C++ Standard Library: The string Class

The **string** class is part of the C++ standard library. A string represents a sequence of characters.

To use the string class, #include the header file:

```
#include <string>
```

Constructors:

- `string ()`
  - creates an empty string ("")
- `string ( other_string )`
  - creates a string identical to `other_string`
- `string ( other_string, position, count )`
  - creates a string that contains `count` characters from `other_string`, starting at `position`. If `count` is missing (only the first two arguments are given), all the characters from `other_string`, starting at `position` and going to the end of `other_string`, are included in the new string.
- `string ( count, character )`
  - create a string containing `character` repeated `count` times

Examples:

```
string s1;                //   s1 = ""

string s2( "abcdef" );    //   s2 = "abcdef"

string s3( s2 );          //   s3 = "abcdef"

string s4( s2, 1 );       //   s4 = "bcdef"

string s5( s2, 3, 2 );    //   s5 = "de"

string s6( 10, '-' );     //   s6 = "----------"
```

The **string** class also has a destructor that takes care of freeing the memory storing the characters when the object is destroyed.

Constant Member Functions:

These functions do not modify the string.

- `const char * data ()`
  - returns a C-style null-terminated string of characters representing the contents of the string
- `unsigned int length ()`
  - returns the length of the string

- `unsigned int size ()`
  - returns the length of the string (i.e., same as the length function)

- `bool empty ()`
  - returns `true` if the string is empty, `false` otherwise

Operators Defined for `string`:

- *Assign* `=`
```
string s1;
string s2;
...
s1 = s2; // the contents of s2 is copied to s1
```

- *Append* `+=`
```
string s1( "abc" );
string s2( "def" );
...
s1 += s2; // s1 = "abcdef" now
```

- *Indexing* `[]`
```
string s( "def" );
char c = s[2]; // c = 'f' now
s[0] = s[1]; // s = "eef" now
```

- *Concatenate* `+`
```
string s1( "abc" );
string s2( "def" );
string s3;
...
s3 = s1 + s2; // s3 = "abcdef" now
```

- *Equality* `==`
```
string s1( "abc" );
string s2( "def" );
string s3( "abc" );
...
bool flag1 = ( s1 == s2 ); // flag1 = false now
bool flag2 = ( s1 == s3 ); // flag2 = true now
```

- *Inequality* `!=`
  - the inverse of equality

- *Comparison* `<, >, <=, >=`
  - performs case-insensitive comparison
```
string s1 = "abc";
string s2 = "ABC";
string s3 = "abcdef";
...
bool flag1 = ( s1 < s2 ); // flag1 = false now
bool flag2 = ( s2 < s3 ); // flag2 = true now
```

Member Functions:

- `void swap ( other_string )`
  - swaps the contents of this string with the contents of `other_string`.
  ```
  string s1( "abc" );
  string s2( "def" );
  s1.swap( s2 ); // s1 = "def", s2 = "abc" now
  ```

- `string & append ( other_string )`
  - appends `other_string` to this string, and returns a reference to the result string.

- `string & insert ( position, other_string )`
  - inserts `other_string` into this string at the given `position`, and returns a reference to the result string.

- `string & erase ( position, count )`
  - removes `count` characters from this string, starting with the character at the given `position`. If `count` is ommitted (only one argument is given), the characters up to the end of the string are removed. If both `position` and `count` are omitted (no arguments are given), the string is cleared (it becomes the empty string). A reference to the result string is returned.

- `unsigned int find ( other_string, position )`
  - finds `other_string` inside this string and returns its position. If `position` is given, the search starts there in this string, otherwise it starts at the beginning of this string.

- `string substr ( position, count )`
  - returns the substring starting at `position` and of length `count` from this string

---