Search...   99+

HTML Cheat Sheet   CSS Cheat Sheet   JS Cheat Sheet   Bootstrap Cheat Sheet   jQuery Cheat Sheet   Ang

C Cheat Sheet

Last Updated : 19 Mar, 2025

This **C Cheat Sheet** provides an overview of both basic and advanced concepts of the C language. Whether you're a beginner or an experienced programmer, this cheat sheet will help you revise and quickly go through the core principles of the C language.

In this Cheat Sheet, we will delve into the basics of the C language, exploring its fundamental concepts that lay the groundwork for programming. We will cover topics such as variables, data types, and operators, providing you with a solid understanding of the building blocks of C programming.

Basic Syntax

Consider the below Hello World program:

```
// C Hello World program
```

```c
{
    printf("Hello World!");
    return 0;
}
```

Output

```
Hello World!
```

Here,

- **#include <stdio.h>:** The header file inclusion to use printf() function.
- **int main():** The main() function is the entry point of any C program.
- **printf("Hello World"):** Function to print hello world.
- **return 0:** Value returned by the main() function.

Variables

A variable is the name given to the memory location that stores some data.

Syntax of Variable

```c
data_type variable_name;
data_type variable_name = initial_value;
```

A variable can be of the following types:

1. Local Variable
2. Global Variable
3. Static Variable
4. Extern Variable
5. Auto Variable
6. Register Variable

Data Types

The data type is the type of data that a given variable can store. Different data types have different sizes. There are 3 types of data types in C:

1. Basic Data Types
2. Derived Data Types
3. User Defined Data Types

1. Basic Data Types

Basic data types are built-in in the C programming language and are independent of any other data type. There are x types of basic data types in C:

1. **char:** Used to represent characters.
2. **int:** Used to represent integral numbers.
3. **float:** Used to represent decimal numbers up to 6-7 precision digits.
4. **double:** Used to represent decimal numbers up to 15 precision digits.
5. **void:** Used to represent the valueless entity.

Example of Basic Data Types

```
double d = 24.3435;
void v;
```

The size of these basic data types can be modified using **data type modifiers** which are:

1. short
2. long
3. signed
4. unsigned

Example of Data Type Modifiers

```
unsigned int var1;
long double var2;
long int var3;
```

2. Derived Data Types

Derived data types are derived from the basic data types. There are 2 derived data types in C:

1. Arrays
2. Pointers

3. User-Defined Data Types

The user-defined data types are the data types that are defined by the programmers in their code. There are 3 user-defined data types in C:

1. Structure
2. Union
3. Enumeration

1. A variable name must only contain alphabets, digits, and underscore.
2. A variable name must start with an alphabet or an underscore only. It cannot start with a digit.
3. No whitespace is allowed within the variable name.
4. A variable name must not be any reserved word or keyword.

Example of Identifiers

```
var, any_other_name, __this_var, var22;
```

Keywords

Keywords are the reserved words that have predefined meanings in the C compiler. They cannot be used as identifiers.

Example of Keywords

```
auto,
float,
int,
return,
switch
```

Basic Input and Output

The basic input and output in C are done using two <stdio.h> functions namely scanf() and print() respectively.

Basic Output - print()

The printf() function is used to print the output on the standard output device which is generally the display screen.

Syntax of printf()

where,

- **formatted-string:** String to be printed. It may contain format specifiers, escape sequences, etc.
- **arguments-list:** It is the list of data to be printed.

Basic Input - scanf()

The scanf() function is used to take input from the standard input device such as the keyboard.

Syntax of scanf()

```
scanf("formatted-string", {address-argument-list});
```

where,

- **formatted-string:** String that describes the format of the input.
- **address-of-arguments-list:** It is the address list of the variables where we want to store the input.

Example of Input and Output

```
// C program to illustrate the basic input and outpu  × IS ▷ ɟ ⧉
// printf() and scanf()
#include <stdio.h>

int main()
{
    int roll_num;
    char name[50];

    // taking input using scanf
    scanf("Enter Roll No.: %d", &roll_num);
    scanf("Enter Name: %s", name);
```

```
        return 0;
}
```

Output

```
Name is ◁ and Roll Number is 0
```

Input

```
Enter Roll No: 20
Enter Name: GeeksforGeeks
```

Output

```
Name is GeeksforGeeks and Roll Number is 20.
```

Format Specifiers

Format specifiers are used to describe the format of input and output in formatted string. It is different for different data types. It always starts with %

The following is the **list of some commonly used format specifiers in C:**

Format Specifier	Description
%c	For b type.
%d	For signed integer type.
%f	For float type.

Format Specifier	Description
%p	Pointer
%s	String
%u	Unsigned int
%%	Prints % character

Escape Sequence

Escape sequences are the characters that are used to represent those characters that cannot by represented normally. They start with **(\ ) backslash** and can be used inside string literals.

The below table list some commonly used escape sequences:

Escape Sequence	Name	Description
\b	Backspace	It is used to move the cursor backward.
\n	New Line	It moves the cursor to the start of the next line.
\r	Carriage Return	It moves the cursor to the start of the current line.
\t	Horizontal Tab	It inserts some whitespace to the left of the cursor and moves the cursor accordingly.

Escape Sequence	Name	Description
\\	Backlash	Use to insert backslash character.
\"	Double Quote	It is used to display double quotation marks.
\0	NULL	It represents the NLL character.

Operators

Operators are the symbols that are used to perform some kind of operation. Operators can be classified based on the type of operation they perform.

There are the following types of operators in C:

S.No.	Operator Type	Description	Example
1.	Arithmetic Operators	Operators that perform arithmetic operations.	+, -, *, /, %
2.	Relational Operators	They are used to compare two values.	<, >, <=, >=, ==, !=
3.	Bitwise Operators	They are used to perform bit-level operations on integers.	&, ^, |, <<, >>, ~
4.	Logical Operators	They perform logical operations such as logical AND, logical OR, etc.	&&, ||, !

S.No.	Operator Type	Description	Example
6.	Assignment Operators	They are used to assign some value to the variables.	=, +=, -=, <<=
7.	Miscellaneous Operators	comma, addressof, sizeof, etc. are some other types of operators.	, sizeof, &, *, ->, .

Conditional Statements

Conditional statements are used to execute some block of code based on whether the given condition is true. There are the following conditional statements in C:

1. if Statement

if statement contains a block of code that will be executed if and only if the given condition is true.

Syntax of if

```c
if (condition) {
    // statements
}
```

2. if-else Statements

The if-else statement contains the else block in addition to the if block which will be executed if the given condition is false.

Syntax if-else

```c
if (expression) {
```

```
        // else block
    }
```

3. if-else-if Ladder

The if-else-if ladder is used when we have to test multiple conditions and for each of these conditions, we have a separate block of code.

Syntax of if-else-if

```
if (expression) {
    // block 1
}
else if (expression) {
    // block 1
}
.
.
.
else {
    // else block
}
```

4. switch Case Statement

The switch case statement is an alternative to the if-else-if ladder that can execute different blocks of statements based on the value of the single variable named switch variable.

Syntax of switch

```
switch (expression) {
    case value1:
        // statements
        break;
```

```
        .
        .
        .
    default:
        // defualt block
        break;
  }
```

5. Conditional Operator

The conditional operator is a kind of single-line if-else statement that tests the condition and executes the true and false statements.

Syntax of Conditional Operator

```
(condition) ? (true-exp) : (false-exp);
```

Example of Conditional Statements

```c
// C program to illustrate conditional statements
#include <stdio.h>

int main()
{
    // conditional operator will assign 10 if 5 < 25,
    // otherwise it will assign 20
    int i = 5 < 25 ? 10 : 20;

    if (i == 10)
        printf("i is 10");
    else if (i == 15)
        printf("i is 15");
    else if (i == 20)
        printf("i is 20");
    else
```

Output

```
i is 10
```

Loops

Loops are the control statements that are used to repeat some block of code till the specified condition is false. There are 3 loops in C:

1. for Loop

The for loop is an entry-controlled loop that consists initialization, condition, and updating as a part of itself.

Syntax of for

```
for (initialization; condition; updation) {
    // statements
}
```

2. while Loop

The while loop is also an entry-controlled loop but only the condition is the part of is syntax.

Syntax of while

```
while (condition) {
    // initialization
}
```

3. do-while Loop

The do-while loop is an exit-controlled loop in which the condition is

```
do {
    // statements
} while (condition);
```

Jump Statements

Jump statements are used to override the normal control flow of the program. There are 3 jump statements in C:

1. break Statement

It is used to terminate the loop and bring the program control to the statements after the loop.

Syntax of break

```
break;
```

It is also used in the switch statement.

2. continue Statement

The continue statement skips the current iteration and moves to the next iteration when encountered in the loop.

Syntax of continue

```
continue;
```

3. goto Statement

The goto statement is used to move the program control to the predefined label.

Syntax of goto

```
       .          |      .
       .          |      .
   label:         |    goto label;
```

Example of Loops and Jump Statements

```c
// C program to illustrate loops
#include <stdio.h>

// Driver code
int main()
{
    int i = 0;

    // for loop with continue
    for (i = 1; i <= 10; i++) {
        if (i == 5 || i == 7) {
            continue;
        }
        printf("%d ", i);
    }
    printf("\n");

    // while loop with break
    i = 1;
    while (i <= 10) {
        if (i == 7) {
            break;
        }
        printf("%d ", i++);
    }
    printf("\n");

    // do_while loop
    i = 1;
    do {
        printf("%d ", i++);
    } while (i <= 10);
```

```c
    i = 1;
any_label:
    printf("%d ", i++);
    if (i <= 10) {
        goto any_label;
    }

    return 0;
}
```

Output

```
1 2 3 4 6 8 9 10
1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Arrays

An array is a fixed-size homogeneous collection of items stored at a contiguous memory location. It can contain elements from type int, char, float, structure, etc. to even other arrays.

- Array provides **random access** using the element index.
- Array **size cannot change.**
- Array can have **multiple dimensions** in which it can grow.

Syntax of Arrays

```c
data_type arr_name [size1];    // 1D array
data_type arr_name [size1][size2];    // 2D array
data_type arr_name [size1][size2][size3];    // 3D array
```

Example of Arrays

```c
int main()
{

    // array declaration and initialization
    int arr[5] = { 10, 20, 30, 40, 50 };

    // modifying element at index 2
    arr[2] = 100;

    // traversing array using for loop
    printf("Elements in Array: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output

```
Elements in Array: 10 20 100 40 50
```

Strings

Strings are the sequence of characters terminated by a '\0' NULL character.
It is stored as the array of characters in C.

Syntax of Strings

```c
char string_name [] = "any_text";
```

Example of Strings

```c
// C program to illustrate strings              ✕  ▷  ⧉

#include <stdio.h>
```

```c
    // declare and initialize string
    char str[] = "Geeks";

    // print string
    printf("%s\n", str);

    int length = 0;
    length = strlen(str);

    // displaying the length of string
    printf("Length of string str is %d", length);

    return 0;
}
```

Output

```
Geeks
Length of string str is 5
```

C String Functions

C language provides some useful functions for string manipulation in **<string.h>** header file. Some of them are as follows:

S. No.	Function	Description
1.	strlen()	Find the length of the string
2.	strcmp()	Compares two strings.
3.	strcpy()	Copy one string to another.
4.	strcat()	Concatenate one string with another.

S. No.	Function	Description
6.	strstr()	Find the given substring in the string.

Pointers

Pointers are the variables that store the address of another variable. They can point to any data type in C

Syntax of Pointers

```
data_type * ptr_name;
```

> **Note:** The addressof (&) operator is used to get the address of a variable.

We can dereference (access the value pointed by the pointer) using the same ***** operator.

Example of Pointers

```c
// C program to illustrate Pointers
#include <stdio.h>

// Driver program
int main()
{
    int var = 10;

    // declare pointer variable
    int* ptr;

    // note that data type of ptr and var must be same
```

```c
        printf("Value at ptr = %p \n", ptr);
        printf("Value at var = %d \n", var);
        printf("Value at *ptr = %d \n", *ptr);
        return 0;
    }
```

Output

```
 Value at ptr = 0x7ffd62e6408c
 Value at var = 10
 Value at *ptr = 10
```

There are different types of pointers based on different classification parameters. Some of them are:

1. Double Pointers
2. Function Pointers
3. Structure Pointers
4. NULL Pointers
5. Dangling Pointers
6. Wild Pointers

Functions

Functions are the block of statements enclosed within **{ } braces** that perform some specific task. They provide code reusability and modularity to the program.

Function Syntax is divided into three parts:

1. Function Prototype

It tells the compiler about the existence of the function.

```
    return_type function_name ( parameter_type_list... );
```

1. **Return Type:** It is the type of optional value returned by the function. Only one value can be returned.

2. **Parameters:** It is the data passed to the function by the caller.

2. Function Definition

It contains the actual statements to be executed when the function is called.

```
return_type function_name ( parameter_type_name_list... ) {
    // block of statements
    .
    .
}
```

3. Function Call

Calls the function by providing arguments. A function call must always be after either function definition or function prototype.

```
function_name (arguments);
```

Example of Function

```c
// C program to show function
// call and definition
#include <stdio.h>

// Function that takes two parameters
// a and b as inputs and returns
// their sum
int sum(int a, int b) { return a + b; }

// Driver code
```

```
    // storing its value in add variable
    int add = sum(10, 30);

    printf("Sum is: %d", add);
    return 0;
}
```

Output

```
Sum is: 40
```

Type of Function

A function can be of 4 types based on return value and parameters:

1. Function with no return value and no parameters.
2. Function with no return value and parameters.
3. Function with return value and no parameters.
4. Function with return value and parameters.

There is another classification of function in which there are 2 types of functions:

1. Library Functions
2. User-Defined Functions

Dynamic Memory Management

Dynamic memory management allows the programmer to allocate the memory at the program's runtime. The C language provides four <stdlib.h> functions for dynamic memory management which are malloc(), calloc(), realloc() and free().

1. malloc()

returns the null pointer.

Syntax

```
malloc (size_t size);
```

2. calloc()

The calloc() function allocates the number of blocks of the specified size in the memory. It returns the void pointer to the memory block. If the allocation is failed, it returns the null pointer.

Syntax

```
calloc (size_t num, size_t size);
```

3. realloc()

The realloc() function is used to change the size of the already allocated memory. It also returns the void pointer to the allocated memory.

Syntax

```
realloc (void *ptr, size_t new_size);
```

4. free()

The free function is used to deallocate the already allocated memory.

Syntax

```
free (ptr);
```

Example of Dynamic Memory Allocation

```c
#include <stdlib.h>

int main()
{
    // using malloc to allocate the int array of size 10
    int* ptr = (int*)malloc(sizeof(int) * 10);

    // allocating same array using calloc
    int* ptr2 = (int*)calloc(10, sizeof(int));

    printf("malloc Array Size: %d\n", 10);
    printf("calloc Array Size: %d\n", 10);

    // reallocating the size of first array
    ptr = realloc(ptr, sizeof(int) * 5);
    printf("malloc Array Size after using realloc: %d", 5);

    // freeing all memory
    free(ptr);

    return 0;
}
```

Output

```
malloc Array Size: 10
calloc Array Size: 10
malloc Array Size after using realloc: 5
```

Structures

A structure is a user-defined data type that can contain items of different types as its members. In C, struct keyword is used to declare structures and we can use **(.) dot operator** to access structure members.

Structure Template

```
struct struct_name {
    member_type1 name1;
    member_type1 name1;
    .
    .
};
```

Structure Variable Syntax

```
...{
    ...structure template...
}var1, var2..., varN;
```

or

```
strcut str_name var1, var2,...varN;
```

Example of Structure

```c
// C program to illustrate the use of structures
#include <stdio.h>

// declaring structure with name str1
struct str1 {
    int i;
    char c;
    float f;
    char s[30];
};

// declaring structure with name str2
struct str2 {
    int ii;
    char cc;
```

```c
// Driver code
int main()
{
    // variable declaration after structure template
    // initialization with initializer list and designated
    // initializer list
    struct str1 var1 = { 1, 'A', 1.00, "GeeksforGeeks" },
                   var2;
    struct str2 var3 = { .ff = 5.00, .ii = 5, .cc = 'a' };

    // copying structure using assignment operator
    var2 = var1;

    printf("Struct 1:\n\ti = %d, c = %c, f = %f, s = %s\n",
            var1.i, var1.c, var1.f, var1.s);
    printf("Struct 2:\n\ti = %d, c = %c, f = %f, s = %s\n",
            var2.i, var2.c, var2.f, var2.s);
    printf("Struct 3\n\ti = %d, c = %c, f = %f\n", var3.ii,
            var3.cc, var3.ff);

    return 0;
}
```

Output

```
Struct 1:
    i = 1, c = A, f = 1.000000, s = GeeksforGeeks
Struct 2:
    i = 1, c = A, f = 1.000000, s = GeeksforGeeks
Struct 3
    i = 5, c = a, f = 5.000000
```

Union

A union is also a user-defined data type that can contain elements of different types. However, unlike structure, a union stores its members in a shared memory location rather than having separate memory for each

```
union union_name {
    // members
    .

    .
 }
```

Union members can be accessed using **dot operator (.)** but only one member can store the data at a particular instance in time.

Example of Union

```c
// C Program to demonstrate how to use union
#include <stdio.h>

// union template or declaration
union un {
    int member1;
    char member2;
    float member3;
};

// driver code
int main()
{

    // defining a union variable
    union un var1;

    // initializing the union member
    var1.member1 = 15;

    printf("The value stored in member1 = %d",
            var1.member1);

    return 0;
}
```

```
The value stored in member1 = 15
```

Enumeration (enum)

Enumeration, also known as enum is a user-defined data type that is used to assign some name to the integral constant. By default, the enum members are assigned values starting from 0 but we can also assign values manually.

Syntax of enum

```
enum { name1, name2, name3 = value };
```

Example of enum

```c
// An example program to demonstrate working
// of enum in C
#include <stdio.h>

enum week { Mon, Tue, Wed, Thur, Fri, Sat, Sun };

int main()
{
    enum week day;
    day = Wed;
    printf("%d", day);
    return 0;
}
```

Output

```
2
```

File Handling

C File Operations

We can perform some set of operations on a file and C language provide some functions for it.

1. Creating a new file – **fopen() with attributes as "a" or "a+" or "w" or "w+"**
2. Opening an existing file – **fopen()**
3. Reading from file – **fscanf() or fgets()**
4. Writing to a file – **fprintf() or fputs()**
5. Moving to a specific location in a file – **fseek()**, **rewind()**
6. Closing a file – **fclose()**

Preprocessor Directives

The preprocessor directives are used to provide instructions to the preprocessor that expands the code before compilation. They start with the # symbol.

The following table lists all the preprocessor directives in C/C++:

S.No.	Preprocessor Directives	Description
1.	#define	Used to define a macro
2.	#undef	Used to undefine a macro
3.	#include	Used to include a file in the source code program
4.	#ifdef	Used to include a section of code if a certain macro is defined by #define

S.No.	Preprocessor Directives	Description
6.	#ifndef	Used to include a section of code if a certain macro is not defined by #define
7.	#if	Check for the specified condition
8.	#else	Alternate code that executes when #if fails
9.	#pragma	This directive is a special purpose directive and is used to turn on or off some features.

Common Library Functions

C languages come bundled with some Standard Libraries that contain some useful functions to make it easier to perform some common operations. These are as follows:
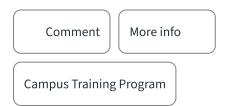
C Math Functions

The **<math.h>** header file contains functions to perform the arithmetic operations. The following table contains some common maths functions in C:

S.No.	Function Name	Function Description
1.	ceil(x)	Returns the smallest integer larger than or equal to x.

S.No.	Function Name	Function Description
3.	fabs(x)	Returns the absolute value of x.
4.	sqrt(x)	Returns the square root of x.
5.	cbrt(x)	Returns the cube root of x.
6.	pow(x , y)	Returns the value of x raised to the power y.
7.	exp(x)	Returns the value of e(Euler's Number) raised to the power x.
8.	fmod(x , y)	Returns the remainder of x divided by y.
9.	log(x)	Returns the natural logarithm of x.
10.	log10(x)	Returns the common logarithm of x.
11.	cos(x)	Returns the cosine of radian angle x.
12.	sin(x)	Returns the sine of radian angle x.
13.	tan(x)	Returns the tangent of radian angle x.

Conclusion

of the core principles of C. With its organized format, code examples, and key syntaxes, it serves as a valuable resource to refresh your knowledge and navigate through the intricacies of C programming. Keep this cheat sheet close by to accelerate your coding journey and streamline your C programming endeavors.

Comment   More info

Campus Training Program

Next Article

CCNA Cheatsheet

Similar Reads

1. GeeksforGeeks Master Sheet - List of all Cheat Sheets

2. Quiz Game in C

3. C Program to Store Information of Students Using Structure

4. Examination Management System in C

5. Non Verbal Reasoning : Paper Folding

6. Puzzle | Number of Sheets to be turned so that Prime Number has a Vowel on the other side

7. Google Sheets Tutorial with PDF 2025 - Everything you need to know

8. 200 Excel Interview Questions & Answers: Beginner to Expert (2025 Updated)

9. Excel: Modifying Columns, Rows, and Cells

10. Amazing Google Sheet Tips and Tricks

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate

K 061, Tower K, Gulshan Vivante
Apartment, Sector 137, Noida, Gautam
Buddh Nagar, Uttar Pradesh, 201305

Advertise with us

Company

About Us

Legal

Privacy Policy

Careers

In Media

Contact Us

Corporate Solution

Campus Training Program

Explore

Job-A-Thon

Offline Classroom Program

DSA in JAVA/C++

Master System Design

Master CP

Videos

Tutorials

Python

Java

C++

PHP

GoLang

SQL

R Language

Android

DSA

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

Python Projects | Computer Network
Python Tkinter | Database Management System
Web Scraping | Software Engineering
OpenCV Tutorial | Digital Logic Design
Python Interview Question | Engineering Maths

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

DevOps Roadmap

System Design

High Level Design

Low Level Design

UML Diagrams

Interview Guide

Design Patterns

OOAD

System Design Bootcamp

Interview Questions

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

Databases

SQL

MYSQL

PostgreSQL

PL/SQL

MongoDB

Preparation Corner

Company-Wise Recruitment Process

Aptitude Preparation

Puzzles

Company-Wise Preparation

More Tutorials

Software Development

Software Testing

Product Management

Project Management

Linux

Excel

All Cheat Sheets

Courses

IBM Certification Courses

DSA and Placements

Web Development

Data Science

Programming Languages

DevOps & Cloud

Programming Languages

C Programming with Data Structures

C++ Programming Course

Java Programming Course

Python Full Course

Clouds/Devops

DevOps Engineering

GATE 2026

GATE CS Rank Booster

GATE Rank Predictor

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy