C++ Cheatsheet

A quick reference guide to C++ syntax, data structures, and common operations. Designed for both beginners and experienced programmers to quickly recall essential information.

Fundamentals

Basic Syntax

`#include <iostream>`	Includes the iostream library for input/output operations.
`int main() { ... }`	The main function, where program execution begins.
`std::cout << "Hello, World!";`	Prints "Hello, World!" to the console.
`std::cin >> variable;`	Reads input from the console into the specified variable.
`return 0;`	Indicates successful program execution.
`;`	Statement terminator.

Variables and Data Types

`int`	Integer data type (e.g., `int age = 30;`).
`float`	Single-precision floating-point number (e.g., `float pi = 3.14;`).
`double`	Double-precision floating-point number (e.g., `double price = 99.99;`).
`char`	Character data type (e.g., `char grade = 'A';`).
`bool`	Boolean data type (e.g., `bool is_valid = true;`).
`std::string`	String data type (e.g., `std::string name = "John";`). Requires `#include <string>`.

Operators

Arithmetic Operators: `+`, `-`, `*`, `/`, `%` (modulus)

Assignment Operators: `=`, `+=`, `-=`, `*=`, `/=`, `%=`

Comparison Operators: `==`, `!=`, `>`, `<`, `>=`, `<=`

Logical Operators: `&&` (AND), `||` (OR), `!` (NOT)

Increment/Decrement Operators: `++`, `--`

Control Flow

Conditional Statements

`if (condition) { ... }`	Executes a block of code if the condition is true.
`if (condition) { ... } else { ... }`	Executes one block of code if the condition is true, and another block if it is false.
`if (condition1) { ... } else if (condition2) { ... } else { ... }`	Chained conditional statements.
`switch (expression) { case value1: ... break; case value2: ... break; default: ... }`	Selects one of several code blocks to execute based on the value of an expression.

Loops

`for (initialization; condition; increment) { ... }`	Executes a block of code repeatedly as long as the condition is true.
`while (condition) { ... }`	Executes a block of code repeatedly as long as the condition is true.
`do { ... } while (condition);`	Executes a block of code at least once, and then repeatedly as long as the condition is true.
`break;`	Exits a loop or switch statement.
`continue;`	Skips the rest of the current iteration of a loop and proceeds to the next iteration.

Examples

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

```
int count = 0;
while (count < 5) {
  std::cout << "Count: " << count << std::endl;
  count++;
}
```

Functions

Function Declaration & Definition

`return_type function_name(parameter_list) { ... }`	Defines a function with a specified return type, name, and parameters.
`void my_function() { ... }`	A function that does not return a value.
`int add(int a, int b) { return a + b; }`	A function that adds two integers and returns the result.

Function Overloading

Function overloading allows you to define multiple functions with the same name but different parameter lists (different number or types of parameters).

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

Parameters

Pass by value	A copy of the variable's value is passed to the function. Changes to the parameter inside the function do not affect the original variable.
Pass by reference	A reference to the variable is passed to the function. Changes to the parameter inside the function do affect the original variable. Use `&`.
Pass by pointer	The memory address of the variable is passed to the function. Changes to the value at the memory address affect the original variable. Use `*`.

Classes and Objects

Class Definition

`class ClassName { ... };`	Defines a new class.
`public:`	Members are accessible from outside the class.
`private:`	Members are only accessible from within the class.
`protected:`	Members are accessible from within the class and its derived classes.

Object Creation

`ClassName objectName;`	Creates an object of the class.
`ClassName* objectPtr = new ClassName();`	Creates an object on the heap and returns a pointer to it. Remember to `delete objectPtr;` to avoid memory leaks.

Example

```
class Dog {
public:
  std::string breed;
  int age;

  void bark() {
    std::cout << "Woof!" << std::endl;
  }
};

int main() {
  Dog myDog;
  myDog.breed = "Labrador";
  myDog.age = 3;
  myDog.bark();
  return 0;
}
```