QuickRef.ME

Stars 4.7k

C++

C++ quick reference cheat sheet that provides basic syntax and methods.

Getting Started

hello.cpp

```cpp
#include <iostream>

int main() {
    std::cout << "Hello QuickRef\n";
    return 0;
}
```

Compiling and running

```
$ g++ hello.cpp -o hello
$ ./hello
Hello QuickRef
```

Variables

```cpp
int number = 5;       // Integer
float f = 0.95;       // Floating number
double PI = 3.14159;  // Floating number
char yes = 'Y';       // Character
std::string s = "ME"; // String (text)
bool isRight = true;  // Boolean

// Constants
```

```
std::cout << age;      // Print 25
```

Primitive Data Types

Data Type	Size	Range
`int`	4 bytes	-2^{31} to $2^{31}-1$
`float`	4 bytes	N/A
`double`	8 bytes	N/A
`char`	1 byte	-128 to 127
`bool`	1 byte	true / false
`void`	N/A	N/A
`wchar_t`	2 or 4 bytes	1 wide character

User Input

```
int num;

std::cout << "Type a number: ";
std::cin >> num;

std::cout << "You entered " << num;
```

Swap

```
int a = 5, b = 10;
std::swap(a, b);

// Outputs: a=10, b=5
std::cout << "a=" << a << ", b=" << b;
```

Comments

```
// A single one line comment in C++

/* This is a multiple line comment
   in C++ */
```

If statement

```
if (a == 10) {
    // do something
}
```

See: Conditionals

Loops

```
for (int i = 0; i < 10; i++) {
    std::cout << i << "\n";
}
```

See: Loops

```
void hello(); // Declaring

int main() {  // main function
    hello();    // Calling
}

void hello() { // Defining
    std::cout << "Hello QuickRef!\n";
}
```

See: Functions

References

```
int i = 1;
int& ri = i; // ri is a reference to i

ri = 2; // i is now changed to 2
std::cout << "i=" << i;

i = 3;   // i is now changed to 3
std::cout << "ri=" << ri;
```

`ri` and `i` refer to the same memory location.

Namespaces

```
#include <iostream>
namespace ns1 {int val(){return 5;}}
int main()
{
    std::cout << ns1::val();
}
```

```
#include <iostream>
namespace ns1 {int val(){return 5;}}
using namespace ns1;
using namespace std;
int main()
{
    cout << val();
}
```

Namespaces allow global identifiers under a name

C++ Arrays

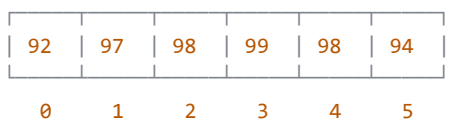
Declaration

```
std::array<int, 3> marks; // Definition
marks[0] = 92;
marks[1] = 97;
marks[2] = 98;

// Define and initialize
```

```
std::array<int, 3> marks = {92, 97};
std::cout << marks[2]; // Outputs: 0
```

Manipulation




```
std::array<int, 6> marks = {92, 97, 98, 99, 98, 94};

// Print first element
std::cout << marks[0];

// Change 2th element to 99
marks[1] = 99;

// Take input from the user
std::cin >> marks[2];
```

Displaying

```
char ref[5] = {'R', 'e', 'f'};

// Range based for loop
for (const int &n : ref) {
    std::cout << std::string(1, n);
}

// Traditional for loop
for (int i = 0; i < sizeof(ref); ++i) {
    std::cout << ref[i];
}
```

Multidimensional


```
int x[2][6] = {
    {1,2,3,4,5,6}, {6,5,4,3,2,1}
};
for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 6; ++j) {
        std::cout << x[i][j] << " ";
    }
}
// Outputs: 1 2 3 4 5 6 6 5 4 3 2 1
```

```
if (a == 10) {
    // do something
}
```

```
int number = 16;

if (number % 2 == 0)
{
    std::cout << "even";
}
else
{
    std::cout << "odd";
}

// Outputs: even
```

Else if Statement

```
int score = 99;
if (score == 100) {
    std::cout << "Superb";
}
else if (score >= 90) {
    std::cout << "Excellent";
}
else if (score >= 80) {
    std::cout << "Very Good";
}
else if (score >= 70) {
    std::cout << "Good";
}
else if (score >= 60)
    std::cout << "OK";
else
    std::cout << "What?";
```

Operators

Relational Operators	
`a == b`	a is equal to b
`a != b`	a is NOT equal to b
`a < b`	a is less than b
`a > b`	a is greater b
`a <= b`	a is less than or equal to b
`a >= b`	a is greater or equal to b
Assignment Operators	
`a += b`	Aka a = a + b
`a -= b`	Aka a = a - b
`a *= b`	Aka a = a * b
`a /= b`	Aka a = a / b

`exp1 && exp2`	Both are true (AND)
`exp1 || exp2`	Either is true (OR)
`!exp`	exp is false (NOT)

Bitwise Operators

`a & b`	Binary AND
`a | b`	Binary OR
`a ^ b`	Binary XOR
`~ a`	Binary One's Complement
`a << b`	Binary Shift Left
`a >> b`	Binary Shift Right

Ternary Operator

```
          ┌── True ──┐
Result = Condition ? Exp1 : Exp2;
          └──── False ────┘
```

```
int x = 3, y = 5, max;
max = (x > y) ? x : y;

// Outputs: 5
std::cout << max << std::endl;
```

```
int x = 3, y = 5, max;
if (x > y) {
    max = x;
} else {
    max = y;
}
// Outputs: 5
std::cout << max << std::endl;
```

Switch Statement

```
int num = 2;
switch (num) {
    case 0:
        std::cout << "Zero";
        break;
    case 1:
        std::cout << "One";
        break;
    case 2:
        std::cout << "Two";
        break;
    case 3:
        std::cout << "Three";
        break;
    default:
        std::cout << "What?";
        break;
```

C++ Loops

While

```
int i = 0;
while (i < 6) {
    std::cout << i++;
}

// Outputs: 012345
```

Do-while

```
int i = 1;
do {
    std::cout << i++;
} while (i <= 5);

// Outputs: 12345
```

Continue statements

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        continue;
    }
    std::cout << i;
} // Outputs: 13579
```

Infinite loop

```
while (true) { // true or 1
    std::cout << "infinite loop";
}
```

```
for (;;) {
    std::cout << "infinite loop";
}
```

```
for(int i = 1; i > 0; i++) {
    std::cout << "infinite loop";
}
```

for_each (Since C++11)

```
#include <iostream>

int main()
{
    auto print = [](int num) { std::cout << num << std::endl; };

    std::array<int, 4> arr = {1, 2, 3, 4};
    std::for_each(arr.begin(), arr.end(), print);
    return 0;
}
```

```
for (int n : {1, 2, 3, 4, 5}) {
    std::cout << n << " ";
}
// Outputs: 1 2 3 4 5
```

```
std::string hello = "QuickRef.ME";
for (char c: hello)
{
    std::cout << c << " ";
}
```

Break statements

```
int password, times = 0;
while (password != 1234) {
    if (times++ >= 3) {
        std::cout << "Locked!\n";
        break;
    }
    std::cout << "Password: ";
    std::cin >> password; // input
}
```

Several variations

```
for (int i = 0, j = 2; i < 3; i++, j--){
    std::cout << "i=" << i << ",";
    std::cout << "j=" << j << ";";
}
// Outputs: i=0,j=2;i=1,j=1;i=2,j=0;
```

C++ Functions

Arguments & Returns

```
#include <iostream>

int add(int a, int b) {
    return a + b;
}

int main() {
    std::cout << add(10, 20);
}
```

add is a function taking 2 ints and returning int

Overloading

```
void fun(string a, string b) {
    std::cout << a + " " + b;
}
void fun(string a) {
    std::cout << a;
```

Built-in Functions

```
#include <iostream>
#include <cmath> // import library

int main() {
    // sqrt() is from cmath
    std::cout << sqrt(9);
}
```

C++ Classes & Objects

Defining a Class

```
class MyClass {
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};
```

Creating an Object

```
MyClass myObj;  // Create an object of MyClass

myObj.myNum = 15;          // Set the value of myNum to 15
myObj.myString = "Hello";  // Set the value of myString to "Hello"

cout << myObj.myNum << endl;         // Output 15
cout << myObj.myString << endl;      // Output "Hello"
```

Constructors

```
class MyClass {
  public:
    int myNum;
    string myString;
    MyClass() {  // Constructor
      myNum = 0;
      myString = "";
    }
};

MyClass myObj;  // Create an object of MyClass

cout << myObj.myNum << endl;         // Output 0
cout << myObj.myString << endl;      // Output ""
```

Destructors

```
class MyClass {
  public:
```

```
    myString = "";
  }
  ~MyClass() {  // Destructor
    cout << "Object destroyed." << endl;
  }
};

MyClass myObj;  // Create an object of MyClass

// Code here...

// Object is destroyed automatically when the program exits the scope
```

Class Methods

```
class MyClass {
  public:
    int myNum;
    string myString;
    void myMethod() {  // Method/function defined inside the class
      cout << "Hello World!" << endl;
    }
};

MyClass myObj;  // Create an object of MyClass
myObj.myMethod();  // Call the method
```

Access Modifiers

```
class MyClass {
  public:     // Public access specifier
    int x;    // Public attribute
  private:    // Private access specifier
    int y;    // Private attribute
  protected:  // Protected access specifier
    int z;    // Protected attribute
};

MyClass myObj;
myObj.x = 25;  // Allowed (public)
myObj.y = 50;  // Not allowed (private)
myObj.z = 75;  // Not allowed (protected)
```

Getters and Setters

```
class MyClass {
  private:
    int myNum;
  public:
    void setMyNum(int num) {  // Setter
      myNum = num;
    }
    int getMyNum() {  // Getter
      return myNum;
    }
};
```

```
cout << myObj.getMyNum() << endl;  // Output 15
```

Inheritance

```
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut!" << endl;
    }
};

class Car : public Vehicle {
  public:
    string model = "Mustang";
};

Car myCar;
myCar.honk();  // Output "Tuut, tuut!"
cout << myCar.brand + " " + myCar.model << endl;  // Output "Ford Mustang"
```

C++ Preprocessor

Preprocessor

if	elif
else	endif
ifdef	ifndef
define	undef
include	line
error	pragma
defined	__has_include
__has_cpp_attribute	export
import	module

Includes

```
#include "iostream"
#include <iostream>
```

Defines

```
#define FOO
#define FOO "hello"

#undef FOO
```

```
  ...
#else
  ...
#endif
```

Error

```
#if VERSION == 2.0
  #error Unsupported
  #warning Not really supported
#endif
```

Macro

```
#define DEG(x) ((x) * 57.29)
```

Token concat

```
#define DST(name) name##_s name##_t
DST(object);   #=> object_s object_t;
```

Stringification

```
#define STR(name) #name
char * a = STR(object);   #=> char * a = "object";
```

file and line

```
#define LOG(msg) console.log(__FILE__, __LINE__, msg)
#=> console.log("file.txt", 3, "hey")
```

Miscellaneous

Escape Sequences

\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null Character

alignas	alignof	and	and_eq	asm
atomic_cancel	atomic_commit	atomic_noexcept	auto	bitand
bitor	bool	break	case	catch
char	char8_t	char16_t	char32_t	class
compl	concept	const	consteval	constexpr
constinit	const_cast	continue	co_await	co_return
co_yield	decltype	default	delete	do
double	dynamic_cast	else	enum	explicit
export	extern	false	float	for
friend	goto	if	inline	int
long	mutable	namespace	new	noexcept
not	not_eq	nullptr	operator	or
or_eq	private	protected	public	reflexpr
register	reinterpret_cast	requires	return	short
signed	sizeof	static	static_assert	static_cast
struct	switch	synchronized	template	this
thread_local	throw	true	try	typedef
typeid	typename	union	unsigned	using
virtual	void	volatile	wchar_t	while
xor	xor_eq	final	override	transaction_safe
transaction_safe_dynamic				

Preprocessor

if	elif
else	endif
ifdef	ifndef
define	undef
include	line
error	pragma
defined	__has_include
__has_cpp_attribute	export
import	module

C++ reference (cppreference.com)
C++ Language Tutorials (cplusplus.com)

Top Cheatsheet

Python **Cheatsheet**
Quick Reference

Vim **Cheatsheet**
Quick Reference

JavaScript **Cheatsheet**
Quick Reference

Bash **Cheatsheet**
Quick Reference

Recent Cheatsheet

Remote Work Revolution **Cheatsheet**
Quick Reference

Homebrew **Cheatsheet**
Quick Reference

PyTorch **Cheatsheet**
Quick Reference

Taskset **Cheatsheet**
Quick Reference