

MODULE 25

C++ CHARACTER AND STRING MANIPULATION PART I

My Training Period: hours

Note:

Program examples in this Module compiled using Visual C++ 6.0 with SP6 and Visual C++ .Net. [g++](#) (GNU C++ run on my Fedora 3 machine) example is given at the end of this Module. Take note also for the codes that span more than one line, which they are not supposed to.

Abilities

- Able to understand and appreciate how the templates are used.
- Able to understand and use the template based C++ headers.
- Able to understand and use string template classes in manipulating character and string in C++.
- Able to understand and use the `basic_string` template class.
- Able to recognize the Containers, Iterators and Algorithms.

25.1 Introduction

- Before we 'jump' into the real STL 'bandwagon', let try the `<string>` C++ Standard Library. This library is constructed using template classes and functions.
- In this Module take note for these words: **container**, **iterator** and **algorithm**. See how these things used in the program examples.
- We will discuss these 'creatures' in more detail later on. May be after completing this Module, you may develop your own simple search routine :o) program.
- Your main task here is 'learn how to use'.

25.2 `<string>` C++ Standard Library

- `<string>` defines the container template class `basic_string` and various supporting templates. You must include the `<string>` header in your program as shown below.

```
#include <string>
```

- The C++ language supports two types of strings:
 - Null-terminated character arrays often referred to as C strings and,
 - Template class objects, of type `basic_string` that handles all char-like template arguments.

25.3 `<string>` Typedefs

- The following table is a list of the `<string>` typedefs.

Type	Description
string typedef basic_string<char> string; e.g. const basic_string <char> str1("TEST"); or string str2("TEST");	A type that describes a specialization of the template class <code>basic_string</code> with elements of type char as a string . This means <code>basic_string<char></code> is synonym to <code>string</code> .
wstring typedef basic_string<wchar_t> wstring; e.g. const basic_string <wchar_t> str1(L"EST"); or wstring str2(L"EST");	A type that describes a specialization of the template class <code>basic_string</code> with elements of type wchar_t as a wstring . This means <code>basic_string<wchar_t></code> is synonym to <code>wstring</code> . This is wide char, normally 2 byte such as Unicode character set (more information on this is discussed HERE and the implementation is HERE).

Table 25.1 string and wstring typedefs

```
//char and wchar_t types
#include <string>
#include <iostream>
using namespace std;

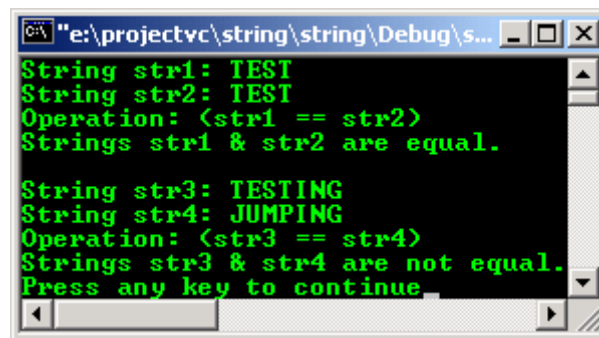
int main( )
{
    const basic_string <char> str1("TEST");
    //Uses the typedef for string word
    //synonym to basic_string <char>
    string str2("TEST");

    //simple comparison between two
    //objects of type basic_string
    cout<<"String str1: "<<str1<<"\nString str2: "<<str2<<endl;
    cout<<"Operation: (str1 == str2)"<<endl;
    if(str1 == str2)
        cout<<"Strings str1 & str2 are equal."<<endl;
    else
        cout<<"Strings str1 & str2 are not equal."<<endl;

    //L - literal qualifier, long
    const basic_string <wchar_t> str3(L"TESTING");
    //Uses the typedef for wstring word
    //synonym to basic_string <wchar_t>
    wstring str4(L"JUMPING");

    //simple comparison between two objects
    //of type basic_string <wchar_t>
    cout<<"\nString str3: TESTING \nString str4: JUMPING"<<endl;
    cout<<"Operation: (str3 == str4)"<<endl;
    if(str3 == str4)
        cout<<"Strings str3 & str4 are equal."<<endl;
    else
        cout<<"Strings str3 & str4 are not equal."<<endl;
    return 0;
}
```

Output:



```
"e:\projectvc\string\string\Debug\s...
String str1: TEST
String str2: TEST
Operation: (str1 == str2)
Strings str1 & str2 are equal.

String str3: TESTING
String str4: JUMPING
Operation: (str3 == str4)
Strings str3 & str4 are not equal.
Press any key to continue
```

25.4 <string> Operators

- String operators are overloaded operators.
- The comparison between string objects is based on what is called a pairwise lexicographical comparison of their characters. Two strings are equal if they have the same number of characters and their respective character values are the same. Otherwise, they are unequal.
- A lexicographical comparison between strings compares them character by character until:
 - It finds two corresponding characters unequal, and the result of their comparison is taken as the result of the comparison between the strings.
 - It finds no inequalities, but one string has more characters than the other, and the shorter string is considered less than the longer string.
 - It finds no inequalities and finds that the strings have the same number of characters, and so the strings are equal.
- The following table is a list of string operators.

Operator	Brief Description
operator!= str1 != str2	Tests if the string object on the left side of the operator is not equal to the string object on the right side.
operator== str1 == str2	Tests if the string object on the left side of the operator is equal to the string object on the right side.
operator< str1 < str2	Tests if the string object on the left side of the operator is less than to the string object on the right side.
operator<< same as in iostream, e.g. cout<<	A template function that inserts a string into the output stream.
operator<= str1 <= str2	Tests if the string object on the left side of the operator is less than or equal to the string object on the right side.
operator> str1 > str2	Tests if the string object on the left side of the operator is greater than to the string object on the right side.
operator>= str1 >= str2	Tests if the string object on the left side of the operator is greater than or equal to the string object on the right side.
operator>> same as in iostream, e.g. cin>>	A template function that extracts a string from the input stream.
operator+ string str13 = str1 + str3	Concatenates two string objects.

Table 25.2: string operators

- Some of the program examples using string operators.

```
//== and != operators
#include <string>
#include <iostream>
using namespace std;

int main( )
{
    //Declaring an objects of type basic_string<char>
    string str1("testingone");
    string str2("testingtwo");
    cout<<"str1 string is = "<<str1<<endl;
    cout<<"str2 string is = "<<str2<<endl;
    //Declaring a C-style string
    char *str3 = "testingone";
    cout<<"C-style str3 string is = "<<str3<<endl;

    //Comparison between left-side object of type basic_string
    //& right-side object of type basic_string
    cout<<"\nOperation: (str1 != str2)"<<endl;
    if(str1 != str2)
        cout<<"str1 & str2 are not equal."<<endl;
    else
        cout<<"str1 & str2 are equal."<<endl;

    //Comparison between left-side object of C-style string
    //type & right-side object of type basic_string
    cout<<"\nOperation: (str3 != str2)"<<endl;
    if(str3 != str2)
        cout<<"str3 & str2 are not equal."<<endl;
    else
        cout<<"str3 & str2 are equal."<<endl;

    //Comparison between left-side object of type basic_string
    //& right-side object of C-style string type
    cout<<"\nOperation: (str1 != str3)"<<endl;
    if(str1 != str3)
        cout<<"str1 & str3 are not equal."<<endl;
    else
        cout<<"str1 & str3 are equal."<<endl;

    //Comparison between left-side object of type basic_string
    //& right-side object of type basic_string
    cout<<"\nOperation: (str1 == str2)"<<endl;
    if(str1 == str2)
        cout<<"str1 & str2 are equal."<<endl;
    else
```

```

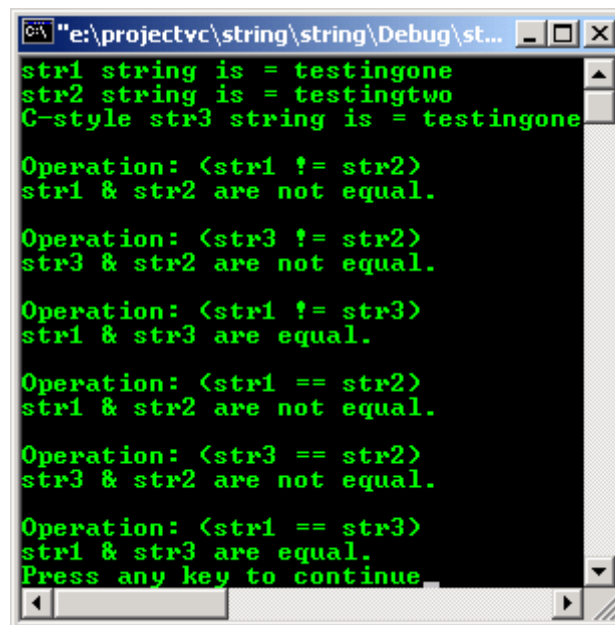
cout<<"str1 & str2 are not equal."<<endl;

//Comparison between left-hand object of C-style string type
//& right-hand object of type basic_string
cout<<"\nOperation: (str3 == str2)"<<endl;
if(str3 == str2)
cout<<"str3 & str2 are equal."<<endl;
else
cout<<"str3 & str2 are not equal."<<endl;

//Comparison between left-hand object of type basic_string
//& right-hand object of C-style string type
cout<<"\nOperation: (str1 == str3)"<<endl;
if(str1 == str3)
cout<<"str1 & str3 are equal."<<endl;
else
cout<<"str1 & str3 are not equal."<<endl;
return 0;
}

```

Output:



```

str1 string is = testingone
str2 string is = testingtwo
C-style str3 string is = testingone

Operation: <str1 != str2>
str1 & str2 are not equal.

Operation: <str3 != str2>
str3 & str2 are not equal.

Operation: <str1 != str3>
str1 & str3 are equal.

Operation: <str1 == str2>
str1 & str2 are not equal.

Operation: <str3 == str2>
str3 & str2 are not equal.

Operation: <str1 == str3>
str1 & str3 are equal.
Press any key to continue

```

```

//< and > operators
#include <string>
#include <iostream>
using namespace std;

int main()
{
//Declaring objects of type basic_string<char>
string str1("testingthree");
string str2("testingtwo");
cout<<"str1 is = "<<str1<<endl;
cout<<"str2 is = "<<str2<<endl;

//Declaring a C-style string
char *str3 = "testingone";
cout<<"str3 C-style string is = "<<str3<<endl;

//Comparison between left-side object of type basic_string
//& right-side object of type basic_string
cout<<"\nOperation: (str1 < str2)"<<endl;
if(str1 < str2)
    cout<<"str1 is less then string str2."<<endl;
else
    cout<<"str1 is not less then string str2."<<endl;

//Comparison between left-side object of C-style string
//type & right-side object of type basic_string
cout<<"\nOperation: (str3 < str2)"<<endl;
if(str3 < str2)
    cout<<"str3 is less then string str2."<<endl;
}

```

```

else
    cout<<"str3 is not less then string str2."<<endl;

//Comparison between left-side object of type basic_string
//& right-side object of C-style string type
cout<<"\nOperation: (str1 < str3)"<<endl;
if(str1 < str3)
    cout<<"str1 is less then string str3."<<endl;
else
    cout<<"str1 is not less then string str3."<<endl;

//Comparison between left-side object of type basic_string
//& right-side object of type basic_string
cout<<"\nOperation: (str1 > str2)"<<endl;
if(str1 > str2)
    cout<<"str1 is greater then string str2."<<endl;
else
    cout<<"str1 is not greater then string str2."<<endl;

//Comparison between left-hand object of C-style string type
//& right-hand object of type basic_string
cout<<"\nOperation: (str3 > str2)"<<endl;
if(str3 > str2)
    cout<<"str3 is greater then string str2."<<endl;
else
    cout<<"str3 is not greater then string str2."<<endl;

//Comparison between left-hand object of type basic_string
//& right-hand object of C-style string type
cout<<"\nOperation: (str1 > str3)"<<endl;
if(str1 > str3)
    cout<<"str1 is greater then string str3."<<endl;
else
    cout<<"str1 is not greater then string str3."<<endl;
return 0;
}

```

Output:

```

e:\projectvc\string\string\Debug\strin...
str1 is = testingthree
str2 is = testingtwo
str3 C-style string is = testingone

Operation: <str1 < str2>
str1 is less then string str2.

Operation: <str3 < str2>
str3 is less then string str2.

Operation: <str1 < str3>
str1 is not less then string str3.

Operation: <str1 > str2>
str1 is not greater then string str2.

Operation: <str3 > str2>
str3 is not greater then string str2.

Operation: <str1 > str3>
str1 is greater then string str3.
Press any key to continue

```

```

//>= and <= operators
#include <string>
#include <iostream>
using namespace std;

int main( )
{
    //Declaring an objects of type basic_string<char>
    string str1("testingone");
    string str2("testingtwo");
    cout<<"str1 string is = "<<str1<<endl;
    cout<<"str2 string is = "<<str2<<endl;
    //Declaring a C-style string

```

```

char *str3 = "testingone";
cout<<"str3 C-style string is = "<<str3<<endl;

//Comparison between left-side object of type basic_string
//& right-side object of type basic_string
cout<<"\nOperation: (str1 <= str2)"<<endl;
if(str1 <= str2)
    cout<<"str1 is less than or equal to str2."<<endl;
else
    cout<<"str1 is not less than or equal to str2."<<endl;

//Comparison between left-side object of C-style string
//type & right-side object of type basic_string
cout<<"\nOperation: (str3 <= str2)"<<endl;
if(str3 <= str2)
    cout<<"str3 is less than or equal to str2."<<endl;
else
    cout<<"str3 is not less than or equal to str2."<<endl;

//Comparison between left-side object of type basic_string
//& right-side object of C-style string type
cout<<"\nOperation: (str1 <= str3)"<<endl;
if(str1 <= str3)
    cout<<"str1 is less than or equal to str3."<<endl;
else
    cout<<"str1 is not less than or equal to str3."<<endl;

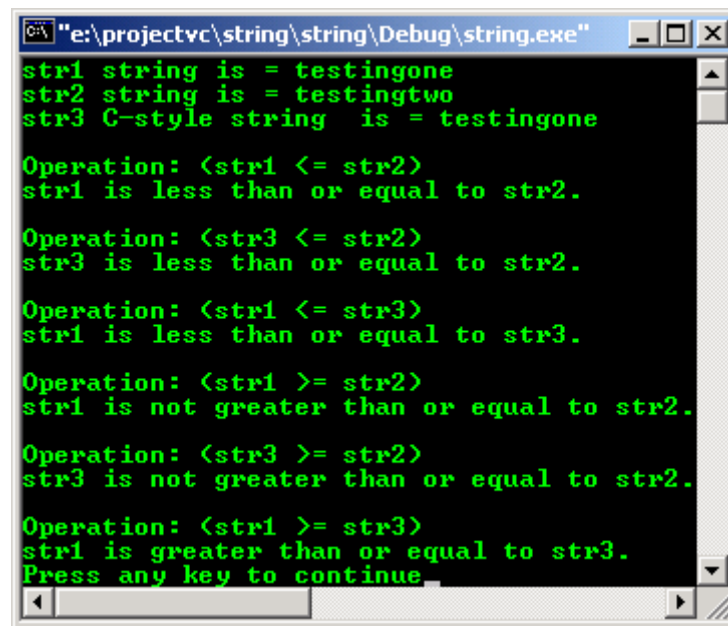
//Comparison between left-side object of type basic_string
//& right-side object of type basic_string
cout<<"\nOperation: (str1 >= str2)"<<endl;
if(str1 >= str2)
    cout<<"str1 is greater than or equal to str2."<<endl;
else
    cout<<"str1 is not greater than or equal to str2."<<endl;

//Comparison between left-hand object of C-style string type
//& right-hand object of type basic_string
cout<<"\nOperation: (str3 >= str2)"<<endl;
if(str3 >= str2)
    cout<<"str3 is greater than or equal to str2."<<endl;
else
    cout<<"str3 is not greater than or equal to str2."<<endl;

//Comparison between left-hand object of type basic_string
//& right-hand object of C-style string type
cout<<"\nOperation: (str1 >= str3)"<<endl;
if(str1 >= str3)
    cout<<"str1 is greater than or equal to str3."<<endl;
else
    cout<<"str1 is not greater than or equal to str3."<<endl;
return 0;
}

```

Output:



```
e:\projectvc\string\string\Debug\string.exe
str1 string is = testingone
str2 string is = testingtwo
str3 C-style string is = testingone

Operation: <str1 <= str2>
str1 is less than or equal to str2.

Operation: <str3 <= str2>
str3 is less than or equal to str2.

Operation: <str1 <= str3>
str1 is less than or equal to str3.

Operation: <str1 >= str2>
str1 is not greater than or equal to str2.

Operation: <str3 >= str2>
str3 is not greater than or equal to str2.

Operation: <str1 >= str3>
str1 is greater than or equal to str3.
Press any key to continue
```

- The operator skips the leading white spaces unless the skipws flag is set. It reads all the following characters until the next character is a white space or the end of the file is reached.

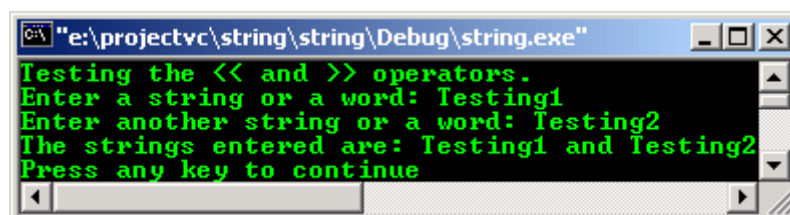
```
//<< and >> operators
#include <string>
#include <iostream>
using namespace std;

int main()
{
    string Sample = "Testing the << and >> operators.";
    string Var1, Var2;

    cout<<Sample<<endl;
    cout<<"Enter a string or a word: ";
    cin>>Var1;
    cout<<"Enter another string or a word: ";
    cin>>Var2;
    cout<<"The strings entered are: "<<Var1<<" and "<<Var2<<endl;

    return 0;
}
```

Output:



```
e:\projectvc\string\string\Debug\string.exe
Testing the << and >> operators.
Enter a string or a word: Testing1
Enter another string or a word: Testing2
The strings entered are: Testing1 and Testing2
Press any key to continue
```

```
//concatenating using '+' operator
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //Declaring an object of type basic_string<char>
    string str1("StringOne");
    string str2("StringTwo");
    //Declaring a C-style string
    char *str3 = "StringThree";
    //Declaring a character constant
    char chr = '?';

    cout<<"str1 string is = "<<str1<<endl;
```

```

cout<<"str2 string is = "<<str2<<endl;
cout<<"str3 C-style string is = "<<str3<<endl;
cout<<"A character constant chr is = "<<chr<<endl;

//Concatenates an object of type basic_string
//with an object of type basic_string
cout<<"\nOperation: str12 = str1 + str2"<<endl;
string str12 = str1 + str2;
cout<<"str12 = "<<str12<<endl;

//Concatenates an object of type basic_string
//with an object of C-style string type
cout<<"\nOperation: str13 = str1 + str3"<<endl;
string str13 = str1 + str3;
cout<<"str13 = "<<str13<<endl;

//Concatenates an object of type basic_string
//with a character constant
cout<<"\nOperation: str13chr = str13 + chr"<<endl;
string str13chr = str13 + chr;
cout<<"str13chr = "<<str13chr<<endl;
return 0;
}

```

Output:

```

C:\e:\projectvc\string\string\Debug\stri...
str1 string is = StringOne
str2 string is = StringTwo
str3 C-style string is = StringThree
A character constant chr is = ?

Operation: str12 = str1 + str2
str12 = StringOneStringTwo

Operation: str13 = str1 + str3
str13 = StringOneStringThree

Operation: str13chr = str13 + chr
str13chr = StringOneStringThree?
Press any key to continue

```

25.5 <string> Specialized Template Functions

Specialized Template Function	Description
swap()	Exchanges the arrays of characters of two strings.

Table 25.3: swap() function

- If the strings being swapped have the same allocator object, the swap() member function:
 - Occurs in constant time.
 - Throws no exceptions.
 - Invalidates no references, pointers, or iterators that designate elements in the two strings.
- Otherwise, it performs a number of element assignments and constructor calls proportional to the number of elements in the two controlled sequences.

```

//swap()
#include <string>
#include <iostream>
using namespace std;

int main()
{
//Declaring an object of type basic_string<char>
string str1("StringOne");
string str2("StringTwo");

```



```

cout<<"Before swapping string str1 and str2:"<<endl;
cout<<"str1 string is = "<<str1<<endl;
cout<<"str2 string is = "<<str2<<endl;

swap(str1, str2);
cout<<"\nOperation: swap(str1, str2)"<<endl;
cout<<"After swapping string str1 and str2:"<<endl;
cout<<"str1 string is = "<<str1<<endl;
cout<<"str2 string is = "<<str2<<endl;
return 0;
}

```

Output :

```

e:\projectvc\string\string\Debug\string.exe
Before swapping string str1 and str2:
str1 string is = StringOne
str2 string is = StringTwo

Operation: swap(str1, str2)
After swapping string str1 and str2:
str1 string is = StringTwo
str2 string is = StringOne
Press any key to continue

```

25.6 <string> Functions

Function	Description
getline()	<p>The getline() function creates a string containing all of the characters from the input stream until one of the following situations occurs:</p> <ul style="list-style-type: none"> - End of file. - The delimiter is encountered. - is.max_str elements have been extracted.

Table 25.4: getline() function

- Program example

```

//getline()
#include <string>
#include <iostream>
using namespace std;

int main()
{
    string str;
    string str1;
    string str2;
    cout<<"Enter a line of text: ";
    getline(cin, str);
    cout<<"You entered: "<<str<<endl;
    cout<<"Enter a line of text, <space> as the delimiter: "<<endl;
    getline(cin, str1, ' ');
    cout<<"You entered: "<<str1<<endl;
    return 0;
}

```

Output :

```

e:\projectvc\string\string\Debug\string.exe
Enter a line of text: Line of text
You entered: Line of text
Enter a line of text, <space> as the delimiter:
Another line of text
You entered: Another
Press any key to continue

```

25.7 <string> Template Classes

- With `<string>`, we are provided with two string template classes as shown in the following table.

Class	Description
<code>basic_string</code>	A template class that describes objects that can store a sequence of arbitrary character-like objects.
<code>char_traits</code>	A template class that describes attributes associated with a character of type <code>CharType</code>

Table 25.5: `<string>` classes

25.8 `basic_string` Template Class

- The sequences controlled by an object of template class `basic_string` are the Standard C++ string class and are usually referred to as strings, but they should not be confused with the null-terminated C-strings used throughout the Standard C++ Library.
- The `string` class is a **container** that enables the use of strings as normal types, such as using comparison and concatenation operations, **iterators** and STL **algorithms**. The `basic_string` template structure is shown below.

```
template <
    class CharType,
    class Traits = char_traits<CharType>,
    class Allocator = allocator<CharType>
>
```

- Where:

Entity	Description
<i>CharType</i>	The data type of a single character to be stored in the string. The Standard C++ Library provides two specializations of this template class: 1. <code>string</code> , for elements of type <code>char</code> , 2. <code>wstring</code> , for elements of type <code>wchar_t</code> .
<i>Traits</i>	Various important properties of the <code>CharType</code> elements in a <code>basic_string</code> specialization are described by the class <code>Traits</code> .
<i>Allocator</i>	The type that represents the stored allocator object that encapsulates details about the string's allocation and de-allocation of memory. The default value is <code>allocator<Type></code> .

Table 25.6: `basic_string` template parameters

25.9 `basic_string` Template Class Typedef

- The following table is the list of the typedef used in `basic_string` template class. Their usages are presented in the program examples part.

Typedef	Brief Description
<code>allocator_type</code>	A type that represents the allocator class for a string object. The type is a synonym for the template parameter <code>Allocator</code> .
<code>const_iterator</code>	A type that provides a random-access iterator that can access and read a const element in the string. A type <code>const_iterator</code> cannot be used to modify the value of a character and is used to iterate through a string in a forward direction.
<code>const_pointer</code>	A type that provides a pointer to a const element in a string. The type is a synonym for <code>allocator_type::const_pointer</code> . For type <code>string</code> , it is equivalent to <code>char*</code> . Pointers that are declared const must be initialized when they are declared. const pointers always point to the same memory location and may point to constant or non constant data.
<code>const_reference</code>	A type that provides a reference to a const element stored in a string for reading and performing const operations. A type <code>const_reference</code> cannot be used to modify the value of an element. The type is a synonym for <code>allocator_type::const_reference</code> . For <code>string</code> type, it is equivalent to <code>const char&</code> .

const_reverse_iterator	A type that provides a random-access iterator that can read any const element in the string. A type const_reverse_iterator cannot modify the value of a character and is used to iterate through a string in reverse.
difference_type	A type that provides the difference between two iterators those refer to elements within the same string. The signed integer type describes an object that can represent the difference between the addresses of any two elements in the controlled sequence. For type string, it is equivalent to ptrdiff_t.
iterator	A type that provides a random-access iterator that can read or modify any element in a string. A type iterator can be used to modify the value of a character and is used to iterate through a string in a forward direction.
npos	An unsigned integral value initialized to -1 that indicates either "not found" or "all remaining characters" when a search function fails. When the return value is to be checked for the npos value, it might not work unless the return value is of type size_type and not either int or unsigned.
pointer	A type that provides a pointer to a character element in a string or character array. The type is a synonym for allocator_type::pointer. For type string, it is equivalent to char*.
reference	A type that provides a reference to an element stored in a string. A type reference can be used to modify the value of an element. The type is a synonym for allocator_type::reference. For type string, it is equivalent to chr&.
reverse_iterator	A type that provides a random-access iterator that can read or modify an element in a reversed string. A type reverse_iterator can be used to modify the value of a character and is used to iterate through a string in reverse.
size_type	An unsigned integral type for the number of elements in a string. It is equivalent to allocator_type::size_type. For type string, it is equivalent to size_t.
traits_type	A type for the character traits of the elements stored in a string. The type is a synonym for the second template parameter Traits. For type string, it is equivalent to char_traits<char>.
value_type	A type that represents the type of characters stored in a string. It is equivalent to traits_type::char_type and is equivalent to char for objects of type string.

Table 25.7: basic_string typedefs

25.10 basic_string Template Class Member Functions

- The following table is a list of member functions available in basic_string template class.

Member Function	Brief Description
append()	Adds characters to the end of a string.
assign()	Assigns new character values to the contents of a string.
at()	Returns a reference to the element at a specified location in the string.
basic_string()	Constructs a string that is empty or initialized by specific characters or that is a copy of all or part of some other string object or C-string.
begin()	Returns an iterator addressing the first element in the string.
c_str()	Converts the contents of a string as a C-style, null-terminated, string.
capacity()	Returns the largest number of elements that could be stored in a string without increasing the memory allocation of the string.
clear()	Erases all elements of a string.
compare()	Compares a string with a specified string to determine if the two strings are equal or if one is lexicographically less than the other.
copy()	Copies at most a specified number of characters from an indexed

	position in a source string to a target character array.
<code>data()</code>	Converts the contents of a string into an array of characters.
<code>empty()</code>	Tests whether the string contains characters or not.
<code>end()</code>	Returns an iterator that addresses the location succeeding the last element in a string.
<code>erase()</code>	Removes an element or a range of elements in a string from specified positions. Notice the different with <code>clear()</code> .
<code>find()</code>	Searches a string in a forward direction for the first occurrence of a substring that matches a specified sequence of characters.
<code>find_first_not_of()</code>	Searches through a string for the first character that is not any element of a specified string.
<code>find_first_of()</code>	Searches through a string for the first character that matches any element of a specified string.
<code>find_last_not_of()</code>	Searches through a string for the last character that is not any element of a specified string.
<code>find_last_of()</code>	Searches through a string for the last character that is an element of a specified string.
<code>get_allocator()</code>	Returns a copy of the allocator object used to construct the string.
<code>insert()</code>	Inserts an element or a number of elements or a range of elements into the string at a specified position.
<code>length()</code>	Returns the current number of elements in a string.
<code>max_size()</code>	Returns the maximum number of characters a string could contain.
<code>push_back()</code>	Adds an element to the end of the string.
<code>rbegin()</code>	Returns an iterator to the first element in a reversed string.
<code>rend()</code>	Returns an iterator that point just beyond the last element in a reversed string.
<code>replace()</code>	Replaces elements in a string at a specified position with specified characters or characters copied from other ranges or strings or C-strings.
<code>reserve()</code>	Sets the capacity of the string to a number at least as great as a specified number.
<code>resize()</code>	Specifies a new size for a string, appending or erasing elements as required.
<code>rfind()</code>	Searches a string in a backward direction for the first occurrence of a substring that matches a specified sequence of characters.
<code>size()</code>	Returns the current number of elements in a string.
<code>substr()</code>	Copies a substring of at most some number of characters from a string beginning from a specified position.
<code>swap()</code>	Exchange the contents of two strings.

Table 25.8: `basic_string` member functions

25.11 `basic_string` Template Class Operators

- The following table is a list of the operators available in `basic_string` template class.

Operator	Brief Description
<code>operator+=</code>	Appends characters to a string.
<code>operator=</code>	Assigns new character values to the contents of a string.
<code>operator[]</code>	Provides a reference to the character with a specified index in a string.

Table 25.9: `basic_string` operator

25.12 `basic_string` Program Examples

- Let try program examples demonstrating the `basic_string` template class member functions and other *creatures* readily available for us.

append()

- Characters may be appended to a string using the operator+= or the member functions append() or push_back().
- operator+= appends single-argument values while the multiple-argument append() member function allows a specific part of a string to be specified for adding.

```
//append()
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //appending a C-string to a string
    string str1("Playing ");
    const char *str2 = "with a string";

    cout<<"str1 is: "<<str1<<endl;
    cout<<"str2, C string is: "<<str2<<endl;
    str1.append(str2);
    cout<<"Operation: str1.append(str2)"<<endl;
    cout<<"Appending str2 to str1: "<<str1<<endl;

    //appending part of a C-string to a string
    string str3 ("Replaying ");
    const char *str4 = "the string ";

    cout<<"\nstr3 string is: "<<str3<<endl;
    cout<<"str4 C-string is: "<<str4<<endl;
    str3.append(str4, 6);
    cout<<"Operation: str3.append(str4, 6)"<<endl;
    cout<<"Appending part of the str4 to string str3: \n"<<str3<<endl;

    //appending part of one string to another
    string str5("Again "), str6("string manipulation");

    cout<<"\nstr5 is: "<<str5<<endl;
    cout<<"str6 is: "<<str6<<endl;
    str5.append(str6, 4, 6);
    cout<<"Operation: str5.append(str6, 4, 6)"<<endl;
    cout<<"The appended string is: "<<str5<<endl;

    //appending one string to another in two ways,
    //comparing append and operator []
    string str7("First "), str8("Second "), str9("Third ");
    cout<<"\nstr7 is: "<<str7<<"\nstr8 is: "<<str8<<"\nstr9 is: "<<str9<<endl;
    str7.append(str8);
    cout<<"Operation: str7.append(str8)"<<endl;
    cout<<"The appended string str7 is: "<<str7<<endl;
    str7 += str9;
    cout<<"Operation: str7 += str9"<<endl;
    cout<<"The re appended string is: "<<str7<<endl;

    //appending characters to a string
    string str10("What string");
    cout<<"\nstr10 string is: "<<str10<<endl;
    str10.append(3, '?');
    cout<<"Operation: str10.append(3, '?)"<<endl;
    cout<<"str10 string appended with ? is: "<<str10<<endl;

    //appending a range of one string to another
    string str11("Finally "), str12("comes the END ");
    cout<<"\nstr11 is: "<<str11<<" str12 is: "<<str12<<endl;
    str11.append(str12.begin() + 6, str12.end() - 1);
    cout<<"Operation:\nstr11.append(str12.begin() + 6, str12.end() - 1)"<<endl;
    cout<<"The appended str11 String is: "<<str11<<endl;
    return 0;
}
```

Output:

```
"e:\projectvc\string\string\Debug\string.exe"
str1 is: Playing
str2, C string is: with a string
Operation: str1.append(str2)
Appending str2 to str1: Playing with a string

str3 string is: Replaying
str4 C-string is: the string
Operation: str3.append(str4, 6)
Appending part of the str4 to string str3:
Replaying the st

str5 is: Again
str6 is: string manipulation
Operation: str5.append(str6 ,4 ,6)
The appended string is: Again ng man

str7 is: First
str8 is: Second
str9 is: Third
Operation: str7.append(str8)
The appended string str7 is: First Second
Operation: str7 += str9
The re appended string is: First Second Third

str10 string is: What string
Operation: str10.append(3, '?')
str10 string appended with ? is: What string???

str11 is: Finally str12 is: comes the END
Operation:
str11.append(str12.begin() + 6 , str12.end() - 1)
The appended str11 String is: Finally the END
Press any key to continue
```

assign()

- The strings can be assigned new character values. The new value can be either a string and C-string or a single character.
- The operator= may be used if the new value can be described by a single parameter; otherwise the member function `assign()`, which has multiple parameters, can be used to specify which part of the string is to be assigned to a target string.

```
//assign(), string assignment
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //assigning the characters of a C-string to a string
    string str1;
    const char *str2 = "StRiNg assign()";

    cout<<"str2, C string is: "<<str2<<endl;
    str1.assign(str2);
    cout<<"Operation: str1.assign(str2)"<<endl;
    cout<<"Assigning the C-string str2 to str1 string: \n"<<str1<<endl;

    //assigning a number of a C-string characters to a string
    string str3;
    const char *str4 = "Another StRiNg assign()";
    cout<<"\nstr4 C string is: "<<str4<<endl;
    str3.assign(str4, 11);
    cout<<"Operation: str3.assign(str4, 11)"<<endl;
    cout<<"Assigning some portion of the str4 "
        <<"to str3 string: \n"<<str3<<endl;

    //assigning a number of the characters from one string to another string
    string str5("First "), str6("Second sTrInG");
    cout<<"\nstr5 string is: "<<str5<<endl;
    cout<<"str6 string is: "<<str6<<endl;
    str5.assign(str6, 7, 6);
    cout<<"Operation: str5.assign(str6, 7, 6)"<<endl;
```

```

cout<<"Newly assigned str5 string is: "<<str5<<endl;

//assigning the characters from one string to another string
//in two equivalent ways, comparing the assign and operator =
string str7("First"), str8("Second"), str9("Third");
cout<<"\nstr7 string is: "<<str7<<endl;
cout<<"str8 string is: "<<str8<<endl;
cout<<"str9 string is: "<<str9<<endl;
str7.assign(str8);
cout<<"Operation: str7.assign(str8)"<<endl;
cout<<"Newly assigned str7 with str8 string is: "<<str7<<endl;
str7 = str9;
cout<<"Operation: str7 = str9"<<endl;
cout<<"String str7 reassigned with str9 string is: "<<str7<<endl;

//assigning a specific number of characters of a certain value to a string
string str10("Working STRInG");
cout<<"\nstr10 string is: "<<str10<<endl;
str10.assign(3, '!');
cout<<"Operation: str10.assign(3, '!)"<<endl;
cout<<"str10 string assigned with character '!' is: "<<str10<<endl;

//assigning a value from a range of one string to another string
string str11("Comes "), str12("the END ");
cout<<"\nstr11 string is: "<<str11<<endl;
cout<<"str12 string is: "<<str12<<endl;
str11.assign(str12.begin() + 4, str12.end() - 1);
cout<<"Operation: str11.assign(str12.begin()+4, str12.end()-1)"<<endl;
cout<<"str11 assigned a range of str12 string is: \n"<<str11<<endl;
return 0;
}

```

Output:

```

e:\projectvc\string\string\Debug\string.exe
str2, C string is: STRInG assign()
Operation: str1.assign(str2)
Assigning the C-string str2 to str1 string:
STRInG assign()

str4 C string is: Another STRInG assign()
Operation: str3.assign(str4, 11)
Assigning some portion of the str4 to str3 string:
Another STR

str5 string is: First
str6 string is: Second sTrInG
Operation: str5.assign(str6 ,7 ,6)
Newly assigned str5 string is: sTrInG

str7 string is: First
str8 string is: Second
str9 string is: Third
Operation: str7.assign(str8)
Newly assigned str7 with str8 string is: Second
Operation: str7 = str9
String str7 reassigned with str9 string is: Third

str10 string is: Working STRInG
Operation: str10.assign(3 , '!')
str10 string assigned with character '!' is: !!!

str11 string is: Comes
str12 string is: the END
Operation: str11.assign(str12.begin()+4, str12.end()-1)
str11 assigned a range of str12 string is:
END
Press any key to continue_

```

at()

- The first element of the string has an index of zero and the following elements are indexed consecutively by the positive integers, so that a string of length n has an n th element indexed by the number $n - 1$.

- The member `operator[]` is faster than the member function `at()` for providing read and write access to the elements of a string.
- The member `operator[]` does not check whether the index passed as a parameter is valid but the member function `at()` does and so should be used in the validity is not certain. An invalid index, which is an index less than zero or greater than or equal to the size of the string, passed to the member function `at()` throws an `out_of_range` class exception.
- An invalid index passed to the `operator[]` results in undefined behavior, but the index equal to the length of the string is a valid index for const strings and the operator returns the null-character when passed this index.
- The reference returned may be invalidated by string reallocations or modifications for the non-const strings.

Output :


```

e:\projectvc\string\string\Debug\string.exe
str1 string is: Operation
str2 string is: Desert Storm
constr1, const string is: Making cakes
constr2, const string is: Start eating

---str1[5]---
The 5th character of str1 is: a
---str2.at(7)---
The 7th character of str2 is: s
---constr1.length()---
The length of the constr1 string is: 12

Testing the null character...
---cRefStr1 = constr1[constr1.length()]---
Operation: (cRefStr1 == '\\0')
The null character is returned.

---constr2.at(8)---
The 8th character of the constr2 is: t
Press any key to continue

```

basic_string::basic_string

- Constructs a string that is empty or initialized by specific characters or that is a copy of all or part of some other string object or C-string.
- The five member functions and their parameters are shown below but it is not our concern here. We just want to know how to use the member functions in our program.
- Then, the following program example tries to describe the string object initialization.

```

basic_string(
    const value_type* _Ptr,
    size_type _Count = npos,
    const allocator_type& _Al = Allocator()
);
basic_string(
    const basic_string& _Right,
    size_type _Roff = 0,
    size_type _Count = npos
);
basic_string(
    const basic_string& _Right,
    size_type _Roff = 0,
    size_type _Count = npos,
    const allocator_type& _Al = Allocator()
);
basic_string(
    size_type _Count,
    value_type _Ch,
    const allocator_type& _Al = Allocator()
);
explicit basic_string(
    const allocator_type& _Al = Allocator()
);
template <class InputIterator>
basic_string(
    InputIterator _First,
    InputIterator _Last,
    const allocator_type& _Al = Allocator()
);

```

- The constructors for class `basic_string` create and initialize strings as follows:
 - The first member function creates a string that is initialized by all or part of a C-string.
 - The second member function creates a string that is initialized by all or part of an object of type `basic_string`.
 - The third member function creates a string that is initialized by a specific number of characters of a parameter stipulated value.
 - The fourth member function creates an empty string.
 - The fifth member function creates a string that is initialized by the characters in the range whose boundaries are delimited by input iterators.

```

//basic_string
#include <string>
#include <iostream>
using namespace std;

int main()
{
//initializing with a C-string
const char *str1 = "The basic_string";
basic_string <char> str2(str1, 5);
cout<<"str1 string is: "<<str1<<endl;
cout<<"Operation: str2(str1, 5)"<<endl;
cout<<"str2 initialized by str1 is: "<<str2<<"\n\n";

//initializing with a string
string str3("Initialize with a StRinG?");
cout<<"str3 string is: "<<str3<<endl;
basic_string <char> str4(str3, 6, 10);
cout<<"Operation: str4(str3, 6, 10)"<<endl;
cout<<"str4 initialized by part of the str3 string is: \n"<<str4<<"\n\n";

//initializing a string with a number of characters of a specific value
basic_string <char> str5(6, '7');
cout<<"Operation: str5(6, '7')"<<endl;
cout<<"str5 initialized by six number of 7s is: "<<str5<<"\n\n";

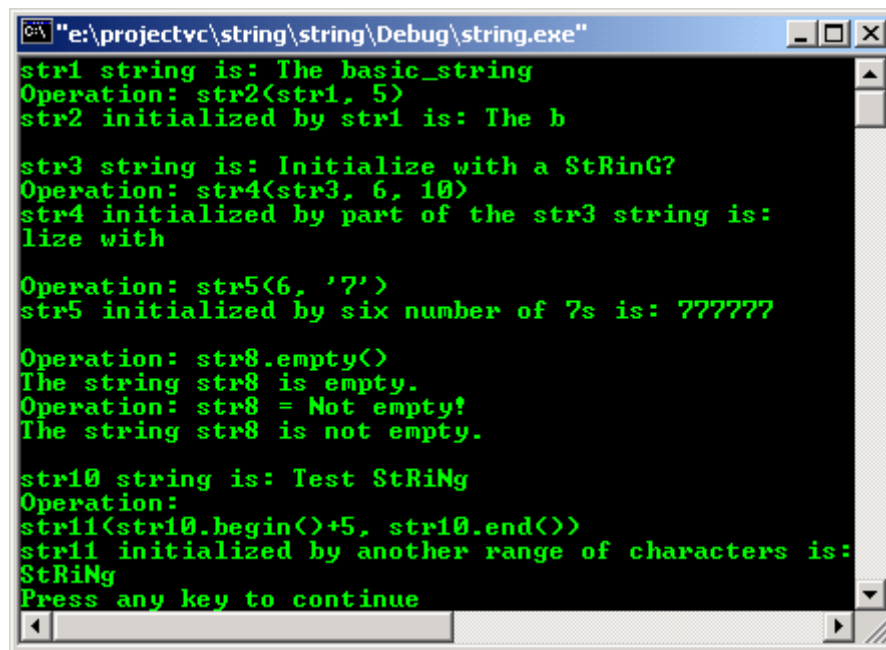
//creating an empty string and string with a specified allocator
basic_string <char> str6;
string str7;
//allocate a storage
basic_string <char> str8(str7.get_allocator());
//test the emptiness
cout<<"Operation: str8.empty()"<<endl;
if(str8.empty())
    cout<<"The string str8 is empty."<<endl;
else
    cout<<"The string str8 is not empty."<<endl;

//fill up some string
str8 = "Not empty!";
cout<<"Operation: str8 = Not empty!"<<endl;
//retest again...
if(str8.empty())
    cout<<"The string str8 is empty."<<endl;
else
    cout<<"The string str8 is not empty."<<endl;
cout<<endl;

//initializing a string from another range of characters
string str10("Test StRinG");
cout<<"str10 string is: "<<str10<<endl;
cout<<"Operation: \nstr11(str10.begin()+5, str10.end())"<<endl;
basic_string <char> str11(str10.begin()+5, str10.end());
cout<<"str11 initialized by another range of characters is: \n"<<str11<<endl;
return 0;
}

```

Output:



```
"e:\projectvc\string\string\Debug\string.exe"
str1 string is: The basic_string
Operation: str2(str1, 5)
str2 initialized by str1 is: The b

str3 string is: Initialize with a StRinG?
Operation: str4(str3, 6, 10)
str4 initialized by part of the str3 string is:
lize with

Operation: str5(6, '7')
str5 initialized by six number of 7s is: 777777

Operation: str8.empty()
The string str8 is empty.
Operation: str8 = Not empty!
The string str8 is not empty.

str10 string is: Test StRiNg
Operation:
str11(str10.begin()+5, str10.end())
str11 initialized by another range of characters is:
StRiNg
Press any key to continue
```

begin()

- If the return value of `begin()` is assigned to a `const_iterator`, the string object cannot be modified.
- If the return value of `begin()` is assigned to an `iterator`, the string object can be modified.

```
//begin(), end()
#include <string>
#include <iostream>
using namespace std;

int main()
{
    string Str1("Testing the begin() and end()"), Str2;
    basic_string<char>::iterator Str1Iter;
    //const_iterator...
    basic_string<char>::const_iterator Str1CIter;

    //...an error because the iterator Str1CIter is const
    /*Str1CIter = 'Z';

    cout<<"String Str1 is: "<<Str1<<endl;
    Str1Iter = Str1.begin();
    cout<<"Operation: Str1Iter = Str1.begin()"<<endl;
    cout<<"The first character of the string Str1 is: "<<*Str1Iter<<"\n\n";

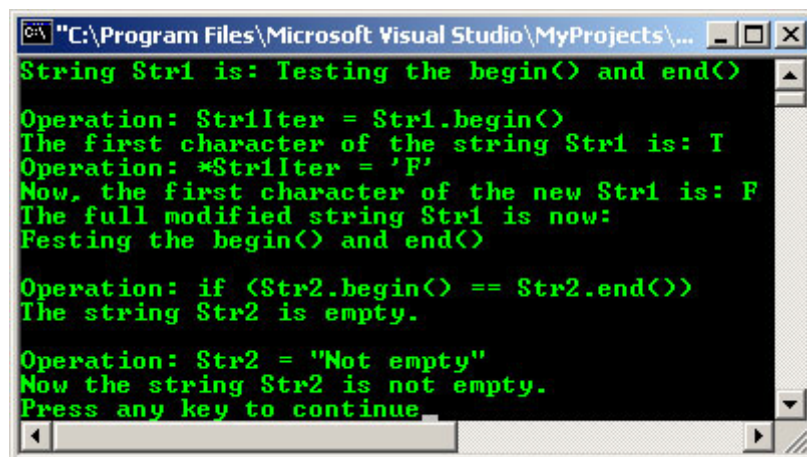
    //using dereferenced iterator to modify a character
    *Str1Iter = 'F';
    cout<<"Operation: *Str1Iter = 'F'"<<endl;
    cout<<"Now, the first character of the new Str1 is: "<<*Str1Iter<<endl;
    cout<<"The full modified string Str1 is now: \n"<<Str1<<"\n\n";

    //For an empty string, begin() == end()
    cout<<"Operation: if(Str2.begin() == Str2.end())<<endl;
    if(Str2.begin() == Str2.end())
        cout<<"The string Str2 is empty."<<endl;
    else
        cout<<"The string Str2 is not empty."<<endl;
    cout<<endl;

    //Fill up some string and retest...
    Str2 = "Not empty";
    cout<<"Operation: Str2 = \"Not empty\""<<endl;
    if(Str2.begin() == Str2.end())
        cout<<"Now the string Str2 is empty."<<endl;
    else
        cout<<"Now the string Str2 is not empty."<<endl;
    return 0;
```

```
}
```

Output:

A screenshot of a Visual Studio console window. The title bar shows the path "C:\Program Files\Microsoft Visual Studio\MyProjects\...". The console output is as follows:

```
String Str1 is: Testing the begin() and end()

Operation: Str1Iter = Str1.begin()
The first character of the string Str1 is: T
Operation: *Str1Iter = 'F'
Now, the first character of the new Str1 is: F
The full modified string Str1 is now:
Festing the begin() and end()

Operation: if (Str2.begin() == Str2.end())
The string Str2 is empty.

Operation: Str2 = "Not empty"
Now the string Str2 is not empty.
Press any key to continue.
```

c_str()

- Objects of type string belonging to the C++ template class `basic_string<char>` are not necessarily null terminated.
- The null character `'\0'` is used as a special character in a C-string to mark the end of the string but has not special meaning in an object of type string and may be a part of the string just like any other character.
- There is an automatic conversion from `const char*` into strings, but the string class does not provide for automatic conversions from C-style strings to objects of type `basic_string<char>`.
- The returned C-style string should not be modified, as this could invalidate the pointer to the string, or deleted, as the string has a limited lifetime and is owned by the class string.

```
//c_str(), length(), data(), strlen()
#include <string>
#include <iostream>
using namespace std;

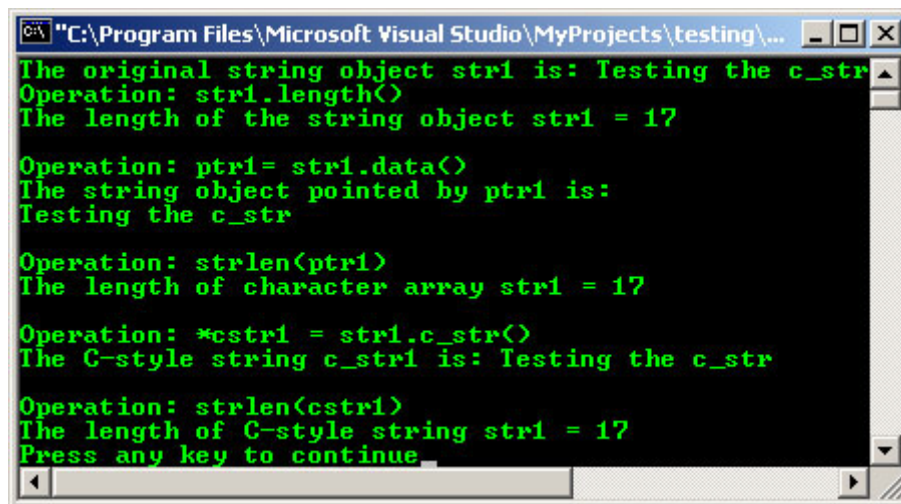
int main( )
{

    string str1("Testing the c_str");
    cout<<"The original string object str1 is: "<<str1<<endl;
    cout<<"Operation: str1.length()"<<endl;
    cout<<"The length of the string object str1 = "<<str1.length()<<"\n\n";

    //A string to an array of characters conversion
    const char *ptr1 = 0;
    ptr1= str1.data();
    cout<<"Operation: ptr1= str1.data()"<<endl;
    cout<<"The string object pointed by ptr1 is: \n"<<ptr1<<endl;
    cout<<"\nOperation: strlen(ptr1)"<<endl;
    cout<<"The length of character array str1 = "<<strlen(ptr1)<<"\n\n";

    //A string to a C-style string conversion
    const char *cstr1 = str1.c_str();
    cout<<"Operation: *cstr1 = str1.c_str()"<<endl;
    cout<<"The C-style string c_str1 is: "<<cstr1<<endl;
    cout<<"\nOperation: strlen(cstr1)"<<endl;
    cout<<"The length of C-style string str1 = "<<strlen(cstr1)<<endl;
    return 0;
}
```

Output:

A screenshot of a Visual Studio console window. The title bar shows the file path: "C:\Program Files\Microsoft Visual Studio\MyProjects\testing\...". The console output is as follows:
The original string object str1 is: Testing the c_str
Operation: str1.length()
The length of the string object str1 = 17

Operation: ptr1= str1.data()
The string object pointed by ptr1 is:
Testing the c_str

Operation: strlen(ptr1)
The length of character array str1 = 17

Operation: *cstr1 = str1.c_str()
The C-style string c_str1 is: Testing the c_str

Operation: strlen(cstr1)
The length of C-style string str1 = 17
Press any key to continue

capacity()

- The member function `capacity()` returns the storage currently allocated to hold the controlled sequence, a value at least as large as `size`.

```
//capacity(), size(), erase()
//length(), max_size()
#include <string>
#include <iostream>
using namespace std;

int main( )
{
    string str1("Testing the capacity()");
    cout<<"str1 string is: "<<str1<<endl;

    //The size and length member functions differ in name only
    basic_string<char>::size_type SizeStr1, LenStr1;
    SizeStr1 = str1.size();
    LenStr1 = str1.length();

    basic_string<char>::size_type CapStr1, MaxSizeStr1;
    CapStr1 = str1.capacity();
    MaxSizeStr1 = str1.max_size();

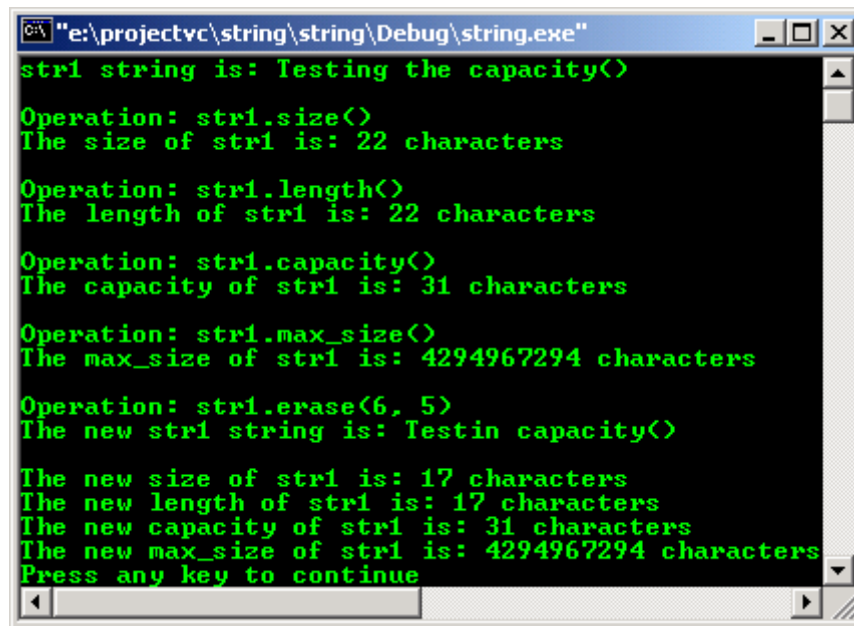
    //Compare size, length, capacity & max_size of a string
    cout<<"\nOperation: str1.size()"<<endl;
    cout<<"The size of str1 is: "<<SizeStr1<<" characters"<<endl;
    cout<<"\nOperation: str1.length()"<<endl;
    cout<<"The length of str1 is: "<<LenStr1<<" characters"<<endl;
    cout<<"\nOperation: str1.capacity()"<<endl;
    cout<<"The capacity of str1 is: "<<CapStr1<<" characters"<<endl;
    cout<<"\nOperation: str1.max_size()"<<endl;
    cout<<"The max_size of str1 is: "<<MaxSizeStr1<<" characters"<<endl;

    //erase some characters...
    str1.erase(6, 5);
    //Re test...
    cout<<"\nOperation: str1.erase(6, 5)"<<endl;
    cout<<"The new str1 string is: "<<str1<<"\n\n";

    SizeStr1 = str1.size();
    LenStr1 = str1.length();
    CapStr1 = str1.capacity();
    MaxSizeStr1 = str1.max_size();

    //Compare size, length, capacity & max_size of a string
    //after erasing part of the original string
    cout<<"The new size of str1 is: "<<SizeStr1<<" characters"<<endl;
    cout<<"The new length of str1 is: "<<LenStr1<<" characters"<<endl;
    cout<<"The new capacity of str1 is: "<<CapStr1<<" characters"<<endl;
    cout<<"The new max_size of str1 is: "<<MaxSizeStr1<<" characters"<<endl;
    return 0;
}
```

Output:



```
C:\e:\projectvc\string\string\Debug\string.exe
str1 string is: Testing the capacity()
Operation: str1.size()
The size of str1 is: 22 characters
Operation: str1.length()
The length of str1 is: 22 characters
Operation: str1.capacity()
The capacity of str1 is: 31 characters
Operation: str1.max_size()
The max_size of str1 is: 4294967294 characters
Operation: str1.erase(6, 5)
The new str1 string is: Testin capacity()
The new size of str1 is: 17 characters
The new length of str1 is: 17 characters
The new capacity of str1 is: 31 characters
The new max_size of str1 is: 4294967294 characters
Press any key to continue
```

clear()

- The string on which the member function `clear()` is called will be empty.

```
//clear()
#include <string>
#include <iostream>
using namespace std;

int main( )
{
    string str1("Testing the clear()");
    basic_string<char>::iterator StrIter;
    //Normal...
    cout<<"str1 string is : "<<str1<<endl;
    //using iterator....iterate character by character...
    cout<<"str1 string is: ";
    for (StrIter = str1.begin(); StrIter != str1.end(); StrIter++)
        cout<<*StrIter;
    cout<<endl;

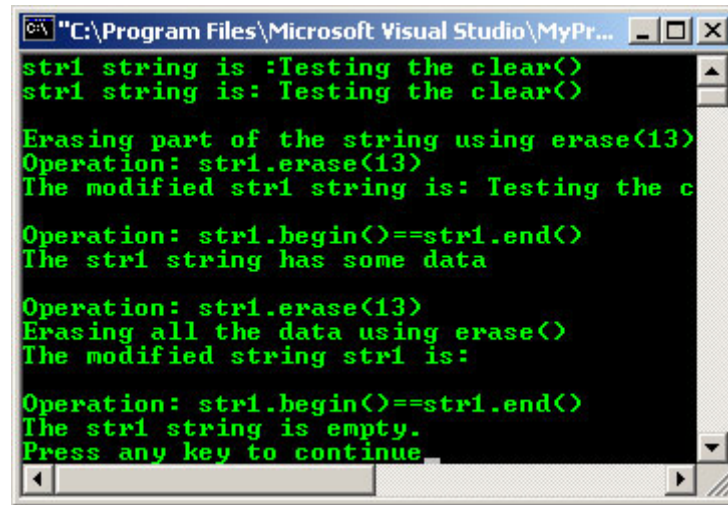
    //str1.clear();
    cout<<"\nErasing part of the string using erase(13)"<<endl;
    str1.erase(13);
    cout<<"Operation: str1.erase(13)"<<endl;
    cout<<"The modified str1 string is: ";
    //using iterator...
    for(StrIter = str1.begin(); StrIter != str1.end(); StrIter++)
        cout<<*StrIter;
    cout<<endl;

    //For an empty string, begin is equivalent to end
    cout<<"\nOperation: str1.begin()==str1.end()"<<endl;
    if(str1.begin() == str1.end())
        cout<<"The str1 string is empty."<<endl;
    else
        cout<<"The str1 string has some data"<<endl;

    //erasing all the data...
    cout<<"\nOperation: str1.erase(13)"<<endl;
    cout<<"Erasing all the data using erase()"<<endl;
    str1.erase();
    //re test...
    cout<<"The modified string str1 is: "<<endl;
    cout<<"\nOperation: str1.begin()==str1.end()"<<endl;
    if(str1.begin() == str1.end())
        cout<<"The str1 string is empty."<<endl;
    else
        cout<<"The str1 string has some data"<<endl;
```

```
return 0;
}
```

Output:



```
str1 string is :Testing the clear()
str1 string is: Testing the clear()

Erasing part of the string using erase(13)
Operation: str1.erase(13)
The modified str1 string is: Testing the c

Operation: str1.begin()==str1.end()
The str1 string has some data

Operation: str1.erase(13)
Erasing all the data using erase()
The modified string str1 is:

Operation: str1.begin()==str1.end()
The str1 string is empty.
Press any key to continue
```

compare()

- The compare() member functions compare either all or part of the parameter and operand strings depending on which in used.
- A negative return value if the operand string is less than the parameter string; zero if the two strings are equal; or a positive value if the operand string is greater than the parameter string.

```
//compare() program example part I
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //Comparing a string to a string...
    //the character index start from 0
    int str1;
    string str2("First");
    string str3("First");
    cout<<"str2 string is: "<<str2<<endl;
    cout<<"str3 string is: "<<str3<<endl;

    //compare str2 and str3 assign the result to str1
    cout<<"Operation: str2.compare(str3)"<<endl;
    str1 = str2.compare(str3);
    if(str1 < 0)
        cout<<"The str2 string is less than the str3 string."<<endl;
    else if(str1 == 0)
        cout<<"The str2 string is equal to the str3 string."<<endl;
    else
        cout<<"The str2 string is greater than the str3 string."<<endl;
    cout<<endl;

    //Comparing part of a string to a string
    int str4, str5;
    string str6("SecondThird");
    string str7("Third");
    cout<<"str6 string is: "<<str6<<endl;
    cout<<"str7 string is: "<<str7<<endl;

    cout<<"Operation: str6.compare(6, 5, str7)"<<endl;
    str4 = str6.compare(6, 5, str7);
    if(str4 < 0)
        cout<<"The last 5 characters of the str6 string are less than\n"
        <<"the str7 string."<<endl;
    else if(str4 == 0)
        cout<<"The last 5 characters of the str6 string are equal to\n"
        <<"the str7 string."<<endl;
    else
```



```

        cout<<"The last 5 characters of the str6 string is greater than\n"
            <<"the str7 string."<<endl;
    cout<<endl;

    cout<<"Operation: str6.compare(0, 6, str7)"<<endl;
    str5 = str6.compare(0, 6, str7);
    if(str5 < 0)
        cout<<"The first 6 characters of the str6 \nstring are less than "
            <<"the str7 string."<<endl;
    else if(str5 == 0)
        cout<<"The first 6 characters of the str6 \nstring are equal to "
            <<"the str7 string."<<endl;
    else
        cout<<"The first 6 characters of the str6 \nstring is greater than "
            <<"the str7 string."<<endl;
    return 0;
}

```

Output:

```

C:\Program Files\Microsoft Visual Studio\MyProjects\testing\...
str2 string is: First
str3 string is: First
Operation: str2.compare(str3)
The str2 string is equal to the str3 string.

str6 string is: SecondThird
str7 string is: Third
Operation: str6.compare(6, 5, str7)
The last 5 characters of the str6 string are equal to
the str7 string.

Operation: str6.compare(0, 6, str7)
The first 6 characters of the str6
string are less than the str7 string.
Press any key to continue

```

```

//compare() program example part II
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //comparing part of a string to part of a string
    int str8;
    string str9("TestFourth");
    string str10("TFourthT");
    cout<<"str9 string is: "<<str9<<endl;
    cout<<"str10 string is: "<<str10<<endl;

    str8 = str9.compare(4, 6, str10, 1, 6);
    cout<<"Operation: str9.compare(4, 6, str10, 1, 6)"<<endl;
    if(str8 < 0)
        cout<<"The 6 characters from position 4 of the str9 string \nare less than "
            <<"the 6 characters str10 string from position 1."<<endl;
    else if(str8 == 0)
        cout<<"The 6 characters from position 4 of the str9 string\nare equal to "
            <<"the 6 characters str10 string from position 1."<<endl;
    else
        cout<<"The 6 characters from position 4 of the str9 string\nis greater than "
            <<"the 6 characters str10 string from position 1."<<endl;
    cout<<endl;

    //comparing a string to a C-string
    int str11;
    string str12("Fifth");
    const char* str13 = "Sixth";
    cout<<"The string str12 is: "<<str12<<endl;
    cout<<"The C-string str13 is: "<<str13<<endl;

    str11 = str12.compare(str13);
    cout<<"Operation: str12.compare(str13)"<<endl;
    if(str11 < 0)
        cout<<"The str12 string is less than the str13 C-string."<<endl;
    else if(str11 == 0)

```



```

    cout<<"The str12 string is equal to the str13 C-string."<<endl;
else
    cout<<"The str12 string is greater than the str13 C-string."<<endl;
cout << endl;

//Comparing part of a string to a C-string
int str14;
string str15("SeventhEight");
const char* str16 = "Eight";
cout<<"str15 string is: "<<str15<<endl;
cout<<"str16 string is: "<<str16<<endl;

str14 = str15.compare(7, 5, str16);
cout<<"Operation: str15.compare(7, 5, str16)"<<endl;
if(str14 < 0)
    cout<<"The last 5 characters of the str15 \nstring are less than "
        <<"the str16 C-string."<<endl;
else if(str14 == 0)
    cout<<"The last 5 characters of the str15 \nstring are equal to "
        <<"the str16 C-string."<<endl;
else
    cout<<"The last 5 characters of the str15 \nstring is greater than "
        <<"the str16 C-string."<<endl;
cout << endl;

//comparing part of a string to part of an equal length of a C-string
int str17;
string str18("ReTestEighth");
const char* str19 = "TestEighth";
cout<<"str18 string is: "<<str18<<endl;
cout<<"str19 C-string is: "<<str19<<endl;

str17 = str18.compare(2, 4, str19, 4);
cout<<"Operation: str18.compare(4, 6, str19, 6)"<<endl;
if(str17 < 0)
    cout<<"The 4 characters from position 2 of the str18 string \nare less than "
        <<"the first 4 characters of the str19 C-string."<<endl;
else if(str17 == 0)
    cout<<"The 4 characters from position 2 of the str18 string \nare equal to "
        <<"the first 4 characters of the str19 C-string."<<endl;
else
    cout<<"The 4 characters from position 2of the str18 string \nis greater than "
        <<"the first 4 characters of the str19 C-string."<<endl;
return 0;
}

```

Output:

```

C:\Program Files\Microsoft Visual Studio\MyProjects\testing\Debug\te...
str9 string is: TestFourth
str10 string is: TFourthI
Operation: str9.compare(4, 6, str10, 1, 6)
The 6 characters from position 4 of the str9 string
are equal to the 6 characters str10 string from position 1.

The string str12 is: Fifth
The C-string str13 is: Sixth
Operation: str12.compare(str13)
The str12 string is less than the str13 C-string.

str15 string is: SeventhEight
str16 string is: Eight
Operation: str15.compare(7, 5, str16)
The last 5 characters of the str15
string are equal to the str16 C-string.

str18 string is: ReTestEighth
str19 C-string is: TestEighth
Operation: str18.compare(4, 6, str19, 6)
The 4 characters from position 2 of the str18 string
are equal to the first 4 characters of the str19 C-string.
Press any key to continue

```

copy()

- A null character is not appended to the end of the copy.
- The return value is the number of characters actually copied.

```
//copy()
#include <string>
#include <iostream>
using namespace std;

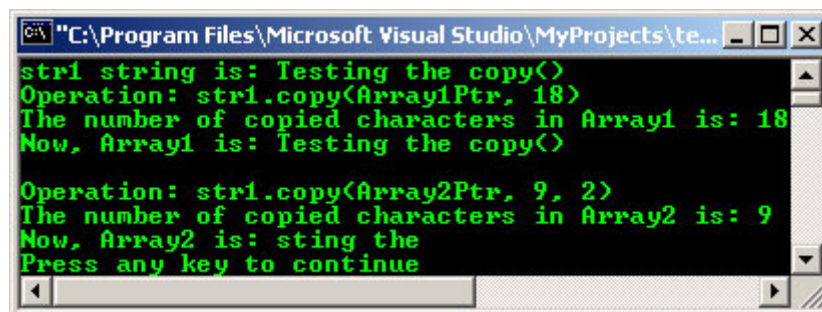
int main()
{
    string str1("Testing the copy()");
    basic_string<char>::iterator StrIter;
    //declare and initialize arrays to 0
    char Array1[20] = {0};
    char Array2[10] = {0};
    basic_string<char>::pointer Array1Ptr = Array1;
    basic_string<char>::value_type *Array2Ptr = Array2;

    //iterate character by character...
    cout<<"str1 string is: ";
    for(StrIter = str1.begin(); StrIter != str1.end(); StrIter++)
        cout<<*StrIter;
    cout<<endl;

    basic_string<char>::size_type NewArray1;
    NewArray1 = str1.copy(Array1Ptr, 18);
    cout<<"Operation: str1.copy(Array1Ptr, 18)"<<endl;
    cout<<"The number of copied characters in Array1 is: "<<unsigned int(NewArray1)<<endl;
    cout<<"Now, Array1 is: "<<Array1<<"\n\n";

    basic_string<char>::size_type NewArray2;
    NewArray2 = str1.copy(Array2Ptr, 9, 2);
    cout<<"Operation: str1.copy(Array2Ptr, 9, 2)"<<endl;
    cout<<"The number of copied characters in Array2 is: "<<unsigned int(NewArray2)<<endl;
    cout<<"Now, Array2 is: "<<Array2Ptr<<endl;
    return 0;
}
```

Output:



```
C:\Program Files\Microsoft Visual Studio\MyProjects\te...
str1 string is: Testing the copy()
Operation: str1.copy(Array1Ptr, 18)
The number of copied characters in Array1 is: 18
Now, Array1 is: Testing the copy()

Operation: str1.copy(Array2Ptr, 9, 2)
The number of copied characters in Array2 is: 9
Now, Array2 is: string the
Press any key to continue
```

data()

- Objects of type string belonging to the C++ template class `basic_string<char>` are not necessarily null terminated.
- The return type for `data()` is not a valid C-string, because no null character gets appended. The null character `'\0'` is used as a special character in a C-string to mark the end of the string, but has **no special meaning in an object of type string** and may be a part of the string object just like any other character.
- There is an automatic conversion from `const char*` into strings, but the string class does not provide for automatic conversions from C-style strings to objects of type `basic_string<char>`.
- The returned string should not be modified, because this could invalidate the pointer to the string, or deleted, because the string has a limited lifetime and is owned by the class string.
- The return value is a pointer to the first element of the array containing the contents of the string, or, for an empty array, a non-null pointer that cannot be dereferenced.

```
//data(), length(), strlen()
//and c_str()
#include <string>
#include <iostream>
using namespace std;
```

```

int main()
{
    string str1("Testing the data()");
    cout<<"str1 string object is: "<<str1<<endl;
    cout<<"Operation: str1.length()"<<endl;
    cout<<"The length of str1 = "<<unsigned int(str1.length())<<"\n\n";

    //Converting a string to an array of characters
    const char *ptr1 = 0;
    ptr1= str1.data();
    cout<<"Operation: str1.data()"<<endl;
    cout<<"The modified ptr1 string object is: "<<ptr1<<endl;
    cout<<"Operation: strlen(ptr1)"<<endl;
    cout<<"The length of character array str1 = "<<unsigned int(strlen(ptr1))<<"\n\n";

    //Converting a string to a C-style string
    const char *cstr1 = str1.c_str();
    cout<<"Operation: str1.c_str()"<<endl;
    cout<<"The C-style string c_str1 is: "<<cstr1<<endl;
    cout<<"Operation: strlen(ptr1)"<<endl;
    cout<<"The length of C-style string str1 = "<<unsigned int(strlen(cstr1))<<endl;
    return 0;
}

```

Output:

```

C:\Program Files\Microsoft Visual Studio\MyProjects\testing\D...
str1 string object is: Testing the data()
Operation: str1.length()
The length of str1 = 18

Operation: str1.data()
The modified ptr1 string object is: Testing the data()
Operation: strlen(ptr1)
The length of character array str1 = 18

Operation: str1.c_str()
The C-style string c_str1 is: Testing the data()
Operation: strlen(ptr1)
The length of C-style string str1 = 18
Press any key to continue

```

empty()

- The return value is **true** if the string object contains no characters; **false** if it has at least one character.
- The empty() is equivalent to size == 0.

```

//empty()
#include <string>
#include <iostream>
using namespace std;

int main()
{
    bool Var1, Var2;
    string str1("Testing the empty()");
    cout<<"str1 string object is: "<<str1<<endl;

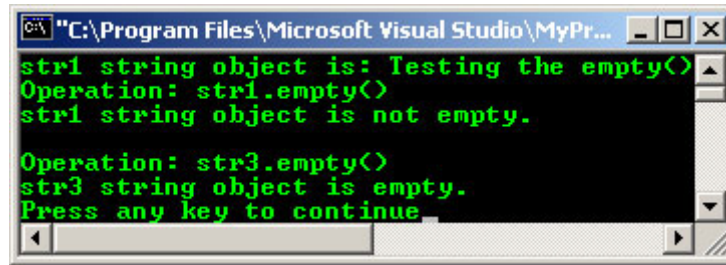
    Var1 = str1.empty();
    //test the emptiness
    cout<<"Operation: str1.empty()"<<endl;
    if(Var1)
        cout<<"str1 string object is empty."<<endl;
    else
        cout<<"str1 string object is not empty."<<"\n\n";

    //An example of an empty string object
    string str3;
    Var2 = str3.empty();
    cout<<"Operation: str3.empty()"<<endl;
    //test the emptiness
    if(Var2)
        cout<<"str3 string object is empty."<<endl;
    else

```

```
cout<<"str3 string object is not empty."<<endl;
return 0;
}
```

Output:



```
str1 string object is: Testing the empty()
Operation: str1.empty()
str1 string object is not empty.

Operation: str3.empty()
str3 string object is empty.
Press any key to continue
```

end()

- The return value is a random-access iterator that addresses the location succeeding the last element in a string.
- end() is often used to test whether an iterator has reached the end of its string. The value returned by end() should not be dereferenced.
- If the return value of end() is assigned to a const_iterator, the string object cannot be modified. If the return value of end() is assigned to an iterator, the string object can be modified.

```
//begin(), end()
#include <string>
#include <iostream>
using namespace std;

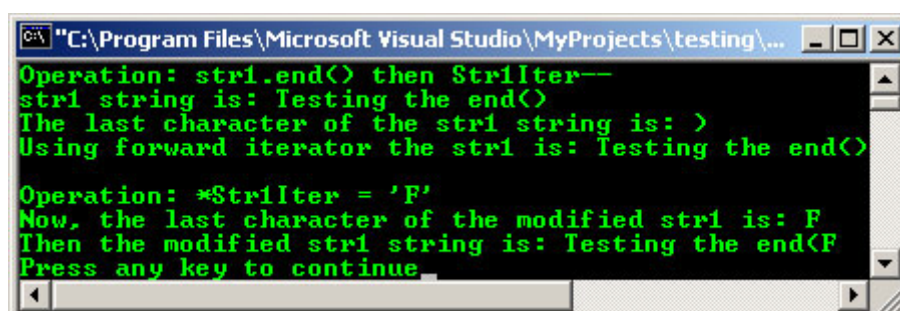
int main( )
{
    string str1("Testing the end()");
    basic_string<char>::iterator StrIter, StrIter;

    StrIter = str1.end();
    //minus the null character, so point to the real
    //last character in the string...
    StrIter--;
    cout<<"Operation: str1.end() then StrIter--"<<endl;
    cout<<"str1 string is: "<<str1<<endl;
    cout<<"The last character of the str1 string is: "<<*StrIter<<endl;

    //end() used to test when an iterator has reached the end of its string
    cout<<"Using forward iterator the str1 is: ";
    for(StrIter = str1.begin(); StrIter != str1.end(); StrIter++)
        cout<<*StrIter;
    cout<<"\n\n";

    //The dereferenced iterator can be used to modify a character
    //The last event, this pointer point to the last character in the string
    *StrIter = 'F';
    cout<<"Operation: *StrIter = 'F'"<<endl;
    cout<<"Now, the last character of the modified str1 is: "<<*StrIter<<endl;
    cout<<"Then the modified str1 string is: "<<str1<<endl;
    return 0;
}
```

Output:



```
Operation: str1.end() then StrIter--
str1 string is: Testing the end()
The last character of the str1 string is: )
Using forward iterator the str1 is: Testing the end()

Operation: *StrIter = 'F'
Now, the last character of the modified str1 is: F
Then the modified str1 string is: Testing the end<F
Press any key to continue
```

erase()

- The return value: For the first two member functions, an iterator addressing the first character after the last character removed by the member function.
- For the third member function, a reference to the string object from which the elements have been erased.
- The third member function returns `*this`.

```
//erase()
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //using a range...
    string str1("Testing the erase() part I");
    basic_string<char>::iterator Str1Iter;
    cout<<"str1 string object is: "<< str1<<endl;

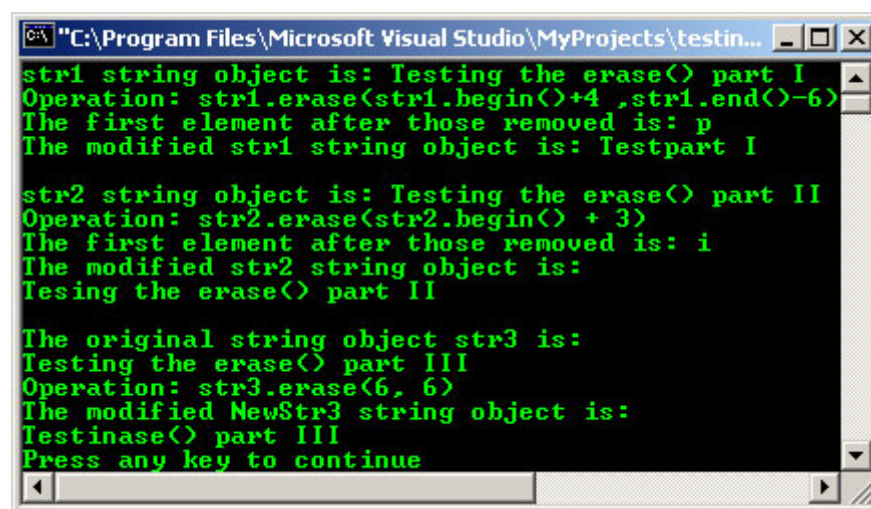
    //don't forget the null character
    Str1Iter = str1.erase(str1.begin() + 4, str1.end() - 6);
    cout<<"Operation: str1.erase(str1.begin()+4, str1.end()-6)"<<endl;
    cout<<"The first element after those removed is: "<<*Str1Iter<<endl;
    cout<<"The modified str1 string object is: "<<str1<<endl;

    //erasing a char pointed to by an iterator
    string str2("Testing the erase() part II");
    basic_string<char>::iterator Str2Iter;

    cout<<"\nstr2 string object is: "<<str2<<endl;
    Str2Iter = str2.erase(str2.begin() + 3);
    cout<<"Operation: str2.erase(str2.begin() + 3)"<<endl;
    cout<<"The first element after those removed is: "<<*Str2Iter<<endl;
    cout<<"The modified str2 string object is: \n"<<str2<<endl;

    //erasing a number of chars after a char
    string str3("Testing the erase() part III"), NewStr3;
    cout<<"\nThe original string object str3 is: \n"<<str3<<endl;
    NewStr3 = str3.erase(6, 8);
    cout<<"Operation: str3.erase(6, 6)"<<endl;
    cout<<"The modified NewStr3 string object is: \n"<<NewStr3<<endl;
    return 0;
}
```

Output:



find()

- The return value is the index of the first character of the substring searched for when successful; otherwise `npos`.

```

//find() part I
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //don't forget the null character
    //searching for a single character in a string
    string str1("Search part I, a character in a string");
    cout<<"str1 string is: "<<str1<<endl;
    basic_string<char>::size_type index1, index2;
    static const basic_string<char>::size_type npos = -1;

    index1 = str1.find('r', 2);
    cout<<"Operation: str1.find('r', 2)"<<endl;
    if(index1 != npos)
        cout<<"The index of the 1st 'r' found after the 2nd"
            <<" position in str1 is: "<<unsigned int(index1)<<endl;
    else
        cout<<"The character 'r' was not found in str1."<<endl;
    cout<<endl;

    index2 = str1.find('t');
    cout<<"Operation: str1.find('t')"<<endl;
    if(index2 != npos)
        cout<<"The index of the 't' found in str1 is: "<<unsigned int(index2)<<endl;
    else
        cout<<"The character 't' was not found in str1."<<endl;
    cout<<endl;

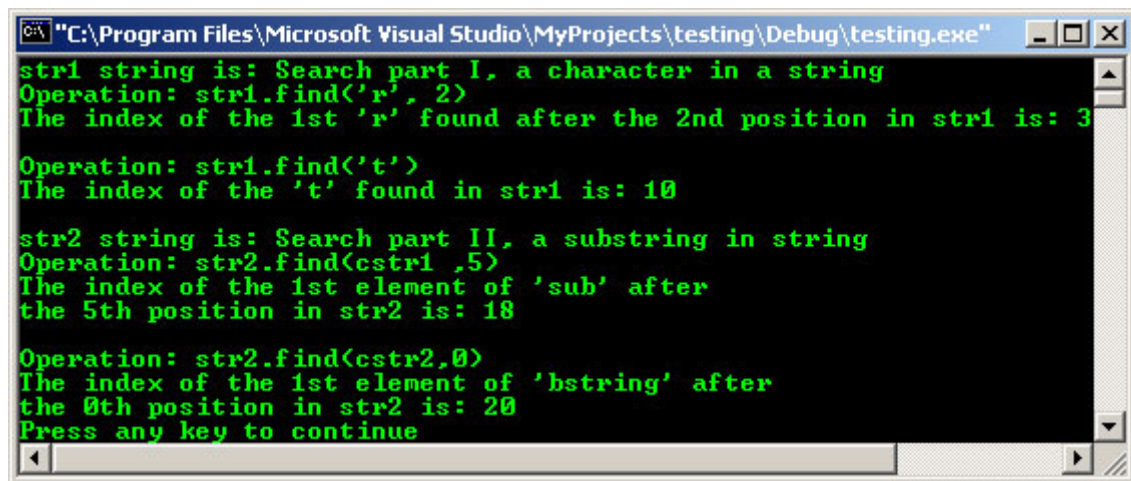
    //-----
    //searching a string for a substring as specified by a C-string
    string str2("Search part II, a substring in string");
    cout<<"str2 string is: "<<str2<<endl;
    basic_string<char>::size_type index3, index4;

    const char *cstr1 = "sub";
    index3 = str2.find(cstr1, 5);
    cout<<"Operation: str2.find(cstr1, 5)"<<endl;
    if(index3 != npos)
        cout<<"The index of the 1st element of 'sub' after\nthe 5th "
            <<"position in str2 is: "<<unsigned int(index3)<<endl;
    else
        cout<<"The substring 'sub' was not found in str2"<<endl;
    cout<<endl;

    const char *cstr2 = "bstring";
    index4 = str2.find(cstr2, 0);
    cout<<"Operation: str2.find(cstr2, 0)"<<endl;
    if(index4 != npos)
        cout<<"The index of the 1st element of 'bstring' "
            <<"after\nthe 0th position in str2 is: "<<unsigned int(index4)<<endl;
    else
        cout<<"The substring 'bstring' was not found in str2"<<endl;
    return 0;
}

```

Output:



```
"C:\Program Files\Microsoft Visual Studio\MyProjects\testing\Debug\testing.exe"
str1 string is: Search part I, a character in a string
Operation: str1.find('r', 2)
The index of the 1st 'r' found after the 2nd position in str1 is: 3

Operation: str1.find('t')
The index of the 't' found in str1 is: 10

str2 string is: Search part II, a substring in string
Operation: str2.find(cstr1, 5)
The index of the 1st element of 'sub' after
the 5th position in str2 is: 18

Operation: str2.find(cstr2, 0)
The index of the 1st element of 'bstring' after
the 0th position in str2 is: 20
Press any key to continue
```

```
//find() part II
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //don't forget the null character
    //searching a string for a substring as specified by a C-string
    static const basic_string <char>::size_type npos = -1;
    string str3("Again, search part III");
    cout<<"str3 string is: "<<str3<<endl;
    basic_string <char>::size_type index5, index6;

    const char *cstr3 = "part";
    index5 = str3.find(cstr3);
    cout<<"Operation: str3.find(cstr3)"<<endl;
    if(index5 != npos)
        cout<<"The index of the 1st element of 'part' "
        <<"in str3 is: "<<unsigned int(index5)<<endl;
    else
        cout<<"The substring 'part' was not found in str3"<<endl;
    cout<<endl;

    const char *cstr4 = "ar";
    index6 = str3.find(cstr4, index5 + 1, 2);
    cout<<"Operation: str3.find(cstr4, index5 + 1, 2)"<<endl;
    if(index6 != npos)
        cout<<"The index of the next occurrence of 'ar' in "
        <<"str3 begins at: "<<unsigned int(index6)<<endl;
    else
        cout<<"There is no next occurrence of 'ar' in str3."<<endl;
    cout<<endl;

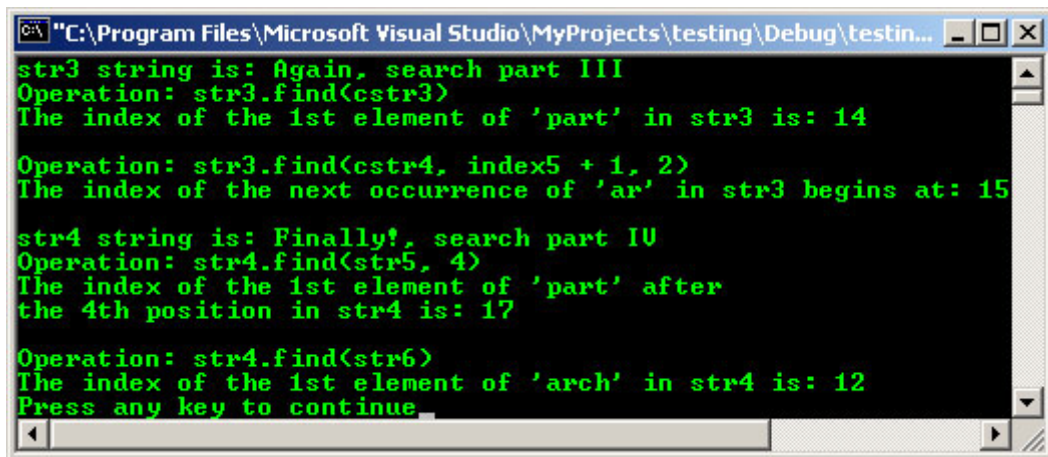
    //-----
    //searching a string for a substring as specified by a string
    string str4("Finally!, search part IV");
    cout<<"str4 string is: "<<str4<<endl;
    basic_string <char>::size_type index7, index8;

    string str5("part");
    index7 = str4.find(str5, 4);
    cout<<"Operation: str4.find(str5, 4)"<<endl;
    if(index7 != npos)
        cout<<"The index of the 1st element of 'part' "
        <<"after\nthe 4th position in str4 is: "<<unsigned int(index7)<<endl;
    else
        cout<<"The substring 'part' was not found in str4"<<endl;
    cout<<endl;

    string str6("arch");
    index8 = str4.find(str6);
    cout<<"Operation: str4.find(str6)"<<endl;
    if(index8 != npos)
        cout<<"The index of the 1st element of 'arch' in "
        <<"str4 is: "<<unsigned int(index8)<<endl;
    else
        cout<<"The substring 'arch' was not found in str4"<<endl;
    return 0;
}
```

```
}
```

Output:



```
C:\Program Files\Microsoft Visual Studio\MyProjects\testing\Debug\testin...
str3 string is: Again, search part III
Operation: str3.find(cstr3)
The index of the 1st element of 'part' in str3 is: 14

Operation: str3.find(cstr4, index5 + 1, 2)
The index of the next occurrence of 'ar' in str3 begins at: 15

str4 string is: Finally!, search part IV
Operation: str4.find(str5, 4)
The index of the 1st element of 'part' after
the 4th position in str4 is: 17

Operation: str4.find(str6)
The index of the 1st element of 'arch' in str4 is: 12
Press any key to continue
```

`find_first_not_of()`

- The return value is the index of the first character of the substring searched for when successful; otherwise `npos`.

```
//find_first_not_of() part I
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //searching a single character in a string
    string str1("Testing the find_first_not_of() part 1");
    cout<<"str1 string is: "<<str1<<endl;
    basic_string<char>::size_type index1, index2;
    static const basic_string<char>::size_type npos = -1;

    index1 = str1.find_first_not_of('_', 3);
    cout<<"Operation: str1.find_first_not_of('_', 3)"<<endl;
    if(index1 != npos)
        cout<<"The index of the 1st '_' found after the 3rd\nposition in str1 is: "<<unsigned
int(index1)<<endl;
    else
        cout<<"The character '_' was not found in str1"<<endl;

    index2 = str1.find_first_not_of('T');
    cout<<"\nOperation: str1.find_first_not_of('T')"<<endl;
    if(index2 != npos)
        cout<<"The index of the 'non T' found in str1 is: "<<unsigned int(index2)<<endl;
    else
        cout<<"The character 'non T' was not found in str1."<<endl;
    cout<<endl;

    //-----
    //searching a string for a substring as specified by a C-string
    string str2("Testing the find_first_not_of() part 2");
    cout<<"str2 string is: "<<str2<<endl;
    basic_string<char>::size_type index3, index4;
    const char *cstr2 = "df";

    index3 = str2.find_first_not_of(cstr2, 4);
    cout<<"Operation: str2.find_first_not_of(cstr2, 4)"<<endl;
    if(index3 != npos)
        cout<<"The index of the 1st occurrence of an element\nof 'df' in str2 after the 4th "
        <<"position is: "<<unsigned int(index3)<<endl;
    else
        cout<<"Elements of the substring 'df' were not\nfound in str2 after the 4th position."<<endl;

    const char *cstr3 = "gz";
    index4 = str2.find_first_not_of(cstr3);
    cout<<"\nOperation: str2.find_first_not_of(cstr3)"<<endl;
    if(index4 != npos)
```



```

    cout<<"The index of the 1st element of 'gz' after\nthe 0th position in str2 is: "
        <<unsigned int(index4)<<endl;
else
    cout<<"The substring 'gz' was not found in str2"<<endl;
return 0;
}

```

Output:

```

e:\projectvc\string\string\Debug\string.exe
str1 string is: Testing the find_first_not_of() part 1
Operation: str1.find_first_not_of('_', 3)
The index of the 1st '_' found after the 3rd
position in str1 is: 3

Operation: str1.find_first_not_of('T')
The index of the 'non T' found in str1 is: 1

str2 string is: Testing the find_first_not_of() part 2
Operation: str2.find_first_not_of(cstr2, 4)
The index of the 1st occurrence of an element
of 'df' in str2 after the 4th position is: 4

Operation: str2.find_first_not_of(cstr3)
The index of the 1st element of 'gz' after
the 0th position in str2 is: 0
Press any key to continue

```

```

//find_first_not_of() part II
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //searching a string for a substring as specified by a C-string
    string str3("Testing the find_first_not_of() part 3");
    cout<<"str3 string is: "<<str3<<endl;
    basic_string<char>::size_type index5, index6;
    static const basic_string<char>::size_type npos = -1;

    const char *cstr4 = "nro";
    index5 = str3.find_first_not_of(cstr4);
    cout<<"Operation: str3.find_first_not_of(cstr4)"<<endl;
    if(index5 != npos)
        cout<<"The index of the 1st occurrence of an "
            <<"element in str3\nother than one of the "
            <<"characters in 'nro' is: "<<unsigned int(index5)<<endl;
    else
        cout<<"Elements in str3 contain only characters "
            <<" in the string 'nro'. "<<endl;

    const char *cstr5 = "nro";
    index6 = str3.find_first_not_of(cstr5, index5+1, 2);
    cout<<"\nOperation: str3.find_first_not_of(cstr5, index5+1, 2)"<<endl;
    if(index6 != npos)
        cout<<"The index of the second occurrence of an "
            <<"element\nof 'nro' in str3 after the 0th "
            <<"position is: "<<unsigned int(index6)<<endl<<endl;
    else
        cout<<"Elements in str3 contain only characters "
            <<" in the string 'nro'"<<endl;
    cout<<endl;

    //-----
    //searching a string for a substring as specified by a string
    string str4("Testing the find_first_not_of() part 4");
    cout<<"str4 string is: "<<str4<<endl;
    basic_string<char>::size_type index7, index8;

    string str5("tf7");
    index7 = str4.find_first_not_of(str5, 3);
    cout<<"Operation: str4.find_first_not_of(str5, 3)"<<endl;

    if(index7 != npos)

```

```

        cout<<"The index of the 1st non occurrence of an element\nof 'tf7' "
            <<"in str4 after the 3rd position is: "<<unsigned int(index7)<<endl;
    else
        cout<<"Elements other than those in the substring 'tf7' "
            <<"were not found in the string str4."<<endl;

    string str6("in");
    index8 = str4.find_first_not_of(str6);
    cout<<"\nOperation: str4.find_first_not_of(str6)"<<endl;

    if(index8 != npos)
        cout<<"The index of the 1st occurrence of an "
            <<"element of\n'in' in str4 after the 0th "
            <<"position is: "<<unsigned int(index8)<<endl;
    else
        cout<<"Elements other than those in the substring"
            <<" 'in' were not found in the string str4."<<endl;
    return 0;
}

```

Output:

```

e:\projectvc\string\string\Debug\string.exe
str3 string is: Testing the find_first_not_of() part 3
Operation: str3.find_first_not_of(cstr4)
The index of the 1st occurrence of an element in str3
other than one of the characters in 'nro' is: 0

Operation: str3.find_first_not_of(cstr5, index5+1, 2)
The index of the second occurrence of an element
of 'nro' in str3 after the 0th position is: 1

str4 string is: Testing the find_first_not_of() part 4
Operation: str4.find_first_not_of(str5, 3)
The index of the 1st non occurrence of an element
of 'tf7' in str4 after the 3rd position is: 4

Operation: str4.find_first_not_of(str6)
The index of the 1st occurrence of an element of
'in' in str4 after the 0th position is: 0
Press any key to continue

```

find_first_of()

- The return value is the index of the first character of the substring searched for when successful; otherwise npos.

```

//find_first_of() part I
#include <string>
#include <iostream>
using namespace std;

int main( )
{
    //searching for a single character in a string
    string str1("find_first_of()");
    cout<<"str1 string is: "<<str1<<endl;
    basic_string <char>::size_type index1, index2;
    static const basic_string <char>::size_type npos = -1;

    index1 = str1.find_first_of('r', 3);
    cout<<"Operation: str1.find_first_of('r', 3)"<<endl;
    if(index1 != npos)
        cout<<"The index of the 1st 'r' found after the 3rd\n"
            <<"position in str1 is: "<<unsigned int(index1)<<endl;
    else
        cout<<"The character 'r' was not found in str1"<<endl;

    index2 = str1.find_first_of('z');
    cout<<"\nOperation: str1.find_first_of('z')"<<endl;
    if(index2 != npos)
        cout<<"The index of the 'z' found in str1 is: "<<unsigned int(index2)<<endl;
    else
        cout<<"The character 'z' was not found in str1."<<endl;
}

```

```
//-----
//searching a string for a substring as specified by a C-string
string str2("Testing 123...Testing 123");
cout<<"\nstr2 string is: "<<str2<<endl;
basic_string<char>::size_type index3, index4;

const char *cstr = "s1";
index3 = str2.find_first_of(cstr, 3);
cout<<"Operation: str2.find_first_of(cstr, 3)"<<endl;
if(index3 != npos)
    cout<<"The index of the 1st occurrence of an "
        <<"element\nof 's1' in str2 after the 3rd "
        <<"position is: "<<unsigned int(index3)<<endl;
else
    cout<<"Elements of the substring 's1' were not\n"
        <<"found in str2 after the 3rd position."<<endl;

const char *cstr1 = "g3";
index4 = str2.find_first_of(cstr1);
cout<<"\nOperation: str2.find_first_of(cstr1)"<<endl;
if(index4 != npos)
    cout <<"The index of the 1st element of 'g3'\n"
        <<"after the 0th position in str2 is: "<<unsigned int(index4)<<endl;
else
    cout<<"The substring 'g3' was not found in str2."<<endl;
return 0;
}
```

Output:

```
e:\projectvc\string\string\Debug\string.exe
str1 string is: find_first_of()
Operation: str1.find_first_of('r', 3)
The index of the 1st 'r' found after the 3rd
position in str1 is: 7

Operation: str1.find_first_of('z')
The character 'z' was not found in str1.

str2 string is: Testing 123...Testing 123
Operation: str2.find_first_of(cstr, 3)
The index of the 1st occurrence of an element
of 's1' in str2 after the 3rd position is: 8

Operation: str2.find_first_of(cstr1)
The index of the 1st element of 'g3'
after the 0th position in str2 is: 6
Press any key to continue
```

```
//find_first_of() part II
#include <string>
#include <iostream>
using namespace std;

int main( )
{
    //searching a string for a substring as specified by a C-string
    string str3("Testing 456...Testing 456...789");
    cout<<"str3 string is: "<<str3<<endl;
    basic_string<char>::size_type index5, index6;
    static const basic_string<char>::size_type npos = -1;

    const char *cstr2 = "t6";
    index5 = str3.find_first_of(cstr2);
    cout<<"Operation: str3.find_first_of(cstr2)"<<endl;
    if(index5 != npos)
        cout<<"The index of the 1st occurrence of an "
            <<"element\nof 't6' in str3 after the 0th "
            <<"position is: "<<unsigned int(index5)<<endl;
    else
        cout<<"Elements of the substring 't6' were not\n"
            <<"found in str3 after the 0th position."<<endl;

    const char *cstr3 = "t68";
    index6 = str3.find_first_of(cstr3, index5 + 1, 2);
```

```

        cout<<"\nOperation: str3.find_first_of(cstr3, index5 + 1, 2)"<<endl;
    if(index6 != npos)
        cout<<"The index of the second occurrence of an "
            <<"element\nof 't68' in str3 after the 0th "
            <<"position is: "<<unsigned int(index6)<<endl;
    else
        cout<<"Elements of the substring 't68' were not\n"
            <<"found in str3 after the first occurrence."<<endl;
    cout<<endl;

//-----
//searching a string for a substring as specified by a string
string str4("find_first_of() and find_first_of()");
cout<<"str4 string is: "<<str4<<endl;
basic_string<char>::size_type index7, index8;

string str5("dfz");
index7 = str5.find_first_of(str4, 3);
cout<<"Operation: str5.find_first_of(str5, 3)"<<endl;
if(index7 != npos)
    cout<<"The index of the 1st occurrence of an "
        <<"element\nof 'dfz' in str4 after the 3rd "
        <<"position is: "<<unsigned int(index7)<<endl;
else
    cout<<"Elements of the substring 'dfz' were not\n"
        <<"found in str4 after the 3rd position."<<endl;

string str6("fo");
index8 = str4.find_first_of(str6);
cout<<"\nOperation: str4.find_first_of(str6)"<<endl;
if(index8 != npos)
    cout<<"The index of the 1st occurrence of an "
        <<"element\nof 'fo' in str4 after the 0th "
        <<"position is: "<<unsigned int(index8)<<endl;
else
    cout<<"Elements of the substring 'fo' were not\n"
        <<"found in str4 after the 0th position."<<endl;
    return 0;
}

```

Output:

```

e:\projectvc\string\string\Debug\string.exe
str3 string is: Testing 456...Testing 456...789
Operation: str3.find_first_of(cstr2)
The index of the 1st occurrence of an element
of 't6' in str3 after the 0th position is: 3

Operation: str3.find_first_of(cstr3, index5 + 1, 2)
The index of the second occurrence of an element
of 't68' in str3 after the 0th position is: 10

str4 string is: find_first_of() and find_first_of()
Operation: str5.find_first_of(str5, 3)
Elements of the substring 'dfz' were not
found in str4 after the 3rd position.

Operation: str4.find_first_of(str6)
The index of the 1st occurrence of an element
of 'fo' in str4 after the 0th position is: 0
Press any key to continue

```

- To be continued on next Module :o).
- For Cs' character and string manipulations please refer to [ModuleX](#).
- The following is a program example compiled using **g++**.

```

//*****string.cpp*****
//append()
#include <string>
#include <iostream>
using namespace std;

int main()
{
    //appending a C-string to a string
}

```

```

string str1("Playing ");
const char *str2 = "with a string";

cout<<"str1 is: "<<str1<<endl;
cout<<"str2, C string is: "<<str2<<endl;
str1.append(str2);
cout<<"Operation: str1.append(str2)"<<endl;
cout<<"Appending str2 to str1: "<<str1<<endl;

//appending part of a C-string to a string
string str3 ("Replaying ");
const char *str4 = "the string ";

cout<<"\nstr3 string is: "<<str3<<endl;
cout<<"str4 C-string is: "<<str4<<endl;
str3.append(str4, 6);
cout<<"Operation: str3.append(str4, 6)"<<endl;
cout<<"Appending part of the str4 to string str3: \n"<<str3<<endl;

//appending part of one string to another
string str5("Again "), str6("string manipulation");

cout<<"\nstr5 is: "<<str5<<endl;
cout<<"str6 is: "<<str6<<endl;
str5.append(str6, 4, 6);
cout<<"Operation: str5.append(str6, 4, 6)"<<endl;
cout<<"The appended string is: "<<str5<<endl;

//appending one string to another in two ways,
//comparing append and operator []
string str7("First "), str8("Second "), str9("Third ");
cout<<"\nstr7 is: "<<str7<<"\nstr8 is: "<<str8<<"\nstr9 is: "<<str9<<endl;
str7.append(str8);
cout<<"Operation: str7.append(str8)"<<endl;
cout<<"The appended string str7 is: "<<str7<<endl;
str7 += str9;
cout<<"Operation: str7 += str9"<<endl;
cout<<"The re appended string is: "<<str7<<endl;

//appending characters to a string
string str10("What string");
cout<<"\nstr10 string is: "<<str10<<endl;
str10.append(3, '?');
cout<<"Operation: str10.append(3, '?)"<<endl;
cout<<"str10 string appended with ? is: "<<str10<<endl;

//appending a range of one string to another
string str11("Finally "), str12("comes the END ");
cout<<"\nstr11 is: "<<str11<<" str12 is: "<<str12<<endl;
str11.append(str12.begin() + 6, str12.end() - 1);
cout<<"Operation:\nstr11.append(str12.begin() + 6, str12.end() - 1)"<<endl;
cout<<"The appended str11 String is: "<<str11<<endl;
return 0;
}

```

[bodo@bakawali ~]\$ g++ string.cpp -o stringapp

[bodo@bakawali ~]\$./stringapp

```

str1 is: Playing
str2, C string is: with a string
Operation: str1.append(str2)
Appending str2 to str1: Playing with a string

```

```

str3 string is: Replaying
str4 C-string is: the string
Operation: str3.append(str4, 6)
Appending part of the str4 to string str3:
Replaying the st

```

```

str5 is: Again
str6 is: string manipulation
Operation: str5.append(str6, 4, 6)
The appended string is: Again ng man

```

```

str7 is: First
str8 is: Second
str9 is: Third
Operation: str7.append(str8)
The appended string str7 is: First Second

```

```
Operation: str7 += str9
The re appended string is: First Second Third

str10 string is: What string
Operation: str10.append(3, '?')
str10 string appended with ? is: What string???

str11 is: Finally str12 is: comes the END
Operation:
str11.append(str12.begin() + 6, str12.end() - 1)
The appended str11 String is: Finally the END
```

-----To be continued-----
---www.tenouk.com---

Further reading and digging:

1. Check the [best selling C / C++ and STL books at Amazon.com](#).