the size and the user has to take care of all of that. Here, all that you need is like addition of int, you do not have to bother what is the size of str and so on, str after the concatenation str is being initialized with that concatenated value.

So, it will be initialized with the proper size, everything is taken care of one line. So, you can see, when you have programs where you will have to do a lot of string manipulation, how easy it becomes when you deal with this string type. So that is a big plus so far as C++ is concerned.

(Refer Slide Time: 21:51)

Further,

- `operator=` can be used on strings in place of `strcpy` function in C
- `operator<=`, `operator<`, `operator>=`, `operator>` operators can be used on strings in place of `strcmp` function in C

Programming in Modern C++ Partha Pratim Das

Now, besides this + you obviously, we have talked about the assignment operator, which is the string copy, and you can compare using less than or equal to, less than, greater than or equal to, greater than and so on operators, which are equivalent of `strcmp` in C. So, several of these can be done as a part of the type itself.

(Refer Slide Time: 22:17)

Function	Description	Used Frequently?
<b>Copying:</b> <code>memcpy</code> ✓	Copy block of memory (function)	Yes
<code>memmove</code> ✓	Move block of memory (function)	Yes
<code>strcpy</code> ✓	Copy string (function)	Yes
<code>strncpy</code> ✓	Copy characters from string (function)	
<b>Concatenation:</b> <code>strcat</code> ✓	Concatenate strings (function)	Yes
<code>strncat</code>	Append characters from string (function)	
<b>Comparison:</b> <code>memcmp</code>	Compare two blocks of memory (function)	
<code>strcmp</code> ✓	Compare two strings (function)	Yes
<code>strcoll</code>	Compare two strings using locale (function)	
<code>strncmp</code>	Compare characters of two strings (function)	
<code>strxfrm</code>	Transform string using locale (function)	
<b>Searching:</b> <code>memchr</code>	Locate character in block of memory (function)	Yes
<code>strchr</code> ✓	Locate first occurrence of character in string (function)	Yes
<code>strcspn</code>	Get span until character in string (function)	
<code>strpbrk</code>	Locate characters in string (function)	
<code>strrchr</code>	Locate last occurrence of character in string (function)	
<code>strspn</code>	Get span of character set in string (function)	
<code>strstr</code> ✓	Locate substring (function)	Yes
<code>strtok</code> ✓	Split string into tokens (function)	Yes
<b>Other:</b> <code>memset</code>	Fill block of memory (function)	
<code>strerror</code>	Get pointer to error message string (function)	
<code>strlen</code> ✓	Get string length (function)	Yes
<b>Macros:</b> <code>NULL</code> ✓	Null pointer (macro)	Yes
<b>Types:</b> <code>size_t</code> ✓	Unsigned integral type (type)	Yes

Programming in Modern C++ Partha Pratim Das

So, here, I mean this is not for kind of memorizing but this is just to highlight to connect you to what are we talking about, this is your `string.h` standard library in C. So, these are the different functions and on the rightmost column I have put yes to mark that these are the functions which are more commonly used and as a C, C++ programmer, you must be familiar with those.

So, for example, you have things like memcpy, which is very common; memmove, these are these are actually not exactly string operations, but very useful, you can copy any chunk of memory buffer and so on. Then you have strcpy which you have you concatenate. Then you have comparison, you have strchr, finding out a character, finding out a token, the null macro itself which sets to 0, size\_t, which talks about integer size and so on.

Other functions are also important here, but these are some of the commonly used, frequently used functions which your kind of must know by use, others you can look up further as and when needed from the user documentation. I need to erase this.

(Refer Slide Time: 23:47)

**Strings in C and C++: C++ Standard Library string Class**

- Strings are objects that represent sequences of characters
- The standard string class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters
- The string class is an instantiation of the `basic_string` class template that uses `char` (that is, bytes) as its character type, with its default `char_traits` and `allocator` types

Function	Description (Member Function)	C Parallel
<i>Member functions</i>		
(constructor)	Construct string object (public)	Initialize <code>string</code> object with a C string
(destructor)	String destructor (public)	
operator=	String assignment (public)	<code>strcpy()</code> . <code>operator=</code> does shallow copy
<i>Iterators</i>		
begin	Return iterator to beginning (public)	
end	Return iterator to end (public)	
rbegin	Return reverse iterator to reverse beginning (public)	
rend	Return reverse iterator to reverse end (public)	
cbegin	Return const_iterator to beginning (public)	
end	Return const_iterator to end (public)	
crbegin	Return const_reverse_iterator to reverse beginning (public)	
crend	Return const_reverse_iterator to reverse end (public)	

Programming in Modern C++ | Partha Pratim Das | M03.16

Now, I am showing you the string in the C++ string library. Now, as we will see that this is a type, so it will have a constructor destructor which will understand over a period of time, but it is just to say that you can take a C string kind of string and initialize to create a string object. It does support assignment and it gives certain things called iterators they are nothing but, iterators are nothing but ways to go over the string.

So, if you have a string here and so, these are the different locations in the string, you can say that I put a marker here and I can say I can put a marker here. So, this is the beginning and this is the end of a simple iterator. So, I can say that I want to iterate from begin to end, which means exclude the end, but start from begin and go up to but not including the end.

So, this gives you, I mean it is kind of a higher level for loop you can think of, we will certainly explain this more, but these are common structures that you have in C++ standard library. So, you have varied forms of these iterators on the type, you can begin, go from begin

to end you can come a form reverse that is left to right, right to left, you can do it for constant arrays, you can do constant strings, you can do it for non-constant strings and so on so forth.

(Refer Slide Time: 25:58)

Function	Description (Member Function)	C Parallel
<b>Capacity</b>		
<code>size</code>	Return length of string (public)	<code>strlen()</code>
<code>length</code>	Return length of string (public)	<code>strlen()</code>
<code>max_size</code>	Return maximum size of string (public)	Fixed at allocation
<code>resize</code>	Resize string (public)	
<code>capacity</code>	Return size of allocated storage (public)	Need to be remembered in the
<code>reserve</code>	Request a change in capacity (public)	
<code>clear</code>	Clear string (public)	<code>strcpy()</code> an empty string
<code>empty</code>	Test if string is empty (public)	<code>strlen() == 0</code>
<code>shrink_to_fit</code>	Shrink to fit (public)	
<b>String operations</b>		
<code>c_str</code>	Get C string equivalent (public)	C string from a string object
<code>data</code>	Get string data (public)	
<code>get_allocator</code>	Get allocator (public)	
<code>copy</code>	Copy sequence of characters from string (public)	<code>strcpy()</code>
<code>find</code>	Find content in string (public)	<code>strchr(), strstr()</code>
<code>rfind</code>	Find last occurrence of content in string (public)	
<code>find_first_of</code>	Find character in string (public)	<code>strchr()</code>
<code>find_last_of</code>	Find character in string from the end (public)	<code>strrchr()</code>
<code>find_first_not_of</code>	Find absence of character in string (public)	
<code>find_last_not_of</code>	Find non-matching character in string from the end (public)	
<code>substr</code>	Generate substring (public)	<code>strcpy()</code>
<code>compare</code>	Compare strings (public)	<code>strcmp()</code>

So, there are some more of what the string type has, and on the rightmost column, and this is something which you will typically not find in regular documentation. In the rightmost column, I have tried to define map the corresponding C function in this string.h, wherever it is there. In some cases, it is there, in some cases, it is not there, because you can do things which are not, like `resize`, you cannot `resize` a string, but you can `resize` a string object. So, I have tried to map, give those mapping, these are for your references.

Again, not to memorize right away, but as and when you will start using lot of strings, you can find this. But the entire thing that I am trying to show is a string is a type in C++, which is not only easy to use, very powerful and it has features which go way beyond what your typical C string in string.h with its functions can do, but you can still free to use some of those functions if you really need to.

(Refer Slide Time: 27:08)

Function	Description (Member Function)	C Parallel
<i>Element access</i>		
<code>operator[]</code>	Get character of string (public)	<code>operator[]</code>
<code>at</code>	Get character in string (public)	<code>operator[]</code>
<code>back</code>	Access last character (public)	Character at <code>strlen()-1</code>
<code>front</code>	Access first character (public)	Character at 0 <sup>th</sup> location
<i>Modifiers</i>		
<code>operator+=</code>	Append to string (public)	<code>strcat()</code>
<code>append</code>	Append to string (public)	<code>strcat()</code>
<code>push_back</code>	Append character to string (public)	Set character to <code>strlen()</code> and <code>NULL</code> to next location
<code>assign</code>	Assign content to string (public)	
<code>insert</code>	Insert into string (public)	
<code>erase</code>	Erase characters from string (public)	
<code>replace</code>	Replace portion of string (public)	
<code>swap</code>	Swap string values (public)	Character by character swapping between two arrays
<code>pop_back</code>	Delete last character (public)	Set location <code>strlen()-1</code> to <code>NULL</code>
<i>Member constants</i>		
<code>npos</code>	Maximum value for <code>size_t</code> (public static)	
<i>Non-member function overloads</i>		
<code>operator+</code>	Concatenate strings (global)	<code>strcat()</code>
<i>relational operators</i>	Relational operators for string (global)	<code>strcmp()</code> followed by <code>test</code> for -
<code>swap</code>	Exchanges the values of two strings (global)	
<code>operator&gt;&gt;</code>	Extract string from stream (global)	<code>format %s</code>
<code>operator&lt;&lt;</code>	Insert string into stream (global)	<code>format %s</code>
<code>getline</code>	Get line from stream into string (global)	<code>getline()</code> in <code>&lt;std&gt;</code>

This is continuation of that same list. And you can do a lot of things like appending, like putting a character at the end, so on so forth, a lot of stuff.

(Refer Slide Time: 27:24)

Module Summary
<ul style="list-style-type: none"> <li>Working with variable sized arrays is more flexible with vectors in C++</li> <li>String operations are easier with C++ standard library</li> </ul>

So, this brings us to the end of this module, where we have primarily focused on handling of arrays and the convenience of using vectors in C++, which is equally efficient and can be seamlessly used in the same form, whether it is fixed size, whether it is small, whether it is large, or whether it is dynamically runtime resized, all of these can be handled in the same syntax in the same structure using vectors which is not so in C, you have to keep track of the size from the static time or do a explicit complicated dynamic allocation using malloc and all that.

And further, we took a look at the string type or string operations in C and C++ when we saw that C++ standard library defines a string type as string component in the standard library, which is very, very useful and lot more compact than our C string equivalent in the string.h. Thank you very much for your attention, and we will meet in the next module.