**Basic Code**

#include <iostream>

using namespace std;

int main ()

{

return 0;

}

**Simple Commands**

|  |  |
| --- | --- |
| cout << variable << line of text << endl | Console output |
| endl; | End line (skip to next line) |
| cin >> variable; | Console input |
| getline (action/location,string-var); | Extracting data from file or user |

**Escape Characters**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| \n | New line | \” | Allows “ in “ “ | \\ | Backslash in “ ” |
| \t | Horizontal tab | \’ | Allows ‘ in “ “ | \a | Sounds an alert (for fun) |

**Variable Types**

|  |  |
| --- | --- |
| **string** **variable-name** = “**Literal String**” | Literal strings |
| **char** **variable-name** = ‘**x**’ | characters |
| **int** **variable-name** = **66** | integers |
| **double variable-name** = **3.14** | Floating point (decimals) (64 bits = bigger range) |
| **float variable-name** = **3.14** | Floating point (decimals) (32 bits = smaller range) |
| **bool variable-name** = **true/false** | True/fase (True == 1, False == 0) |
| **const variable-type variable-name** | Unchanging variable |
|  |  |
| **auto variable-name = expression** | Automatically deduces variable type by given expression |

|  |  |
| --- | --- |
| cout.setf(ios::fixed);  cout.precision(**NumberOfDecimalPoints**); | Specifies the amount of decimal points to be outputted |

Character to integer conversions - <http://www.asciitable.com/> (32 is magic number for lowercase-uppercase conversion)

**Variable Intitalization Techniques**

|  |  |  |
| --- | --- | --- |
| ***Copy Initialization*** | ***Direct Initialization*** | ***C++11 Uniform Initialization*** |
| int x = 10; | int x(10); | int x{10}; |
| Pass by value | Pass by reference | Does not implicitly cast |

**Type Casting (Type Conversion)**

Changing a variable type to another type.

**Implicit Casting:**

When the compiler converts the variable for you. It’s fine for small conversions, but the larger and more intense it gets, this casting method loses precision.

Ex:

int x = 5;

double y = 10.5;

casting double sum = x + y;

**Explicit Casting:**

When the programmer directly

Ex:

int x = 5;

char y;

casting y = (char) x;

**OR**

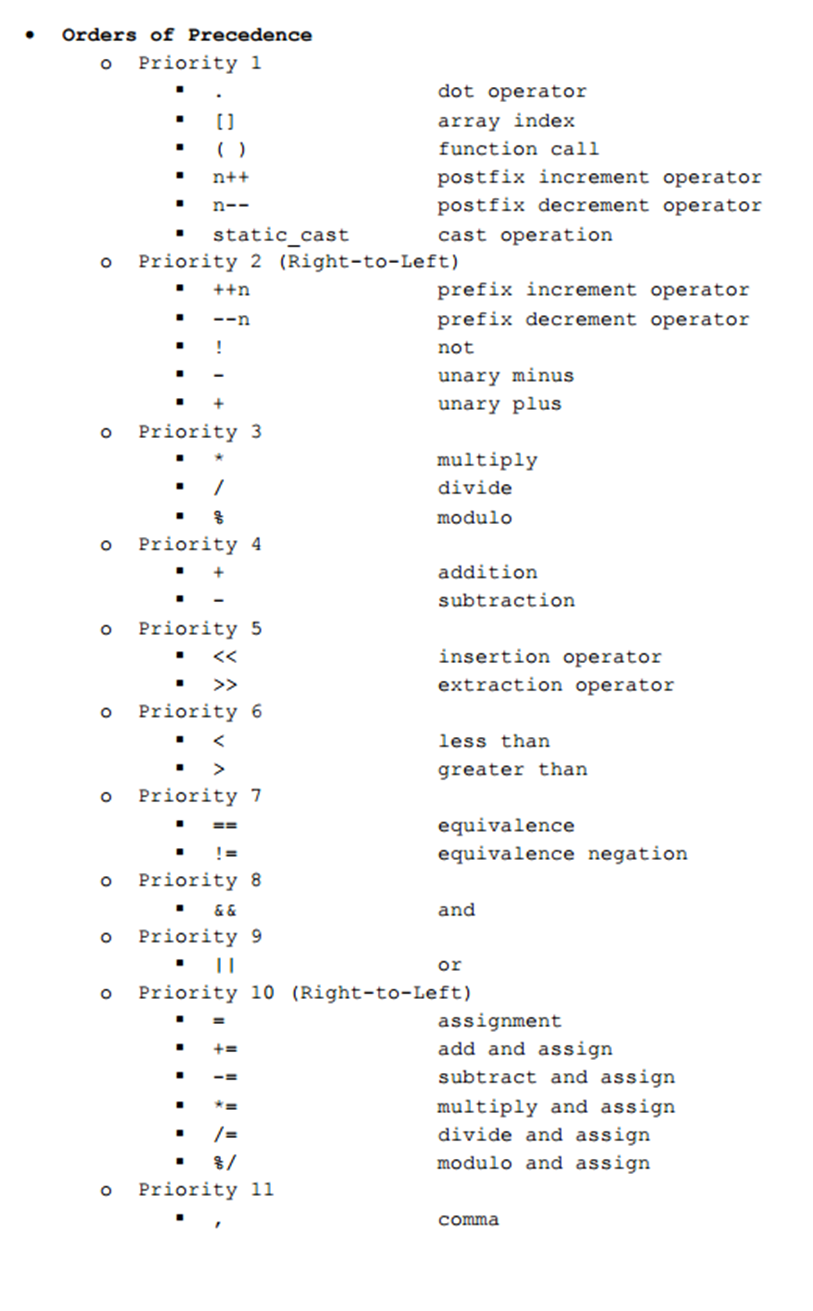
specific casting static\_cast<**new-type**>(**variable/expression**);

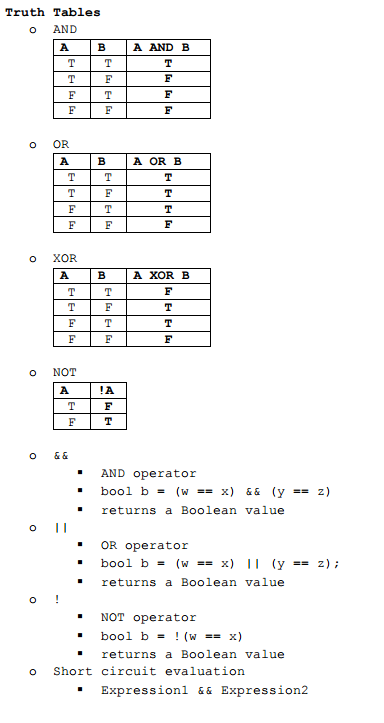
**Operators**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Arithmetic Operators | | | | Conditional Operators | | | |
| + | Addition | ++x | Increment x | > | Greater Than | == | Equals |
| - | Subtraction | x++ | Copy x and increment | < | Less Than | ! | Not |
| \* | Multiplication | --x | Decrement x | >= | Greater or Equal to | != | Not Equal |
| / | Divison | x-- | Copy x and decrement | <= | Less or Equal to | = | Assignment |
| % | Modulus | ?= | (+ - \* /) and assign | && | AND | || | OR |
| () {} | Assignment | << | Output Stream | >> | Input Stream | [ ] | Array |

// : to ‘comment out’ a single line.

/\* ------ \*/ : to ‘comment out’ an entire section.





**If Statements** An if statement is a tool to separate different code into doing things based upon certain perameters.

**Statement Template:**

if (expression) {

commands;

}

// If the expression is true, do something.

else if(expression) {

commands;

}

// If this expression is also true, also do this.

else {

commands;

}

// If all else expressions are not true, do this.

**Switch Statements** A Switch statement is a tool that is used to make menus.

**Statement Template:**

switch (Variable-Option-Choice) {

case value\_1 *(ex. 1, “String”, ‘C’)*:

commands;

break;

case value\_2:

commands;

break;

default:

Commands if non-existant option is selected;

}

Updated: 10/17/2017