Illustrations List ***(Main Page)***

Fig. 19.1 Demonstrating **`string`** assignment and concatenation.
Fig. 19.2 Comparing **`string`**s.
Fig. 19.3 Using function **`substr`** to extract a substring from a **`string`**.
Fig. 19.4 Using function **`swap`** to swap two **`string`**s.
Fig. 19.5 Printing **`string`** characteristics.
Fig. 19.6 Program that demonstrates the **`string find`** functions.
Fig. 19.7 Demonstrating functions **`erase`** and **`replace`**.
Fig. 19.8 Demonstrating the **`string insert`** functions.
Fig. 19.9 Converting **`string`**s to C-style strings and character arrays.
Fig. 19.10 Using an iterator to output a **`string`**.
Fig. 19.11 Using a dynamically allocated **`ostringstream`** object.
Fig. 19.12 Demonstrating input from an **`istringstream`** object.

```
1  // Fig. 19.1: fig19_01.cpp
2  // Demonstrating string assignment and concatenation
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string s1( "cat" ), s2, s3;
10
11    s2 = s1;          // assign s1 to s2 with =
12    s3.assign( s1 );  // assign s1 to s3 with assign()
13    cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: "
14         << s3 << "\n\n";
15
16    // modify s2 and s3
17    s2[ 0 ] = s3[ 2 ] = 'r';
18
19    cout << "After modification of s2 and s3:\n"
20         << "s1: " << s1 << "\ns2: " << s2 << "\ns3: ";
21
22    // demonstrating member function at()
23    int len = s3.length();
24    for ( int x = 0; x < len; ++x )
25       cout << s3.at( x );
26
27    // concatenation
28    string s4( s1 + "apult" ), s5;  // declare s4 and s5
29
30    // overloaded +=
31    s3 += "pet";             // create "carpet"
32    s1.append( "acomb" );    // create "catacomb"
33
34    // append subscript locations 4 thru the end of s1 to
35    // create the string "comb" (s5 was initially empty)
36    s5.append( s1, 4, s1.size() );
37
38    cout << "\n\nAfter concatenation:\n" << "s1: " << s1
39         << "\ns2: " << s2 << "\ns3: " << s3 << "\ns4: " << s4
40         << "\ns5: " << s5 << endl;
41
42    return 0;
43 }
```

Fig. 19.1 Demonstrating **string** assignment and concatenation (part 1 of 2).

```
s1: cat
s2: cat
s3: cat

After modification of s2 and s3:
s1: cat
s2: rat
s3: car

After concatenation:
s1: catacomb
s2: rat
s3: carpet
s4: catapult
s5: comb
```

Fig. 19.1 Demonstrating **string** assignment and concatenation (part 2 of 2).

```
1  // Fig. 19.2: fig19_02.cpp
2  // Demonstrating string comparison capabilities
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string s1( "Testing the comparison functions." ),
10           s2("Hello" ), s3( "stinger" ), z1( s2 );
11
12    cout << "s1: " << s1 << "\ns2: " << s2
13         << "\ns3: " << s3 << "\nz1: " << z1 << "\n\n";
```

Fig. 19.2 Comparing **string**s (part 1 of 3).

```
14
15    // comparing s1 and z1
16    if ( s1 == z1 )
17       cout << "s1 == z1\n";
18    else { // s1 != z1
19       if ( s1 > z1 )
20          cout << "s1 > z1\n";
21       else // s1 < z1
22          cout << "s1 < z1\n";
23    }
24
25    // comparing s1 and s2
26    int f = s1.compare( s2 );
27
28    if ( f == 0)
29       cout << "s1.compare( s2 ) == 0\n";
30    else if ( f > 0 )
31       cout << "s1.compare( s2 ) > 0\n";
32    else  // f < 0
33       cout << "s1.compare( s2 ) < 0\n";
34
35    // comparing s1 (elements 2 - 5) and s3 (elements 0 - 5)
36    f = s1.compare( 2, 5, s3, 0, 5 );
37
38    if ( f == 0 )
```

```
39          cout << "s1.compare( 2, 5, s3, 0, 5 ) == 0\n";
40       else if ( f > 0 )
41          cout << "s1.compare( 2, 5, s3, 0, 5 ) > 0\n";
42       else  // f < 0
43          cout << "s1.compare( 2, 5, s3, 0, 5 ) < 0\n";
44
45       // comparing s2 and z1
46       f = z1.compare( 0, s2.size(), s2 );
47
48       if ( f == 0 )
49          cout << "z1.compare( 0, s2.size(), s2 ) == 0" << endl;
50       else if ( f > 0 )
51          cout << "z1.compare( 0, s2.size(), s2 ) > 0" << endl;
52       else  // f < 0
53          cout << "z1.compare( 0, s2.size(), s2 ) < 0" << endl;
54
55       return 0;
56    }
```

Fig. 19.2 Comparing **string**s (part 2 of 3).

```
s1: Testing the comparison functions.
s2: Hello
s3: stinger
z1: Hello

s1 > z1
s1.compare( s2 ) > 0
s1.compare( 2, 5, s3, 0, 5 ) == 0
z1.compare( 0, s2.size(), s2 ) == 0
```

Fig. 19.2 Comparing **string**s (part 3 of 3).

```
1  // Fig. 19.3: fig19_03.cpp
2  // Demonstrating function substr
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string s( "The airplane flew away." );
10
11    // retrieve the substring "plane" which
12    // begins at subscript 7 and consists of 5 elements
13    cout << s.substr( 7, 5 ) << endl;
14
15    return 0;
16 }
```

```
plane
```

Fig. 19.3 Using function **substr** to extract a substring from a **string**.

```
1  // Fig. 19.4: fig19_04.cpp
2  // Using the swap function to swap two strings
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string first( "one" ), second( "two" );
10
11    cout << "Before swap:\n first: " << first
12         << "\nsecond: " << second;
13    first.swap( second );
14    cout << "\n\nAfter swap:\n first: " << first
15         << "\nsecond: " << second << endl;
16
17    return 0;
18 }
```

```
Before swap:
 first: one
second: two

After swap:
 first: two
second: one
```

Fig. 19.4 Using function **swap** to swap two **string**s.

```
1  // Fig. 19.5: fig19_05.cpp
2  // Demonstrating functions related to size and capacity
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  void printStats( const string & );
```

Fig. 19.5 Printing **string** characteristics (part 1 of 3).

```
8
9  int main()
10 {
11    string s;
12
13    cout << "Stats before input:\n";
14    printStats( s );
15
16    cout << "\n\nEnter a string: ";
17    cin >> s;  // delimited by whitespace
18    cout << "The string entered was: " << s;
19
20    cout << "\nStats after input:\n";
21    printStats( s );
22
23    s.resize( s.length() + 10 );
24    cout << "\n\nStats after resizing by (length + 10):\n";
25    printStats( s );
26
27    cout << endl;
28    return 0;
```

```
}

void printStats( const string &str )
{
   cout << "capacity: " << str.capacity()
        << "\nmax size: " << str.max_size()
        << "\nsize: " << str.size()
        << "\nlength: " << str.length()
        << "\nempty: " << ( str.empty() ? "true": "false" );
}
```

Fig. 19.5 Printing `string` characteristics (part 2 of 3).

```
Stats before input:
capacity: 0
max size: 4294967293
size: 0
length: 0
empty: true

Enter a string: tomato soup
The string entered was: tomato
Stats after input:
capacity: 31
max size: 4294967293
size: 6
length: 6
empty: false

Stats after resizing by (length + 10):
capacity: 31
max size: 4294967293
size: 16
length: 16
empty: false
```

Fig. 19.5 Printing `string` characteristics (part 3 of 3).

```
// Fig. 19.6: fig19_06.cpp
// Demonstrating the string find functions
#include <iostream>
#include <string>
using namespace std;

int main()
{
   // compiler concatenates all parts into one string literal
   string s( "The values in any left subtree"
             "\nare less than the value in the"
             "\nparent node and the values in"
             "\nany right subtree are greater"
             "\nthan the value in the parent node" );
```

Fig. 19.6 Program that demonstrates the `string` `find` functions (part 1 of 2).

```

   // find "subtree" at locations 23 and 102
   cout << "Original string:\n" << s
```

```
         << "\n\n(find) \"subtree\" was found at: "
         << s.find( "subtree" )
         << "\n(rfind) \"subtree\" was found at: "
         << s.rfind( "subtree" );

   // find 'p' in parent at locations 62 and 144
   cout << "\n(find_first_of) character from \"qpxz\" at: "
        << s.find_first_of( "qpxz" )
        << "\n(find_last_of) character from \"qpxz\" at: "
        << s.find_last_of( "qpxz" );

   // find 'b' at location 25
   cout << "\n(find_first_not_of) first character not\n"
        << "   contained in \"heTv lusinodrpayft\": "
        << s.find_first_not_of( "heTv lusinodrpayft" );

   // find '\n' at location 121
   cout << "\n(find_last_not_of) first character not\n"
        << "   contained in \"heTv lusinodrpayft\": "
        << s.find_last_not_of( "heTv lusinodrpayft" ) << endl;

   return 0;
}
```

```
Original string:
The values in any left subtree
are less than the value in the
parent node and the values in
any right subtree are greater
than the value in the parent node

(find) "subtree" was found at: 23
(rfind) "subtree" was found at: 102
(find_first_of) character from "qpxz" at: 62
(find_last_of) character from "qpxz" at: 144
(find_first_not_of) first character not
   contained in "heTv lusinodrpayft": 25
(find_last_not_of) first character not
   contained in "heTv lusinodrpayft": 121
```

Fig. 19.6 Program that demonstrates the `string` `find` functions (part 2 of 2).

```
// Fig. 19.7: fig19_07.cpp
// Demonstrating functions erase and replace
#include <iostream>
#include <string>
using namespace std;

int main()
{
   // compiler concatenates all parts into one string
   string s( "The values in any left subtree"
             "\nare less than the value in the"
             "\nparent node and the values in"
             "\nany right subtree are greater"
             "\nthan the value in the parent node" );

   // remove all characters from location 62
   // through the end of s
   s.erase( 62 );

   // output the new string
   cout << "Original string after erase:\n" << s
```

```
22          << "\n\nAfter first replacement:\n";
23
24       // replace all spaces with a period
25       int x = s.find( " " );
26       while ( x < string::npos ) {
27          s.replace( x, 1, "." );
28          x = s.find( " ", x + 1 );
29       }
30
31       cout << s << "\n\nAfter second replacement:\n";
32
33       // replace all periods with two semicolons
34       // NOTE: this will overwrite characters
35       x = s.find( "." );
36       while ( x < string::npos ) {
37          s.replace( x, 2, "xxxxx;;yyy", 5, 2 );
38          x = s.find( ".", x + 1 );
39       }
40
41       cout << s << endl;
42       return 0;
43    }
```

Fig. 19.7 Demonstrating functions `erase` and `replace` (part 1 of 2).

```
Original string after erase:
The values in any left subtree
are less than the value in the


After first replacement:
The.values.in.any.left.subtree
are.less.than.the.value.in.the


After second replacement:
The;;alues;;n;;ny;;eft;;ubtree
are;;ess;;han;;he;;alue;;n;;he
```

Fig. 19.7 Demonstrating functions `erase` and `replace` (part 2 of 2).

```
1  // Fig. 19.8: fig19_08.cpp
2  // Demonstrating the string insert functions.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string s1( "beginning end" ),
10           s2( "middle " ), s3( "12345678" ), s4( "xx" );
11
12    cout << "Initial strings:\ns1: " << s1
13         << "\ns2: " << s2 << "\ns3: " << s3
14         << "\ns4: " << s4 << "\n\n";
15
16    // insert "middle" at location 10
17    s1.insert( 10, s2 );
18
19    // insert "xx" at location 3 in s3
20    s3.insert( 3, s4, 0, string::npos );
```

```
21
22     cout << "Strings after insert:\ns1: " << s1
23          << "\ns2: " << s2 << "\ns3: " << s3
24          << "\ns4: " << s4 << endl;
25
26     return 0;
27  }
```

```
Initial strings:
s1: beginning end
s2: middle
s3: 12345678
s4: xx

Strings after insert:
s1: beginning middle end
s2: middle
s3: 123xx45678
s4: xx
```

Fig. 19.8 Demonstrating the **string** **insert** functions.

```
1  // Fig. 19.9: fig19_09.cpp
2  // Converting to C-style strings.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string s( "STRINGS" );
10    const char *ptr1 = 0;
11    int len = s.length();
12    char *ptr2 = new char[ len + 1 ]; // including null
13
14    // Assign to pointer ptr1 the const char * returned by
15    // function data(). NOTE: this is a potentially dangerous
16    // assignment. If the string is modified, the pointer
17    // ptr1 can become invalid.
18    ptr1 = s.data();
19
20    // copy characters out of string into allocated memory
21    s.copy( ptr2, len, 0 );
22    ptr2[ len ] = 0;  // add null terminator
23
24    // output
25    cout << "string s is " << s
26         << "\ns converted to a C-Style string is "
27         << s.c_str() << "\nptr1 is ";
28
29    for ( int k = 0; k < len; ++k )
30       cout << *( ptr1 + k );   // use pointer arithmetic
31
32    cout << "\nptr2 is " << ptr2 << endl;
33    delete [] ptr2;
34    return 0;
35 }
```

Fig. 19.9 Converting **string**s to C-style strings and character arrays (part 1 of 2).

```
string s is STRINGS
s converted to a C-Style string is STRINGS
ptr1 is STRINGS
ptr2 is STRINGS
```

Fig. 19.9 Converting **string**s to C-style strings and character arrays (part 2 of 2).

```
1  // Fig. 19.10: fig19_10.cpp
2  // Using an iterator to output a string.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string s( "Testing iterators" );
10    string::const_iterator i1 = s.begin();
11
12    cout << "s = " << s
13         << "\n(Using iterator i1) s is: ";
14
15    while ( i1 != s.end() ) {
16       cout << *i1;     // dereference iterator to get char
17       ++i1;           // advance iterator to next char
18    }
19
20    cout << endl;
21    return 0;
22 }
```

```
s = Testing iterators
(Using iterator i1) s is: Testing iterators
```

Fig. 19.10 Using an iterator to output a **string**.

```
1  // Fig. 19.11: fig19_11.cpp
2  // Using a dynamically allocated ostringstream object.
3  #include <iostream>
4  #include <string>
5  #include <sstream>
6  using namespace std;
7
8  main()
9  {
10    ostringstream outputString;
11    string s1( "Output of several data types " ),
12           s2( "to an ostringstream object:" ),
13           s3( "\n        double: " ),
14           s4( "\n           int: " ),
15           s5( "\naddress of int: " );
16    double d = 123.4567;
17    int i = 22;
18
19    outputString << s1 << s2 << s3 << d << s4 << i << s5 << &i;
20    cout << "outputString contains:\n" << outputString.str();
21
```

```
22     outputString << "\nmore characters added";
23     cout << "\n\nafter additional stream insertions,\n"
24          << "outputString contains:\n" << outputString.str()
25          << endl;
26
27     return 0;
28  }
```

```
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0068FD0C

after additional stream insertions,
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0068FD0C
more characters added
```

Fig. 19.11 Using a dynamically allocated **ostringstream** object.

```
1  // Fig. 19.12: fig19_12.cpp
2  // Demonstrating input from an istringstream object.
3  #include <iostream>
4  #include <string>
5  #include <sstream>
6  using namespace std;
7
```

Fig. 19.12 Demonstrating input from an **istringstream** object (part 1 of 2).

```
8  main()
9  {
10    string input( "Input test 123 4.7 A" );
11    istringstream inputString( input );
12    string string1, string2;
13    int i;
14    double d;
15    char c;
16
17    inputString >> string1 >> string2 >> i >> d >> c;
18
19    cout << "The following inputs were extracted\n"
20         << "from the istringstream object:"
21         << "\nstring: " << string1
22         << "\nstring: " << string2
23         << "\n   int: " << i
24         << "\ndouble: " << d
25         << "\n  char: " << c;
26
27    // attempt to read from empty stream
28    long l;
29
30    if ( inputString >> l )
31       cout << "\n\nlong value is: " << l << endl;
32    else
```

```
33        cout << "\n\ninputString is empty" << endl;
34
35     return 0;
36  }
```

```
The following items were extracted
from the istringstream object:
String: Input
String: test
   int: 123
double: 4.7
  char: A

inputString is empty
```

Fig. 19.12 Demonstrating input from an **istringstream** object (part 2 of 2).