

C++ and CS106 Library Reference

Written by Julie Zelenski.

A couple of decent C++ web resources you might want to bookmark:

```
http://www.cppreference.com  
http://www.cplusplus.com/ref/  
http://msdn2.microsoft.com/en-us/library/csc687y.aspx
```

These can be useful for anything in standard C++, which includes the language itself and all of its standard libraries (**string**, **stream**, **ctype**, **math**, etc.)

The standard C++ **string** class

The **string** class is defined in `<string>`. The **string** type is actually a **typedef** shorthand. The underlying full name is

```
std::basic_string<char, std::char_traits<char>, std::allocator<char>>
```

You don't need to worry about this sort of low-level goop, but you will see the full name in compiler error messages and will want to recognize it as such.

The default constructor initializes a string variable to the empty string, thus declaring a string variable ensures that its contents start empty. This is unlike the built-in types (**int**, **double**, etc.) that have random contents until explicitly initialized. Assigning one string to another via `=` or passing/returning a string makes a new distinct copy of the same character sequence. Strings are mutable, unlike Java strings.

A string literal, i.e., sequence of characters within double-quotes such as **"binky"**, is actually an old-style C-string. You can typically use a C-string wherever a string object is required since there is an automatic conversion from C-string to new-style C++ string object. If ever need to force this conversion, you can do so using a syntax similar to a **typeid**: `string("binky")`. This is invoking the string class constructor that takes a C-string argument.

In general, operations on strings are designed to be very efficient and, as a result, some do not check parameters for validity. It is the client's job to ensure positions/lengths are in bounds for calls to **substr**, **find**, **replace**, and so on. The behavior on incorrect calls is implementation-dependent, but unlikely to be pleasant in any situation.

<i>str.length()</i> <i>str.size()</i>	Returns number of characters in receiver string (length and size are synonyms)
<i>str[index]</i> <i>str.at(index)</i>	Access character at specified index in receiver string. Indexes start at 0. at throws an exception if out of bounds, operator [] does not bounds-check (for efficiency).
<i>str.empty()</i>	Returns true if receiver string is equal to "", false otherwise
<i>str1 + str2</i> <i>str1 + ch</i>	+ is overloaded to allow strings to be concatenated with other strings and single chars. The result is a new string containing concatenation of the operands.
<i>str.find(key, pos)</i>	Searches for key (which can be either string or single character) within receiver string, starting search at index pos . If pos not specified, default value of 0 is used. Returns index of key if found or string::npos otherwise.
<i>str.substr(pos, len)</i>	Returns a new string containing len chars starting from index pos in receiver string. If len is not given, takes all characters to end of string.
<i>str.insert(pos, text)</i>	Inserts text starting at index pos into the receiver string. Modifies receiver string.
<i>str.replace(pos, count, text)</i>	Removes count chars from receiver string starting at index pos , and replaces with text . Modifies receiver string.
<i>str1 < str2</i> == != < > <= >=	String comparison uses standard relational operators. Ordering is lexicographic (dictionary ordering) and case-sensitive.
<i>str.c_str()</i>	Returns receiver string in old-style C-string form. Used when you need backward compatibility with an older function.

CS106 string utility functions

strutils.h contains a few conveniences for handling string conversions. These are free functions (i.e. not member functions invoked on a receiver string).

string RealToString (double d) double StringToReal (string str)	Convert double value to string form and vice versa. StringToReal raises an error if string is not well-formed.
string IntegerToString (int i) int StringToInteger (string str)	Convert integer value to string form and vice versa. StringToInteger raises an error if string is not well-formed.
string ConvertToUpperCase (string s) string ConvertToLowerCase (string s)	Returns a new string which is a copy of input string where all alphabetic characters have been converted to upper/lower case equivalents, non-letter characters are unchanged

Standard C++ stream classes

The global streams **cin/cout** and the basic stream classes are defined in **<iostream>**. The file stream classes are defined in **<fstream>**. There are many variants of stream classes in the standard library, we typically will use **ifstream** for input file streams, and **ofstream** for output file streams. There are many more features available on streams than I will list here. I/O isn't particularly interesting to study and we will mostly just use the simple features, so no need to dig deep.

Like strings, the stream class names are also shortened with a **typedef**. The full, underlying name for **ifstream** is **std::basic_ifstream<char, std::char_traits<char>>** and **ofstream** is same with **ofstream** substituted for **ifstream**.

Copying of stream objects is discouraged. Streams should typically be passed by reference. In most library implementations, copying a stream (either from direct assignment or pass-by-value) is specifically disallowed and will not compile.

These member functions apply to both input and output streams:

<i>stream.open</i> (<i>filenameAsCString</i>)	Opens named file and attaches to receiver stream. If unsuccessful, sets stream error state. The filename parameter is expected to be an old-style C-string! (see c_str above for how to convert a C++ string to C-string)
<i>stream.close</i> ()	Closes file. This is automatically done by stream destructor, but if you open another file on the stream, you first explicitly close any open one.
<i>stream.fail</i> ()	Returns true if the receiver stream is in an error state, e.g a previous stream operation was not successful. Once a stream gets into an error state, the error state persists and no further operations on that stream can succeed until the error state is cleared (see clear below)
<i>stream.clear</i> ()	Clears error state of the receiver stream

These operations are specific to output streams.

<i>ostream << num << str << ch</i>	Stream insertion << does formatted output. See <iomanip> for all the fancy features for controlling width/precision/alignment/format.
<i>ostream.put</i> (<i>ch</i>)	Outputs a single char onto receiver stream

These operations are specific to input streams.

<i>istream >> num >> str >> ch</i>	Stream extraction >> reads formatted input. By default, skips white space. Puts stream into fail state if read doesn't match expected.
<i>istream.peek</i> () <i>istream.get</i> ()	Read next character from receiver stream. Return EOF (-1) if no more characters to read. Returns an int rather than char because of need to represent EOF. peek returns the next character but doesn't remove it from the stream

<code>istream.unget()</code>	Pushes last character read back onto the receiver stream
<code>getline(istream & in, string & str, char delimiter = '\n')</code>	Reads next line of input (up to delimiter) and stores in str reference parameter. Note: this is a free function not a stream member function! You pass the stream to read from as the first argument.

CS106 simple input functions

Handling user input can be a little messy (i.e. retrying on errors, etc.), so these simplified input routines are provided in our **simpio.h** to make your life a little easier. These are supplied as free functions.

<code>string GetLine()</code> <code>int GetInteger()</code> <code>long GetLong()</code> <code>double GetReal()</code>	Each reads a line of input from the user and returns the value. In case of the numeric versions, if user's input is not well-formed, re-prompts and tries again until input is valid.
--	---

CS106 random library

random.h contains a set of functions that generate pseudo-random events. The implementation is layered on top of the standard C functions **rand/srand** from **<stdlib>**.

<code>void Randomize()</code>	Seeds random number generator
<code>int RandomInteger(int low, int high)</code> <code>double RandomReal(double low, double high)</code>	Returns int /real from random range
<code>bool RandomChance(double prob)</code>	Returns true/false based on random probability

CS106 Graphics Library

Our graphics library provides support for drawing in a simple graphics window. The canvas uses a Cartesian-coordinate system and units are expressed in inches. The base graphics library supports drawing lines and arcs. The extended graphics library adds text, filled regions, elliptical arcs, color, and support for event handling. There are now directives to switch the origin to the upper left corner and to change the implicit dimension from inches to pixels, but we won't worry about them this quarter.

These are eight functions exported by **graphics.h**.

<code>void InitGraphics()</code>	Set up empty graphics window
<code>void MovePen(double x, double y)</code>	Move pen to location x , y
<code>void DrawLine(double dx, double dy)</code>	Draw line from pen position moving dx , dy
<code>void DrawArc(double r, double start, double sweep)</code>	Draw arc described by radius , start angle, and sweep angle
<code>double GetWindowWidth()</code> <code>double GetWindowHeight()</code>	Returns graphics window size
<code>double GetCurrentX()</code> <code>double GetCurrentY()</code>	Returns current pen position

There are some 40 functions exported by **extgraph.h**. I listed the prototypes below, but did not attempt to repeat what is already well commented in the header file. Please refer to **extgraph.h** for notes on how to use these functions.

<code>void DrawEllipticalArc(double rx, double ry, double start, double sweep)</code>	Elliptical arcs
<code>void StartFilledRegion(double density)</code> <code>void EndFilledRegion()</code>	Filled region support
<code>void DrawTextString(string text)</code> <code>double TextStringWidth(string text)</code> <code>void SetFont(string font)</code> <code>string GetFont()</code> <code>void SetPointSize(int size)</code> <code>int GetPointSize()</code> <code>void SetStyle(int style)</code> <code>int GetStyle()</code> <code>double GetFontAscent()</code> <code>double GetFontDescent()</code> <code>double GetFontHeight()</code>	Text/font support
<code>double GetMouseX()</code> <code>double GetMouseY()</code> <code>bool MouseButtonIsDown()</code> <code>void WaitForMouseDown()</code> <code>void WaitForMouseUp()</code>	Mouse event-handling support
<code>void SetPenColor(string color)</code> <code>string GetPenColor()</code> <code>void SetPenColorRGB(double r, double g, double b)</code> <code>void DefineColor(string name, double r, double g, double b)</code>	Color support
<code>void DrawNamedPicture(string name)</code> <code>double GetPictureWidth(string name)</code> <code>double GetPictureHeight(string name)</code>	Image support
<code>void Pause(double seconds)</code> <code>void UpdateDisplay()</code>	Delay, force buffered drawing onscreen

<code>void SaveGraphicsState()</code> <code>void RestorergraphicsState()</code>	Save/restore settings
<code>string GetWindowTitle()</code> <code>void SetWindwoTitle(string str)</code> <code>double GetFullScreenWidth()</code> <code>double GetFullScreenHeight()</code> <code>void SetWindowSize(double w, double h)</code> <code>double GetXResolution()</code> <code>double GetyResoution()</code>	Window setup