C++ Quick Reference Guide

Last Update: Saturday, April 2, 2011

C++ Fundamental Data Types

bool	Boolean type (values are **true** or **false**)
char	Character
short	Usually a 2-byte integer
int	Usually a 4-byte integer
long	Same as long int (long size >= int size)
float	Single precision floating point usually 4 bytes
double	double precision floating point usually 8 bytes
<type> *	declares a pointer to a variable of type <type>
<type> &	declares a reference to a variable of type <type>

C++ has type modifiers **unsigned** and **long**. **unsigned** may be applied to integral types (including char). **long** may be applied to **int** or **double**. An unsigned data type only allows non-negative numbers to be stored.

Default integral data type is **int.**
Default floating-point data type is **double.**

"integral" and **"floating-point"** are examples of categories of data. bool, char, int, double etc. are C++ data types.

Examples of literals in C++	Data Type
true, false	bool
'2', '\n', 'A'	char
2, 3, 75	int
2.0, .5, 5.	double
"2", "", "Hello World!"	char * (string)

Strings are represented in C++ as either char arrays or string objects.

```
#include<cstring>  functions for c-strings.
#include<string>   functions for string objects

char firstNname[16];  //firstName is a c-string
string lastName;      //lastName  is a string object
```

Commonly Used C++ Operators

Assignment Operators

=	Assignment
+=	Combined addition/assignment
-=	Combined subtraction/assignment
*=	Combined multiplication/assignment
/=	Combined division/assignment
%=	Combined modulus/assignment

Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division (floating-point or integer) **2.0/3.0 =.666667 (floating-point), 2/3 = 0 (integer)**
%	Modulus (integer remainder) **17 % 3 = 2, 12 % 15 = 12**

Relational Operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Logical Operators

&&	AND
||	OR
!	NOT

Increment/Decrement

++	Increment
--	Decrement

Increment/Decrement (used in prefix and postfix mode)

prefix: inc(dec) variable, then use in larger expression
postfix: use in larger expression, then inc(dec) variable

Pointers in C++:

A "pointer" is a variable that is used to store the address of an object of the type the pointer points to.

int x=5, *xPtr = &x;

* is the indirection operator
*, [ ] dereference a pointer
& is the address-of operator

Selection Structures

- Unary or single selection
  - if
- Binary or dual selection
  - if-else
- Case structure
  - switch
- Simple selection
  - One condition
- Compound selection
  - Multiple conditions joined with AND / OR operators
  - if (score < 0 || score > 100)

Forms of the if Statement

Simple if

```
if (expression)
    statement;
```

Example

```
if (x < y)
    x++;
```

if/else

```
if (expression)
    statement;
else
    statement;
```

Example

```
if (x < y)
    x++;
else
    x--;
```

if/else if (nested if)

```
if (expression)
    statement;
else
     if (expression)
          statement;
     else
          statement;
```

Example

```
 if (x < y)
     x++;
 else
   if (x < z)
      x--;
   else
      y++;
```

To conditionally execute more than one statement, you must create a **compound statement** (block) by enclosing the statements in braces (this is true for loops as well):

Form

```
if (expression)
{
   statement;
   statement;
}
```

Example

```
if (x < y)
{
   x++;
   cout << x << endl;
}
```

The **"expression"** in the parentheses for an

if statement

or

loop

is often also referred to as a **"condition"**

Conditional Operator ? :
(Simplified if-else)

Form: expr1 ? expr2 : expr3;

Example: x = a < b ? a : b;

The statement above works like:

```
if (a < b) x = a;
else       x = b;
```

Escape Sequences

Special characters in Java

\n	newline character	'\n'
\t	tab character	'\t'
\"	double quote	'\"'
\'	single quote	'\''
\\	backslash	'\\'

Operator Precedence

```
    ( )
 ----------
  *, /, %      [ mathematical ]
 ----------
    +, -
```

Logical operators: !, &&, ||, &, |
(1) mathematical (2) relational (3) logical

C++ Quick Reference Guide

Last Update: Saturday, April 2, 2011

Loop Structures

- C++ **Pre-test** loops
  - while
  - for
- C++ **Post-test** loop
  - do…while

Loop Control:

- Counter-controlled aka definite loops have 3 expressions:
  - Initialize (init)
  - Test
  - Update
- Sentinel-controlled aka indefinite loops have 2 expressions:
  - Test
  - Update
- C++ Loop Early Exit:
  - **break** statement
- C++ also has a **continue** statement to skip statements and proceed to the test-expression.

The switch statement (case structure) (**break** and **default** are optional)

Form:

```
switch (expression)
{
  case int-constant :
     statement(s);
   [ break; ]

  case int-constant :
     statement(s);
  [ break; ]

  [ default  :
     statement; ]
}
```

Example:

```
switch (choice)
{
   case 0 :
      cout << "You selected 0."  << endl;
      break;

   case 1:
      cout << "You selected 1." << endl;
      break;

  default :
      cout <<   "Select 0 or 1."  << endl;
}
```

The type of the "*expression*" is integral - usually an expression of type **int** but it could also be an expression of type **char**.

Use the **break** keyword to exit the structure (avoid "falling through" other cases).

Use the **default** keyword to provide a default case if none of the case expressions match (similar to trailing "else" in an if-else-if statement).

The for Loop

Form:

```
for (init; test; update)
     statement;

for (init; test; update)
{
     statement;
     statement;
}
```

Example:

```
for (count = 0; count < 10; count++)
    cout << count << endl;

for (count = 0; count < 10; count++)
{
    cout << "The value of count is ";
    cout << count << endl;
}
```

The while Loop

Form:

```
init;
while (test)
{
   statement(s);
   update;
}
```

Example:

```
int x=0;
while (x < 100)
{
   cout << x << endl;
   x++;
}
```

The do-while Loop

Form:

```
do
   statement;
while (expression);

do
{
   statement;
   statement;
} while (expression);
```

Example:

```
do
   cin >> x;
while (x != 0);

do
{
   cout << x << endl;
   x++;
} while (x < 100);
```

Using cin / cout **Requires iostream header file:** **#include<iostream>**

Note: Default for numeric output is six (6) significant digits

Stream Manipulators: **Requires iomanip header file:** **#include<iomanip>**

Manipulator	Description
fixed	changes mode to fixed-point; displays with integer and decimal digits
setprecision( )	sets the number of significant digits or decimal digits if used with fixed
setw( )	sets field width (used for input and output)
left	sets left justification
right	sets right justification
showpoint	forces decimal point & trailing zeros to display
scientific	sets scientific notation
resetiosflags( )	"turn off" a manipulator. Common use **resetiosflags(ios::fixed)**

Creating and using file stream objects: Requires header file: **#include <fstream>**

Class	Description
ifstream	create a file stream object for use with an input file
ofstream	create a file stream object for use with an output file

Member Functions for file stream classes

Function	Description
open( )	infile.open("data.txt")
close( )	infile.close()
fail( )	infile.fail() test for stream failure (T/F)
clear( )	infile.clear() //reset stream status to good
eof( )	infile.eof() //test for end of file condition (T/F)
peek( )	read next character but don't remove it from the input buffer
unget( )	put last character read back into the input buffer. Replaces the putback( ) function.

Member functions for input formatting using a stream object (such as cin)

Name	Description
.getline(array, size)	Reads at most size-1 characters. Appends '\0'. Stops at '\n' by default. Consumes the newline character
.get(array, size)	Reads at most size-1 characters. Appends '\0'. Stops at '\n' by default. Does not consume the newline character
.get(ch)	reads a character (including whitespace)
.ignore( )	removes last character entered from buffer
.ignore(50,'\n')	removes last 50 characters from input buffer or until it sees a newline character