Programming Cheat Sheet

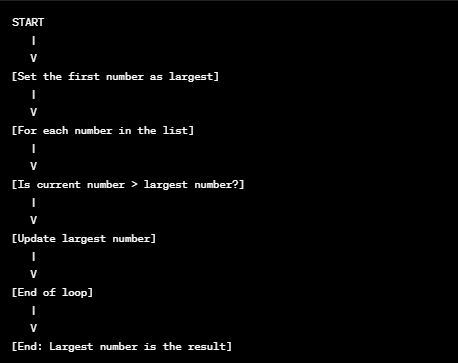
A project by Alliance Of Coders

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

**Algorithm**

An algorithm is a step-by-step set of instructions for solving a particular problem. Here's a simple algorithm to find the largest number in a list:  
  
Start:  
Set the first number in the list as the largest.  
For each number in the list:  
If the current number is greater than the largest number, update the largest number.  
End: The largest number is the result.

**Flowgorithm**

  
 Flowgorithm is a graphical way of representing algorithms. It uses shapes to denote different actions and decisions. Let's visualize the above algorithm in Flowgorithm:

Algorithm and Flowgorithm:

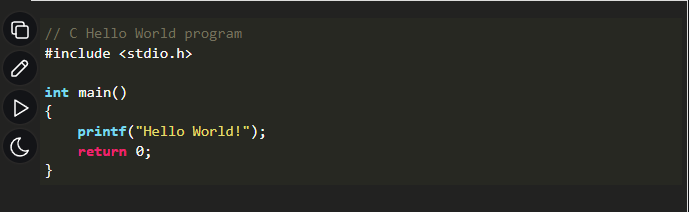
* Ovals: Start and end points.
* Rectangles: Actions or processes.
* Diamonds: Decisions or conditions.
* Arrows: Show process flow.

Simplified Explanation:  
  
Start: Begin.  
Set first number as largest: Assume it's the largest.  
For each number: Go through each.  
If current number > largest: Check if it's bigger.  
Update largest number: If yes, update.  
End of loop: Repeat for all.  
End: Largest number is the result.

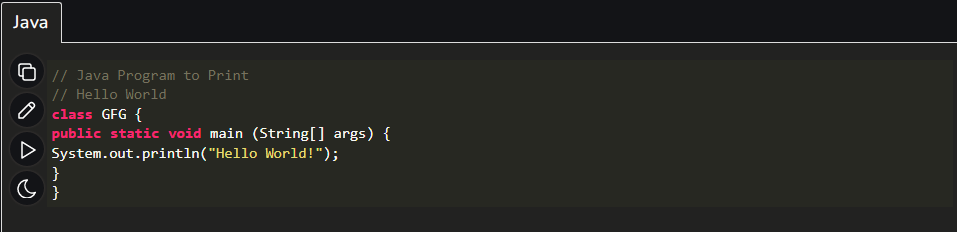
Essence:  
  
An algorithm is like a recipe.  
Flowgorithm visually structures each step.

We’re going to tackle up code sheets from evert language that we can provide. And put each sample definition and a little bit of code.

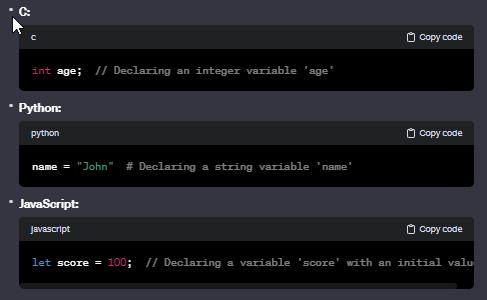
Basic Syntax:   
Consider the below Hello World program Using C Programming:



* #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 a storage spot in a program with a name (identifier) holding a value. It can change during program execution. You declare it before use, specifying its name and data type. Examples in different languages:

**Basic Data Types**

**Integer (int)**: Whole numbers without decimals, like 1, 42, or -10.  
**Float/Double (float, double)**: Numbers with decimals, e.g., 3.14 or -0.5.  
**String**: Character sequences, like "hello" and "world."  
**Boolean**: Represents true or false values.  
**Character (char)**: Single characters, such as "a" or "1."  
**Array**: A group of elements with the same data type.  
**Object/Dictionary**: Depending on the language, represents key-value pairs.  
**Null/Undefined**: Indicates the absence of a value.

Examples of Variable Declaration:

int num = 10;

float pi = 3.14;

char letter = 'A';

String happiness = “Uyab”;

int scores[4] = { 10, 14, 12, 17 }; // integer array

String motorcycles[3] = { “NMAX”, “XRM”, “Smash” }; // string array

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

1. short – range is from -32,768 to 32,767
2. long – range is from -2,147,483,648 to 2,147,483,647
3. signed – you can store both positive and negative values
4. unsigned – you cannot store negative values

Aside from basic data types, there are also the **derived data types** and **user-defined data types**.

* Derived Data Types

Arrays: Think of it like a row of boxes, each holding one item.  
Pointers: These are like variables that remember where to find other data.

* User-Defined Data Types

Structure: It is a way to group several types of data under one name.  
Union: Like a structure but holds only one type of data at a time.  
Enumeration: Used to create a list of named items with corresponding numbers.

**Identifiers**

Identifiers are the names given to the variables, functions, structure, etc. Names for elements in code, following rules:

* Use letters, digits, and underscores.
* Must start with a letter or underscore.
* No spaces allowed.
* Cannot be the same as reserved words or keywords in the programming language.

**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 Sequences**

Escape sequences represent special characters using a backslash ( \ ) within string literals.  
  
Common Escape Sequences:  
  
\n: Newline  
\t: Tab  
\": Double quote  
\': Single quote  
\\: Backslash  
In short, escape sequences handle special characters in strings.

**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:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Operator Type** | **Description** | **Example** |
| **1.** | **Arithmetic Operators** | Operators that perform arithmetic operations. | +, -, \*, /, % |
| **2.** | **Relational Operators** | They are used to compare two values. | <, >, <=, >=, ==, != |
| **3.** | **Logical Operators** | They perform logical operations such as logical AND, logical OR, etc. | &&, ||, ! |
| **4.** | **Conditional Operators** | The conditional Operator is used to insert conditional code. | ? : |
| **5.** | **Assignment Operators** | They are used to assign some value to the variables. | =, +=, -=, <<= |

**Conditional Statements**

Conditional statements execute code based on whether a given condition is true. In any programming language, we have:

**If Statement**: Executes code if the condition is true.

Syntax:

**if** (condition) {

// *statements*

}

**Else Statement**: Executes alternative code if the condition in the if statement is false.

Syntax:

**if** (expression) {

*// if block*

}

**else** {

*// else block*

}

**If Else Ladder**: Checks additional conditions if the previous ones are false.

Syntax:

**if** (expression) {

*// if block*

} **else** **if** (expression) {

*// else if block*

}

.

.

.

**else** {

*// else block*

}

**Nested If Else Statement**: Checks additional conditions inside a condition.

Syntax:

**if** (expression) {

*// if block*

**if** (expression) {

*// if block*

}

}

**else** {

*// else block*

}

**Switch Case Statement**: An alternative to if-else-ladder that executes different blocks of statement.

Syntax:

**switch** (expression) {

**case** value1:

*// statements*

break;

**case** value2:

*// statements*

break;

.

.

.

**default:**

*// default block*

break;

}

**Conditional Operator**: Tests the condition and executes true and false statements in a single line.

Syntax:

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

These statements help control the flow of code based on conditions.

**Loops**

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

**for loop**: Executes a block of code a specific number of times.

Syntax:

**for** (initialization; condition; update) {

// Code to repeat

}

Example:

**for** (int i = 0; i < 5; i++) {

// Code to repeat five times

}

**while loop:** Repeats a block of code if a specified condition is true.

Syntax:

**while** (condition) {

// Code to repeat

}

Ex.

int count = 0;

**while** (count < 3) {

// Code to repeat three times

count++;

}

**do while**: An exit-controlled loop in which the condition is checked after the body of the loop.

Syntax:

**do** {

*// statements*

} **while** (condition);

Example:

int count = 0;

**do** {

count+=1;

} **while** (count < 5);

Loops are essential for automating repetitive tasks and iterating through data structures in programming.

## **Jump Statements**

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

### 1. **break** Statement

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

Syntax:

**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:

**continue;**

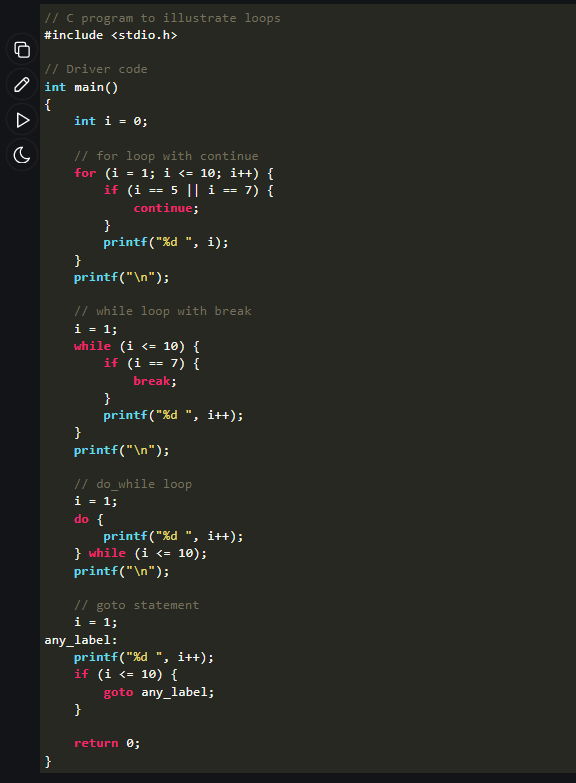
### 3. **goto** Statement

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

Syntax:

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

Look at this C Program using Jump statements.



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:

*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:

String mgaUyab[5] = { “Joy”, “Michaella”, “Diana”, “Mae”, “Rose” } // 1D array

int arr[5][3] = { 1, 2, 3, 4, 5 }, { 2, 4, 6 };

char crushInitials[2][4][3] = { ‘A’, ‘C’ }, { ‘Q’, ‘E’, ‘R’, ‘J’ }, { ‘T’, ‘R’, ‘I’ };

**String Functions**

C language provides some useful functions for string manipulation in **<string.h>** header file. However, in Java, declaring a library is unnecessary. You can outright used the functions without a library in the header file. The following are one of these functions:

|  |  |  |
| --- | --- | --- |
| **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. |

### **C Math Functions**

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

|  |  |  |
| --- | --- | --- |
| **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)**](https://www.geeksforgeeks.org/fabs-function-in-c/) | 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. |

## **Pointers**

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

Syntax:

*data\_type* ***\* ptr\_name;***

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

Example:

int var = 10;

int \*ptr;

ptr = &var;

*Note: Whatever the value of* ***var*** *variable will be the value of the pointer* ***ptr****.*

## **Functions**

Functions are the block of statements enclosed within **{ }** **braces** that perform some specific tasks. 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,

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:

**int** sum(**int** a, **int** b) { **return** a + b; }

### **Types of Function**

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

* Function with no return value and no parameters.
* Function with no return value and parameters.
* Function with return value and no parameters.
* Function with return value and parameters.

**User-Defined Data Types**

## **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 must define its template.

**struct** *struct\_name* {  
 *member\_type1 name1;*  
 *member\_type1 name1;*  
 .  
 .  
};

### Syntax:

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

or

**struct** *str\_name* var1, var2,...varN;

Example:

**struct** str1 { //declaring structure

**int** i;

**char** c;

**float** f;

**char** s[30];

}var; //variable declaration with structure template

**struct** str1 var1 = { 1, 'A', 1.00, "Sanaol May Jowa" },

## **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 member.

### Syntax:

**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:

**union** un {

**int** member1;

**char** member2;

**float** member3;

};

**union** un var1;

var1.member1 = 31; // the value of **member1** will be 31

## **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:

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

Example:

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

**enum** week day; // defining variable

day = Wed; // **day** will be **2** since Mon is **0** and Tue is **1**