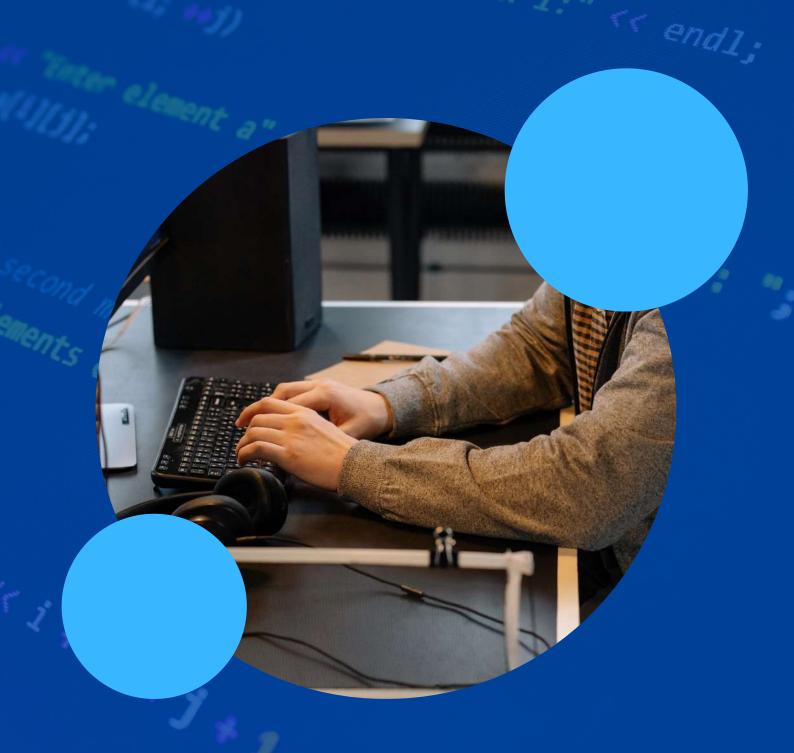# C PROGRAMMING
CHEATSHEET

For Quicker Revision & Faster Implementation

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
// C Hello World program
#include <stdio.h>

int main()
{
    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

data_type **variable_name** ;
data_type **variable_name** = *initial_value*
A variable can be of the following types:

1st Local Variable

2nd Global Variable

3rd Static Variable

4th Extern Variable

5th Auto Variable

6th Register Variable

Note: There are a few rules which we have to follow while naming a 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:

1st Basic Data Types

2nd Derived Data Types

3rd 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:

1st **char:** Used to represent characters.

2nd **int:** Used to represent integral numbers.

3rd **float:** Used to represent decimal numbers up to 6-7 precision

digits.

4th **double:** Used to represent decimal numbers up to 15 precision

digits.

5th **void:** Used to represent the valueless entity.

Example of Basic Data Types
```
char c = 'a';
int integer = 24;
float f = 24.32;
double d = 24.3435;
void v;
```

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

1st short

2nd long

3rd signed

4th 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:

1st Arrays

2nd 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:

1st Structure

Identifiers

Identifiers is the name given to the variables, functions, structure, etc. Identifiers should follow the following set of rules:

1st A variable name must only contain alphabets, digits, and

underscore.

   2nd A variable name must start with an alphabet or an underscore

only. It cannot start with a digit.

3rd No whitespace is allowed within the variable name.

4th 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()

printf("*formatted-string*", ...{*arguments-list*});
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
// C program to illustrate the basic input and output using
// 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);

 // printing output using printf
 printf("Name is %s and Roll Number is %d", name,
 roll_num);

 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.
%lf	Double
%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.
\v	Vertical Tab	It is used to insert vertical space.
\\	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.	&&, ||, !
5.	Conditional Operators	The conditional ? : Operator is used to insert conditional code.	
6.	Assignment Operators	They are used to assign some value to the variables.	=, +=, -=, <<=
7.	Miscellaneous Operators		comma, , sizeof, &, *, addressof,->, .

S.No.	Operator Type	Description	Example
		sizeof, etc. are some other types of operators.	

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

```
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

```
if   (expression) {
    // if block
}
else
{
    // 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;
case  value2:
      // 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
// 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
printf("i is not present");
}
```

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 checked after the body of the loop.

Syntax of do-while

```
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

goto label; | label:
. | .
. | .
. | .
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);
 printf("\n");

 // goto statement
 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

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
// C Program to demonstrate the use of array
#include <stdio.h>

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

char string_name[] = " *any_text*";

Example of Strings

```
// C program to illustrate strings

#include <stdio.h>
#include <string.h>

int main()
{
// 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.
5.	strchr()	Find the given character in the string.
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
ptr = &var;

 // assign the address of a variable to a pointer
 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:

1st Double Pointers

2nd Function Pointers

3rd Structure Pointers

4th NULL Pointers

5th Dangling Pointers

6th 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...*);
where,

    1st **Return Type:** It is the type of optional value returned by the

function. Only one value can be returned.

     2nd **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 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
int main()
{
// Calling sum function and
 // 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:

1stFunction with no return value and no parameters.

2ndFunction with no return value and parameters.

3rdFunction with return value and no parameters.

4thFunction with return value and parameters.

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

1stLibrary Functions

2ndUser-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()

The **malloc() function** allocates the block of a specific size in the memory. It returns the void pointer to the memory block. If the allocation is failed, it 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
// C program to illustrate the dynamic memory
allocation #include <stdio.h>
#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

To use structure, we first have to define its template.

struct *struct_name*
        member_type1 name1;
 member_type1 name1;

 .
 .
 }
 ;

Structure Variable Syntax

...{
 ...structure template... }
Or *var1, var2..., varN;*

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

Example of Structure

```
// 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];
```

```c
};

// declaring structure with name str2
struct str2 {
 int ii;
 char cc;
 float ff;
} var; // variable declaration with structure template

// 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
```

File Handling

File handling is the process of performing input and output on a file instead of the console. We can store, retrieve, and update data in a file. C supports text and binary files.

C File Operations

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

1stCreating a new file – **fopen() with attributes as "a" or "a+"** or "w" or "w+"

2ndOpening an existing file – **fopen()**

3rdReading from file – **fscanf() or fgets()**

4thWriting to a file – **fprintf() or fputs()**

5thMoving to a specific location in a file – **fseek(), rewind()**

6thClosing 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
5.	#endif	Used to mark the end of #endif
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

S.No.	Preprocessor Directives	Description
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 largest integer smaller than or equal to x.
2.	floor(x)	Returns the smallest integer larger than or equal to x.
3.	fabs(x)	Returns the absolute value of x.
4.	sqrt(x)	Returns the square root of x.

S.No.	Function Name	Function Description
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.

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.