MODULE 26
C++ CHARACTER AND STRING MANIPULATION PART II

My Training Period: hours

Note:
This is continuation from the previous Module. Program examples in this Module compiled using Visual C++ 6.0 with SP6 and Visual C++ .Net. Some program examples may generate warning and runtime errors caused by buffer/stack overflow. These good compilers have some protection for errors :o). **g++** (run on Fedora 3 machine) examples, given at the end of this Module.

Abilities

- Able to understand and use string template classes of the <`string`> in manipulating character and string in C++.
- Able to understand the functionalities string template classes of the <`string`> in manipulating character and string in C++.
- Able to appreciate the usefulness and use these string template classes in your own programs.

26.1 Continuation from previous Module…

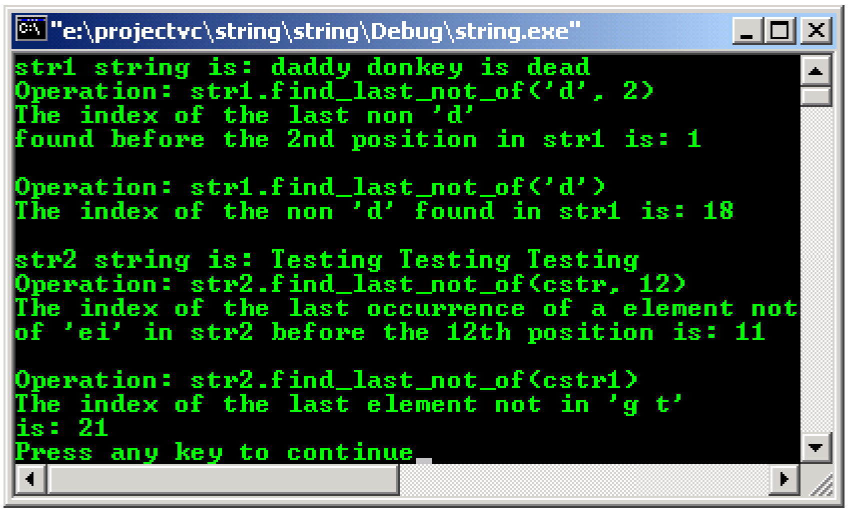
find_last_not_of()

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

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

int main()
{
  //searching for a single character in a string
   string str1("daddy donkey is dead");
   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_last_not_of('d', 2);
   cout<<"Operation: str1.find_last_not_of('d', 2)"<<endl;
   if(index1 != npos)
      cout<<"The index of the last non 'd'\nfound before the "
           <<"2nd position in str1 is: "<<unsigned int(index1)<< endl;
   else
      cout<<"The non 'd' character was not found."<<endl;

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

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

   const char *cstr = "ei";
   index3 = str2.find_last_not_of(cstr, 12);
   cout<<"Operation: str2.find_last_not_of(cstr, 12)"<<endl;
   if(index3 != npos)
      cout<<"The index of the last occurrence of a "
           <<"element not\nof 'ei' in str2 before the 12th "
           <<"position is: "<<unsigned int(index3)<<endl;
   else
      cout<<"Elements not of the substring 'ei' were not "
```

```
          <<"\n found in str2 before the 12th position."<<endl;

   const char *cstr1 = "g t";
   index4 = str2.find_last_not_of(cstr1);
   cout<<"\nOperation: str2.find_last_not_of(cstr1)"<<endl;
   if(index4 != npos)
      cout<<"The index of the last element not "
          <<"in 'g t'\nis: "<<unsigned int(index4)<<endl;
   else
      cout<<"The elements of the substring 'g t' were "
          <<"not found in str2"<<endl;
   return 0;
}
```

Output:

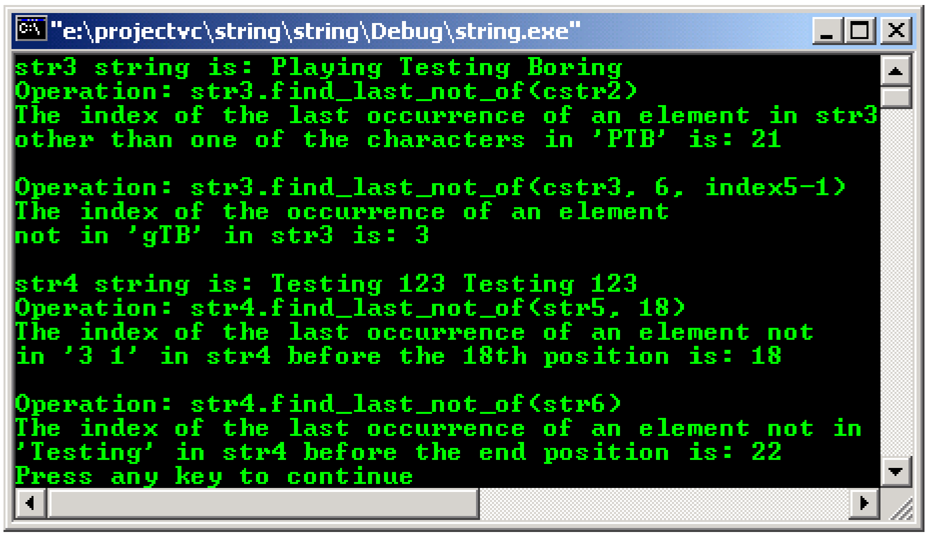


```
str1 string is: daddy donkey is dead
Operation: str1.find_last_not_of('d', 2)
The index of the last non 'd'
found before the 2nd position in str1 is: 1

Operation: str1.find_last_not_of('d')
The index of the non 'd' found in str1 is: 18

str2 string is: Testing Testing Testing
Operation: str2.find_last_not_of(cstr, 12)
The index of the last occurrence of a element not
of 'ei' in str2 before the 12th position is: 11

Operation: str2.find_last_not_of(cstr1)
The index of the last element not in 'g t'
is: 21
Press any key to continue
```

```
//find_last_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("Playing Testing Boring");
   cout<<"str3 string is: "<<str3<<"\n";
   basic_string <char>::size_type index5, index6;
   static const basic_string <char>::size_type npos = -1;

   const char *cstr2 = "PTB";
   index5 = str3.find_last_not_of(cstr2);
   cout<<"Operation: str3.find_last_not_of(cstr2)"<<endl;
   if(index5 != npos)
      cout<<"The index of the last occurrence of an "
          <<"element in str3\nother than one of the "
          <<"characters in 'PTB' is: "<<unsigned int(index5)<<endl;
   else
      cout<<"Elements in str3 contain only characters in the string 'PTB'"<<endl;

   const char *cstr3 = "gTB";
   index6 = str3.find_last_not_of(cstr3, 6, index5-1);
   cout<<"\nOperation: str3.find_last_not_of(cstr3, 6, index5-1)"<<endl;
   if(index6 != npos)
      cout<<"The index of the occurrence of an "
          <<"element\nnot in 'gTB' in str3 is: "
          <<unsigned int(index6)<<endl;
   else
      cout<<"Elements in str3 contains only characters "
          <<"in the string 'gTB'."<<endl;
cout<<endl;

//-------------------------------------------------------------
//searching a string for a substring as specified by a string
   string str4("Testing 123 Testing 123");
   cout<<"str4 string is: "<<str4<<"\n";
   basic_string <char>::size_type index7, index8;
```

```
    string str5("3 1");
    index7 = str4.find_last_not_of(str5, 18);
    cout<<"Operation: str4.find_last_not_of(str5, 18)"<<endl;
    if(index7 != npos)
       cout<<"The index of the last occurrence of an "
            <<"element not\nin '3 1' in str4 before the 18th "
            <<"position is: "<<unsigned int(index7)<<endl;
    else
       cout<<"Elements other than those in the substring"
            <<" '3 1' were not found in the string str4"<<endl;

    string str6("Testing");
    index8 = str4.find_last_not_of(str6);
    cout<<"\nOperation: str4.find_last_not_of(str6)"<<endl;
    if(index8 != npos)
       cout<<"The index of the last occurrence of an "
           <<"element not in\n'Testing' in str4 before the end "
           <<"position is: "<<unsigned int(index8)<<endl;
    else
       cout<<"Elements other than those in the substring\n"
           <<"'Testing' were not found in the string str4"<<endl;
    return 0;
 }
```

Output:



```
str3 string is: Playing Testing Boring
Operation: str3.find_last_not_of(cstr2)
The index of the last occurrence of an element in str3
other than one of the characters in 'PTB' is: 21

Operation: str3.find_last_not_of(cstr3, 6, index5-1)
The index of the occurrence of an element
not in 'gTB' in str3 is: 3

str4 string is: Testing 123 Testing 123
Operation: str4.find_last_not_of(str5, 18)
The index of the last occurrence of an element not
in '3 1' in str4 before the 18th position is: 18

Operation: str4.find_last_not_of(str6)
The index of the last occurrence of an element not in
'Testing' in str4 before the end position is: 22
Press any key to continue
```

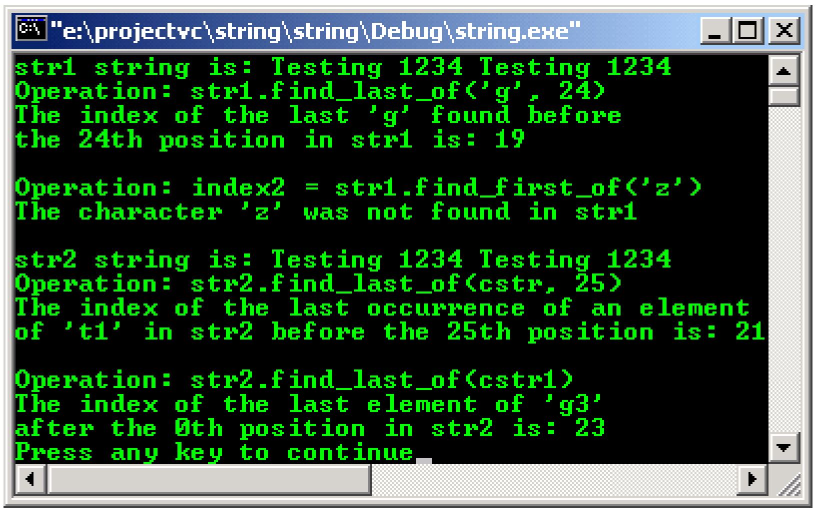
find_last_of()

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

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

int main()
{
   //searching for a single character in a string
   string str1("Testing 1234 Testing 1234");
   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_last_of('g', 24);
   cout<<"Operation: str1.find_last_of('g', 24)"<<endl;
   if(index1 != npos)
      cout<<"The index of the last 'g' found before\nthe 24th"
          <<" position in str1 is: "<<unsigned int(index1)<<endl;
   else
      cout<<"The character 'g' was not found in str1"<<endl;

   index2 = str1.find_first_of('z');
   cout<<"\nOperation: index2 = 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;
   cout<<endl;

   //---------------------------------------------------
   //searching a string for a substring as specified by a C-string
   string str2("Testing 1234 Testing 1234");
   cout<<"str2 string is: "<<str2<<endl;
   basic_string <char>::size_type index3, index4;

   const char *cstr = "t1";
   index3 = str2.find_last_of(cstr, 25);
   cout<<"Operation: str2.find_last_of(cstr, 25)"<<endl;
   if(index3 != npos)
      cout<<"The index of the last occurrence of an "
          <<"element\nof 't1' in str2 before the 25th "
          <<"position is: "<<unsigned int(index3)<<endl;
   else
      cout<<"Elements of the substring 't1' were not\n"
          <<"found in str2 before the 25th position."<<endl;

   const char *cstr1 = "g3";
   index4 = str2.find_last_of(cstr1);
   cout<<"\nOperation: str2.find_last_of(cstr1)"<<endl;
   if(index4 != npos)
      cout<<"The index of the last 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:

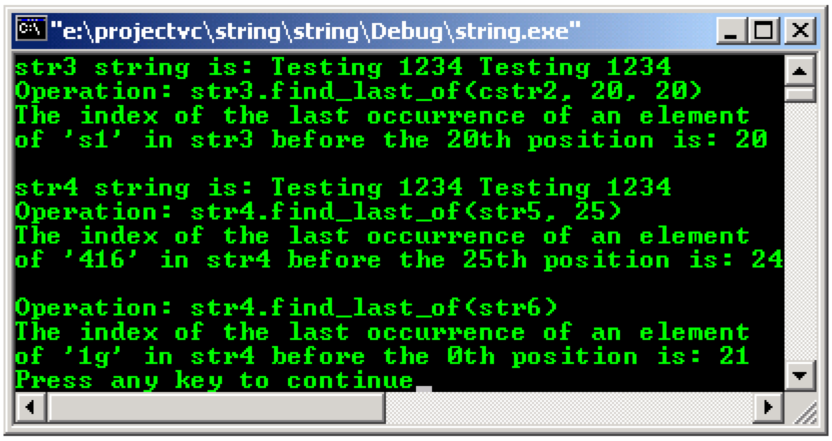


```
str1 string is: Testing 1234 Testing 1234
Operation: str1.find_last_of('g', 24)
The index of the last 'g' found before
the 24th position in str1 is: 19

Operation: index2 = str1.find_first_of('z')
The character 'z' was not found in str1

str2 string is: Testing 1234 Testing 1234
Operation: str2.find_last_of(cstr, 25)
The index of the last occurrence of an element
of 't1' in str2 before the 25th position is: 21

Operation: str2.find_last_of(cstr1)
The index of the last element of 'g3'
after the 0th position in str2 is: 23
Press any key to continue
```

```
//find_last_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 1234 Testing 1234");
   cout<<"str3 string is: "<<str3<<endl;
   basic_string <char>::size_type index5;
   static const basic_string <char>::size_type npos = -1;

   const char *cstr2 = "s1";
   index5 = str3.find_last_of(cstr2, 20, 20);
   cout<<"Operation: str3.find_last_of(cstr2, 20, 20)"<<endl;
   if(index5 != npos)
      cout<<"The index of the last occurrence of an "
          <<"element\nof 's1' in str3 before the 20th "
          <<"position is: "<<unsigned int(index5)<<endl;
```

```
    else
       cout<<"Elements of the substring 's1' were not\n"
           <<"found in str3 before the 20th position."<<endl;
    cout<<endl;

 //---------------------------------------------------------
    //searching a string for a substring as specified by a string
    string str4("Testing 1234 Testing 1234");
    cout<<"str4 string is: "<<str4<<endl;
    basic_string <char>::size_type index6, index7;

    string str5("416");
    index6 = str4.find_last_of(str5, 25);
    cout<<"Operation: str4.find_last_of(str5, 25)"<<endl;
    if(index6 != npos)
       cout<<"The index of the last occurrence of an "
           <<"element\nof '416' in str4 before the 25th "
           <<"position is: "<<unsigned int(index6)<<endl;
    else
       cout<<"Elements of the substring '416' were not\n"
           <<"found in str4 after the 0th position"<<endl;

    string str6("1g");
    index7 = str4.find_last_of(str6);
    cout<<"\nOperation: str4.find_last_of(str6)"<<endl;
    if(index7 != npos)
       cout<<"The index of the last occurrence of an "
           <<"element\nof '1g' in str4 before the 0th "
           <<"position is: "<<unsigned int(index7)<<endl;
    else
       cout<<"Elements of the substring '1g' were not\n"
            <<"found in str4 after the 0th position"<< endl;
    return 0;
 }
```

Output:


```
str3 string is: Testing 1234 Testing 1234
Operation: str3.find_last_of(cstr2, 20, 20)
The index of the last occurrence of an element
of 's1' in str3 before the 20th position is: 20

str4 string is: Testing 1234 Testing 1234
Operation: str4.find_last_of(str5, 25)
The index of the last occurrence of an element
of '416' in str4 before the 25th position is: 24

Operation: str4.find_last_of(str6)
The index of the last occurrence of an element
of '1g' in str4 before the 0th position is: 21
Press any key to continue
```

get_allocator()

- This member function returns the stored allocator object.
- Allocators for the string class specify how the class manages storage. The default allocators supplied with container classes are sufficient for most programming needs.
- Writing and using your own allocator class is an advanced C++ topic and its usage is very specific.
- The return value is the allocator used by the string.

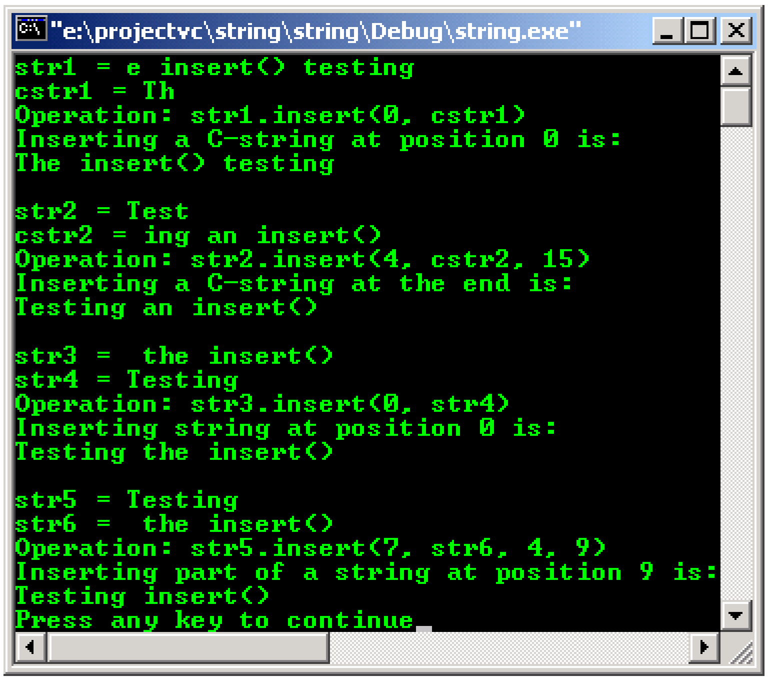
```
//get_allocator()
#include <string>
#include <iostream>
using namespace std;

int main()
{
   //using the default allocator.
   string str1;
   basic_string <char> str2;
   basic_string <char, char_traits<char>, allocator<char> > str3;

   //str4 will use the same allocator class as str1
   basic_string <char> str4(str1.get_allocator());

   basic_string <char>::allocator_type xchar = str1.get_allocator();
   //You can now call functions on the allocator class xchar used by str1
  string str5(xchar);
  return 0;
}
```

insert()

- The return value is either a reference to the string object that is being assigned new characters by the member function or, in the case of individual character insertions, an iterator addressing the position of the character inserted, or none, depending on the particular member function.

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

int main()
{
//inserting a C-string at a given position
basic_string <char> str1("e insert() testing");
const char *cstr1 = "Th";

cout<<"str1 = "<<str1<<endl;
cout<<"cstr1 = "<<cstr1<<endl;
str1.insert(0, cstr1);
cout<<"Operation: str1.insert(0, cstr1)"<<endl;
cout<<"Inserting a C-string at position 0 is:\n"<<str1<<endl;
cout<<endl;

//inserting a C-string at a given position for a specified number of elements
basic_string <char> str2("Test");
const char *cstr2 = "ing an insert()";

cout<<"str2 = "<<str2<<endl;
cout<<"cstr2 = "<<cstr2<<endl;
str2.insert(4, cstr2, 15);
cout<<"Operation: str2.insert(4, cstr2, 15)"<<endl;
cout<<"Inserting a C-string at the end is:\n"<<str2<<endl;
cout<<endl;

//inserting a string at a given position
basic_string <char> str3(" the insert()");
string str4("Testing");

cout<<"str3 = "<<str3<<endl;
cout<<"str4 = "<<str4<<endl;
str3.insert(0, str4);
cout<<"Operation: str3.insert(0, str4)"<<endl;
cout<<"Inserting string at position 0 is:\n"<<str3<<endl;
cout<<endl;

//inserting part of a string at a given position
basic_string <char> str5("Testing ");
string str6(" the insert()");

cout<<"str5 = "<<str5<<endl;
cout<<"str6 = "<<str6<<endl;
str5.insert(7, str6, 4, 9);
cout<<"Operation: str5.insert(7, str6, 4, 9)"<<endl;
cout<<"Inserting part of a string at position 9 is:\n"<<str5<<endl;
return 0;
}
```

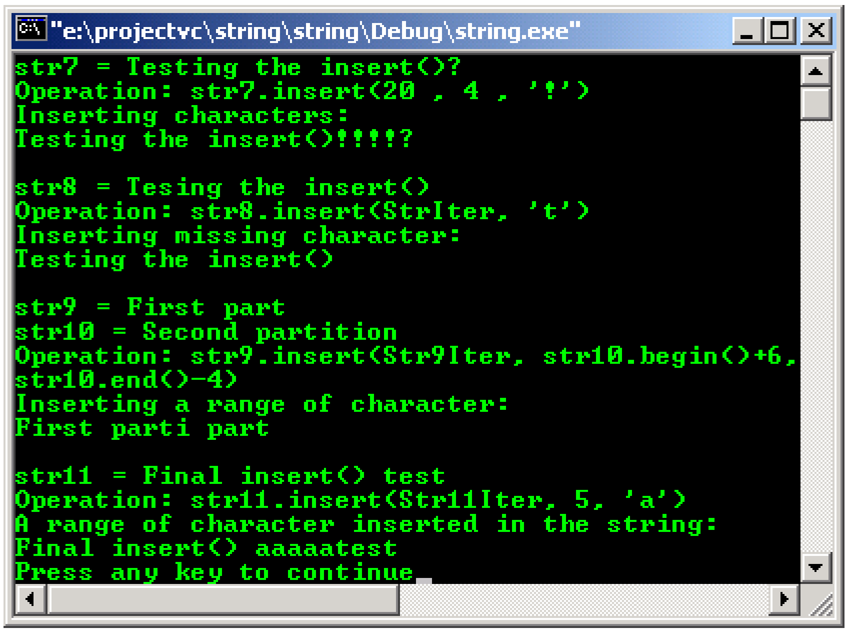
Output:



```
str1 = e insert() testing
cstr1 = Th
Operation: str1.insert(0, cstr1)
Inserting a C-string at position 0 is:
The insert() testing

str2 = Test
cstr2 = ing an insert()
Operation: str2.insert(4, cstr2, 15)
Inserting a C-string at the end is:
Testing an insert()

str3 =  the insert()
str4 = Testing
Operation: str3.insert(0, str4)
Inserting string at position 0 is:
Testing the insert()

str5 = Testing
str6 =  the insert()
Operation: str5.insert(7, str6, 4, 9)
Inserting part of a string at position 9 is:
Testing insert()
Press any key to continue
```

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

int main()
{
//inserting a number of characters at a specified position in the string
string str7("Testing the insert()?");
cout<<"str7 = "<<str7<<endl;
str7.insert(20, 4, '!');
cout<<"Operation: str7.insert(20, 4, '!')"<<endl;
cout<<"Inserting characters: \n"<<str7<<endl;
cout<<endl;

//inserting a character at a specified position in the string
string str8("Tesing the insert()");
cout<<"str8 = "<<str8<<endl;
basic_string <char>::iterator StrIter = (str8.begin() + 3);
str8.insert(StrIter, 't');
cout<<"Operation: str8.insert(StrIter, 't')"<<endl;
cout<<"Inserting missing character: \n"<<str8<<endl;
cout<<endl;

//inserts a range at a specified position in the string
string str9("First part");
string str10("Second partition");
cout<<"str9 = "<<str9<<endl;
cout<<"str10 = "<<str10<<endl;
basic_string <char>::iterator Str9Iter = (str9.begin() + 5);
str9.insert(Str9Iter, str10.begin()+6, str10.end()-4);
cout<<"Operation: str9.insert(Str9Iter, str10.begin()+6,\nstr10.end()-4)"<<endl;
cout<<"Inserting a range of character: \n"<<str9<<endl;
cout<<endl;

//inserting a number of characters at a specified position in the string
string str11("Final insert() test");
cout<<"str11 = "<<str11<<endl;
basic_string <char>::iterator Str11Iter = (str11.begin() + 15);
str11.insert(Str11Iter, 5, 'a');
cout<<"Operation: str11.insert(Str11Iter, 5, 'a')"<<endl;
cout<<"A range of character inserted in the string: \n"<<str11<<endl;
return 0;
}
```

Output:

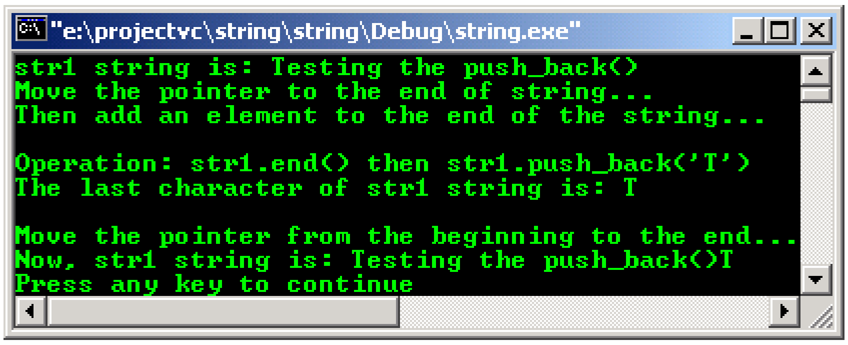


```
str7 = Testing the insert()?
Operation: str7.insert(20 , 4 , '!')
Inserting characters:
Testing the insert()!!!!?

str8 = Tesing the insert()
Operation: str8.insert(StrIter, 't')
Inserting missing character:
Testing the insert()

str9 = First part
str10 = Second partition
Operation: str9.insert(Str9Iter, str10.begin()+6,
str10.end()-4)
Inserting a range of character:
First parti part

str11 = Final insert() test
Operation: str11.insert(Str11Iter, 5, 'a')
A range of character inserted in the string:
Final insert() aaaaatest
Press any key to continue
```

push_back()

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

int main()
{
   string str1("Testing the push_back()");
   basic_string <char>::iterator StrIter, Str1Iter;

   cout<<"str1 string is: ";
   for(StrIter = str1.begin(); StrIter != str1.end(); StrIter++)
      cout<<*StrIter;
   cout<<endl;
   cout<<"Move the pointer to the end of string..."<<endl;
   Str1Iter = str1.end();
   cout<<"Then add an element to the end of the string..."<<endl;
   str1.push_back('T');
   cout<<"\nOperation: str1.end() then str1.push_back('T')"<<endl;
   cout<<"The last character of str1 string is: "
        <<*Str1Iter;
   cout<<endl;

   cout<<"\nMove the pointer from the beginning to the end..."<<endl;
   cout<<"Now, str1 string is: ";
   for(StrIter = str1.begin(); StrIter != str1.end(); StrIter++)
      cout<<*StrIter;
   cout<<endl;
   return 0;
}
```

Output:



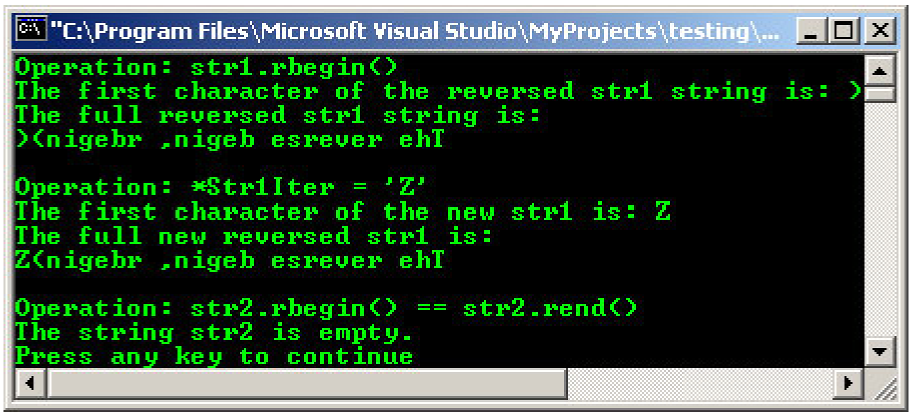
```
str1 string is: Testing the push_back()
Move the pointer to the end of string...
Then add an element to the end of the string...

Operation: str1.end() then str1.push_back('T')
The last character of str1 string is: T

Move the pointer from the beginning to the end...
Now, str1 string is: Testing the push_back()T
Press any key to continue
```

rbegin() and rend()

- `rbegin()` is used with a reversed string just as begin is used with a string.
- If the return value of `rbegin()` is assigned to a `const_reverse_iterator`, the string object cannot be modified. If the return value of `rbegin()` is assigned to a `reverse_iterator`, the string object can be modified.
- `rbegin()` can be used to initialize an iteration through a string backwards.
- The return value of the `rbegin()` is a random-access iterator to the first element in a reversed string, addressing what would be the last element in the corresponding un-reversed string.
- `rend()` is used with a reversed string just as `end` is used with a string.
- If the return value of `rend()` is assigned to a `const_reverse_iterator`, the string object cannot be modified. If the return value of `rend()` is assigned to a `reverse_iterator`, the string object can be modified.
- `rend()`can be used to test whether a reverse iterator has reached the end of its string.
- The return value of the `rend()` is a reverse random-access iterator that addresses the location succeeding the last element in a reversed string. The value returned by `rend()`should not be dereferenced.

```
//rbegin() and rend()
#include <string>
#include <iostream>
using namespace std;

int main()
{
   string str1("The reverse begin, rbegin()"), str2;
   basic_string <char>::reverse_iterator StrIter, Str1Iter;
   basic_string <char>::const_reverse_iterator str1_rcIter;

   //well, no need to minus the null character huh?
   cout<<"Operation: str1.rbegin()"<<endl;
   Str1Iter = str1.rbegin();
   cout<<"The first character of the reversed str1 string is: "
        <<*Str1Iter<<endl;
   cout<<"The full reversed str1 string is:\n";
   //rbegin() should be with rend()
   for(StrIter = str1.rbegin(); StrIter != str1.rend(); StrIter++)
      cout<<*StrIter;
   cout<<"\n\n";

   //The dereferenced iterator can be used to modify a character
   cout<<"Operation: *Str1Iter = 'Z'"<<endl;
   *Str1Iter = 'Z';
   cout<<"The first character of the new str1 is: "
        <<*Str1Iter<<endl;
   cout<<"The full new reversed str1 is:\n";
   for(StrIter = str1.rbegin(); StrIter != str1.rend(); StrIter++)
      cout<<*StrIter;
   cout<<"\n\n";

   //The following line will generate error because iterator is const
   //*str1_rcIter = 'T';

   //For an empty string, rbegin() is equivalent to rend()
   cout<<"Operation: str2.rbegin() == str2.rend()"<<endl;
   if(str2.rbegin() == str2.rend())
      cout<<"The string str2 is empty."<<endl;
   else
      cout<<"The stringstr2 is not empty."<<endl;
   return 0;
}
```

Output:


"C:\Program Files\Microsoft Visual Studio\MyProjects\testing\...

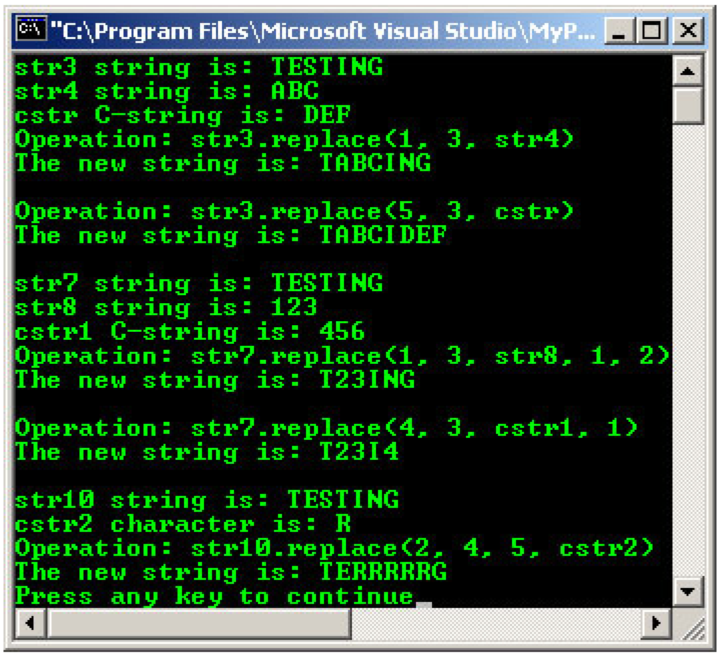
```
Operation: str1.rbegin()
The first character of the reversed str1 string is: )
The full reversed str1 string is:
)(nigebr ,nigeb esrever ehT

Operation: *Str1Iter = 'Z'
The first character of the new str1 is: Z
The full new reversed str1 is:
Z(nigebr ,nigeb esrever ehT

Operation: str2.rbegin() == str2.rend()
The string str2 is empty.
Press any key to continue
```

replace()

- The return value is the operand string with the replacement made.

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

int main()
{

   //replacing part of the string with
   //characters of a string or C-string
   //remember that index start from 0!
   string str1, str2;
   string str3("TESTING");
   string str4("ABC");
   const char* cstr = "DEF";
   cout<<"str3 string is: "<<str3<<endl;
   cout<<"str4 string is: "<<str4<<endl;
   cout<<"cstr C-string is: "<<cstr<<endl;

   str1 = str3.replace(1, 3, str4);
   cout<<"Operation: str3.replace(1, 3, str4)"<<endl;
   cout<<"The new string is: "<<str1<<endl;
   cout<<"\nOperation: str3.replace(5, 3, cstr)"<<endl;
   str2 = str3.replace(5, 3, cstr);
   cout<<"The new string is: "<<str2<<"\n\n";

   //replacing part of the string with characters
   //form part of a string or C-string
   string str5, str6;
   string str7 ("TESTING");
   string str8 ("123");
   const char* cstr1 = "456";

   cout<<"str7 string is: "<<str7<<endl;
   cout<<"str8 string is: "<<str8<<endl;
   cout<<"cstr1 C-string is: "<<cstr1<<endl;
   cout<<"Operation: str7.replace(1, 3, str8, 1, 2)"<<endl;
   str5 = str7.replace(1, 3, str8, 1, 2);
   cout<<"The new string is: "<<str5<<endl;
   cout<<"\nOperation: str7.replace(4, 3, cstr1, 1)"<<endl;
   str6 = str7.replace(4, 3, cstr1, 1);
   cout<<"The new string is: "<<str6<<"\n\n";

   //replacing part of the string with characters
   string str9;
   string str10 ("TESTING");
   char cstr2 = 'R';
   cout<<"str10 string is: "<<str10<<endl;
   cout<<"cstr2 character is: "<<cstr2<<endl;
   cout<<"Operation: str10.replace(2, 4, 5, cstr2)"<<endl;
   str9 = str10.replace(2, 4, 5, cstr2);
   cout<<"The new string is: "<<str9<<endl;
   return 0;
}
```

Output:

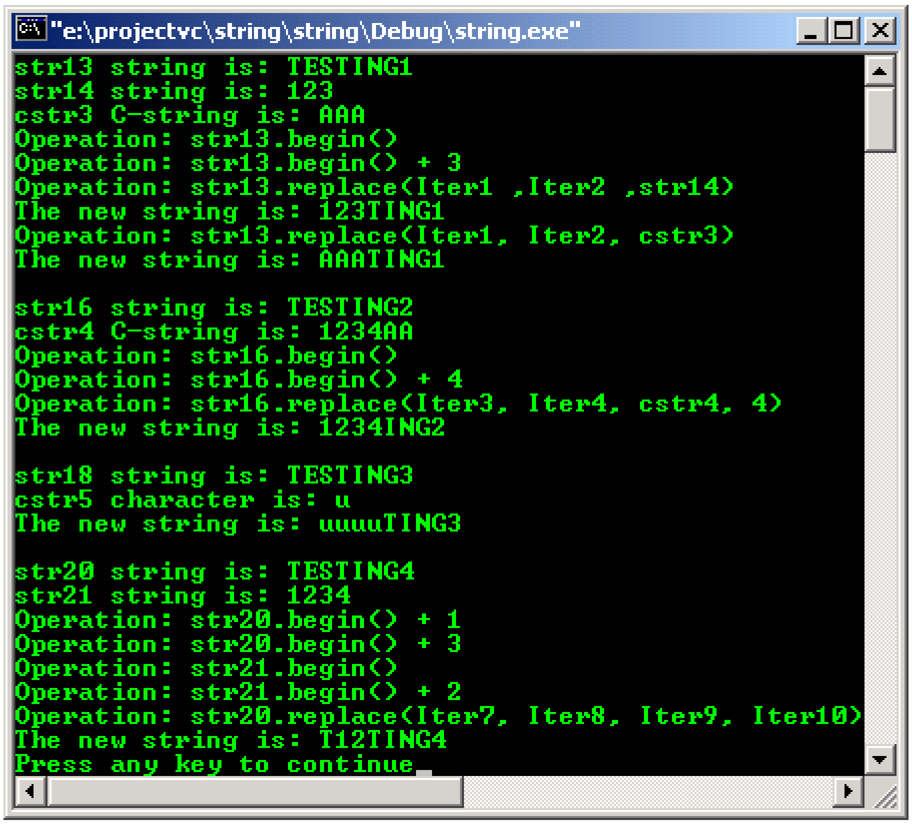


```
str3 string is: TESTING
str4 string is: ABC
cstr C-string is: DEF
Operation: str3.replace(1, 3, str4)
The new string is: TABCING

Operation: str3.replace(5, 3, cstr)
The new string is: TABCIDEF

str7 string is: TESTING
str8 string is: 123
cstr1 C-string is: 456
Operation: str7.replace(1, 3, str8, 1, 2)
The new string is: T23ING

Operation: str7.replace(4, 3, cstr1, 1)
The new string is: T23I4

str10 string is: TESTING
cstr2 character is: R
Operation: str10.replace(2, 4, 5, cstr2)
The new string is: TERRRRRG
Press any key to continue
```

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

int main()
{

//replacing part of the string, delineated with iterators,
//with a string or C-string
   string str11, str12;
   string str13("TESTING1");
   string str14("123");
   const char* cstr3 = "AAA";

   cout<<"str13 string is: "<<str13<<endl;
   cout<<"str14 string is: "<<str14<<endl;
   cout<<"cstr3 C-string is: "<<cstr3<<endl;
   basic_string<char>::iterator Iter1, Iter2;

   cout<<"Operation: str13.begin()"<<endl;
   cout<<"Operation: str13.begin() + 3"<<endl;
   cout<<"Operation: str13.replace(Iter1, Iter2, str14)"<<endl;

   Iter1 = str13.begin();
   Iter2 = str13.begin() + 3;
   str11 = str13.replace(Iter1, Iter2, str14);
   cout<<"The new string is: "<<str11<<endl;
   cout<<"Operation: str13.replace(Iter1, Iter2, cstr3)"<<endl;
   str12 = str13.replace(Iter1, Iter2, cstr3);
   cout<<"The new string is: "<<str12<<"\n\n";

   //replacing part of the string delineated with iterators
   //with a number of C-string characters
   string str15;
   string str16("TESTING2");
   const char* cstr4 = "1234AA";
   cout<<"str16 string is: "<<str16<<endl;
   cout<<"cstr4 C-string is: "<<cstr4<<endl;
   basic_string<char>::iterator Iter3, Iter4;

   cout<<"Operation: str16.begin()"<<endl;
   cout<<"Operation: str16.begin() + 4"<<endl;
   cout<<"Operation: str16.replace(Iter3, Iter4, cstr4, 4)"<<endl;
   Iter3 = str16.begin();
   Iter4 = str16.begin() + 4;
   str15 = str16.replace(Iter3, Iter4, cstr4, 4);
   cout<<"The new string is: "<<str15<<"\n\n";

   //replacing part of the string delineated with iterators
   //with specified characters
```

```
    string str17;
    string str18("TESTING3");
    char cstr5 = 'u';

    cout<<"str18 string is: "<<str18<<endl;
    cout<<"cstr5 character is: "<<cstr5<<endl;
    basic_string<char>::iterator Iter5, Iter6;
    Iter5 = str18.begin();
    Iter6 = str18.begin() + 3;

    str17 = str18.replace(Iter5, Iter6, 4, cstr5);
    cout<<"The new string is: "<<str17<<"\n\n";

    //replacing part of the operand string delineated with iterators
    //with part of a parameter string delineated with iterators
    string str19;
    string str20("TESTING4"); //operand
    string str21("1234");  //parameter
    cout<<"str20 string is: "<<str20<<endl;
    cout<<"str21 string is: "<<str21<<endl;
    basic_string<char>::iterator Iter7, Iter8, Iter9, Iter10;

    cout<<"Operation: str20.begin() + 1"<<endl;
    cout<<"Operation: str20.begin() + 3"<<endl;
    cout<<"Operation: str21.begin()"<<endl;
    cout<<"Operation: str21.begin() + 2"<<endl;

    Iter7 = str20.begin() + 1;
    Iter8 = str20.begin() + 3;
    Iter9 = str21.begin();
    Iter10 = str21.begin() + 2;
    cout<<"Operation: str20.replace(Iter7, Iter8, Iter9, Iter10)"<<endl;
    str19 = str20.replace(Iter7, Iter8, Iter9, Iter10);
    cout<<"The new string is: "<<str19<<endl;
    return 0;
}
```

Output:



"e:\projectvc\string\string\Debug\string.exe"

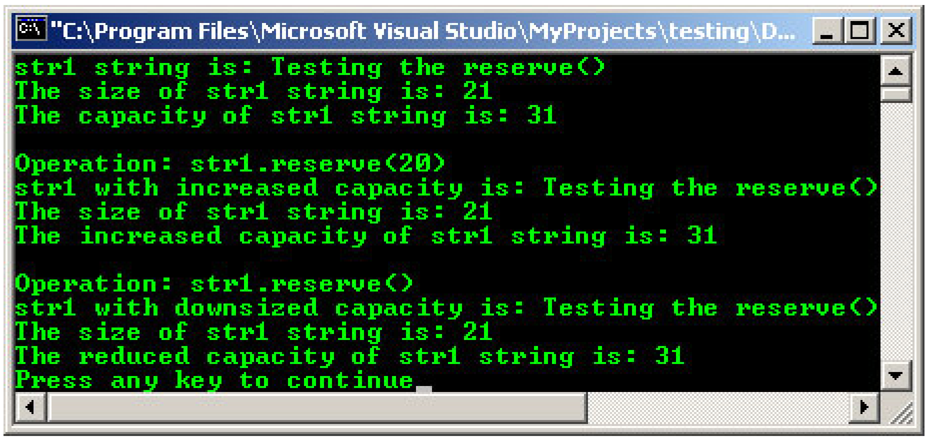
```
str13 string is: TESTING1
str14 string is: 123
cstr3 C-string is: AAA
Operation: str13.begin()
Operation: str13.begin() + 3
Operation: str13.replace(Iter1 ,Iter2 ,str14)
The new string is: 123TING1
Operation: str13.replace(Iter1, Iter2, cstr3)
The new string is: AAATING1

str16 string is: TESTING2
cstr4 C-string is: 1234AA
Operation: str16.begin()
Operation: str16.begin() + 4
Operation: str16.replace(Iter3, Iter4, cstr4, 4)
The new string is: 1234ING2

str18 string is: TESTING3
cstr5 character is: u
The new string is: uuuuTING3

str20 string is: TESTING4
str21 string is: 1234
Operation: str20.begin() + 1
Operation: str20.begin() + 3
Operation: str21.begin()
Operation: str21.begin() + 2
Operation: str20.replace(Iter7, Iter8, Iter9, Iter10)
The new string is: T12TING4
Press any key to continue_
```

reserve()

- Having sufficient capacity is important because reallocations is a time-consuming process and invalidates all references, pointers, and iterators that refer to characters in a string.
- Unlike `vector` (another STL), the member function `reserve()` may be called to shrink the capacity of an object. The request is non binding and may or may not happen.
- The default value for the parameter is zero, a call of `reserve()` is a non-binding request to shrink the capacity of the string to fit the number of characters currently in the string.
- The capacity is never reduced below the current number of characters.

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

int main()
{

   string str1("Testing the reserve()");
   cout<<"str1 string is: "<<str1<<endl;

   basic_string <char>::size_type SizeStr1, Size1Str1;
   SizeStr1 = str1.size();
   basic_string <char>::size_type CapaStr1, Capa1Str1;
   CapaStr1 = str1.capacity();

   //Compare size & capacity of the original string
   cout<<"The size of str1 string is: "<<SizeStr1<<endl;
   cout<<"The capacity of str1 string is: "<<CapaStr1<<"\n\n";

   //Compare size & capacity of the string
   //with added capacity
   cout<<"Operation: str1.reserve(20)"<<endl;
   str1.reserve(20);
   Size1Str1 = str1.size();
   Capa1Str1 = str1.capacity();

   cout<<"str1 with increased capacity is: "<<str1<<endl;
   cout<<"The size of str1 string is: "<<Size1Str1<<endl;
   cout<<"The increased capacity of str1 string is: "<<Capa1Str1<<"\n\n";

   //Compare size & capacity of the string
   //with downsized capacity. Without any parameter,
   //it should shrink to fit the number of the characters
   //currently in the string
   cout<<"Operation: str1.reserve()"<<endl;
   str1.reserve();
   basic_string <char>::size_type Size2Str1;
   basic_string <char>::size_type Capa2Str1;
   Size2Str1 = str1.size();
   Capa2Str1 = str1.capacity();

   cout<<"str1 with downsized capacity is: "<<str1<<endl;
   cout<<"The size of str1 string is: "<<Size2Str1<<endl;
   cout<<"The reduced capacity of str1 string is: "<<Capa2Str1<<endl;
   return 0;
}
```

Output:


"C:\Program Files\Microsoft Visual Studio\MyProjects\testing\D...

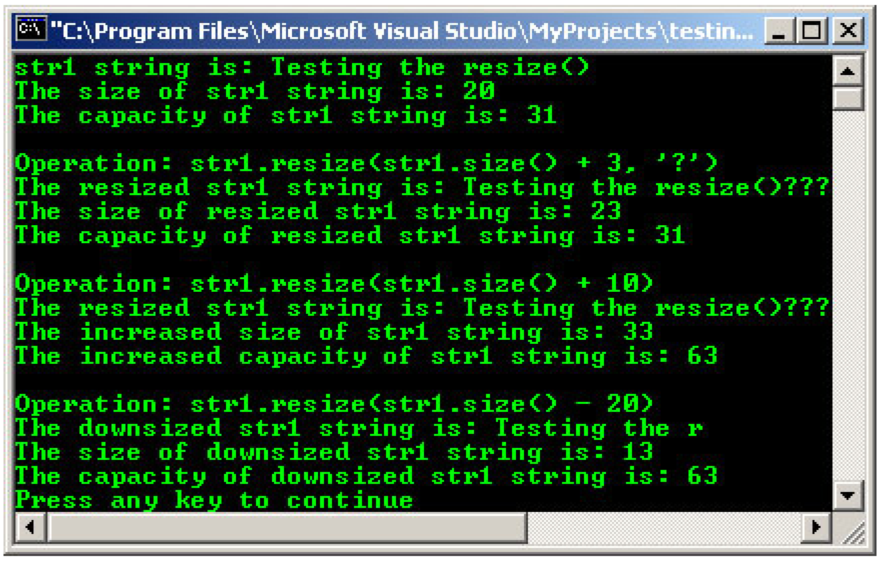
```
str1 string is: Testing the reserve()
The size of str1 string is: 21
The capacity of str1 string is: 31

Operation: str1.reserve(20)
str1 with increased capacity is: Testing the reserve()
The size of str1 string is: 21
The increased capacity of str1 string is: 31

Operation: str1.reserve()
str1 with downsized capacity is: Testing the reserve()
The size of str1 string is: 21
The reduced capacity of str1 string is: 31
Press any key to continue_
```

resize() and size()

- If the resulting size exceeds the maximum number of characters, the form throws `length_error` exception handler.

```
//resize() and size()
#include <string>
#include <iostream>
using namespace std;

int main()
{

   string  str1("Testing the resize()");
   cout<<"str1 string is: "<<str1<<endl;

   basic_string <char>::size_type SizeStr1;
   SizeStr1 = str1.size();
   basic_string <char>::size_type CapaStr1;
   CapaStr1 = str1.capacity();

   //Compare size & capacity of the original string
   cout<<"The size of str1 string is: "<<SizeStr1<<endl;
   cout<<"The capacity of str1 string is: "<<CapaStr1<<endl;

   //Use resize() to increase size by 3 elements
   //of the question mark
   cout<<"\nOperation: str1.resize(str1.size() + 3, '?')"<<endl;
   str1.resize(str1.size() + 3, '?');
   cout<<"The resized str1 string is: "<<str1<<endl;

   SizeStr1 = str1.size();
   CapaStr1 = str1.capacity();

   //Compare size & capacity of a string after resizing
   cout<<"The size of resized str1 string is: "<<SizeStr1<<endl;
   cout<<"The capacity of resized str1 string is: "<<CapaStr1<<endl;

   //Use resize() to increase size by 10 elements:
   cout<<"\nOperation: str1.resize(str1.size() + 10)"<<endl;
   str1.resize(str1.size() + 10);
   cout<<"The resized str1 string is: "<<str1<<endl;

   SizeStr1 = str1.size();
   CapaStr1 = str1.capacity();

   //Compare size & capacity of a string after resizing
   //note capacity increases automatically as required
   cout<<"The increased size of str1 string is: "<<SizeStr1<<endl;
   cout<<"The increased capacity of str1 string is: "<<CapaStr1<<endl;

   //Use resize() to downsize by 20 elements:
   cout<<"\nOperation: str1.resize(str1.size() - 20)"<<endl;
   str1.resize(str1.size() - 20);
   cout<<"The downsized str1 string is: "<<str1<<endl;

   SizeStr1 = str1.size();
   CapaStr1 = str1.capacity();

   //Compare size & capacity of a string after downsizing
   cout<<"The size of downsized str1 string is: "<<SizeStr1<<endl;
   cout<<"The capacity of downsized str1 string is: "<<CapaStr1<<endl;
   return 0;
}
```

Output:


"C:\Program Files\Microsoft Visual Studio\MyProjects\testin...

```
str1 string is: Testing the resize()
The size of str1 string is: 20
The capacity of str1 string is: 31

Operation: str1.resize(str1.size() + 3, '?')
The resized str1 string is: Testing the resize()???
The size of resized str1 string is: 23
The capacity of resized str1 string is: 31

Operation: str1.resize(str1.size() + 10)
The resized str1 string is: Testing the resize()???
The increased size of str1 string is: 33
The increased capacity of str1 string is: 63

Operation: str1.resize(str1.size() - 20)
The downsized str1 string is: Testing the r
The size of downsized str1 string is: 13
The capacity of downsized str1 string is: 63
Press any key to continue
```

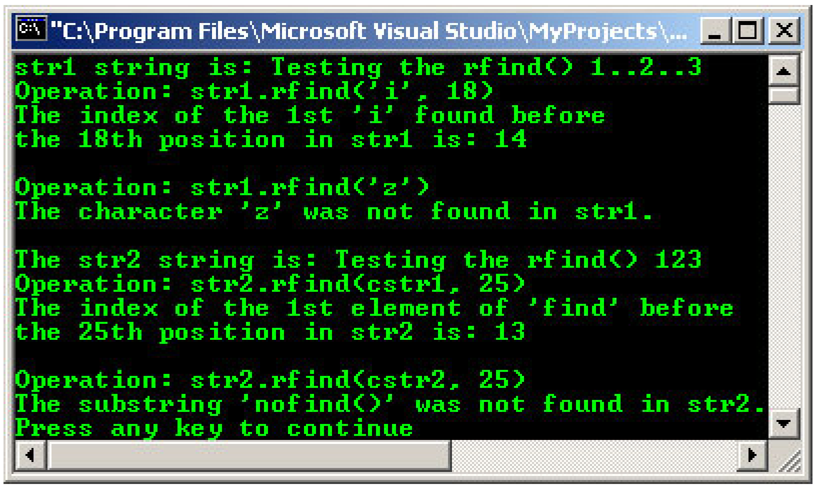
rfind()

- The return value is the index of the last occurrence when searched backwards, of the first character of the substring when successful; otherwise `npos`.

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

int main()
{
   //searching for a single character in a string
   string str1("Testing the rfind() 1..2..3");
   cout<<"str1 string is: "<<str1<<endl;
   basic_string <char>::size_type index1, index2;
   static const basic_string <char>::size_type npos = -1;

   cout<<"Operation: str1.rfind('i', 18)"<<endl;
   index1 = str1.rfind('i', 18);
   if(index1 != npos)
      cout<<"The index of the 1st 'i' found before\nthe 18th"
          <<" position in str1 is: "<<index1<<endl;
   else
      cout<<"The character 'i' was not found in str1."<<endl;

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

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

   const char *cstr1 = "find";
   cout<<"Operation: str2.rfind(cstr1, 25)"<<endl;
   index3 = str2.rfind(cstr1, 25);
   if(index3 != npos)
      cout<<"The index of the 1st element of 'find' "
          <<"before\nthe 25th position in str2 is: "<<index3<<endl;
   else
      cout<<"The substring 'find' was not found in str2."<<endl;

   const char *cstr2 = "nofind()";
   cout<<"\nOperation: str2.rfind(cstr2, 25)"<<endl;
   index4 = str2.rfind(cstr2, 25);
   if(index4 != npos)
      cout<<"The index of the 1st element of 'nofind()' "
```

```
            <<"before\n the 25th position in str3 is: "<<index4<<endl;
      else
         cout<<"The substring 'nofind()' was not found in str2."<<endl;
      return 0;
   }
```

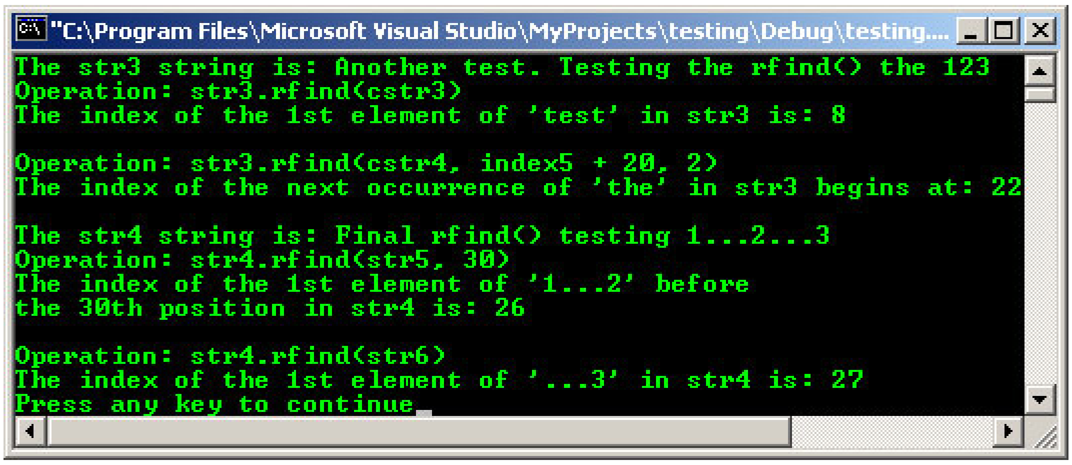
Output:



```
str1 string is: Testing the rfind() 1..2..3
Operation: str1.rfind('i', 18)
The index of the 1st 'i' found before
the 18th position in str1 is: 14

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

The str2 string is: Testing the rfind() 123
Operation: str2.rfind(cstr1, 25)
The index of the 1st element of 'find' before
the 25th position in str2 is: 13

Operation: str2.rfind(cstr2, 25)
The substring 'nofind()' was not found in str2.
Press any key to continue
```

```
//reversed find, rfind() 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("Another test. Testing the rfind() the 123");
   cout<<"The str3 string is: "<<str3<<endl;
   static const basic_string <char>::size_type npos = -1;
   basic_string <char>::size_type index5, index6;

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

   const char *cstr4 = "the";
   cout<<"\nOperation: str3.rfind(cstr4, index5 + 20, 2)"<<endl;
   index6 = str3.rfind(cstr4, index5 + 20, 2);
   if(index6 != npos)
      cout<<"The index of the next occurrence of 'the' in str3 begins at:"
<<index6<<endl;
   else
      cout<<"There is no next occurrence of 'the' in str3"<<endl;
   cout<<endl;

   //------------------------------------------------------------
   //searching string for a substring as specified by a string
   string str4("Final rfind() testing 1...2...3");
   cout<<"The str4 string is: "<<str4<<endl;
   basic_string <char>::size_type index7, index8;

   string str5("2...3");
   cout<<"Operation: str4.rfind(str5, 30)"<<endl;
   index7 = str4.rfind(str5, 30);
   if(index7 != npos)
      cout<<"The index of the 1st element of '1...2' "
          <<"before\nthe 30th position in str4 is: "<<index7<<endl;
   else
      cout<<"The substring '1...2' was not found in str4\n"
          <<"before the 30th position."<<endl;

   string str6("...3");
   cout<<"\nOperation: str4.rfind(str6)"<<endl;
   index8 = str4.rfind(str6);
   if(index8 != npos)
      cout<<"The index of the 1st element of '...3' in str4 is: "<<index8<<endl;
   else
      cout<<"The substring '...3' was not found in str4."<<endl;
   return 0;
}
```

Output:



"C:\Program Files\Microsoft Visual Studio\MyProjects\testing\Debug\testing....

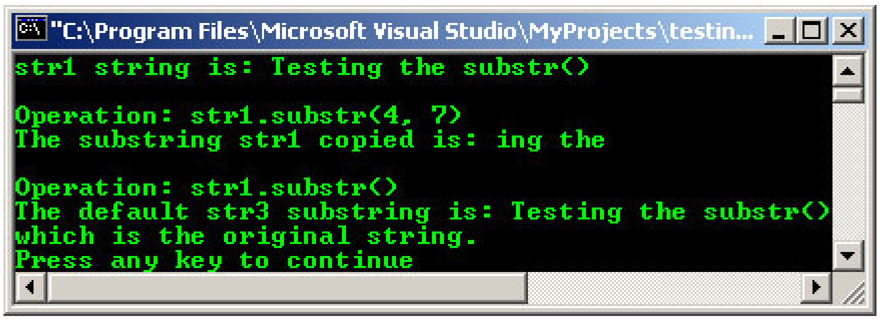
```
The str3 string is: Another test. Testing the rfind() the 123
Operation: str3.rfind(cstr3)
The index of the 1st element of 'test' in str3 is: 8

Operation: str3.rfind(cstr4, index5 + 20, 2)
The index of the next occurrence of 'the' in str3 begins at: 22

The str4 string is: Final rfind() testing 1...2...3
Operation: str4.rfind(str5, 30)
The index of the 1st element of '1...2' before
the 30th position in str4 is: 26

Operation: str4.rfind(str6)
The index of the 1st element of '...3' in str4 is: 27
Press any key to continue
```

substr()

- The return value is a substring object that is a copy of elements of the string operand beginning at the position specified by the first argument.

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

int main()
{

   string  str1("Testing the substr()");
   cout<<"str1 string is: "<<str1<<endl;

   cout<<"\nOperation: str1.substr(4, 7)"<<endl;
   basic_string <char> str2 = str1.substr(4, 7);
   cout<<"The substring str1 copied is: "<<str2<<endl;

   cout<<"\nOperation: str1.substr()"<<endl;
   basic_string <char> str3 = str1.substr();
   cout<<"The default str3 substring is: "<<str3
        <<"\nwhich is the original string."<<endl;
   return 0;
}
```

Output:



"C:\Program Files\Microsoft Visual Studio\MyProjects\testin...

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

Operation: str1.substr(4, 7)
The substring str1 copied is: ing the

Operation: str1.substr()
The default str3 substring is: Testing the substr()
which is the original string.
Press any key to continue
```

26.2 char_traits Class

- The `char_traits` class describes **attributes associated with a character**. The template class structure for `char_traits` is shown below.

```
template <class CharType> struct char_traits;
```

- Where:

  `CharType` is the type of the data element.

- The template class describes various character traits for type `CharType`. The template class `basic_string` as well as several `iostream` template classes, including `basic_ios`, use this information to manipulate elements of type `CharType`.
- Such an element type must not require explicit construction or destruction. It must supply a default constructor, a copy constructor, and an assignment operator, with the expected semantics.
- A bitwise copy must have the same effect as an assignment. None of the member functions of class `char_traits` can throw exceptions.

26.3 char_traits Class Typedefs

- The following table is a list of the `char_traits` class template `typedef`, the synonym name.

Typedef	Brief Description
`char_type`	A type of character.
`int_type`	An integer type that can represent a character of type **char_type** or an end-of-file (EOF) character.
`off_type`	An integer type that can represent offsets between positions in a stream.
`pos_type`	An integer type that can represent positions in a stream.
`state_type`	A type that represents the conversion state in for multibyte characters in a stream.

Table 26.1: `char_traits` typedef

26.4 char_traits Class Member Functions

- The following table is a list of the `char_traits` class template member function.

Member Function	Brief Description
`assign()`	Assigns one character value to another.
`compare()`	Compares up to a specified number of characters in two strings.
`copy()`	Copies a specified number of characters from one string to another.
`eof()`	Returns the end-of-file (EOF) character.
`eq()`	Tests whether two **char_type** characters are equal.
`eq_int_type()`	Tests whether two characters represented as **int_type**s are equal.
`find()`	Searches for the first occurrence of a specified character in a range of characters.
`length()`	Returns the length of a string.
`lt()`	Tests whether one character is less than another.
`move()`	Copies a specified number of characters in a sequence to another, possible overlapping sequence.
`not_eof()`	Tests whether a character is the end-of-file (EOF) character.
`to_char_type()`	Converts an **int_type** character to the corresponding **char_type** character and returns the result.
`to_int_type()`	Converts a **char_type** character to the corresponding **int_type** character and returns the result.

Table 26.2: `char_traits` class template member functions

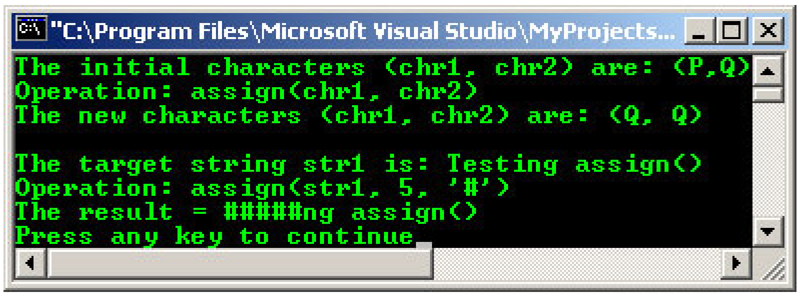
26.5 char_traits Member Function Program Examples

Note:
The following program example may generate a runtime error regarding the buffer overflow, because there are no explicit exception handling code used in the program, the exceptions should be ‘fully’ handled by the compiler. Good compiler should warn us regarding the exceptions. If the problem persists, try changing some of the pointer variables to arrays variables as shown in **`move()`** program example. It should be OK.

assign()

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

int main()
{
   //assigning a character value to another character
   char chr1 = 'P';
   const char chr2 = 'Q';

   cout<<"The initial characters (chr1, chr2) are: ("<<chr1<<","<<chr2<<")"<<endl;
   cout<<"Operation: assign(chr1, chr2)"<<endl;
   char_traits<char>::assign(chr1, chr2);
   cout<<"The new characters (chr1, chr2) are: ("<<chr1<< ", "<<chr2<<")"<<endl;


   //assigning character values to initial part of a string
   char_traits<char>::char_type* str1 = "Testing assign()";
   char_traits<char>::char_type* result;

   cout<<"\nThe target string str1 is: "<<str1<<endl;
   cout<<"Operation: assign(str1, 5, \'#\')"<<endl;
   result = char_traits<char>::assign(str1, 5, '#');
   cout<<"The result = "<<result<<endl;
   return 0;
}
```

Output:



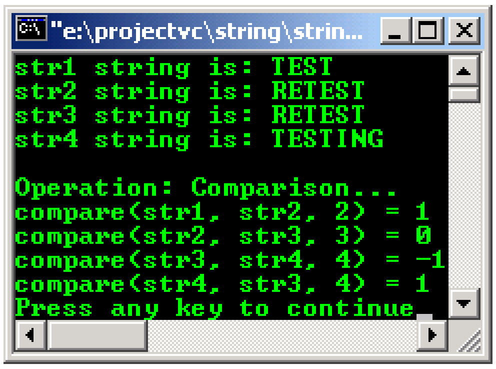
```
The initial characters (chr1, chr2) are: (P,Q)
Operation: assign(chr1, chr2)
The new characters (chr1, chr2) are: (Q, Q)

The target string str1 is: Testing assign()
Operation: assign(str1, 5, '#')
The result = #####ng assign()
Press any key to continue
```

compare()

- The comparison between the strings is made element by element, first testing for equality and then, if a pair of elements in the sequence tests not equal, they are tested for less than.
- If two strings compare equal over a range but one is longer than the other, then the shorter of the two is less than the longer one.
- The return value is a negative value if the first string is less than the second string, 0 if the two strings are equal, or a positive value if the first string is greater than the second string.

```
//char_traits, compare()
#include <string>
#include <iostream>
using namespace std;

int main()
{
   char_traits<char>::char_type* str1 = "TEST";
   char_traits<char>::char_type* str2 = "RETEST";
   char_traits<char>::char_type* str3 = "RETEST";
   char_traits<char>::char_type* str4 = "TESTING";

   cout<<"str1 string is: "<<str1<<endl;
   cout<<"str2 string is: "<<str2<<endl;
   cout<<"str3 string is: "<<str3<<endl;
   cout<<"str4 string is: "<<str4<<endl;

   cout<<"\nOperation: Comparison..."<<endl;
   int comp1, comp2, comp3, comp4;
   comp1 = char_traits<char>::compare(str1, str2, 2);
   comp2 = char_traits<char>::compare(str2, str3, 3);
   comp3 = char_traits<char>::compare(str3, str4, 4);
   comp4 = char_traits<char>::compare(str4, str3, 4);
   cout<<"compare(str1, str2, 2) = "<<comp1<<endl;
   cout<<"compare(str2, str3, 3) = "<<comp2<<endl;
   cout<<"compare(str3, str4, 4) = "<<comp3<<endl;
   cout<<"compare(str4, str3, 4) = "<<comp4<<endl;
   return 0;
}
```

Output:



```
str1 string is: TEST
str2 string is: RETEST
str3 string is: RETEST
str4 string is: TESTING

Operation: Comparison...
compare(str1, str2, 2) = 1
compare(str2, str3, 3) = 0
compare(str3, str4, 4) = -1
compare(str4, str3, 4) = 1
Press any key to continue
```

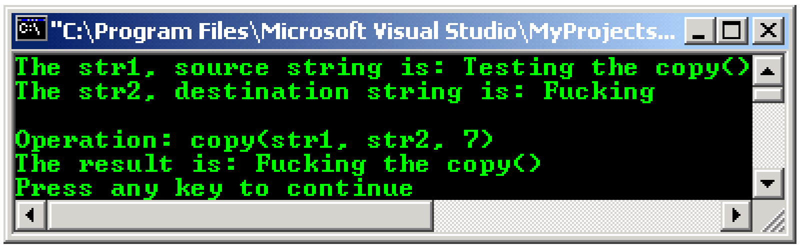
Note:
The following program example may generate a runtime error regarding the buffer overflow, because there are no explicit exception handling code use in the program, the exceptions should be 'fully' handled by the compiler. Good compiler should warn us regarding the exceptions. If the problem persists, try changing some of the pointer variables to arrays variables as shown in **move()** program example. It should be OK.

copy()

- The source and destination character sequences must not overlap. Compare with the `move()` member function.

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

int main()
{
   char_traits<char>::char_type* str1 = "Testing the copy()";
   char_traits<char>::char_type* str2 = "Fucking";
   char_traits<char>::char_type* result;

   cout<<"The str1, source string is: "<<str1<<endl;
   cout<<"The str2, destination string is: "<<str2<<endl;
   cout<<"\nOperation: copy(str1, str2, 7)"<<endl;
   result = char_traits<char>::copy(str1, str2, 7);
   cout<<"The result is: "<<result<<endl;
   return 0;
}
```

Output:



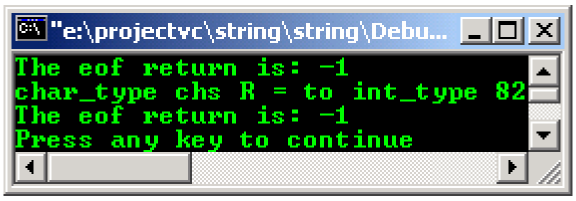
```
The str1, source string is: Testing the copy()
The str2, destination string is: Fucking

Operation: copy(str1, str2, 7)
The result is: Fucking the copy()
Press any key to continue
```

eof()

- The return value is a value that represents end of file character (such as `EOF` or `WEOF`).
- If the value is represented as type `char_type`, it must correspond to no valid value of that type.

```
//char_traits, eof()
#include <string>
#include <iostream>
using namespace std;

int main( )
{
   char_traits <char>::int_type int0 = char_traits<char>::eof();
   cout<<"The eof return is: "<<int0<<endl;

   char_traits<char>::char_type chs = 'R';
   char_traits<char>::int_type int1;
   int1 =char_traits<char>::to_int_type(chs);
   cout<<"char_type chs "<<chs<<" = to int_type "<<int1<<endl;

   char_traits <char>::int_type int2 = char_traits<char>::eof();
   cout<<"The eof return is: "<<int2<<endl;
   return 0;
}
```

Output:



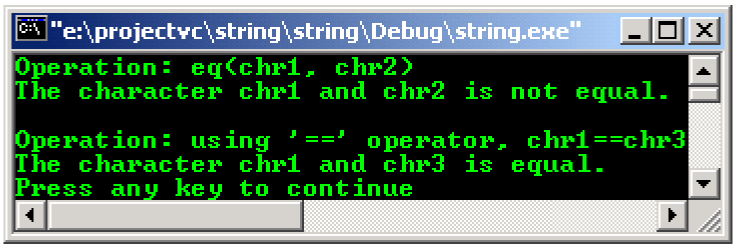
```
The eof return is: -1
char_type chs R = to int_type 82
The eof return is: -1
Press any key to continue
```

eq()

- The return value is **true** if the first character is equal to the second character; otherwise **false**.

```
//char_traits, eq()
#include <string>
#include <iostream>
using namespace std;

int main()
{
   char_traits<char>::char_type chr1 = 'P';
   char_traits<char>::char_type chr2 = 'Q';
   char_traits<char>::char_type chr3 = 'P';

   //Testing for equality
   bool Var1 = char_traits<char>::eq(chr1, chr2);
   cout<<"Operation: eq(chr1, chr2)"<<endl;
   if(Var1)
      cout<<"The character chr1 and chr2 is equal."<<endl;
   else
      cout<<"The character chr1 and chr2 is not equal."<<endl;

   //alternatively...
   cout<<"\nOperation: using \'==\' operator, chr1==chr3"<<endl;
   if(chr1 == chr3)
      cout<<"The character chr1 and chr3 is equal."<<endl;
   else
      cout<<"The character chr1 and chr3 is not equal."<<endl;
   return 0;
}
```

Output:



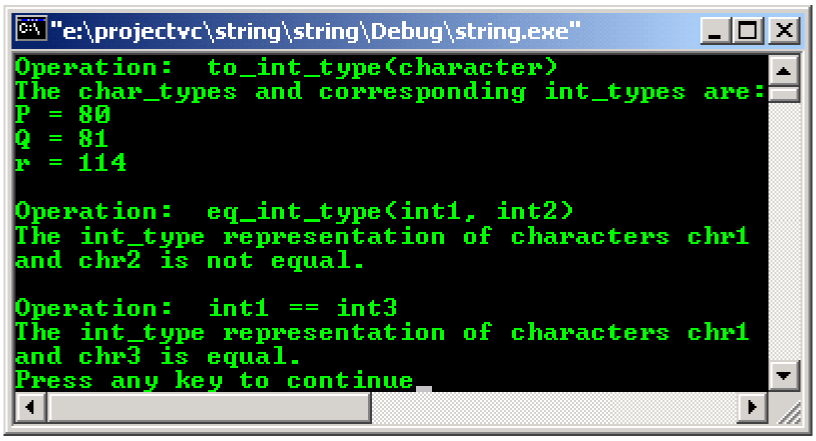
```
"e:\projectvc\string\string\Debug\string.exe"
Operation: eq(chr1, chr2)
The character chr1 and chr2 is not equal.

Operation: using '==' operator, chr1==chr3
The character chr1 and chr3 is equal.
Press any key to continue
```

eq_int_type()

- The return value is **true** if the first character is equal to the second character; otherwise **false**.

```
//char_traits, eq_int_type()
//and to_int_type()
#include <string>
#include <iostream>
using namespace std;

int main()
{
   char_traits<char>::char_type chr1 = 'P';
   char_traits<char>::char_type chr2 = 'Q';
   char_traits<char>::char_type chr3 = 'P';
   char_traits<char>::char_type chr4 = 'r';

   //char_type to int_type conversion
   char_traits<char>::int_type int1, int2, int3, int4;
   int1 = char_traits<char>::to_int_type(chr1);
   int2 = char_traits<char>::to_int_type(chr2);
   int3 = char_traits<char>::to_int_type(chr3);
   int4 = char_traits<char>::to_int_type(chr4);

   cout<<"Operation:  to_int_type(character)"<<endl;
   cout<<"The char_types and corresponding int_types are:\n";
   cout<<chr1<<" = "<<int1<<endl;
   cout<<chr2<<" = "<<int2<<endl;
   cout<<chr4<<" = "<<int4<<endl;

   //equality of int_type representations test
   cout<<"\nOperation:  eq_int_type(int1, int2)"<<endl;
   bool var1 = char_traits<char>::eq_int_type(int1, int2);
   if(var1)
      cout<<"The int_type representation of characters chr1\n"
          <<"and chr2 is equal."<<endl;
   else
     cout<<"The int_type representation of characters chr1\n"
          <<"and chr2 is not equal."<<endl;

   //alternatively...
   cout<<"\nOperation:  int1 == int3"<<endl;
   if(int1 == int3)
      cout<<"The int_type representation of characters chr1\n"
          <<"and chr3 is equal."<<endl;
   else
      cout<<"The int_type representation of characters chr1\n"
          <<"and chr3 is not equal."<<endl;
   return 0;
}
```

Output:

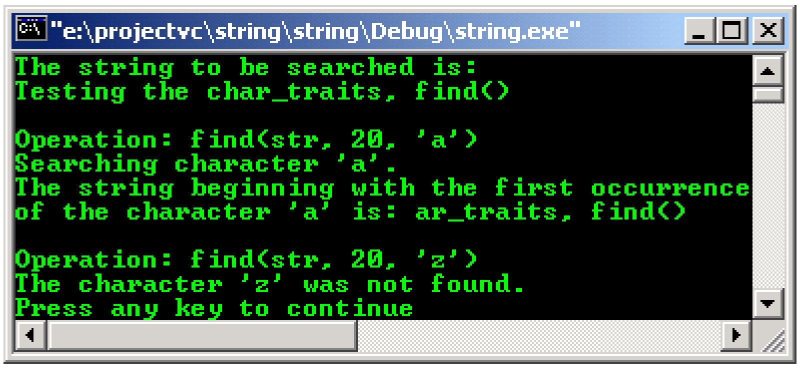


```
Operation:  to_int_type(character)
The char_types and corresponding int_types are:
P = 80
Q = 81
r = 114

Operation:  eq_int_type(int1, int2)
The int_type representation of characters chr1
and chr2 is not equal.

Operation:  int1 == int3
The int_type representation of characters chr1
and chr3 is equal.
Press any key to continue_
```

find()

```
//char_traits, find()
#include <string>
#include <iostream>
using namespace std;

int main( )
{
   const char* str = "Testing the char_traits, find()";
   const char* result1;
   cout<<"The string to be searched is:\n"<<str<<endl;

   //Searching for a 'a' in the first 20 positions of string str
   cout<<"\nOperation: find(str, 20, 'a')"<<endl;
   result1 = char_traits<char>::find(str, 20, 'a');
   cout<<"Searching character \'"<<*result1<<"\'."<<endl;
   cout<<"The string beginning with the first occurrence\n"
        <<"of the character 'a' is: "<<result1<<endl;

   //When no match is found the NULL value is returned
   const char* result2;
   result2 = char_traits<char>::find(str, 20, 'z');
   cout<<"\nOperation: find(str, 20, 'z')"<<endl;
   if(result2 == NULL)
      cout<<"The character 'z' was not found."<<endl;
   else
      cout<<"The result of the search is: "<<result2<<endl;
   return 0;
}
```

Output:



```
The string to be searched is:
Testing the char_traits, find()

Operation: find(str, 20, 'a')
Searching character 'a'.
The string beginning with the first occurrence
of the character 'a' is: ar_traits, find()

Operation: find(str, 20, 'z')
The character 'z' was not found.
Press any key to continue
```

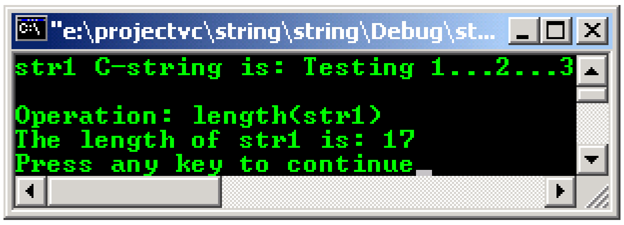
length()

- The return value is the number of elements in the sequence being measured, not including the null terminator.

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

```
int main()
{
   const char* str1= "Testing 1...2...3";
   cout<<"str1 C-string is: "<<str1<<endl;

   size_t LenStr1;
   cout<<"\nOperation: length(str1)"<<endl;
   LenStr1 = char_traits<char>::length(str1);
   cout<<"The length of str1 is: "<<unsigned int(LenStr1)<<endl;
   return 0;
}
```

Output:



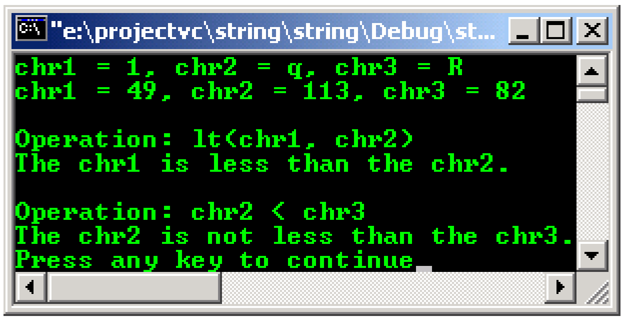
```
str1 C-string is: Testing 1...2...3

Operation: length(str1)
The length of str1 is: 17
Press any key to continue
```

lt()

- The return value is **true** if the first character is less than the second character; otherwise **false**.

```
//char_traits, lt()
#include <string>
#include <iostream>
using namespace std;

int main()
{
   char_traits<char>::char_type chr1 = '1';
   char_traits<char>::char_type chr2 = 'q';
   char_traits<char>::char_type chr3 = 'R';

   char_traits<char>::int_type int1, int2, int3;
   int1 = char_traits<char>::to_int_type(chr1);
   int2 = char_traits<char>::to_int_type(chr2);
   int3 = char_traits<char>::to_int_type(chr3);

   //char_type to int_type conversion, for testing
   cout<<"chr1 = "<<chr1<<", chr2 = "<<chr2<<", chr3 = "<<chr3<<endl;
   cout<<"chr1 = "<<int1<<", chr2 = "<<int2<<", chr3 = "<<int3<<endl;

   //Testing for less than
   cout<<"\nOperation: lt(chr1, chr2)"<<endl;
   bool var1 = char_traits<char>::lt(chr1, chr2);
   if(var1)
      cout<<"The chr1 is less than "
          <<"the chr2."<<endl;
   else
      cout<<"The chr1 is not less "
          <<"than the chr2."<<endl;
   //alternatively...
   cout<<"\nOperation: chr2 < chr3"<<endl;
   if(chr2 < chr3)
      cout<<"The chr2 is less than "
          <<"the chr3."<<endl;
   else
      cout<<"The chr2 is not less "
          <<"than the chr3."<<endl;
   return 0;
}
```

Output:



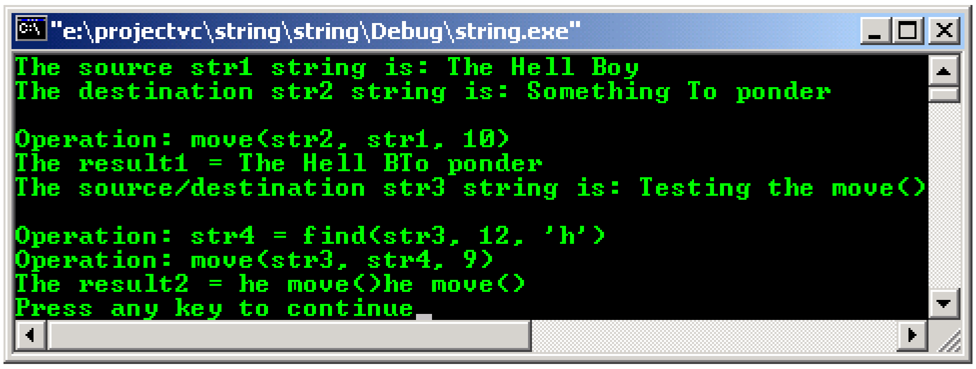
```
chr1 = 1, chr2 = q, chr3 = R
chr1 = 49, chr2 = 113, chr3 = 82

Operation: lt(chr1, chr2)
The chr1 is less than the chr2.

Operation: chr2 < chr3
The chr2 is not less than the chr3.
Press any key to continue
```

move()

- The source and destination may overlap. Compare with `copy()`.
- The return value is the first element is copied into the string or character array targeted to receive the copied sequence of characters.

```
//char_traits, move(), find()
#include <string>
#include <iostream>
using namespace std;

int main()
{
   char_traits<char>::char_type str1[25] = "The Hell Boy";
   char_traits<char>::char_type str2[25] = "Something To ponder";
   char_traits<char>::char_type *result1;

   cout<<"The source str1 string is: "<<str1<<endl;
   cout<<"The destination str2 string is: "<<str2<<endl;
   result1 = char_traits<char>::move(str2, str1, 10);
   cout<<"\nOperation: move(str2, str1, 10)"<<endl;
   cout<<"The result1 = "<<result1<<endl;

   //When source and destination overlap
   char_traits<char>::char_type str3[30] = "Testing the move()";
   char_traits<char>::char_type *result2;
   cout << "The source/destination str3 string is: "<<str3<<endl;

   cout<<"\nOperation: str4 = find(str3, 12, 'h')"<<endl;
   const char *str4 = char_traits<char>::find(str3, 12, 'h');
   cout<<"Operation: move(str3, str4, 9)"<<endl;
   result2 = char_traits<char>::move(str3, str4, 9);
   cout<<"The result2 = "<<result2<<endl;
   return 0;
}
```

Output:



```
The source str1 string is: The Hell Boy
The destination str2 string is: Something To ponder

Operation: move(str2, str1, 10)
The result1 = The Hell BTo ponder
The source/destination str3 string is: Testing the move()

Operation: str4 = find(str3, 12, 'h')
Operation: move(str3, str4, 9)
The result2 = he move()he move()
Press any key to continue
```

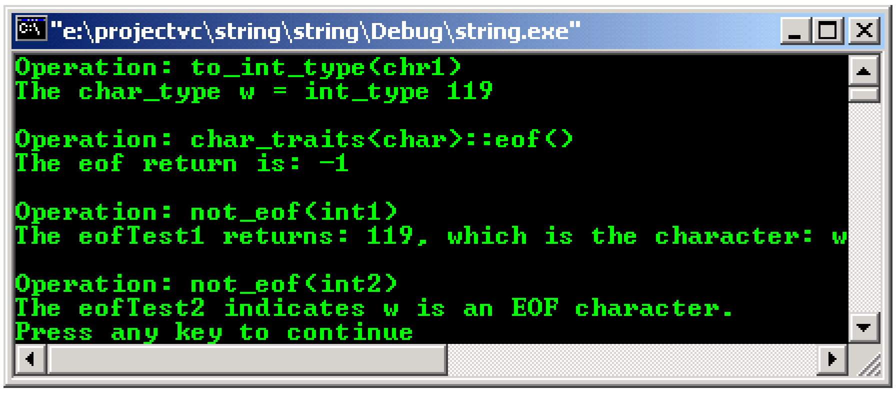
not_eof()

- The return value is the `int_type` representation of the character tested, if the `int_type` of the character is not equal to that of the EOF character.
- If the character `int_type` value is equal to the EOF `int_type` value, then it is **false**.

```
//char_traits, not_eof()
```

```
#include <string>
#include <iostream>
using namespace std;

int main()
{
   char_traits<char>::char_type chr1 = 'w';
   char_traits<char>::int_type int1;
   int1 = char_traits<char>::to_int_type(chr1);
   cout<<"Operation: to_int_type(chr1)"<<endl;
   cout<<"The char_type "<<chr1<<" = int_type "<<int1<<endl;

   //EOF
   char_traits <char>::int_type int2 = char_traits<char>::eof();
   cout<<"\nOperation: char_traits<char>::eof()"<<endl;
   cout<<"The eof return is: "<<int2<<endl;

   //Testing for EOF
   char_traits <char>::int_type eofTest1, eofTest2;
   eofTest1 = char_traits<char>::not_eof(int1);
   cout<<"\nOperation: not_eof(int1)"<<endl;
   if(!eofTest1)
      cout<<"The eofTest1 indicates "<<chr1<<" is an EOF character."<<endl;
   else
      cout<<"The eofTest1 returns: "<<eofTest1
         <<", which is the character: "
          <<char_traits<char>::to_char_type(eofTest1)<<endl;


   eofTest2 = char_traits<char>::not_eof(int2);
   cout<<"\nOperation: not_eof(int2)"<<endl;
   if(!eofTest2)
      cout<<"The eofTest2 indicates "<<chr1<<" is an EOF character."<<endl;
   else
      cout<<"The eofTest1 returns: "<<eofTest2
           <<", which is the character "
           <<char_traits<char>::to_char_type(eofTest2)<<endl;
   return 0;
}
```

Output:



"e:\projectvc\string\string\Debug\string.exe"

```
Operation: to_int_type(chr1)
The char_type w = int_type 119

Operation: char_traits<char>::eof()
The eof return is: -1

Operation: not_eof(int1)
The eofTest1 returns: 119, which is the character: w

Operation: not_eof(int2)
The eofTest2 indicates w is an EOF character.
Press any key to continue
```

`to_char_type()` and `to_int_type()`

- The conversion operations `to_int_type` and `to_char_type` are inverse operation to each other. For example:

   `to_int_type(to_char_type(x)) == x`

- And for any `int_type` **x**:

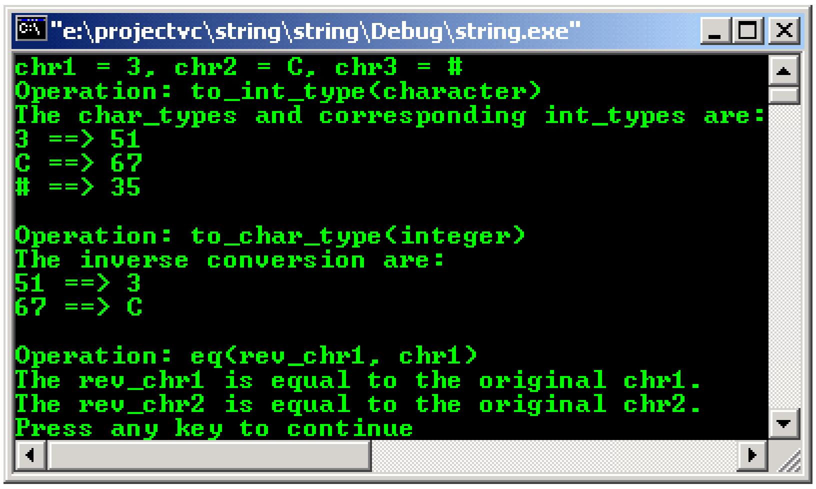
   `to_char_type(to_int_type(x)) == x` for any `char_type` **x**.

- The return value is the `char_type` character corresponding to the `int_type` character.
- A value that cannot be represented by the conversion will yield an unspecified result.

```
//char_traits, to_char_type(),
//to_int_type and eq()
```

```
#include <string>
#include <iostream>
using namespace std;

int main()
{
   char_traits<char>::char_type chr1 = '3';
   char_traits<char>::char_type chr2 = 'C';
   char_traits<char>::char_type chr3 = '#';

   cout<<"chr1 = "<<chr1<<", chr2 = "<<chr2<<", chr3 = "<<chr3<<endl;
   //Converting from char_type to int_type
   char_traits<char>::int_type int1, int2, int3;
   int1 =char_traits<char>::to_int_type(chr1);
   int2 =char_traits<char>::to_int_type(chr2);
   int3 =char_traits<char>::to_int_type(chr3);

   cout<<"Operation: to_int_type(character)"<<endl;
   cout<<"The char_types and corresponding int_types are:\n";
   cout<<chr1<<" ==> "<<int1<<endl;
   cout<<chr2<<" ==> "<<int2<<endl;
   cout<<chr3<<" ==> "<<int3<<endl;

   //int_type to char_type re conversion
   char_traits<char>::char_type rev_chr1;
   rev_chr1 = char_traits<char>::to_char_type(int1);
   char_traits<char>::char_type rev_chr2;
   rev_chr2 = char_traits<char>::to_char_type(int2);

   cout<<"\nOperation: to_char_type(integer)"<<endl;
   cout<<"The inverse conversion are:\n";
   cout<<int1<<" ==> "<<rev_chr1<<endl;
   cout<<int2<<" ==> "<<rev_chr2<<endl;

   //test for conversions, they are just inverse operations
   cout<<"\nOperation: eq(rev_chr1, chr1)"<<endl;
   bool var1 = char_traits<char>::eq(rev_chr1, chr1);
   if(var1)
      cout<<"The rev_chr1 is equal to the original chr1."<<endl;
   else
     cout<<"The rev_chr1 is not equal to the original chr1."<<endl;

   //alternatively...
   if(rev_chr2 == chr2)
     cout<<"The rev_chr2 is equal to the original chr2."<<endl;
   else
     cout<<"The rev_chr2 is not equal to the original chr2."<<endl;
   return 0;
}
```

Output:



"e:\projectvc\string\string\Debug\string.exe"

```
chr1 = 3, chr2 = C, chr3 = #
Operation: to_int_type(character)
The char_types and corresponding int_types are:
3 ==> 51
C ==> 67
# ==> 35

Operation: to_char_type(integer)
The inverse conversion are:
51 ==> 3
67 ==> C

Operation: eq(rev_chr1, chr1)
The rev_chr1 is equal to the original chr1.
The rev_chr2 is equal to the original chr2.
Press any key to continue
```

26.6 `char_traits` Specializations

- The following table is a list of the `char_traits` class template specialization.

Class Specialization	Brief Description

`char_traits<char>` class	A class that is a specialization of the template class `char_traits<CharType>` to an element of type `char`.
`char_traits<wchar_t>` class	A class that is a specialization of the template class `char_traits<CharType>` to an element of type `wchar_t`.

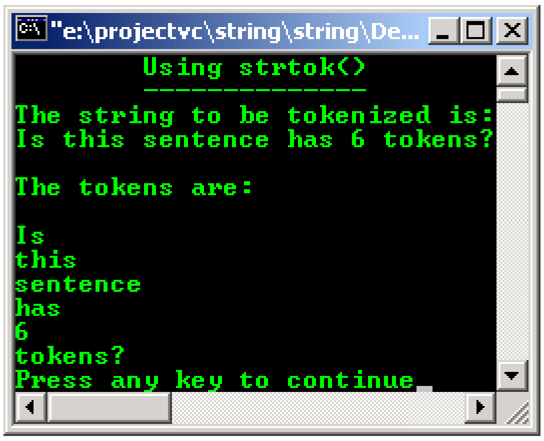
Table 26.3: `char_traits` class template specialization

26.7 Using C++ wrapper for C string and character manipulation

- The following are recompiling and re running of the C characters and strings program examples. Program examples are taken from Module X. You can try other program examples as well.

```
//strtok()
//using the C++ wrappers
#include <cstdio>
#include <string>
using namespace std;

int main()
{
       char string[] = "Is this sentence has 6 tokens?";
       char *tokenPtr;

       printf("      Using strtok()\n");
       printf("      --------------\n");

       printf("The string to be tokenized is:\n%s\n", string);
       printf("\nThe tokens are: \n\n");

       tokenPtr = strtok(string, " ");
       while (tokenPtr != NULL)
              {
                     printf("%s\n", tokenPtr);
                     tokenPtr  =  strtok(NULL, " ");
              }
       return   0;
}
```

Output:

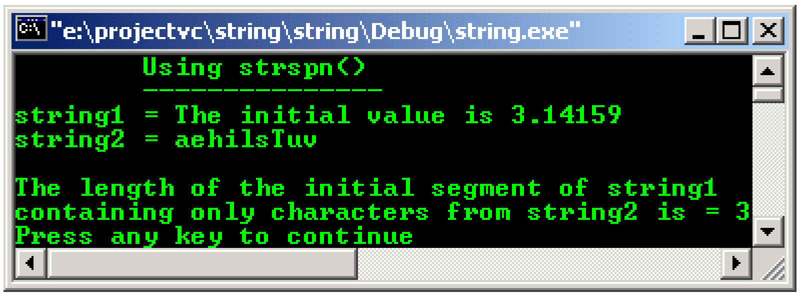


```
          Using strtok()
          --------------
The string to be tokenized is:
Is this sentence has 6 tokens?

The tokens are:

Is
this
sentence
has
6
tokens?
Press any key to continue
```

```
//Using strspn()
#include <cstdio>
#include <string>
using namespace std;

int main()
{
       char *string1 = "The initial value is 3.14159";
       char *string2 = "aehilsTuv";


       printf("      Using strspn()\n");
       printf("      ---------------\n");

       printf("string1 = %s\n", string1);
```

```
        printf("string2 = %s\n", string2);
        printf("\nThe length of the initial segment of string1\n");
        printf("containing only characters from string2 is = %u\n", strspn(string1,
string2));
        return 0;
}
```

Output:



```
        Using strspn()
        ---------------
string1 = The initial value is 3.14159
string2 = aehilsTuv

The length of the initial segment of string1
containing only characters from string2 is = 3
Press any key to continue
```

- Program example compiled using **g++**. Portability is not an issue here :o). g++ warmly warn you for constructs that are obsolete!

```
//*********string2.cpp************
//insert() part I
#include <string>
#include <iostream>
using namespace std;

int main()
{
//inserting a C-string at a given position
basic_string <char> str1("e insert() testing");
const char *cstr1 = "Th";

cout<<"str1 = "<<str1<<endl;
cout<<"cstr1 = "<<cstr1<<endl;
str1.insert(0, cstr1);
cout<<"Operation: str1.insert(0, cstr1)"<<endl;
cout<<"Inserting a C-string at position 0 is:\n"<<str1<<endl;
cout<<endl;

//inserting a C-string at a given position for a specified number of elements
basic_string <char> str2("Test");
const char *cstr2 = "ing an insert()";

cout<<"str2 = "<<str2<<endl;
cout<<"cstr2 = "<<cstr2<<endl;
str2.insert(4, cstr2, 15);
cout<<"Operation: str2.insert(4, cstr2, 15)"<<endl;
cout<<"Inserting a C-string at the end is:\n"<<str2<<endl;
cout<<endl;

//inserting a string at a given position
basic_string <char> str3(" the insert()");
string str4("Testing");

cout<<"str3 = "<<str3<<endl;
cout<<"str4 = "<<str4<<endl;
str3.insert(0, str4);
cout<<"Operation: str3.insert(0, str4)"<<endl;
cout<<"Inserting string at position 0 is:\n"<<str3<<endl;
cout<<endl;

//inserting part of a string at a given position
basic_string <char> str5("Testing ");
string str6(" the insert()");

cout<<"str5 = "<<str5<<endl;
cout<<"str6 = "<<str6<<endl;
str5.insert(7, str6, 4, 9);
cout<<"Operation: str5.insert(7, str6, 4, 9)"<<endl;
cout<<"Inserting part of a string at position 9 is:\n"<<str5<<endl;
return 0;
}
```

```
[bodo@bakawali ~]$ g++ string2.cpp -o string2
[bodo@bakawali ~]$ ./string2

str1 = e insert() testing
cstr1 = Th
Operation: str1.insert(0, cstr1)
Inserting a C-string at position 0 is:
The insert() testing

str2 = Test
cstr2 = ing an insert()
Operation: str2.insert(4, cstr2, 15)
Inserting a C-string at the end is:
Testing an insert()

str3 =  the insert()
str4 = Testing
Operation: str3.insert(0, str4)
Inserting string at position 0 is:
Testing the insert()

str5 = Testing
str6 =  the insert()
Operation: str5.insert(7, str6, 4, 9)
Inserting part of a string at position 9 is:
Testing insert()
```

---------------------------------o0o---------------------------------------

---www.tenouk.com---

Further reading and digging:

1. Check the best selling C / C++ and STL books at Amazon.com.