**Basics of C for Embedded Systems**

C is a general-purpose programming language created by Dennis Ritchie

**Difference between C and C++**

* [C++](https://cn.w3schools.com/cpp/default.asp) was developed as an extension of C, and both languages have almost the same syntax
* The main difference between C and C++ is that C++ supports classes and objects, while C does not.

**Syntex:**

#include <stdio.h>   
int main() {  
  printf("Hello World!");  
  return 0;  
}

\*\*return 0 ends the main() function.

\*\*#include <stdio.h> -> **header file library** that lets us work with input and output functions, such as printf()

\*\* printf()- f is formate specifier

**New Lines: \n** is called escape sequence,

**printf("Hello World!\n");**

**\*\*** **You can also output multiple lines with a single printf() - > printf("Hello World!\nI am learning C.\nAnd it is awesome!");**

**\*\*** [**\\n**](file:///\\n) **– create two line space ->**

printf("Hello World!\n\n");  
  printf("I am learning C.");

**OUTPUT:**

Hello World!  
  
I am learning C.

**\*\* \t it create horizontal gap**

printf("Hello World!\t");

printf("I am learning C.");

**output:**

Hello World!    I am learning C.

**\*\* Addidng special character**

|  |  |
| --- | --- |
| **\** **\** | **Inserts a backslash character (\)- print \** |

**Code:**

**int main() {**

printf("Hello World!\@");

printf("I am learning C.");

return 0;

**}**

**Print**

hello world \I am learning C

**C format specifier:**

Int -  format specifier %d + double quotes ("") = "%d"

int myNum = 15;  
printf("%d", myNum);

int – “%d” or “%i”

char – “%c”

float- “%f” or “%F”

string – “%s”

double – “ %1f”

 read **memory addresses** (pointers).- %p in hex format

char greetings[] = "Hello World!";

printf("%s", greetings);

**\*\* To print different types in a single printf() function**

int myNum = 15;  
char myLetter = 'D';  
printf("My number is %d and my letter is %c", myNum, myLetter);

output;

My number is 15 and my letter is D

**\*\* Change Variable Values**

int myNum = 15;  // myNum is 15  
myNum = 10;  // Now myNum is 10

int myNum = 15;  
int myOtherNum = 23;  
myNum = myOtherNum;

**Rules:**

* **Names can contain letters, digits and underscores**
* **Names must begin with a letter or an underscore (\_)**
* **Names are case-sensitive (myVar and myvar are different variables)**
* **Names cannot contain whitespaces or special characters like !, #, %, etc.**
* **Reserved words (such as int) cannot be used as names**

**Float vs double :**

**Float- 6-7 digit after point – 6.444333**

**Double – 15 digit after point**

If you want to remove the extra zeros (set decimal precision), you can use a dot (.1)

float myFloatNum = 3.5;  
  
printf("%f\n", myFloatNum);   // Default will show 6 digits after the decimal point  
printf("%.1f\n", myFloatNum); // Only show 1 digit  
printf("%.2f\n", myFloatNum); // Only show 2 digits  
printf("%.4f", myFloatNum);   // Only show 4 digits

**Memory Size of data type :**

* **Use sizeof() operator**

int myInt;  
float myFloat;  
double myDouble;  
char myChar;  
  
printf("%lu\n", sizeof(myInt)); // 4  
printf("%lu\n", sizeof(myFloat)); // 4  
printf("%lu\n", sizeof(myDouble)); //8  
printf("%lu\n", sizeof(myChar)); //1

**why we use %1u( long unsigned int) instead of %d or normal format specifier ?**

ans : becoz we are using sizeof() so it return long unsinged int

* It give positive number only
* It has larger range and store larger positive values
* %1u use everywhere doesnot matter int ,string ,char

1 bytes – 8 bit

Int – 2 or 4 bytes

Float – 4 bytes

Double – 8 bytes

Char – 1 byte

**Type conversion :** convert one data type to another data type

Implicit conversion – automatically

Explicit conversion - manually

Implicit conversion : it is done automatically by compiler when value assign .

// float m = 5;

// printf( "%f", m);

// return 0;

int n = 9.99;

printf("%d",n);

return 0;

float sum = 5 / 2;  
  
printf("%f", sum); // 2.000000

acutal ans is 2.5 why is the result ? ans : 5/2 is considered as interger , you need to conver integer to float use manual way is called Explicit conversion

Explicit Conversion : Explicit conversion is done manually by placing the type in parentheses () in front of the value.

float n = (float)5/2;

printf("%f",n);

return 0;

constant : unchangeable and real only

#include<stdio.h>

int main(){

const int m = 5;

m = 3;

printf("%i",m);

return 0;

}

Output: A screen shot of a computer error

AI-generated content may be incorrect.

Note : When you declare a constant variable, it must be assigned with a value.

const int minutesPerHour;  
minutesPerHour = 60; // error

const int minutesPerHour = 60;

Boolean variable :

Import header file for Boolean : #include <stdbool.h>

#include<stdio.h>

#include<stdbool.h>

int main(){

bool m = true;

printf("%i",m);

return 0;

}

Ternary operator :

(*condition*) ? *expressionTrue* : *expressionFalse*;

Code:

#include<stdio.h>

int main(){

int age =20;

( age>18)? printf("you are eligible for collage"): printf("you are not eligible for collage ");

}

**Switch condition :**

switch (*expression*) {  
  case x:  
    *// code block*  
    break;  
  case y:  
    *// code block*  
    break;  
  default:  
    *// code block*  
}

**loop :**

**while loop** : for infinite looping whenever condition is not true .

intilization

while (*condition*) {  
*// code block to be executed*

*increment++;*  
}

code:

int i = 0;  
  
while (i < 5) {  
  printf("%d\n", i);  
  i++;  
}

Do not forget to increase the variable used in the condition (i++), otherwise the loop will never end!

**Do/while loop**: The do/while loop executes the code block at least once before checking the condition

Intilization ;

do {  
*// code block to be executed*

*i++;*}  
while (*condition*);

**for loop** : finite time

for( intiilization ; termination condition ; increment ){

// body

}

**Break and continue**

The break statement can also be used to jump out of a loop.

Code: int i;  
  
for (i = 0; i < 10; i++) {  
  if (i == 4) {  
    break;  
  }  
  printf("%d\n", i);  
}

Output: 0 1 2 3

**Continue**

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Code:

int i;  
for (i = 0; i < 10; i++) {  
  if (i == 4) {  
    continue;  
  }  
  printf("%d\n", i);  
}

**Array**

int myNumbers[] = {25, 50, 75, 100}; // c/c++/ java same

it is heterogeneous array

**2d array – c/c++/ java same**

int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

**Array size / length**

* Use **sizeof()** operator
* sizeof operator returns the size of a data type in **bytes**. int type is usually 4 bytes ,  4 x 5 (*4 bytes x 5 elements*) = **20 bytes**.

int myNumbers[] = {10, 25, 50, 75, 100};  
int length = **sizeof(myNumbers) / sizeof(myNumbers[0])**;  
  
printf("%d", length);  // Prints 5

forloop-

#include<stdio.h>

int main(){

        int arr[] = { 4 ,3,5,6,6,6};

      int length = sizeof(arr) / sizeof(arr[0]);

      for(int i=0; i<length ; i++){

          printf("%d\n" , arr[i]);

      }

  return 0;}

**Multidimensional array :**

Specified matrix[2][3] in c but in java matrix[][] it is totally blank

Java :

Int matrix[][] = { 1,2,3,4,5};

C:

int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

code:

int main()

{

int arr[3][3] = {

{1,2,3 } ,

{ 4,5,6} ,

{7,8,9}

};

printf("%d" , arr[0][2]);

return 0;

}

**Stirng :**

- C does not have a **String type** to easily create string variables. Instead, you must use the char type and create an [array](https://www.w3schools.com/c/c_arrays.php) of characters to make a string in C:

- Note that you have to use double quotes ("").

-To output the string, you can use the printf() function together with the format specifier %s to tell C that we are now working with strings:

char greetings[] = "Hello World!";

code: - %s- full string show in output

#include <stdio.h>

int main()

{

char arr[] = "hello world";

printf("%s" , arr);

return 0;

}

Another way creating string

char arr[] = {

'h', 'e','l','l','l','o', ' ', 'w','a','d',’\0’

};

Note- **Why do we include the \0 character at the end?** This is known as the "null terminating character", and must be included when creating strings using this method. It tells C that this is the end of the string.

Access Strings: use %c only **single**character show in output

int main()

{

char arr[] = "hello world";

printf("%c" , arr[0]);

return 0;

}

**Modify Strings**

It modify some character of string but it print the whole string so we use %s

int main()

{

char arr[] = "hello world";

arr[0]='j';

printf("%s" , arr);

return 0;

}

Output: jello world

Note: use strlen() for length of string but use library <string.h> for tabhi kaam karega

Code:

#include <stdio.h>

#include <string.h>

int main()

{

char arr[] = "hello world";

// int length = sizeof(arr)/ sizeof(arr[0]);

int length = strlen(arr);

for(int i=0;i<length;i++){

printf("%c \n" , arr[i]);

}

return 0;

}

**User input :** **scanf() use for argument**

& - this operator is imp you want store data in variable so add operator with variable

Eg, scanf("%d", &myNum);

Code:

int main()

{

printf("sum calculator \n");

int a ;

int b ;

scanf("%d %d" , &a , &b);

int result = a +b ;

printf("result:%d and value of %d and %d" , result , a,b);

return 0;}

**Take String Input**

**Note-When working with strings in scanf(), you must specify the size of the string/array**

**char arr[50] ;**

**char arr2[50];**

**printf("enter you name ");**

**scanf("%s ", &arr);**

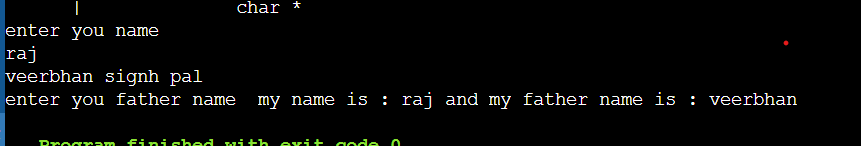
**printf("enter you father name ");**

**scanf("%s \n", &arr2);**

**printf(" my name is : %s and my father name is : %s ",arr , arr2);**

**notes :**

**scanf() function has some limitations:**

****

Notes  scanf() function to get a single word as input, and use fgets() for multiple words.

Fgets() function :

// Fgets( variable , sizeof() , stdin);

char arr[50] ;

printf("enter you name ");

fgets( arr , sizeof(arr) , stdin);

printf(" my name is : %s ",arr);



pointer : The memory address is the location of where the variable is stored on the computer.

When we assign a value to the variable, it is stored in this memory address.

&- reference operator

Memory address : “ %p “ , &var

Code: int age = 46;

printf("%p", &age); // 0x7ffx8ab

printf("%d", age); // 46

return 0;

code:

int age = 46;

int \*ptr = &age;

printf("%p \n", &age); // 0x7ffx8ab

printf("%p" , ptr); // 0x7ffx8ab

**Pointers & Memory Management**

**Good To Know:** There are two ways to declare pointer variables in C:

int\* myNum;  
int \*myNum;

POINTER : A **pointer**is a variable that stores the **memory address** of another variable.  it holds the address where the value is stored in memory.

* **Dereferencing operator**(\*) used to declare pointer variable and access the value stored in the address.
* **Address operator(&)**used to returns the address of a variable or to access the address of a variable to a pointer.

#include <stdio.h>

int main() {

int num = 10; // A normal integer variable

int \*ptr = &num; // A pointer that stores the address of 'num'

printf("Value of num: %d\n", num); // Output: 10

printf("Address of num: %p\n", &num); // Output: Memory address (e.g., 0x7ffd57d5b5fc)

printf("Value stored in ptr: %p\n", ptr); // Same as &num

printf("Value pointed by ptr: %d\n", \*ptr); // Output: 10 (dereferencing)

return 0;

}

* **%p format specifier** is used to print the address stored in pointer variables.
* **Printing a pointer with %d format specifier** may result in a warning or undefined behaviour because the size of a pointer (usually 4 or 8 bytes) may not match that of an integer.
* The memory address format will always be in **hexadecimal format**(starting with 0x).
* C does not use the term “**reference**” explicitly (**unlike C++**), “**referencing**” in C usually refers to obtaining the address of a variable using the **address operator (&)**.
* Pointers are essential for **dynamic memory allocatio**n, providing control over memory usage with functions like **malloc**, **calloc**, and **free.**

**Pointer declaration: \*ptr**

**Pointer initialization: &ptr- addressof operator**

A diagram of a number

AI-generated content may be incorrect.

* ( Arrow Operator): Used with pointers to **structs/classes** to access members.

**Pointer Arithmetic :**

**Increment/Decrement (++, --):**

****

**Addition/Subtraction of an Integer (ptr + n, ptr - n)**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Subtracting Two Pointers (ptr1 - ptr2)**

**A white background with black text

AI-generated content may be incorrect.**

**Comparing Two Pointers (ptr1 == ptr2, ptr1 > ptr2)**

**A screenshot of a computer code

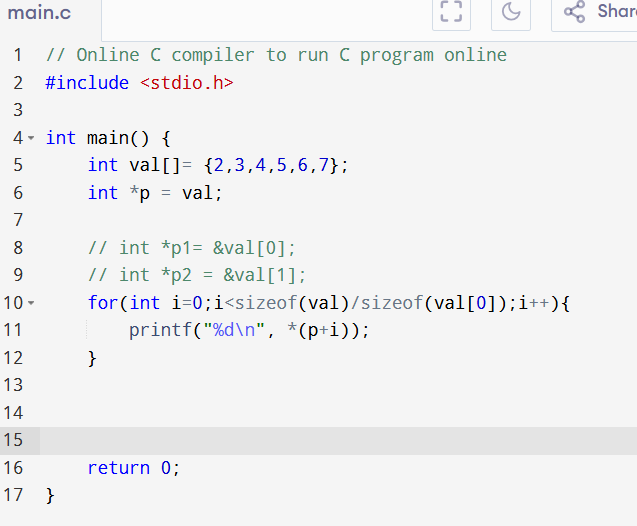
AI-generated content may be incorrect.**

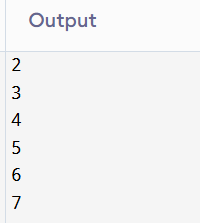
**Pointers & Arrays**

Pointers and arrays are closely related in C. Arrays internally work as pointers, and pointer arithmetic helps in efficient memory handling, making them crucial in embedded systems for buffer management, hardware register access, and low-level programming.

A screenshot of a computer program

AI-generated content may be incorrect.

****

****

**Arrays as Function Arguments (Passing by Reference)**

**A screenshot of a computer program

AI-generated content may be incorrect.**