

# C++ Cheatsheet

## Basics

Basic syntax and functions from the C++ programming language.

## Boilerplate

```
#include <iostream>
using namespace std;

int main() {
    cout << "Welcome To CodeWithHarry";
    return 0;
}
```

## cout <<

It prints output on the screen

```
cout << "This is C++ Programming";
```

## cin >>

It takes input from the user

```
cin >> variable_name
```

## Data types

The data type is the type of data

## Character type

Typically a single octet(one byte). It is an integer type

```
char variable_name;
```

## Integer type

The most natural size of integer for the machine

```
int variable_name;
```

## Float type

A single-precision floating-point value

```
float variable_name;
```

## Double type

A double-precision floating-point value

```
double variable_name;
```

## Void type

Represents the absence of the type

```
void
```

## Boolean type

```
bool
```

## Escape Sequences

It is a sequence of characters starting with a backslash, and it doesn't represent itself when used inside string literal.

### Alarm or Beep

It produces a beep sound

```
\a
```

### Backspace

It adds a backspace

```
\b
```

### Form feed

```
\f
```

### Newline

Newline Character

```
\n
```

### Carriage return

```
\r
```

## Tab

It gives a tab space

```
\t
```

## Backslash

It adds a backslash

```
\\
```

## Single quote

It adds a single quotation mark

```
\'
```

## Question mark

It adds a question mark

```
\?
```

## Octal No.

It represents the value of an octal number

```
\nnn
```

## Hexadecimal No.

It represents the value of a hexadecimal number

```
\xhh
```

## Null

The null character is usually used to terminate a string

```
\0
```

## Comments

A comment is a code that is not executed by the compiler, and the programmer uses it to keep track of the code.

## Single line comment

```
// It's a single line comment
```

## Multi-line comment

```
/* It's a  
multi-line  
comment  
*/
```

## Strings

It is a collection of characters surrounded by double quotes

## Declaring String

```
// Include the string library  
#include <string>  
  
// String variable  
string variable1 = "Hello World";
```

## append function

It is used to concatenate two strings

```
string firstName = "Harry ";  
string lastName = "Bhai";  
string fullName = firstName.append(lastName);  
cout << fullName;
```

## length function

It returns the length of the string

```
string variable1 = "CodeWithHarry";  
cout << "The length of the string is: " << variable1.length();
```

## Accessing and changing string characters

```
string variable1 = "Hello World";  
variable1[1] = 'i';  
cout << variable1;
```

# Maths

C++ provides some built-in math functions that help the programmer to perform mathematical operations efficiently.

## max function

It returns the larger value among the two

```
cout << max(25, 140);
```

## min function

It returns the smaller value among the two

```
cout << min(55, 50);
```

## sqrt function

It returns the square root of a supplied number

```
#include <cmath>

cout << sqrt(144);
```

## ceil function

It returns the value of x rounded up to its nearest integer

```
ceil(x)
```

## floor function

It returns the value of x rounded down to its nearest integer

```
floor(x)
```

## pow function

It returns the value of x to the power of y

```
pow(x, y)
```

# Decision Making Instructions

Conditional statements are used to perform operations based on some condition.

## If Statement

```
if (condition) {  
  // This block of code will get executed, if the condition is True  
}
```

## If-else Statement

```
if (condition) {  
  // If condition is True then this block will get executed  
} else {  
  // If condition is False then this block will get executed  
}
```

## if else-if Statement

```
if (condition) {  
  // Statements;  
}  
else if (condition){  
  // Statements;  
}  
else{  
  // Statements  
}
```

## Ternary Operator

It is shorthand of an if-else statement.

```
variable = (condition) ? expressionTrue : expressionFalse;
```

## Switch Case Statement

It allows a variable to be tested for equality against a list of values (cases).

```
switch (expression)  
{  
  case constant-expression:  
    statement1;  
    statement2;  
    break;  
  case constant-expression:  
    statement;  
    break;  
  ...  
  default:  
    statement;  
}
```

# Iterative Statements

Iterative statements facilitate programmers to execute any block of code lines repeatedly and can be controlled as per conditions added by the programmer.

## while Loop

It iterates the block of code as long as a specified condition is True

```
while (/* condition */)
{
    /* code block to be executed */
}
```

## do-while loop

It is an exit controlled loop. It is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the condition is False

```
do
{
    /* code */
} while (/* condition */);
```

## for loop

It is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

```
for (int i = 0; i < count; i++)
{
    /* code */
}
```

## Break Statement

break keyword inside the loop is used to terminate the loop

```
break;
```

## Continue Statement

continue keyword skips the rest of the current iteration of the loop and returns to the starting point of the loop

```
continue;
```

## References

Reference is an alias for an already existing variable. Once it is initialized to a variable, it cannot be changed to refer to another variable. So, it's a const pointer.

## Creating References

```
string var1 = "Value1"; // var1 variable
string &var2 = var1; // reference to var1
```

## Pointers

Pointer is a variable that holds the memory address of another variable

### Declaration

```
datatype *var_name;

var_name = &variable2;
```

## Functions & Recursion

Functions are used to divide an extensive program into smaller pieces. It can be called multiple times to provide reusability and modularity to the C program.

### Function Definition

```
return_type function_name(data_type parameter...){
//code to be executed
}
```

### Function Call

```
function_name(arguments);
```

## Recursion

Recursion is when a function calls a copy of itself to work on a minor problem. And the function that calls itself is known as the Recursive function.

```
void recurse()
{
... ..
recurse();
... ..
}
```

## Object-Oriented Programming

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.



## class

```
class Class_name {  
public: // Access specifier  
// fields  
// functions  
// blocks  
};
```

## object

```
Class_name ObjectName;
```

## Constructors

It is a special method that is called automatically as soon as the object is created.

```
class className { // The class  
public: // Access specifier  
className() { // Constructor  
cout << "Code With Harry";  
}  
};  
  
int main() {  
className obj_name;  
return 0;  
}
```

## Encapsulation

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

```

#include<iostream>
using namespace std;
class ExampleEncap{
private:
/* Since we have marked these data members private,
* any entity outside this class cannot access these
* data members directly, they have to use getter and
* setter functions.
*/
int num;
char ch;
public:
/* Getter functions to get the value of data members.
* Since these functions are public, they can be accessed
* outside the class, thus provide the access to data members
* through them
*/
int getNum() const {
return num;
}
char getCh() const {
return ch;
}
/* Setter functions, they are called for assigning the values
* to the private data members.
*/
void setNum(int num) {
this->num = num;
}
void setCh(char ch) {
this->ch = ch;
}
};
int main(){
ExampleEncap obj;
obj.setNum(100);
obj.setCh('A');
cout<<obj.getNum()<<endl;
cout<<obj.getCh()<<endl;
return 0;
}

```

## File Handling

File handling refers to reading or writing data from files. C provides some functions that allow us to manipulate data in the files.

### Creating and writing to a text file

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
// Create and open a text file
ofstream MyFile("filename.txt");

// Write to the file
MyFile << "File Handling in C++";

// Close the file
MyFile.close();
}
```

## Reading the file

It allows us to read the file line by line

```
getline()
```

## Opening a File

It opens a file in the C++ program

```
void open(const char* file_name,ios::openmode mode);
```

## OPEN MODES

### in

Opens the file to read(default for ifstream)

```
fs.open ("test.txt", std::fstream::in)
```

### out

Opens the file to write(default for ofstream)

```
fs.open ("test.txt", std::fstream::out)
```

## binary

Opens the file in binary mode

```
fs.open ("test.txt", std::fstream::binary)
```

## app

Opens the file and appends all the outputs at the end

```
fs.open ("test.txt", std::fstream::app)
```

## ate

Opens the file and moves the control to the end of the file

```
fs.open ("test.txt", std::fstream::ate)
```

## trunc

Removes the data in the existing file

```
fs.open ("test.txt", std::fstream::trunc)
```

## nocreate

Opens the file only if it already exists

```
fs.open ("test.txt", std::fstream::nocreate)
```

## noreplace

Opens the file only if it does not already exist

```
fs.open ("test.txt", std::fstream::noreplace)
```

## Closing a file

It closes the file

```
myfile.close()
```

## Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

### try and catch block

A basic try-catch block in python. When the try block throws an error, the control goes to the except block

```
try {  
  // code to try  
  throw exception; // If a problem arises, then throw an exception  
}  
catch () {  
  // Block of code to handle errors  
}
```