C++ Strings

Constructors

Syntax:

```
string();
string( size_type length, char ch );
string( const char *str );
string( const char *str, size_type length );
string( string &str, size_type index, size_type length );
string( input_iterator start, input_iterator end );
```

The string constructors create a new string containing:

- *length* copies of *ch*,
- a duplicate of *str* (optionally up to *length* characters long),
- a substring of *str* starting at *index* and *length* characters long, or
- elements from *start* to *end*.

For example,

```
string str1( 5, 'c' );
string str2( "Now is the time..." );
string str3( str2, 11, 4 );
cout << str1 << endl;
cout << str2 << endl;
cout << str3 << endl;
```

displays

```
ccccc
Now is the time...
time
```

compare

Syntax:

```
  int compare( const basic_string &str );
  int compare( const char *str );
  int compare( size_type index, size_type length, const basic_string &str );
  int compare( size_type index, size_type length, const basic_string &str, size_type
index2,
  size_type length2 );
  int compare( size_type index, size_type length, const char *str, size_type
length2 );
```

The compare() function either compares *str* to the current string in a variety of ways, returning

Return Value	Case
less than zero	this < str
zero	this == str
greater than zero	this > str

The various functions either:

- compare *str* to the current string,
- compare *str* to a substring of the current string,starting at *index* for *length* characters,
- compare a substring of *str* to a substring of the current string, where *index2* and *length2* refer to *str* and *index* and *length* refer to the current string,
- or compare a substring of *str* to a substring of the current string, where the substring of *str* begins at zero and is *length2* characters long, and the substring of the current string begins at *index* and is *length* characters long.

data

Syntax:

```
  const char *data();
```

The function data() returns a pointer to the first character in the current string.

empty

Syntax:

```
  bool empty();
```

The empty() function returns **true** if the current string is empty, and **false** otherwise.

erase

Syntax:

```
  iterator erase( iterator pos );
  iterator erase( iterator start, iterator end );
  basic_string &erase( size_type index = 0, size_type num = npos );
```

The erase() function either:

- removes the character pointed to by *pos*, returning an iterator to the next character,
- removes the characters between *start* and *end*, returning an iterator to the character after the last character removed,
- or removes *num* characters from the current string, starting at *index*, and returns ***this**.

The parameters *index* and *num* have default values, which means that erase() can be called with just *index* to erase all characters after *index* or with no arguments to erase all characters. For example:

```
    string s("So, you like donuts, eh? Well, have all the donuts in the world!");
    cout << "The original string is '" << s << "'" << endl;

    s.erase( 50, 14 );
    cout << "Now the string is '" << s << "'" << endl;
    s.erase( 24 );
    cout << "Now the string is '" << s << "'" << endl;
    s.erase();
    cout << "Now the string is '" << s << "'" << endl;
```

will display

```
    The original string is 'So, you like donuts, eh? Well, have all the donuts in the
world!'
    Now the string is 'So, you like donuts, eh? Well, have all the donuts'
    Now the string is 'So, you like donuts, eh?'
    Now the string is ''
```

find

Syntax:

```
  size_type find( const basic_string &str, size_type index );
  size_type find( const char *str, size_type index );
  size_type find( const char *str, size_type index, size_type length );
  size_type find( char ch, size_type index );
```

The function find() either:

- returns the first occurrence of *str* within the current string, starting at *index*, **string::npos** if nothing is found,
- returns the first occurrence of *str* within the current string and within *length* characters, starting at *index*, **string::npos** if nothing is found,
- or returns the index of the first occurrence *ch* within the current string, starting at *index*, **string::npos** if nothing is found.

For example,

```
    string str1( "Alpha Beta Gamma Delta" );
    unsigned int loc = str1.find( "Omega", 0 );
    if( loc != string::npos )
      cout << "Found Omega at " << loc << endl;
    else
      cout << "Didn't find Omega" << endl;
```

find_first_of

Syntax:

```
  size_type find_first_of( const basic_string &str, size_type index = 0 );
  size_type find_first_of( const char *str, size_type index = 0 );
  size_type find_first_of( const char *str, size_type index, size_type num );
  size_type find_first_of( char ch, size_type index = 0 );
```

The find_first_of() function either:

- returns the index of the first character within the current string that matches any character in *str*, beginning the search at *index*, **string::npos** if nothing is found,
- returns the index of the first character within the current string that matches any character in *str*, beginning the search at *index* and searching at most *num* characters, **string::npos** if nothing is found,
- or returns the index of the first occurrence of *ch* in the current string, starting the search at *index*, **string::npos** if nothing is found.

find_first_not_of

Syntax:

```
  size_type find_first_not_of( const basic_string &str, size_type index = 0 );
  size_type find_first_not_of( const char *str, size_type index = 0 );
  size_type find_first_not_of( const char *str, size_type index, size_type num );
  size_type find_first_not_of( char ch, size_type index = 0 );
```

The find_first_not_of() function either:

- returns the index of the first character within the current string that does not match any character in *str*, beginning the search at *index*, **string::npos** if nothing is found,
- returns the index of the first character within the current string that does not match any character in *str*, beginning the search at *index* and searching at most *num* characters, **string::npos** if nothing is found,
- or returns the index of the first occurrence of a character that does not match *ch* in the current string, starting the search at *index*, **string::npos** if nothing is found.

find_last_of

Syntax:

```
  size_type find_last_of( const basic_string &str, size_type index = npos );
  size_type find_last_of( const char *str, size_type index = npos );
  size_type find_last_of( const char *str, size_type index, size_type num );
  size_type find_last_of( char ch, size_type index = npos );
```

The find_last_of() function either:

- returns the index of the first character within the current string that matches any character in *str*, doing a reverse search from *index*, **string::npos** if nothing is found,
- returns the index of the first character within the current string that matches any character in *str*, doing a reverse search from *index* and searching at most *num* characters, **string::npos** if nothing is found,
- or returns the index of the first occurrence of *ch* in the current string, doing a reverse search from *index*, **string::npos** if nothing is found.

find_last_not_of

Syntax:

```
  size_type find_last_not_of( const basic_string &str, size_type index = npos );
  size_type find_last_not_of( const char *str, size_type index = npos);
  size_type find_last_not_of( const char *str, size_type index, size_type num );
  size_type find_last_not_of( char ch, size_type index = npos );
```

The find_last_not_of() function either:

- returns the index of the first character within the current string that does not match any character in *str*, doing a reverse search from *index*, **string::npos** if nothing is found,
- returns the index of the first character within the current string that does not match any character in *str*, doing a reverse search from *index* and searching at most *num* characters, **string::npos** if nothing is found,
- or returns the index of the first occurrence of a character that does not match *ch* in the current string, doing a reverse search from *index*, **string::npos** if nothing is found.

---

insert

Syntax:

```
  iterator insert( iterator i, const char &ch );
  basic_string &insert( size_type index, const basic_string &str );
  basic_string &insert( size_type index, const char *str );
  basic_string &insert( size_type index1, const basic_string &str, size_type index2,
size_type num );
  basic_string &insert( size_type index, const char *str, size_type num );
  basic_string &insert( size_type index, size_type num, char ch );
  void insert( iterator i, size_type num, const char &ch );
  void insert( iterator i, iterator start, iterator end );
```

The very multi-purpose insert() function either:

- inserts *ch* before the character denoted by *i*,
- inserts *str* into the current string, at location *index*,
- inserts a substring of *str* (starting at *index2* and *num* characters long) into the current string, at location *index1*,
- inserts *num* characters of *str* into the current string, at location *index*,
- inserts *num* copies of *ch* into the current string, at location *index*,
- inserts *num* copies of *ch* into the current string, before the character denoted by *i*,
- or inserts the characters denoted by *start* and *end* into the current string, before the character specified by *i*.

---

length

Syntax:

```
  size_type length();
```

The function length() returns the length of the current string. This number should be the same as the one returned from size().

---

substr

Syntax:

```
  basic_string substr( size_type index, size_type num = npos );
```

The substr() function returns a substring of the current string, starting at *index*, and *num* characters long. If *num* is omitted, it will default to **string::npos**, and the substr() function will simply return the remainder of the string starting at *index*. For example:

```
    string s("What we have here is a failure to communicate");
    string sub = s.substr(21);
    cout << "The original string is " << s << endl;
    cout << "The substring is " << sub << endl;
```

displays

```
    The original string is What we have here is a failure to communicate
    The substring is a failure to communicate
```
