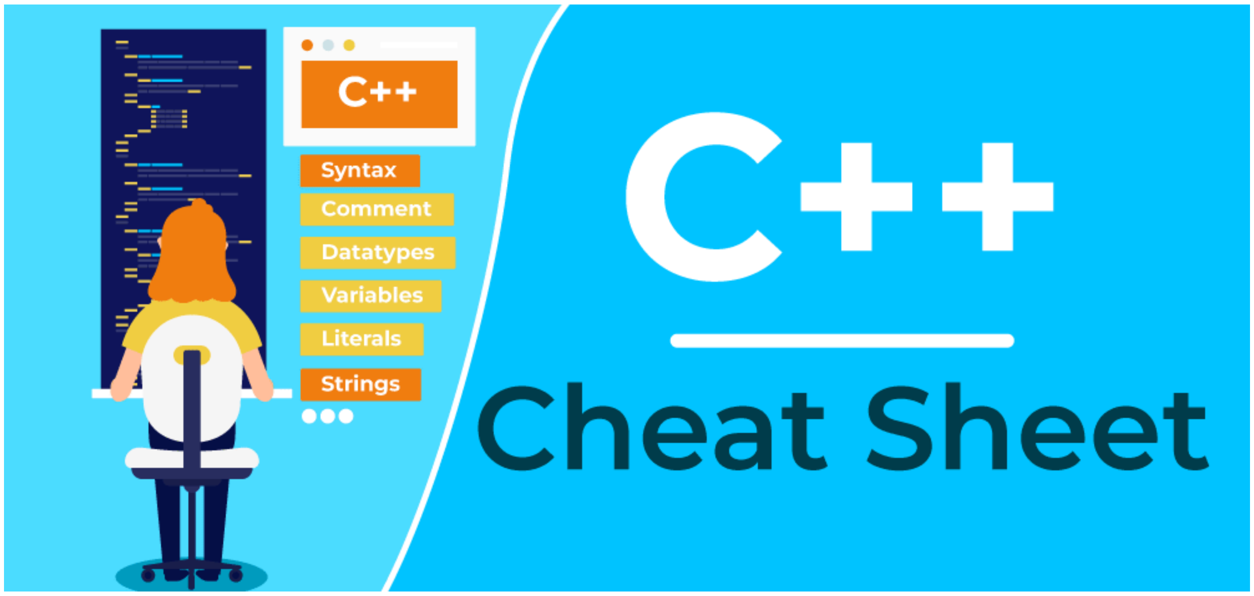
C++ Cheatsheet

Last Updated : 01 May, 2025

This is a C++ programming cheat sheet. It is useful for beginners and intermediates looking to learn or revise the concepts of C++ programming. While learning a new language, it feels annoying to switch pages and find different websites for different concepts that are easily understandable. You can learn C++ concepts very easily using this cheat sheet.



C++ is a high-level programming language. It was developed in 1983 by Bjarne Stroustrup at Bell Labs. It is used for developing various applications.

Basic Structure of C++ Programs

```
// Header files
#include<bits/stdc++.h>

// std namespace contains
// various standard library components
```

```
int main() {

    // This is the section where
    // we write code statements
    return 0;
}
```

Comments

Comments in C++ are used for providing an explanation of the code that makes it easier for others to understand the functionality of the code. They are not executed by the compiler. Comments can also be used to temporarily disable specific statements of code without deleting them.

There are two types of comments:

1. Single-lined

We use two forward slashes **//** to indicate the single-line comment. For example,

```
// This is a comment
```

2. Multi-lined

We use /* to start a multi-line comment and */ to end it. For example,

Variables

In C++, a variable is a container used to store data values:

- Variables must be declared before they can be used.
- Multiple variables can also be declared at one time.
- The name of a variable can contain alphabets, digits, and an underscore but the name of a variable must start with an alphabet or an underscore.

> ***Identifiers****: All variables in a program must be given unique names which are known as the identifiers so each variable can be identified uniquely.*
>
> ***Constants****: Constants are the fixed values that remain unchanged during the execution of the program.*

Syntax

```
// Declaring a single variable
data_type var_name;

// Declaring multiple variables
data_type var1_name, var2_name, var3_name;
```

Data Types

Data types are the type of data that a variable can store in a program.

1. Integer

- It is used to store integers.
- Integers take 4 bytes of memory.

Example

```
int var = 123;
```

- It is used to store characters.
- It takes 1 bytes of memory.

Example

```
char var = 'a';
```

3. Floating Point

- It is used for storing single-precision floating-point numbers.
- It takes 4 bytes of memory.

Example

```
float num = 1.23;
```

4. Double

- It is used to store double-precision floating point numbers.
- It takes 8 bytes of memory.

Example

```
double num = 1.2345;
```

5. Boolean

- It is used to store logical values that can be either true or false.

Example

```
bool b = false;
```

- A string is a collection of characters surrounded by double quotes. The string data type is used to store words or sentences.
- The string data type is part of the Standard Library and is defined in the **<string>** header file.
- We have to include **<string>** header file for using string class.

Example

```
string str = "GeeksforGeeks";
```

Input and Output

1. Input from user: We can take input from the user using **cin** from the **iostream** library. For example,

```
int var;
cin >> var;
```

2. Output on the console: We can print output on the console using **cout** from the **iostream** library. For example,

```
cout << “Hello World”;
```

New Lines

We can use **\n** character or **endl** to insert a new line. For example,

```
cout << "Hello World! \n";
cout << "Hello World!" << endl;
```

Conditional Statements

Conditional statements allow us to control the flow of the program based on certain conditions. It helps us to run a specific section of code based on a condition.

If statement executes a block of code if and only if the given condition is true.

Syntax

```
if (condition) {
    // Code to be executed if the condition is true
}
```

2. Nested if statement

Syntax

```
if (condition1) {
    // Code to be executed
    if (condition2) {
        // Code to be executed
    }
}
```

3. The if-else statement

In if-else statement, condition inside the **if** statement is true, then the code inside the if block will get executed, otherwise code inside the else block will get executed.

Syntax

```
if (condition) {
    // Code to be executed
    // if the condition is true
} else {
    // Code to be executed
    // if the condition is false
}
```

4. The else-if statement

The else if statement allows you to check for multiple conditions

```
    // Code to be executed
    // if condition1 is true
} else if (condition2) {
    // Code to be executed if
    // condition1 is false and
    // condition2 is true
} else {
    // Code to be executed if
    // all conditions are false
}
```

5. Shorthand if else (Ternary Operator)

Shorthand if else also known as the Ternary operator **(?:)** works just like if-else statements that can be used to reduce the number of lines of code.

Syntax

```
(condition) ? expression1 : expression2;
```

Condition **inside round brackets ()** is true, expression1 will be evaluated and it will become the result of the expression. Otherwise, if the condition is false, expression2 will be evaluated and it will become the result.

6. Switch statement

The switch statement evaluates the expression and compares the value of the expression with the cases. If the expression matches the value of any of the cases, the code associated with that case will be executed.

7. Break and Default

The **break** keyword is used to exit the switch statement when one of the cases matches, while the **default** keyword, which is optional, executes when none of the cases match the value of the expression.

Syntax

```
        // Code to be executed if
        // expression matches value1
        break;
    case value2:
        // Code to be executed if
        // expression matches value2
        break;
    // ...
    default:
        // Code to be executed if
        // expression does not match any case
        break;
}
```

Note: *Default keywords is used with switch statements.*

Loops

Loops are used to repeatedly execute a block of code multiple times.

Types of Loops

1. For Loop

For loop helps us to execute a block of code a fixed number of times.

Syntax:

```
for (initialization expr; test expr; update expr) {
    // body of the for loop
}
```

2. While Loop

While loop repeatedly executes a block of code till the given condition is true.

Syntax

```
while (condition) {
```

3. Do-While Loop

Do-while loop also executes the block of code till the condition is true but the difference between a while and a do-while loop is that the do-while executes the code once without checking the condition and the test condition is tested at the end of the loop body.

Syntax

```
do {
    // Body of do-while loop
} while (condition);
```

Arrays

An array is a data structure that allows us to store a fixed number of elements of the same data type in contiguous memory locations.

Syntax:

```
dataType array_name[size];
```

where,

- **data_type**: Type of data to be stored in the array.
- **array_name**: Name of the array.
- **size**: Size of array.

Example

```
#include <iostream>
#include <string>
using namespace std;
int main() {

    // Declare and initialize an array of strings
    string fruits[] = {"Apple", "Banana", "Orange", "Grapes"};

    // Access and print each element of the array
    return 0;
}
```

Output

```
Fruit at index 0: Apple
Fruit at index 1: Banana
Fruit at index 2: Orange
Fruit at index 3: Grapes
```

Multi-Dimensional Arrays

Multi-dimensional arrays are known as arrays of arrays that store similar types of data in tabular form.

Syntax:

```
data_type array_name[size1][size2]....[sizeN];
```

where **size1, size2,..., sizeN are** size of each dimension.

2-Dimensional arrays are the most commonly used multi-dimensional arrays in C++.

Example

```
#include <iostream>
using namespace std;
int main()
{
    // Declaration and initialization of a 2D array
    int arr[3][4] = { { 1, 2, 3, 4 },
                      { 5, 6, 7, 8 },
                      { 9, 10, 11, 12 } };

    // Accessing elements in the 2D array
    // Output: 1
    cout << "Element at arr[0][0]: " << arr[0][0] << endl;
    // Changing the value of an element
    // Output: 20
    arr[2][3] = 20;
    cout << "Modified element at arr[2][3]: " << arr[2][3]
         << endl;

    // Nested loops for iterating through the 2D array
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output

```
Element at arr[0][0]: 1
Element at arr[1][2]: 7
Modified element at arr[2][3]: 20
1 2 3 4
5 6 7 8
9 10 11 20
```

Vectors

Vectors are a dynamic array-like data structure that stores elements of the same data type in a contiguous fashion that can resize itself automatically unlike arrays which mean vectors can grow when an element is inserted or shrink when an element is deleted.

- Vectors are present in C++ Standard Template Library (STL).
- We have to #include <vector> header file in our C++ program to use vectors.

Commonly used Vector Functions

- *push_back()* - *It is used to insert the elements at the end of the vector.*
- ***pop_back()*** - *It is used to pop or remove elements from the end of the vector.*
- ***clear()*** - *It is used to remove all the elements of the vector.*
- ***empty()*** - *It is used to check if the vector is empty.*
- ***at(i)*** - *It is used to access the element at the specified index 'i'.*
- ***front()*** - *It is used to access the first element of the vector.*
- ***back()*** - *It is used to access the last element of the vector.*
- ***erase()*** - *It is used to remove an element at a specified position.*

Example:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    // Create an empty vector
    vector<int> numbers;

    // push_back()
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);

    // Accessing elements using at()
    // Output: 10
    cout << "Element at index 0: " << numbers.at(0) << endl;
    // Output: 20
    cout << "Element at index 1: " << numbers.at(1) << endl;
    cout << "Last element: " << numbers.back() << endl;

    // pop_back()
    // Remove the last element
    numbers.pop_back();

    // erase()
    // Remove the element at index 1
    numbers.erase(numbers.begin() + 1);

    // empty()
    if (numbers.empty()) {
        cout << "Vector is empty" << endl;
    }
    else {
        cout << "Vector is not empty" << endl;
    }

    // clear()
    // Remove all elements
    numbers.clear();

    if (numbers.empty()) {
        cout << "Vector is empty" << endl;
    }
    else {
        cout << "Vector is not empty" << endl;
    }

    return 0;
}
```

Output

```
Element at index 0: 10
Element at index 1: 20
First element: 10
```

References and Pointers

References

References provide an alias for an existing variable. We can manipulate the original value using the reference variable. The reference variable is declared using **&** operator.

Example

```
int var= 12;
// A reference variable to var
int& ref= var;
```

where**, ref** is a reference to **var** variable.

Pointers

A pointer is a variable that stores the memory address of another variable. It can be created using the ***** operator and the address of another variable can be assigned using the address-of operator **&**.

Example

```
int i = 3;
// A pointer to variable i or "stores the address of i"
int *ptr = &i;
```

Functions

Functions are the reusable block of a set of statements that performs a specific task. Functions can be used to organize the logic of the program.

Syntax for function declaration

```
return_type function_name(parameters);
```

Syntax for function definition

```
return_type function_name(parameters) {
    // return statement (if applicable)
}
```

- **return_type**: It is the data type of the value that a function returns.
- **function_name**: It is the name of the function.
- **parameters**: parameters are the input values provided when the function is called. parameters are optional.

Example

Program to add two numbers.

```
#include<bits/stdc++.h>
using namespace std;

// Function declaration
int sum(int a, int b);

// Function definition
int sum(int a, int b) {
    return a + b;
}

int main()
{
    // Function call
    int result = sum(3, 4);
      cout << result;
}
```

Output

```
7
```

Explanation: The function **sum** takes two integers as parameters and returns the sum of the two numbers. The return type of the function is **int**. The parameters of the function are two integers that are 3 and 4. The

There are several string functions present in Standard Template Library in C++ that are used to perform operations on strings. Some of the commonly used string functions are:

1. length() Function

Returns the length of a string.

Example

```
string str = "GeeksforGeeks";
cout << "String length: " << str.length();
```

2. substr() Function

It is used to extract a substring from a given string.

Syntax

```
string substr (size_t pos, size_t len) const;
```

- **pos**: Position of the first character to be copied
- **len**: Length of the sub-string.
- **size_t**: It is an unsigned integral type.

Example

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string str = "GeeksforGeeks";

    // Extracts a substring starting from
    // index 1 with a length of 5
    string sub = str.substr(1, 5);
}
```

Output

```
Substring: eeksf
```

3. append() Function

Appends a string at the end of the given string.

Example

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str = "Geeksfor";

    str.append("Geeks");

    cout << "Appended string: " << str << endl;

    return 0;
}
```

Output

```
Appended string: GeeksforGeeks
```

4. compare() Function

It is used to compare two strings lexicographically.

Example

```
#include <iostream>
#include <string>
```

```
    string str2 = "for";
    string str3 = "Geeks";

    int result1 = str1.compare(str2);
    cout << "Comparison result: " << result1 << endl;

    int result2 = str1.compare(str3);
    cout << "Comparison result: " << result2 << endl;

    return 0;
}
```

Output

```
Comparison result: -31
Comparison result: 0
```

5. empty() Function

It is used to check if a string is empty.

Example

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    string str1 = "GeeksforGeeks";
    string str2 = "";

    if (str1.empty())
        cout << "str1 is empty" << endl;
    else
        cout << "str1 is not empty" << endl;

    if (str2.empty())
        cout << "str2 is empty" << endl;
}
```

Output

```
str1 is not empty
str2 is empty
```

Math Functions

Function	Description	Example
min(x, y)	Returns the minimum value of x and y.	cout << min(10, 20);
max(x, y)	Returns the maximum value of x and y.	cout << max(10, 20);
sqrt(x)	Returns the square root of x.	cout << sqrt(25);
ceil(x)	It rounds up the value x to its nearest integer.	double ceilX = ceil(3.14159);
floor(x)	It rounds the value of x downwards to the nearest integer.	double floorX = floor(3.14159);
pow(x,n)	It returns the value x raised to the power of y	double result = pow(3.0, 2.0);

Object-Oriented Programming

Object-oriented programming generally means storing data in the form of

- Class: A class is a user-defined data type that contains its data members and member functions. A class is a blueprint for objects having similar attributes and behavior.
- **Objects:** An object is an instance or a variable of the class.

Pillars of OOPs

1. Encapsulation

Encapsulation is wrapping up the data and methods together within a single entity. In C++, classes are used for encapsulation.

2. Abstraction

Showing only the necessary details and hiding the internal details is known as **abstraction**.

4. Inheritance

Deriving the properties of a class (Parent class) to another class (Child class) is known as Inheritance. It is used for code reusabilty.

Types of Inheritance:

- **Single Inheritance**: When a derived class inherits the properties of a single base class, it is known as Single Inheritance.
- **Multiple Inheritance**: When a derived class inherits the properties of multiple base classes, it is known as Multiple Inheritance.
- **Multilevel Inheritance**: When a derived class inherits the properties of another derived class, it is known as Multilevel Inheritance.
- **Hierarchical Inheritance**: When more than one derived class inherits the properties of a single base class, it is known as Hierarchical Inheritance.

Combining Multilevel and Hierarchical inheritance.

3. Polymorphism

Providing different functionalities to the functions or operators of the same name is known as Polymorphism.

C++ provides two types of polymorphism:

- Compile-time Polymorphism can be achieved using:
  - Operator overloading
  - Function overloading
- Runtime Polymorphism can be achieved using:
  - Function overriding
  - Virtual Functions

4. Inheritance

Deriving the properties of a class (Parent class) to another class (Child class) is known as Inheritance. It is used for code reusability.

Types of Inheritance:

- **Single Inheritance**: When a derived class inherits the properties of a single base class, it is known as Single Inheritance.
- **Multiple Inheritance**: When a derived class inherits the properties of multiple base classes, it is known as Multiple Inheritance.
- **Multilevel Inheritance**: When a derived class inherits the properties of another derived class, it is known as Multilevel Inheritance.
- **Hierarchical Inheritance**: When more than one derived class inherits the properties of a single base class, it is known as Hierarchical Inheritance.
- **Hybrid (Virtual) Inheritance**: When we combine more than one type of

File Handling

File handling means reading data from a file and manipulating the data of a file.

File Handling Operations

*1. **Open a file**: We can use **open()** member function of **ofstream** class to open a file.*

*2. **Read a file**: We can use **getline()** member function of **ifstream** class to read a file.*

*3. **Write to a file**: We can use **<<** operator to write to a file after opening a file with the object of **ofstream** class.*

Example:

```
#include <fstream>
#include <iostream>
#include <string>

using namespace std;

int main()
{
    ofstream outputFile("example.txt");

    // Open the file for writing
    outputFile.open("example.txt");
    if (outputFile.is_open()) {

        // Write data to the file
        outputFile << "Hello, World!" << endl;
        outputFile << 42 << endl;
        outputFile.close(); // Close the file
    }
    else {
```

```
             << endl;
        return 1;
    }

    // Reading from a file
    ifstream inputFile("example.txt");
    if (inputFile.is_open()) {
        string line;
        while (getline(inputFile, line)) {
            // Print each line
            cout << line << endl;
        }
        // Close the file
        inputFile.close();
    }
    else {

        // Failed to open the file
        cout << "Error opening the file for reading."
             << endl;
        return 1;
    }

    return 0;
}
```

This C++ cheat sheet can serve as a reference guide for programmers that provides quick access to concepts of C++.

Comment More info

Campus Training Program

Next Article

C Cheat Sheet

Similar Reads

4. C++ <cstdio>
5. GeeksforGeeks Master Sheet - List of all Cheat Sheets
6. C++ Programming Multiple Choice Questions
7. C++ printf() Function
8. C++ Tutorial | Learn C++ Programming
9. Output of C programs | Set 60 (Constants)
10. Output of C++ programs | Set 42

Python
Java
C++
PHP
GoLang
SQL
R Language
Android

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
DSA Interview Questions
Competitive Programming

Data Science & ML
Data Science With Python
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies
HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
NodeJs
Bootstrap
Tailwind CSS