# C++ Reference Card

## C++ Data Types

| Data Type | Description |
|---|---|
| `bool` | boolean (true or false) |
| `char` | character ('a', 'b', etc.) |
| `char[]` | character array (C-style string if `null` terminated) |
| `string` | C++ string (from the STL) |
| `int` | integer (1, 2, -1, 1000, etc.) |
| `long int` | long integer |
| `float` | single precision floating point |
| `double` | double precision floating point |

These are the most commonly used types; this is not a complete list.

## Operators

The most commonly used operators in order of precedence:

| | |
|---|---|
| 1 | `++` (post-increment), `--` (post-decrement) |
| 2 | `!` (not), `++` (pre-increment), `--` (pre-decrement) |
| 3 | `*`, `/`, `%` (modulus) |
| 4 | `+`, `-` |
| 5 | `<`, `<=`, `>`, `>=` |
| 6 | `==` (equal-to), `!=` (not-equal-to) |
| 7 | `&&` (and) |
| 8 | `\|\|` (or) |
| 9 | `=` (assignment), `*=`, `/=`, `%=`, `+=`, `-=` |

## Console Input/Output

| | |
|---|---|
| `cout <<` | console out, printing to screen |
| `cin >>` | console in, reading from keyboard |
| `cerr <<` | console error |

Example:
```
cout << "Enter an integer: ";
cin >> i;
cout << "Input: " << i << endl;
```

## File Input/Output

Example (input):
```
ifstream inputFile;
inputFile.open("data.txt");
inputFile >> inputVariable;
// you can also use get (char) or
// getline (entire line) in addition to >>
  ...
inputFile.close();
```

Example (output):
```
ofstream outFile;
outfile.open("output.txt");
outFile << outputVariable;
  ...
outFile.close();
```

## Decision Statements

if

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

Example
```
if (x < y)
    cout << x;
```

if / else

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

Example
```
if (x < y)
    cout << x;
else
    cout << y;
```

switch / case

```
switch(int expression)
{
  case int-constant:
    statement(s);
    break;
  case int-constant:
    statement(s);
    break;
  default:
    statement;
}
```

Example
```
switch(choice)
{
  case 0:
    cout << "Zero";
    break;
  case 1:
    cout << "One";
    break;
  default:
    cout << "What?";
}
```

## Looping

while Loop

```
while (expression)
    statement;

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

Example
```
while (x < 100)
    cout << x++ << endl;

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

do-while Loop

```
do
    statement;
while (expression);

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

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

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

for Loop

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

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

Example
```
for (count = 0; count < 10; count++)
{
    cout << "count equals: ";
    cout << count << endl;
}
```

## Functions

Functions return at most one value. A function that does not return a value has a return type of `void`. Values needed by a function are called parameters.

```
return_type function(type p1, type p2, ...)
{
    statement;
    statement;
    ...
}
```

Examples
```
int timesTwo(int v)
{
    int d;
    d = v * 2;
    return d;
}

void printCourseNumber()
{
    cout << "CSE1284" << endl;
    return;
}
```

Passing Parameters by Value

```
return_type function(type p1)
```

Variable is passed into the function but changes to *p1* are not passed back.

Passing Parameters by Reference

```
return_type function(type &p1)
```

Variable is passed into the function and changes to *p1* are passed back.

Default Parameter Values

```
return_type function(type p1=val)
```

*val* is used as the value of *p1* if the function is called without a parameter.

## Pointers

A pointer variable (or just pointer) is a variable that stores a memory address. Pointers allow the indirect manipulation of data stored in memory.

Pointers are declared using `*`. To set a pointer's value to the address of another variable, use the `&` operator.

Example
```
char c = 'a';
char* cPtr;
cPtr = &c;
```

Use the indirection operator (*) to access or change the value that the pointer references.

Example
```
// continued from example above
*cPtr = 'b';
cout << *cPtr << endl;  // prints the char b
cout << c << endl;      // prints the char b
```

Array names can be used as constant pointers, and pointers can be used as array names.

Example
```
int numbers[]={10, 20, 30, 40, 50};
int* numPtr = numbers;
cout << numbers[0] << endl;     // prints 10
cout << *numPtr << endl;        // prints 10
cout << numbers[1] << endl;     // prints 20
cout << *(numPtr + 1) << endl;  // prints 20
cout << numPtr[2] << endl;      // prints 30
```

## Dynamic Memory

Allocate Memory

```
ptr = new type;

ptr = new type[size];
```

Examples
```
int* iPtr;
iPtr = new int;

int* intArray;
intArray = new int[5];
```

Deallocate Memory

```
delete ptr;
delete [] ptr;
```

Examples
```
delete iPtr;
delete [] intArray;
```

Once a pointer is used to allocate the memory for an array, array notation can be used to access the array locations.

Example
```
int* intArray;
intArray = new int[5];
intArray[0] = 23;
intArray[1] = 32;
```

## Structures

Declaration

```
struct name
{
  type1 element1;
  type2 element2;
};
```

Example
```
struct Hamburger
{
  int patties;
  bool cheese;
};
```

Definition

```
name varName;

name* ptrName;
```

Example
```
Hamburger h;

Hamburger* hPtr;
hPtr = &h;
```

Accessing Members

```
varName.element=val;

ptrName->element=val;
```

Example
```
h.patties = 2;
h.cheese = true;

hPtr->patties = 1;
hPtr->cheese = false;
```

Structures can be used just like the built-in data types in arrays.

## Include Headers

| | |
|---|---|
| #include | <headerfile> |
| **Common Headers / Libraries** | |
| #include <stdio.h> | I / O functions |
| #include <string.h> | string functions |
| #include <time.h> | time functions |
| #include <stdlib.h> | memory, rand, ... |
| #include <math.h> | math functions |
| #include <iostream.h> | |
| #include <fstream.h> | I / O file functions |
| #include "myfile.h" | Insert file in current directory |

## Namespaces

```
using namespace std;
```

## Comments

```
// One line comment text
/* multiple line block comment text */
```

## Basic Variable Types

NUMBER

```
int a; float a;
```

CHARACTER

```
char car; string s;
char car = 'c'; string s = "hola mon";
```

BOOL

```
bool b = false/true;
```

## Basic input / Output Operators

| | |
|---|---|
| cin | cin >> var |
| cout | cout<<"The variable has"<<var |

## Basic Operators / Math Operators

| | | | |
|---|---|---|---|
| + | Add | - | Less |
| * | Mult | / | Div |
| % | Mod | | |
| ++var / --var | | var++ / var-- | |

## Conditionals

| | |
|---|---|
| A == B | if A is equal to B, this is true; otherwise, it's false |
| A != B | if A is NOT equal to B, this is true; otherwise, it's false |
| A < B | if A is less than B, this is true; otherwise, it's false |
| A > B | if A is greater B, this is true; otherwise, it's false |
| A <= B | if A is less than or equal to B, this is true; otherwise, it's false |
| A >= B | if A is greater or equal to B, this is true; otherwise, it's false |
| A ! B | if A |
| A && B | if condition A and condition B are true, this is true; otherwise, it's false. |
| A \|\| B | if condition A or condition B is true, this is true; otherwise, it's false. |

Boolean expressions in C++ are evaluated left t o right!

## Arrays

type array_name [ # of elements ];

```
int price [10];
```

type array_name [# elements] [# elements];

```
int price [5] [10];
```

· Array index starts at 0.
· Ex: Access 3rd element : cout<<price [2];

## Control Flow

### if sentence

```
if ( conditional ) {
  // do something
}
else if ( another_conditional ) {
  // do something else
}
else {
  // do something as default
}
```

### while sentence

```
while ( conditional ) {
// do something
}
```

placing **"break;"** breaks out of the loop.
placing **"continue;"** jumps to next loop.

### for sentence

```
for ( init; test; command ) {
  // do something
}
```

**"break;"** and **"continue;"** identical effects.

### do while sentence

```
do {
  //do something
} while (bool expression);
```

### switch case sentence

```
switch ( variable )
{
case value1:
  // do something;
  break;
case value2:
  // do something else;
  break;
[default:
  // do something by default:
  break; ]
}
```