1. HOME

C programming language is developed by Dennis Ritchie at the Bell Laboratories. it is machine-independent, structured programming language. C is one of the most widely used computer languages.

C programming languageis considered as the base for other programming language, that is why it is also known as mother language. If you know C you can easily grasp the knowledge of the other programming language that uses the concept of C.

**Why learn C?**

As we studied earlier C is a base language for many programming languages. So learning C as the main language will help to learn other languages easily. Some of the key advantages of learning C Programming are as follow

* C is a simple language, it is easy to learn.
* It provide faster execution.
* It is machine independent.
* It is Structured programming language which makes it easy for testing, maintaining and debugging.

**Applications of C :**

1. ‘C’ language is used for compiler productions.
2. It is used for Assemblers.
3. It is widely used in embedded system.
4. It is used for developing desktop applications.
5. It is used for developing system applications.
6. It is used for developing databases.
7. It is used in IOT applications.

**First Programme in C :**

A quick look at the example of Hello, World! In C programming is given below with output.

**Code :**

#include <stdio.h>

int main() {

//this will print “Hello world” on screen

printf(“Hello World! ”);

return 0;

}

**Output:**

Hello World!

**Prerequisites:**

Before learning C Programming, you must have the basic knowledge of Computer Fundamental.

**Audience:**

This C tutorial series has been designed for those who want to learn C programming. whether you are beginners or experts, this tutorials will help you.

Overview of C Language

C is structured, general-purpose programming language developed by Dennis Ritchie in 1973 at Bell Laboratories. It was developed to write the UNIX operating system, hence it is strongly associated with UNIX, which is one of the most popular network operating system in use today. C is one of the most popular computer languages today because of it structure, high-level abstraction, machine independent,machine efficient feature.

**Features of C Programming Language**

* It is structured programming language hence it is very easy to learn for programmer.
* It is a procedure oriented language (i.e. in this procedure or method or function are given more significance) .
* C programming language has become popular because of its major features like reliable, simple and easy to use.
* It is robust language with set of large built-in functions and operators that can be used to write any complex program.
* Programs Written in C are efficient and fast.
* It is Easy to extend.

C - Environment Setup

In this chapter, we are going to learn how to setup your C language Environment. To setup C language Environment there are two ways:

1. Download a IDE (Integrated Development Environment) like Code::Blocks or Dev C++ , which come along with a code editor, compiler, debugger etc.
2. Or, you can use any text editor to edit the program files like Notepad or vim and download the C compiler separately.

* **What Is C Compiler ?**
* C Compiler is a program that converts human readable code into machine readable code. The process of converting human readable code to machine readable code is called compilation.
* Human readable code is a program that consists of letters, digits and special characters that we type in program window. Machine readable code is in 0’s and 1’s form.
* So, compiler converts source code into 0’s and 1’s during compilation.
* Output produced by compiler is in the form of 0’s and 1’s which is saved in .exe file. This file is called as executable or binary file.
* This binary file is executed by processor as per logic written in source code and output is displayed in output window.
* **C Compiler Installation on Windows**

To use C compiler in Windows, you can install any one software mentioned below.

1. You can download [Code::Blocks](http://www.codeblocks.org/downloads) IDE to develop C application.
2. You can also download [Dev-C++](http://www.bloodshed.net/dev/devcpp.html) IDE.
3. You Can install [MinGW](http://www.mingw.org/) compiler.

* **C Compiler Installation on UNIX/Linux**

If you use are using Linux or UNIX, then check whether GCC is installed on your system or not by entering the command line -

*$gcc -v* : if you have GNU compiler installed on your machine, then it should print a message as follows -

*Using built-in specs. Target: i386-redhat-linux Configured with: ../configure --prefix=/usr ....... Thread model: posix gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)*

if GCC is not installed, then you will have to install it yourself using the detailed instructions available at [GCC GNU Website](http://gcc.gnu.org/install/).

* **C Compiler Installation on Windows**

If you use Mac OS X, the easiest way to obtain GCC is to download the [Xcode](developer.apple.com/technologies/tools/) development environment from Apple's web site and follow the simple installation instructions. Once you have Xcode setup, you will be able to use GNU compiler for C/C++.

C - Program Structure

Before starting with building blocks of C programming lets look at simple C program and its structure.

**Example :**

The Simple C program consist of following parts -

* Preprocessor Commands
* Functions
* Variables
* Statements
* Comments

**Code:**

*#include <stdio.h> //preprocessor commands*

*int main() //main functions*

*{*

*printf(“Hello, World! \n”);*

*return 0;*

*}*

* First line of Code is *#include<stdio.h>* which is preprocessor commands it tells the C compiler to include stdio.h file, all statements which start with***#*** symbol are preprocessor commands.
* *In second line of Code we are defining the functions whose name is “main” and return type is int (integer) .where from programme execution would begin.*
* ***printf()*** *is predefined function in c language which print the text on screen.*
* *return 0;* this line terminate the programme and return integer value 0.
* **//** it is single line comment in C programming.

C - Tokens

* One of very important thing required for programming is its Tokens.
* The C programming made up of different things termied as tokens.
* The different tokens of C are following -
* Character set
* Keywords
* Identifiers
* Constants and variables
* Data types
* Operator

**Character Set of C :**

* The character set of any programming language indicates the different characters the programming can contain. Character set include all the alphabets ,digits and special symbols supported by the processor.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Characters** | **List included** |
| 1. | Alphabets | A, B, C …….Z  a, b, c ……..z |
| 2. | Digits | 0,1,2,………9 |
| 3. | Special symbols | <>{}()[],.;:!?’”/+\*=%&#@|\-`$^\_- |
| 4. | Other special symbol | Blank Space, Tab, Carriage Return |

**Keywords :**

* There are some special words that have a predefined meaning for the C compiler. Hence, these words cannot be used as identifier.
* These are a set of words which are reserved for the certain operations and hence are also sometimes referred as reserved words.
* All keywords are in lower case.
* The keywords used in C are as following -

|  |  |  |  |
| --- | --- | --- | --- |
| auto | else | long | switch |
| break | enum | register | typedef |
| case | extern | return | union |
| char | float | short | unsigned |
| const | for | signed | void |
| continue | goto | sizeof | volatile |
| default | if | static | while |
| do | int | struct | \_Packed |
| double |  |  |  |

**Identifiers :**

Identifiers are names given to different user defined things like variables, constants, functions, classes, objects, structures, unions, etc. While making these identifiers we need to follow some rule. These rules are as following -

1. The identifier can consist of alphabets, digit and special symbol i.e. ‘\_’ (underscore).
2. An identifier can not start with digit. It can start with an alphabet or underscore.
3. It can’t contain any special symbol except underscore .
4. It is case sensitive.

C - Data Types

The data type decides the type of data and the memory locations required for storing that type of data in the memory.

**Types :**

1. Primitive
2. Derived
3. User Defined data type

* The Primitive data type further divided into integer, character, fraction type numbers, etc.
* Different type of primitive data type and memory required to store them are as follow -

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Data type** | **Range** | **Required memory space (bytes)** |
| 1. | char | -127 to 128 | 1 |
| 2. | signed char | -127 to 128 | 1 |
| 3. | unsigned char | 0 to 255 | 1 |
| 4. | int | -32,768 to 32,767  Or  -2,147,483,648 to 2,147,483,647 | 2 or 4 |
| 5. | unsigned int | 0 to 65,535  Or  0 to 4,294,967,295 | 2 or 4 |
| 6. | short int | -32,768 to 32,767 | 2 |
| 7. | long | -9223372036854775808 to 9223372036854775807 | 8 |
| 8. | unsigned long | 0 to 18446744073709551615 | 8 |
| 9. | float | 1.2E-38 to 3.4E+38 | 4 |
| 10. | double | 2.3E-308 to 1.7E+308 | 8 |
| 11. | long double | 3.4E-4932 to 1.1E+4932 | 10 |

* The user defined data types are structure, union, class and enumeration.
* Derived data types are array, function, pointer, and reference.

C - Constants

* Constants are value given to the identifiers that do not change their value throughout the execution of the program.
* Constant can be defined either by writing the keyword *const* before the data type or using *#define*.
* Constant are used to declare the values that remain constant throughout the execution of the program, for e.g. value of pi.

**Example :**

**Code :**

#include <stdio.h>

#define Pi 3.14

int main() {

const int radius = 3;

float area = Pi\*radius\*radius;

printf("Area of Circle = %f",area);

return 0;

}

**Output:**

Area of Circle = 28.260000

C - Variables

Variables are values given to identifier that can change their values during the execution of the program.

**Variable Declaration :**

It tells the compiler where and how much storage create for the variable.

**Syntax for declaration :**

data\_type variable\_name\_list ;

Here, data\_type must be valid C-Data Type (char, int, long, float, double, etc.) or any user define Data Type and variable\_name\_list it may consist of one or more identifier separated by commas .

**Example :**

float height;

int a, b;

**Syntax for declaration and initialization :**

data\_type variable\_name\_list = value ;

**Example :**

float height = 5.6;

int a = 97, b = 98;

C - Escape Sequence

* Escape Sequence are used to perform some special operation like going to new line, providing horizontal tab, providing single or double quote etc.
* Escape sequence is a character followed by a backslash ( \ ) .
* List of Escape Sequence is given below -

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Symbol** | **Description** |
| 1. | \n | Newline |
| 2. | \t | Horizontal Tab |
| 3. | \v | Vertical Tab |
| 4. | \b | Backspace |
| 5. | \r | Carriage Return |
| 6. | \f | Form feed |
| 7. | \a | Audible Alert |
| 8. | \\ | Backslash |
| 9. | \? | Question mark |
| 10. | \’ | Single quote |
| 11. | \” | Double quote |

**Example :**

**Code :**

#include <stdio.h>

int main() {

printf("Title:\n This is example program to show use of \"Escape Sequence\" ");

return 0;

}

**Output:**

Title:

 This is example program to show use of "Escape Sequence"

C - Storage Classes

Storage classes are used to describe scope, visibility and life-time of a variable/function. Which help us to trace the existence of a variable during the runtime of a program.

**C language has 4 Storage Classes :**

* auto
* register
* static
* extern

1. **Automatic Storage Class (auto) :** All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned. Variables having automatic storage class are local to the block which they are defined in, and get destroyed on exit from the block.

Eg. int a = 1; OR auto int a = 1;

1. **Register Storage Class (register) :** variable declare with register are very similar to auto storage class except that in this variable store in register (if a free register is available) for faster access. Address of this type of variable can’t be access .

Eg. register int a = 1;

1. **Static Storage Class (static) :** The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

Eg. static int a = 1;

1. **External Storage Class (extern) :** Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

Eg. extern int a;

**Code :**

*#include <stdio.h>*

*static int c = 3; //static class*

*extern int e; //extern class*

*int main()*

*{*

*int a = 1; // by default auto*

*auto int b = 2; //auto externally defined*

*register int d = 4; //register class*

*printf("%d %d %d %d %d",a,b,c,d,e);*

*return 0;*

*}*

*int e = 5;*

**Output :**

1 2 3 4 5

C - Operator

An operator is a symbol that operates on a value or a variable. The data on which the operation is performed are called as operands. If an operator requires one operand, it is called as Unary operator. If an operator requires two operands, then it is called as Binary operator and if it require three operands, it is called as Ternary operator.

**C has following built in operator types -**

1. **Arithmetic Operators :** This set includes the basic arithmetic operators to perform basic arithmetic operations like addition, subtraction, multiplication and division.

Suppose variable A = 10 and variable B = 15.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | To find the sum | A + B = 25 |
| - | To find the difference | A - B = -5 |
| \* | To find the product | A \* B = 150 |
| / | To find quotient after division | B / A = 1 |
| % | To find remainder after division | B % A = 5 |

1. **Relational Operators :** The relational operators are used to test the relation between two variables or a variable and constant.

Suppose variable A = 10 and variable B = 15.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Used to check if two things are equal | (A == B) is false |
| != | Used to check if two things are not equal | (A != B) is true |
| < | Used to check if first data is less than second one | (A < B) is true |
| > | Used to check if first data is greater than second one | (A > B) is false |
| <= | Used to check if first data is less than or equal to second one | (A <= B) is true |
| >= | Used to check if first data is greater than or equal to second one | (A >= B) is false |

1. **Logical Operators :** Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a boolean value either true or false.

Suppose variable A = true and variable B = false.

(**Note :** any non zero value is treated as true and zero is treated as false.)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical And Operator, if both operands are true then true else false. | (A && B) is false |
| || | Logical Or Operator, if any of two operands is true then true else false. | (A || B) is true |
| ! | Logical Not Operator, Used to reverse the operand. | ( !A ) is false |

1. **Bi[twise Operators](https://www.geeksforgeeks.org/interesting-facts-bitwise-operators-c/)** : The Bitwise operators is used to perform bit-level operations on the operands.

Suppose variable A = 10 (i.e. 1010 in binary) and variable B=15 (i.e. 1111 in binary).

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | To perform bitwise And operation. | A & B = 10 (i.e. 1010 in binary) |
| | | To perform bitwise Or operation. | A | B = 15 (i.e. 1111 in binary) |
| ^ | To perform bitwise EXOR operation. | A ^ B = 5 (i.e. 0101 in binary) |
| ~ | To perform bitwise Not operation. | ~A = -11 (i.e. -1011 in binary) |
| << | To perform bitwise left shift operation. | A << 1 = 20 (i.e. 10100 in binary) |
| >> | To perform bitwise right shift operation. | A >> 1 = 5 (i.e. 0101 in binary) |

1. **Assignment Operators :** These operators are used to assign the value of the expression or variable on the right of the assignment operator to the variable on its left.

Suppose variable A = 10 , variable B = 15 and variable C = 0.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | This operator assigns the value of the expression or variable on its right to the variable on its left. | C=A will assign value of A to C |
| += | This operator adds the variable on its left and right and the result is put into the variable on its left. | B += A is same as B = B + A |
| -= | This operator subtracts the variable on its right from variable on its left and result is put into variable on its left. | B -= A is same as B = B - A |
| \*= | This operator multiplies the variable on its left and right and the result is put into the variable on its left. | B \*= A is same as B = B \* A |
| /= | This operator divides the variable on its right from the variable on its left and the result is put into the variable on its left. | B /= A is same as B = B / A |
| %= | This operator finds the remainder by dividing the variable on its right from the variable on its left and the result is put into the variable on its left. | B %= A is same as B = B % A |
| &= | This operator ANDs the variable on its left and right and the result is put into the variable on its left. | B &= A is same as B = B & A |
| |= | This operator ORs the variable on its left and right and the result is put into the variable on its left. | B |= A is same as B = B | A |
| ^= | This operator EXORs the variable on its left and right and the result is put into the variable on its left. | B ^= A is same as B = B ^ A |
| <<= | This operator shifts in left direction the variable on its left for the number of times indicated by the variable or value on right and the result is put into the variable on its left. | B <<= A is same as B = B << A |
| >>= | This operator shifts in right direction the variable on its left for the number of times indicated by the variable or value on right and the result is put into the variable on its left. | B >>= A is same as B = B >> A |

1. **Other Operators :** Apart from above operator there are some other operator is available in C as following -

Suppose variable a = 10 (a is integer).

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| sizeof() | Used to compute size of variable. | sizeof(a) it will return 4 |
| ?: | Conditional operator | If condition is true ? then X : otherwise Y |
| & | Return address of variable | &a; this will return address of variable a. |
| \* | Pointer to variable | int \*b=&a;  \*b this will return value of a. |

**Precedence and Associativity of Operators :**

* The precedence of the operators means the sequence in which the operators will be operated on, in case of multiple operators in a statement i.e. which operator will be executed first and which operator will be executed later.
* The associativity of operators refers to the direction in which the operation will be performed in case of equal precedence operators.

|  |  |  |
| --- | --- | --- |
| **Category** | **Operator** | **Associativity** |
| Postfix | () [] -> . ++ -- | Left to right |
| Unary | + - ! ~ ++ -- (type) \* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %= >>= <<= &= ^= |= | Right to left |
| Comma | , | Left to right |

C - Decision Making

Decision Making are use to perform the operations based on a particular condition i.e. if a condition is true perform one task else perform another task.

C language provide following Decision Making statement -

* **If Statement :**

if (condition)

{

//statement

}

1. The if statement evaluates the condition inside parenthesis.
2. If condition is evaluated to true, statements inside the body of if is executed.
3. If condition is evaluated to false, statements inside the body of if is skipped.

* **If…else Statement :**

if (condition)

{

// statement to be executed if condition is true

}

else

{

// statement to be executed if condition is false

}

1. The if…else executes the codes inside the body of if statement if the condition is true and skips the codes inside the body of else.
2. If the condition is false, it executes the codes inside the body of else statement and skips the codes inside the body of if.

* **Nested If…else Statement :**

if (condition1)

{

// statements to be executed if condition1 is true

}

else if (condition2)

{

// statements to be executed if condition1 is false and condition2 is true

}

else if (condition3)

{

// statements to be executed if condition1 and condition2 is false and condition3 is true

}

.

.

else

{

// statements to be executed if all conditions are false

}

1. The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.
2. The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.

* **Ternary Operator ?:**

A ternary operator operates on 3 operands which can be used instead of a if...else statement.

Consider this code:

If (a<b)

{

min = a;

}

else

{

min = b;

}

You can replace the above code with:

min = (a<b) ? a : b ;

The ternary operator is more readable than a if...else statement for short conditions.

* **Switch...case Statement :**

1. The ladder if..else..if statement allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in ladder if..else..if, it is better to use switch statement.
2. The switch statement is often faster than if...else (not always). Also, the syntax of switch statement is cleaner and easier to understand.

switch (n)

​{

case constant1:

// code to be executed if n is equal to constant1;

break;

case constant2:

// code to be executed if n is equal to constant2;

break;

.

.

.

default:

// code to be executed if n doesn't match any constant

}

When a case constant is found that matches the switch expression, control of the program passes to the block of code associated with that case.

In the above pseudo code, suppose the value of n is equal to constant2. The compiler will execute the block of code associated with the case statement until the end of switch block, or until the [break statement](https://www.programiz.com/cpp-programming/break-continue" \o "C++ break statement) is encountered.

The break statement is used to prevent the code running into the next case.

C - Loops

Loops are used in programming to repeat a specific block of code.

Loops are used in programming to repeat a specific block until some end condition is met. There are three type of loops in C programming:

1. **For Loop :** for is a iterative statement. It is used to repeat a set of statements number of times. The syntax of **for** statement is given below -

for ( initialization ; condition ; updating )

{

//statements inside for loop

}

where, only condition is mandatory.

**How For Loop Works :**

1. The initialization statement is executed only once at the beginning.
2. Then, condition is evaluated, if condition is false then for loop is terminated. But if condition is true, then statements inside for loop is executed.
3. Then, updating operation is executed.
4. After updating, controls goes back to the second step and then third and then fourth again and again until for loop got terminated.

**Code :**

#include <stdio.h>

int main ()

{

int start = 1, end = 10;

for (int i = start; i <= end; i++)

{

printf ("%d ", i);

}

return 0;

}

**Output :**

1 2 3 4 5 6 7 8 9 10

1. **While Loop :** Like for loop while loop is also used for iterative statements. The syntax for **while** loop is given below -

while (condition)

{

//statements

}

**How While Loop Works :**

1. First the condition is checked, If condition is true statement inside while loop are executed.
2. Above step repeated until condition is false.
3. When condition is false while loop terminated.

**Code :**

#include <stdio.h>

int main ()

{

int start = 1, end = 10;

int i=start;

while(i<=end)

{

printf ("%d ", i);

i++;

}

return 0;

}

**Output :**

1 2 3 4 5 6 7 8 9 10

1. **Do…while Loop :** do…while loop is like while loop with slightly difference i.e. in do…while first the statements are executed and then the condition is checked. If condition is true then statements are executed again. If the condition is false, the statements are not executed again.

The Syntax for do…file is given below -

do

{

//statements

} while (condition);

In do…while loop statements got execute once even if starting condition is false. Which is not true in case of while loop.

**Code :**

#include <stdio.h>

int main ()

{

int start = 1, end = 10;

int i=start;

do

{

printf ("%d ", i);

i++;

}while (i <= end);

return 0;

}

**Output :**

1 2 3 4 5 6 7 8 9 10

C - Functions

Function is a self-contained block of statements that perform a given task. The programs we were writing until now had only one function i.e. the main function.

Depending on whether a function is predefined or created by programmer; there are two types of function:

1. User-defined Function
2. Library Function
3. **User-defined Function :**

The syntax for a user-defined function is given below -

return\_type function\_name (argument\_list)

{

.

.

//statements

.

.

}

Let’s see the meaning of each of these terms used in the syntax-

1. **return\_type** is the data type of the data to be returned by the function. The return type of the data can be any of the data types like int, char, float, etc.
2. The **function\_name** can be anything as far as the rules of identifiers.
3. In the parenthesis we write **argument\_list**. Argument list is a set of data types along with the identifiers to accept a set of data passed to the function.
4. Statements inside the function are the statements to be executed when that functions is called.

**Code :**

#include <stdio.h>

int max (int a, int b)

{

if (a > b)

{

return a;

}

else

{

return b;

}

}

int main ()

{

int a = 3, b = 8;

int max\_ab = max (a, b);

printf ("maximum between %d and %d is %d", a, b, max\_ab);

return 0;

}

**Output :**

maximum between 3 and 8 is 8

**Explanation :** In above code max functions takes two argument a and b and return maximum between a and b.

Note : max functions can be written in one line using ternary operator. ( eg. return a>b ? a : b;)

**Function Declarations :**

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts −

return\_type function\_name( parameter list );

For the above defined function max(), the function declaration is as follows −

int max(int a, int b);

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration −

int max(int, int);

Function declaration is required when you define a function in one source file and you call that function in another file or You are calling the function before defining it. In such case, you should declare the function at the top of the file.

1. **Library Functions :** C Standard library functions or simply C Library functions are inbuilt functions in C programming.

The prototype and data definitions of these functions are present in their respective header files. To use these functions we need to include the header file in our program. For example,

If you want to use the printf() function, the header file <stdio.h> should be included.

#include <stdio.h>

int main()

{

printf("Printing this using library function.");

}

If you try to use printf() without including the stdio.h header file, you will get an error.

C - Scope

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable cannot be accessed. There are three places where variables can be declared in C programming language −

1. Inside a function or a block which is called local variables.
2. Outside of all functions which is called global variables.
3. In the definition of function parameters which are called formal parameters.

* **Local Variables :**

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

* **Global Variables :**

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

**Note** : If a program have same name for local and global variables then the value of local variable inside a function will take preference.

* **Formal Parameters :**

Formal parameters, are treated as local variables with-in a function and they take precedence over global variables.

**Code :**

#include <stdio.h>

int z = 0; //global variable z

void max (int a, int b) //formal variable a and b

{

z = a>b ? a : b;

}

int main ()

{

int x = 4, y= 8; //local variable x and y

printf("global variable z before calling max is z = %d",z);

max (x, y);

printf ("\nglobal variable z after calling max is z = %d",z);

return 0;

}

**Output :**

global variable z before calling max is z = 0                                      global variable z after calling max is z = 8

**Explanation :**

In above example max function has two formal variable a and b and storing maximum between a and b into global variable z.

Main function has two local variable x and y which is calling max with parameter x and y and printing value of z.

C - Array

In the programming language C, an array is a collection of values of a given data type. Arrays can be created from any of the C data types, such as int, float, and char. Arrays can only hold data of their designated type: for example, an integer array can only hold integer values.

**Syntax for Array declaration -**

Data\_type array\_name[array\_size] ;

For example, if you want to store 100 integers, you can create an array for it.

int data[100];

Here, we declared an array, data, of int type. And its size is 100. Meaning, it can hold 100 integer values.

It's important to note that the size and type of an array cannot be changed once it is declared.

**Access Array Elements :**

You can access elements of an array by indices.

Suppose you declared an array data as above. The first element is data[0], the second element is data[1] and so on.

**Few keynotes:**

1. Arrays have 0 as the first index, not 1. In this example, data[0] is the first element.
2. If the size of an array is n, to access the last element, the n-1 index is used. In this example, data[4].
3. Suppose the starting address of data[0] is 2120d. Then, the address of the data[1] will be 2124d. Similarly, the address of data[2] will be 2128d and so on.  
   This is because the size of a int is 4 bytes.

**How to initialize an array?**

It is possible to initialize an array during declaration. For example,

int data[5] = {19, 10, 8, 17, 9};

You can also initialize an array like this.

int data[] = {19, 10, 8, 17, 9};

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

**Code :**

#include <stdio.h>

int main ()

{

int data[5]={5,6,7,8,9};

int increment\_val=2;

printf("Arrays value before increment : \n");

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

printf("%d ",data[i]);

}

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

data[i]+=increment\_val;

printf("\nArrays value after increment : \n");

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

printf("%d ",data[i]);

}

return 0;

}

**Output :**

Arrays value before increment :

5  6  7  8  9

Arrays value after increment :

7  8  9  10  11

C - Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. We can get address of any variable using & (reference) operator.

The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

**Syntax for Pointers -**

type \*var-name;

Consider the following example to define a pointer which stores the address of an integer.

int n = 10;

int\* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.

**Declaring a pointer :**

The pointer in c language can be declared using \* (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

int \*a; //pointer to int

char \*c; //pointer to char

**Reference (&) and Deference (\*) operator :**

Reference operator (&) as discussed above gives the address of a variable.

To get the value stored in the memory address, we use the dereference operator (\*).

For example: If a number variable is stored in the memory address 0x123, and it contains a value 5.

The reference (&) operator gives the value 0x123, while the dereference (\*) operator gives the value 5.

Note: The (\*) sign used in the declaration of C pointer is not the dereference pointer. It is just a similar notation that creates a pointer.

**Code :**

#include <stdio.h>

int main ()

{

int \*pc, c = 5;

printf ("Address of c (&c): %d \n", &c);

printf ("Value of c (c): %d \n\n", c);

pc = &c;

printf ("Address that pointer pc holds (pc): %d \n", pc);

printf ("Content of the address pointer pc holds (\*pc): %d \n\n", \*pc);

return 0;

}

**Output :**

Address of c (&c): -1672236636

Value of c (c): 5

Address that pointer pc holds (pc): -1672236636

Content of the address pointer pc holds (\*pc): 5

**Null Pointer :** A pointer that is assigned NULL is called a null pointer.

Example:

int \*a = NULL;

**Pointer to Pointer :** A variable that store (points to) address of another pointer is known as pointer to pointer.

Example:

int \*b = &a;

int \*\*c = &b;

In above, example variable c is pointer to pointer as it is pointing to address of another pointer i.e. b .

**Code :**

#include <stdio.h>

int main ()

{

int a = 3;

int \*pa = &a; //pointer to a

int \*\*ppa = &pa; //pointer to pointer to pa

printf(" a = %d \n pa = %d \n ppa = %d \n \*pa = %d \n \*ppa = %d \n \*\*ppa = %d",a,pa,ppa,\*pa,\*ppa,\*\*ppa);

return 0;}

**Output :**

 a = 3

 pa = -658014868

 ppa = -658014864

 \*pa = 3

\*ppa = -658014868

\*\*ppa = 3

**Pointer to Array :** [Pointers](https://www.programiz.com/cpp-programming/pointers" \o "C++ Pointers) are the variables that hold address. Not only can pointers store address of a single variable, it can also store address of cells of an [array](https://www.programiz.com/cpp-programming/arrays" \o "C++ arrays).

**Consider this example:**

int\* ptr;

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

ptr = &a[0]; // &a[0] is the address of first element of a.

Suppose, pointer needs to point to the second element of an array, that is, hold address of second array element in above case.

Since ptr points to the first element in the above example, (ptr + 1) will point to the second element, (ptr+2) will point to third element and so on.

Here, in above example :

\*ptr = 1

\*(ptr+1) = 2

\*(ptr+2) = 3

\*(ptr+3) = 4

\*(ptr+4) =5 .

As ptr is int pointer ptr and (ptr+1) is differ by 4 bytes because size of int is 4 bytes.

C - Strings

In C strings are arrays of type char terminated with null character, that is, \0 (ASCII value of null character is 0).

**How to define string :**

char str[] = "CString";

In the above code, str is a string and it holds 8 characters.

Although, "CString" has 7 character, the null character \0 is added to the end of the string automatically.

**Alternative ways of defining a string :**

char str[8] = "CString";

char str[] = {'C', 'S', 't', 'r', 'i', 'n', 'g', '\0'};

char str[8] = {'C', 'S', 't', 'r', 'i', 'n', 'g','\0'};

Like arrays, it is not necessary to use all the space allocated for the string. For example:

char str[100] = "CString";

C support wide range of functions for manipulation of strings -

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Functions** | **Purpose** |
| 1. | strcpy(s1,s2) | Copies string s2 into string s1 |
| 2. | strcat(s1, s2) | Concatenates string s2 onto the end of string s1. |
| 3. | strlen(s1) | Return the length of string |
| 4. | strcmp(s1, s2) | Return 0 if s1 equals s2; negative value if s1<s2; positive value if s1>s2 |

**Code :**

#include <stdio.h>

int main ()

{

char str1[] = "Hello";

char str2[] = "Programmer";

printf("length(str1) = %d \nlength(str2) = %d \n", strlen(str1), strlen(str2));

strcat(str1,str2);

printf("after concatenation str1 = %s",str1);

return 0;

}

**Output :**

length(str1) = 5

length(str2) = 10

after concatenation str1 = HelloProgrammer

C - Structures

In C programming, a struct (or structure) is a collection of variables can be of different types under a single name.

Before you can create structure variables, you need to define its data type. To define a struct, the struct keyword is used.

Syntax of structure declaration is as follow -

struct structure\_name

{

data\_type member1;

data\_type member2;

...

};

Here is an example:

struct Person

{

char name[50];

int citNo;

float salary;

};

Here, a derived type **struct Person** is defined. Now, you can create variables of this type.

**Create struct variables :**

When a struct type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

Here's how we create structure variables:

struct Person

{

char name[50];

int citNo;

float salary;

};

int main()

{

struct Person person1, person2, p[20];

return 0;

}

Another way of creating a struct variable is:

struct Person

{

char name[50];

int citNo;

float salary;

} person1, person2, p[20];

In both cases, two variables person1, person2, and an array variable p having 20 elements of type struct Person are created.

**Access members of a structure :**

There are two types of operators used for accessing members of a structure.

1. .  - Member operator

Suppose, you want to access the salary of person2. Here's how you can do it.

person2.salary

1. ->  - To access members of a structure using pointers, we use the -> operator.

**Code :**

#include <stdio.h>

struct Person

{

int citNo;

float salary;

};

int main ()

{

struct Person person, \*personptr;

personptr = &person;

personptr->citNo = 10;

personptr->salary = 999;

printf ("citiNo = %d \nsalary = %f", person.citNo, person.salary);

return 0;

}

**Output :**

citiNo = 10

salary = 999.000000

C - Unions

* Unions is a collections of multiple data elements that can be of different data types But, only one of these data items can be stored in the union variable.
* The memory space required to store one variable of union is equal to the memory space required by the largest element in the union.
* Syntax for Union declaration is as follows -

union union\_name

{

data\_type variable\_name;

data\_type variable\_name;

...

};

Here is an example :

union student

{

char name[20];

int roll\_no;

float id;

};

* Syntax of declaration of a variable of the union :

union union\_name union\_varable\_name;

Here is an example :

union student s1;

* In union **student** the memory space required by the variable “name” of the union is 20 bytes, while for “roll no” and “id” the space required is 4 bytes each. Hence, the space allotted to the variable of the union is 20 bytes as the maximum space is required to store the “name”, which is 20 bytes.

**Access members of a Union:**

There are two types of operators used for accessing members of a structure.

1. .  - Member operator

Suppose, you want to access the roll\_no of student. Here's how you can do it.

student.roll\_no;

1. ->  - To access members of a structure using pointers, we use the -> operator.

**Code :**

#include <stdio.h>

union student

{

char name[20];

int roll\_no;

float id;

};

int main ()

{

union student s1,\*ptrs1;

ptrs1 = &s1;

s1.roll\_no = 21;

s1.id = 2;

printf("roll\_no = %d\nid = %f",ptrs1->roll\_no,ptrs1->id);

return 0;

}

**Output :**

roll\_no = 1073741824

id = 2.000000

*Note that 21 was not stored in s1.roll\_no. As You can access all members of a structure at once as sufficient memory is allocated for all members. However, it's not the case in unions. You can only access a single member of a union at one time.*

C - Bit Fields

* Bit fields are use to consume memory efficiently when we know the value of field or group of field we never exceed a limit or is within a small range.
* It can be used in structure and union.
* Syntax for Bit Fields is as follow -

struct structure\_name

{

type [member\_name] : width ;

};

Here is an example :

struct Student

{

unsigned int roll\_no: 10;

};

* Suppose you have made structure data type to store date as follow -

struct Date

{

unsigned int day;

unsigned int month;

unsigned int year;

};

And create the variable **d1** of Date data type as follow -

struct Date d1 = {30,5,2020};

The total memory consume by variable d1 is 12 bytes as there are three int variable inside it each of 4 bytes.

As we already know maximum value in day would be 31(which takes 5 bits i.e. 11111) and maximum value in the month would be 12(which takes 4 bits i.e. 1100) so we can use Bit Fields here to make it more memory efficient as follows -

struct Date

{

unsigned int day : 5;

unsigned int month : 4;

unsigned int year;

};

Creating variable of Date -

struct Date d1 = {30,5,2020};

This will consume 8 bytes as memory is assigned in group of 4 bytes, so first 4 bytes is assigned to year variable, and in other 4 bytes (i.e. 32 bits) 5 bits are assigned to day variable and 4 bits are assigned to month variable. So total 2 group of 4 bytes are consumed by d1 so the memory required for it is 8 bytes.

C - Enumerators

Enumerators is user defined data type, an enumeration type (also called enum) is a data type that consists of integral constants. To define enums, the enum keyword is used. Syntax to define enum is given below -

enum enum\_name

{

const1, const2, ..., constN

};

By default, const1 is 0, const2 is 1 and so on. You can change default values of enum elements during declaration (if necessary), like as shown below -

enum suit

{

club = 1,

diamonds = 5,

hearts = 9,

spades = 3,

};

**Note :** You can write any code in C language without enumerator but, enumerator help in writing clear code and simplify programming.

**Code :**

#include <stdio.h>

enum boolean

{

true,false

};

enum boolean isPositive(int n)

{

if(n < 0) return false;

else return true;

}

int main ()

{

int num = -1;

enum boolean result;

result = isPositive(num);

if(result == true){

printf("%d is positive number",num);

}else{

printf("%d is negative number",num);

}

return 0;

}

**Output :**

-1 is negative number

C - typedef

* typedef is a keyword
* You can use typedef to give name to a type
* Syntax for using typedef is as follow -

typedef type type\_name;

Here is an example :

typedef unsigned int DATE;

Now you can use DATE as a data type which is just same as unisigned int

* By convention, uppercase letters are used for these definitions (type\_name) to remind the user that type\_name is just symbol abbreviation but you can use lowercase.

typedef unsigned int date;

* You can use typedef with user define data type as well. One example of typedef with user define data type is given below -

**Code :**

#include <stdio.h>

typedef struct student

{

int roll\_no;

int id;

} STUDENT;

int main ()

{

STUDENT s1;

s1.roll\_no = 1;

s1.id = 119;

printf ("roll\_no = %d \nid = %d", s1.roll\_no, s1.id);

return 0;

}

**Output :**

roll\_no = 1

id = 119

C - Preprocessor

* The C preprocessor is a micro processor that is used by compiler to transform your code before compilation. It is called micro preprocessor because it allows us to add macros.

Preprocessor

Compiler

C Source Code

* Preprocessor only notice # start statements.
* # is called preprocessor directives.
* The word after # is called pre-processor command.
* Let’s see the list of preprocessor directives.

1. #include
2. #define
3. #undef
4. #ifdef
5. #ifndef
6. #if
7. #else
8. #elif
9. #endif
10. #error
11. #pragma

C - #include

* #include one of the most popular preprocessor command.
* It can be used include any file content to your source code.
* There are two variants to use #include directive.

1. #include <filename>
2. #include "filename"

* The #include <filename> tells the compiler to look for the directory where system header files are held. In UNIX, it is \usr\include directory.
* The #include "filename" tells the compiler to look in the current directory from where program is running or you can specify the path.
* Suppose you have a header file “list.h” to a path “c:\myprog\list.h” then,

#include <c:\myprog\list.h> is wrong

#include “c:\myprog\list.h> is correct

#include <stdio.h>

#include "math.h"

int main()

{

printf("Hello World"); //printf declaration is present in stdio.h

printf("%f",floor(2.99)); //foor declaration is present in math.h

return 0;

}

Note: if you removed the 1st line (i.e. #include<stdio.h>) then printf() this will give error.

C - #define

* The #define preprocessor directive is used to define constant or macro substitution.
* It can be used with any basic data type.
* Syntax for defining constant is as follow-

#define token value

* Example of Creating constant :

#include <stdio.h>

#define PI 3.14

int main()

{

printf("%f",PI);

return 0;

}

**Output :**

3.140000

* Example of Creating macro :

#include <stdio.h>

#define MAX(a,b) (a>b?a:b)

int main()

{

printf("%d",MAX(2,3));

return 0;

}

**Output :**

3

C - #undef

The #undef preprocessor directive is used to undefine the constant or macro defined by #define.

Syntax:

#undef token

**Examples :**

#include <stdio.h>

#define PI 3.14

float PIsquare = PI\*PI;

#undef PI

int main()

{

printf("%f",PI);

return 0;

}

**Output :**

error: ‘PI’ undeclared

#include <stdio.h>

#define PI 3.14

float PIsquare = PI\*PI;

#undef PI

int main()

{

printf("%f",PIsquare);

return 0;

}

**Output :**

9.859600

C - #ifdef

* The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code otherwise #else code is executed, if present.
* Syntax:

#ifdef MACRO

//code

#endif

* Syntax with #else:

#ifdef MACRO

//if code

#else

//else code

#endif

* **Examples :**

#include <stdio.h>

#define PI 3.14

int main()

{

#ifdef PI

printf("PI is defined");

#else

printf("PI is not defined");

#endif

return 0;

}

**Output :**

PI is defined

C - #ifndef

* The #ifndef preprocessor directive checks if macro is not defined by #define. If yes, it executes the code otherwise #else code is executed, if present.
* Syntax:

#ifndef MACRO

//code

#endif

* Syntax with #else:

#ifndef MACRO

//successful code

#else

//else code

#endif

* **Examples :**

#include <stdio.h>

#define PI 3.14

int main()

{

#ifndef PI

printf("PI is not defined");

#else

printf("PI is defined");

#endif

return 0;

}

**Output :**

PI is defined

C - #if

* The #if preprocessor directive evaluates the expression or condition. If condition is true, it executes the code otherwise #elseif or #else or #endif code is executed.

Syntax:

#if expression

//if code

#elif expression

//elif code

#else

//else code

#endif

Note : After #if, #endif is compulsory others are optional.

* **Examples :**

#include <stdio.h>

#define ID 1

int main()

{

#if(ID == 0)

printf("ID is 0");

#elif(ID == 1)

printf("ID is 1");

#else

printf("ID is not known");

#endif

return 0;

}

**Output :**

ID is 1

C - #else

* The #else preprocessor directive evaluates the expression or condition if condition of #if is false. It can be used with #if, #elif, #ifdef and #ifndef directives.
* Syntax:

#if expression

//if code

#elif expression

//elif code

#else

//else code

#endif

* **Examples :**

#include <stdio.h>

#define ID 10

int main()

{

#if(ID == 0)

printf("ID is 0");

#elif(ID == 1)

printf("ID is 1");

#else

printf("ID is not known");

#endif

return 0;

}

**Output :**

ID is not known

C - #error

* The #error preprocessor directive indicates error. The compiler gives fatal error if #error directive is found and skips further compilation process.
* Syntax:

#error error\_message

* **Examples :**

#include<stdio.h>

#ifndef PI

#error Define PI first

#endif

int main ()

{

printf ("PI = %f",PI);

return 0;

}

**Output :**

Compile Time Error: Define PI first

C - #pragma

* The preprocessor directive #pragma is used to provide the additional information to the compiler in C language. This is used by the compiler to provide some special features.
* Here is the syntax of #pragma directive in C language,

#pragma token\_name

* Following are some of #pragma directives in C language -

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **#pragma Directives** | **Description** |
| 1 | #pragma startup | Before the execution of main(), the function specified in pragma is needed to run. |
| 2 | #pragma exit | Before the end of program, the function specified in pragma is needed to run. |
| 3 | #pragma warn | Used to hide the warning messages. |
| 4 | #pragma GCC dependency | Checks the dates of current and other file. If other file is recent, it shows a warning message. |
| 5 | #pragma GCC system\_header | It treats the code of current file as if it came from system header. |
| 6 | #pragma GCC poison | Used to block an identifier from the program. |

* **Examples :**

#include<stdio.h>

void func ();

#pragma startup func

#pragma exit func

int main ()

{

printf ("\nI am in main");

return 0;

}

void func ()

{

printf ("\nI am in func");

}

**Output :**

I am in func

I am in main

I am in func

C - Command line Argument

* The arguments passed from command line are called command line arguments. These arguments are handled by main() function.
* To accept command line argument, you need to change the structure of main() function as given follows -

int main(int argc, char \*argv[] )

* Here, argc counts the number of arguments. It counts the file name as the first argument.
* The argv[] contains the total number of arguments. The first argument is the file name always.
* **Examples :**

#include <stdio.h>

int main(int argc, char \*argv[])

{

if(argc < 2)

printf("No argument\n");

else

{

for (int i = 1;i < argc; i++){

printf("Argument%d = %s\n",i,argv[i]);

}

}

return 0;

}

Run this program as follows in Linux:

./main command line argument

Run this program as follows in Windows from command line:

main.exe command line argument

**Output :**

Argument1 = command

Argument2 = line

Argument3 = argument

C - Input and Output

**Input** means read the data from keyboard, file or command line. C provides various built in function for reading the data.

**Output** means write the data to screen, file or printer. C provides various built in function to format the data and to write it on screen or file.

**printf() and scanf() Functions :**

* The scanf() and printf() functions are used for input and output respectively in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).
* The printf() function is used for output. It prints the given statement to the console.

Syntax for printf is as follow -

printf("format specifier",argument\_list);

* The scanf() function is used for input. It reads the input data from the console.

Syntax for scanf is as follow -

scanf("format specifier",argument\_list);

* Here's a list of commonly used C data types and their format specifiers.

|  |  |
| --- | --- |
| **Data Type** | **Format Specifier** |
| int | %d |
| char | %c |
| float | %f |
| double | %lf |
| short int | %hd |
| unsigned int | %u |
| long int | %li |
| long long int | %lli |
| unsigned long int | %lu |
| unsigned long long int | %llu |
| signed char | %c |
| unsigned char | %c |
| long double | %Lf |

* **Examples :**

#include <stdio.h>

int main()

{

int num1,num2,sum;

printf("Enter two integer number : ");

scanf("%d %d",&num1,&num2);

sum = num1 + num2;

printf("num1 = %d\nnum2 = %d\n",num1,num2);

printf("sum = %d",sum);

return 0;

}

Notice, that we have used &num1 and &num2 inside scanf. It is because &num1 and &num2 gets address of num1 and num2 respectively, and the value entered by the user are stored in those address.

**Output :**

Enter two integer number : 3

8

num1 = 3

num2 = 8

sum = 11

C - File I/O

* A file is a container in computer storage device used for storing data.
* Sometimes we want to store data even after program end, to use for the next time (e.g. in banking balance of a person) so in that case we have to use file because variable of program stored in RAM which is volatile memory, once program end all variable is destroyed. Storing in a file will preserve data even after program terminates.
* If you have to enter a large number of data, it will take a lot of time to enter them all.  
  However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C.
* To perform file related operation there is a predefined data type **FILE** in C language, declaration of FILE is present in stdio.h
* There is many function present in C library to work with file, a list of file function is given below -

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Function** | **Description** |
| 1 | fopen() | opens new or existing file |
| 2 | fprintf() | write data into the file |
| 3 | fscanf() | reads data from the file |
| 4 | fputc() | writes a character into the file |
| 5 | fgetc() | reads a character from file |
| 6 | fclose() | closes the file |
| 7 | fseek() | sets the file pointer to given position |
| 8 | fputw() | writes an integer to file |
| 9 | fgetw() | reads an integer from file |
| 10 | ftell() | returns current position |
| 11 | rewind() | sets the file pointer to the beginning of the file |

1. **Opening File : fopen()**

* One of most common used function in file handling is fopen().
* You can use the fopen() function to create a new file or to open an existing file.
* Function fopen() call, initialize an object of the type file and return the FILE pointer which point to initialized file object.
* The syntax of fopen() function is given below :

FILE \*fopen( const char \* filename, const char \* mode );

1. The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored.
2. The mode in which the file is to be opened. It is a string.

* List of available modes used in fopen() function are given below.

|  |  |
| --- | --- |
| **Mode** | **Description** |
| r | opens a text file in read mode |
| w | opens a text file in write mode |
| a | opens a text file in append mode |
| r+ | opens a text file in read and write mode |
| w+ | opens a text file in read and write mode |
| a+ | opens a text file in read and write mode |
| rb | opens a binary file in read mode |
| wb | opens a binary file in write mode |
| ab | opens a binary file in append mode |
| rb+ | opens a binary file in read and write mode |
| wb+ | opens a binary file in read and write mode |
| ab+ | opens a binary file in read and write mode |

* Example Code to open a file name “hello.txt” in reading mode is given below.

FILE \*fp;

fp = fopen("hello.txt","r");

1. **Closing File: fclose()**

* The fclose() function is used to close a file. The file must be closed after performing all the operations on it.
* The syntax of fclose() function is given below :

int fclose( FILE \*fp );

* The fclose() function returns zero on success, or EOF if there is an error in closing the file.

1. **Writing File : fprintf()**

* The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.
* The syntax of fclose() function is given below :

int fprintf(FILE \*stream, const char \*format [, argument, ...]);

* Example Code to write “Hello World” text in file “hello.txt” is given below.

#include <stdio.h>

int main ()

{

FILE \*fp;

fp = fopen ("hello.txt", "w");

fprintf (fp, "Hello World");

fclose (fp);

return 0;

}

1. **Reading File : fscanf()**

* The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.
* The syntax of fscanf() function is given below :

int fscanf(FILE \*stream, const char \*format [, argument, ...]);

* Example Code to read whole text present in the file “hello.txt” is given below.

#include <stdio.h>

int main ()

{

FILE \*fp;

char buff[255];

fp = fopen ("hello.txt", "r");

while (fscanf (fp, "%s", buff) != EOF)

{

printf ("%s ", buff);

}

fclose (fp);

}

1. **Writing File : fputc()**

* The fputc() function is used to write a single character into file. It outputs a character to a stream.
* The syntax of fputc() function is given below :

int fputc(int c, FILE \*stream)  ;

* Example Code to write “a” char in file “hello.txt” is given below.

#include <stdio.h>

int main ()

{

FILE \*fp;

fp = fopen ("hello.txt", "w");

fputc (‘a’,fp);

fclose (fp);

return 0;

}

1. **Reading File : fgetc()**

* The fgetc() function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.
* The syntax of fgetc() function is given below :

int fgetc(FILE \*stream) ;

1. **Writing File : fputs()**

* The fputs() function writes a line of characters into file. It outputs string to a stream.
* The syntax of fputs() function is given below :

int fputs(const char \*s, FILE \*stream)  ;

1. **Reading File : fgets()**

* The fgets() function reads a line of characters from file. It gets string from a stream.
* The syntax of fgets() function is given below :

char\* fgets(char \*s, int n, FILE \*stream) ;

* Example Code to read text from file “hello.txt” is given below.

#include <stdio.h>

int main ()

{

FILE \*fp;

char text[300];

fp = fopen ("hello.txt", "r");

printf ("%s", fgets (text, 200, fp));

fclose (fp);

return 0;

}

C - Type Casting

* Type casting is a way to convert a variable from one data type to another data type.
* For example, if you want to store a long value into a simple integer then you can typecast long to int.
* You can convert values from one type to another explicitly using the cast operator.
* There are two types of type casting in c language that are Implicit conversions and Explicit Conversions.
* Syntax for type casting :

(type)value;

1. **Implicit Conversion :** Implicit conversions do not require any operator for converted. They are automatically performed when a value is copied to a compatible type in the program.

Example -

#include <stdio.h>

int main ()

{

int a = 1;

float a\_float;

a\_float = a;

printf ("%f", a\_float);

return 0;

}

Here, the value of variable ‘a’ has been promoted from int to float and we have not had to specify any type-casting operator. This is known as a standard conversion.

**Output -**

1.000000

1. **Explicit Conversions :** In C language, Many conversions, especially those that imply a different interpretation of the value, require an explicit conversion.

Example -

#include <stdio.h>

int main ()

{

int a = 5, b = 2;

float division = (float)a/b;

printf("%f",division);

return 0;

}

Here, the value of variable ‘a’ has been converted from int to float before actual division, to get final result in floating number (as float/int = float). If we have not used type casting here answer would be 5/2 = 2 (as int/int = int) and then this 2 would be internally converted into float so final answer would be 2.00000 .

**Output -**

2.500000

C - Recursion

* A function that calls itself is known as a recursive function. And, this technique is known as recursion.
* Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem.
* It is very important to have some base condition (termination point) in recursive function otherwise function will run for infinite time.
* Code of recursion is look like as given below -

#include <stdio.h>

void recursion(){

//some base condition (termination point)

...

recursion();

...

}

int main ()

{

recursion();

return 0;

}

* Example Code to print factorial of given number using recursion is given below -

#include <stdio.h>

int factorial (int n)

{

if (n == 0 || n == 1) //base condition

return 1;

else

return n \* factorial (n - 1); //recursive call

}

int main ()

{

int num, ans;

printf ("Enter a number : ");

scanf ("%d", &num);

ans = factorial (num);

printf ("factorial of %d = %d", num, ans);

return 0;

}

**Output :**

Enter a number : 4

factorial of 4 = 24

**Explanation :**

factorial(n) = n \* (n-1) \* (n-2) \* (n-3) \* ... \* 2 \* 1 ;

factorial(4) = 4 \* 3 \* 2 \* 1 = 24

C - Variable Arguments

* Perhaps you would like to have a function that will accept any number of arguments, perform some operation on that and then return the calculated answer(e.g. to find sum of given numbers).
* You don't know how many arguments will be passed in to the function. One way you could make the function would be to accept a pointer to an array.
* Another way would be to write a function that can take any number of arguments. So you could write sum(4, 12.2, 23.3, 33.3, 12.1); or you could write sum(2, 2.3, 34.4); The advantage of this approach is that it's much easier to change the code if you want to change the number of arguments.
* Whenever a function is declared to have an indeterminate number of arguments, in place of the last argument you should place an ellipsis ( … )

float function\_name(int n,…)

* We'll need to use some macros from the stdarg.h header file to extract the values stored in the variable argument list. va\_start, which initializes the list, va\_arg, which returns the next argument in the list, and va\_end, which cleans up the variable argument list.
* To use these functions, we need a variable capable of storing a variable-length argument list. this variable will be of type va\_list.
* va\_start is a macro which accepts two arguments, a va\_list and the name of the variable that directly precedes the ellipsis ( … )
* va\_arg takes a va\_list and a variable type, and returns the next argument in the list in the form of whatever variable type it is told.
* Example Code to find sum of all given number using variable length argument is given below -

#include <stdio.h>

#include<stdarg.h>

int sum(int n, ...)

{

va\_list a\_list;

va\_start(a\_list,n);

int ans = 0;

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

ans += va\_arg(a\_list, int);

}

va\_end(a\_list);

return ans;

}

int main()

{

int ans;

ans = sum(4 ,1,2,3,9);

printf("sum = %d\n",ans);

ans = sum(2 ,9,14);

printf("sum = %d",ans);

return 0;

}

Notice, first argument while calling sum function is number of argument.

**Output :**

sum = 15

sum = 23

C - Dynamic Memory Allocation

* The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime.
* Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

* Static vs dynamic memory allocation -

|  |  |
| --- | --- |
| **Static memory allocation** | **Dynamic memory allocation** |
| memory is allocated at compile time. | memory is allocated at run time. |
| memory can't be increased while executing program. | memory can be increased while executing program. |
| used in array. | used in linked list. |

* Methods used for dynamic memory allocation -

|  |  |
| --- | --- |
| **Functions** | **Description** |
| malloc() | allocates single block of requested memory. |
| calloc() | allocates multiple block of requested memory. |
| realloc() | reallocates the memory occupied by malloc() or calloc() functions. |
| free() | frees the dynamically allocated memory. |

1. **malloc() function in C :**

* The malloc() function allocates single block of requested memory.
* It doesn't initialize memory at execution time, so it has garbage value initially.
* It returns NULL if memory is not sufficient.
* The syntax of malloc() function is given below:

ptr = (cast-type\*)malloc(byte-size)

Here, is an Example :

int \*ptr;

ptr=(int\*)malloc(10\*sizeof(int));

1. **calloc() function in C :**

* The calloc() function allocates multiple block of requested memory.
* It initially initialize all bytes to zero.
* It returns NULL if memory is not sufficient.
* The syntax of calloc() function is given below:

ptr = (cast-type\*)calloc(number, byte-size)

Here, is an Example :

int \*ptr;

ptr=(int\*)calloc(n,sizeof(int));

1. **realloc() function in C :**

* If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.
* The syntax of realloc() function is given below:

ptr = realloc(ptr, new-size)

1. **free() function in C :**

* The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.
* The syntax of free() function is given below:

free(ptr)

* **Code :**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int \*ptr, i, size=3, sum = 0;

ptr = (int\*)calloc(size,sizeof(int));

if(ptr == NULL){

printf("Unalable to allocate memory");

exit(0);

}

else{

printf("Enter %d Elements : ",size);

for(i=0; i<size;i++){

scanf("%d",ptr+i);

sum += \*(ptr+i);

}

printf("sum of given %d number = %d",size,sum);

printf("\n\nincreasing size by 1 ......");

size += 1;

ptr = realloc(ptr, size);

printf("\nEnter next variable: ");

scanf("%d",ptr+size-1);

sum += \*(ptr+size-1);

printf("new sum of given number = %d",sum);

}

return 0;

}

**Output :**

Enter 3 Elements : 2

2

2

sum of given 3 number = 6

increasing size by 1 ......

Enter next variable: 3

new sum of given number = 9

C - Questions And Answers

1. Who developed the c programming ?

Answer : Dennis Ritchie

1. A name having a few letters, numbers and special character underscore (\_) is called ?

Answer : identifiers

1. The words if, else, auto, float etc. Have predefined meaning and users cannot use them as variables. These words are called ?

Answer : Keywords

1. Define Ternary operator

Answer : An operator use to check a condition and select a value depending on the value of the condition.

1. Which operators perform operations on data in binary level ?

Answer : Bitwise operator

1. The operator ‘&’ is used for ?

Answer : Bitwise AND

1. The size of character variable in C is ?

Answer : 1 bytes

1. Maximum value of unsigned integer is ?

Answer : 65535

1. The typedef is used to ?

Answer : create a new datatype name

1. What are the key features in the C programming language?

Answer : Features are as follows:

* **Modularity**: Possibility to break down large programs into small modules.
* **Portability**: It is a platform-independent language.
* **Flexibility**: The possibility of a programmer to control the language.
* **Speed**: C comes with support for system programming and hence it compiles and executes with high speed when compared with other high-level languages.
* **Extensibility**: Possibility to add new features by the programmer.

1. What are the basic data types associated with C?

Answer :

* Int – Represent the number (integer)
* Float – Number with a fraction part.
* Double – Double-precision floating-point value
* Char – Single character
* Void – Special purpose type without any value.

1. What is the description for syntax errors?

Answer : The mistakes/errors that occur while creating a program are called syntax errors. Misspelled commands or incorrect case commands, an incorrect number of parameters in calling method /function, data type mismatches can be identified as common examples for syntax errors.

1. What are reserved words with a programming language?

Answer : The words that are a part of the standard C language library are called reserved words. Those reserved words have special meaning and it is not possible to use them for any activity other than its intended functionality.

Example: void, return int.

1. Describe static function with its usage?

Answer : A function, which has a function definition prefixed with a static keyword is defined as a static function. The static function should be called within the same source code.

1. Describe Wild Pointers in C?

Answer : Uninitialized pointers in the C code are known as Wild Pointers. They point to some arbitrary memory location and can cause bad program behavior or program crash.

1. What is the difference between ++a and a++?

Answer : ‘++a”  is called prefixed increment and the increment will happen first on a variable. ‘a++' is called postfix increment and the increment happens after the value of a variable used for the operations.

1. Describe the difference between = and == symbols in C programming?

Answer :  ‘==' is the comparison operator which is used to compare the value or expression on the left-hand side with the value or expression on the right-hand side.

‘=' is the assignment operator which is used to assign the value of the right-hand side to the variable on the left-hand side.

1. What is the explanation for prototype function in C?

Answer :  Prototype function is a declaration of a function with the following information to the compiler.

* Name of the function.
* The return type of the function.
* Parameters list of the function.

1. What is the explanation for the cyclic nature of data types in C?

Answer : Some of the data types in C have special characteristic nature when a developer assigns value beyond the range of the data type. There will be no compiler error and the value changes according to a cyclic order. This is called cyclic nature. Char, int, long int data types have this property. Further float, double and long double data types do not have this property.

1. Describe the header file and its usage in C programming?

Answer : The file containing the definitions and prototypes of the functions being used in the program are called a header file. It is also known as a library file.

Example: The header file contains commands like printf and scanf is from the stdio.h library file.

1. A statement that allows the execution of statements or groups of statements in a repeated way is defined as a loop.

Answer :A statement that allows the execution of statements or groups of statements in a repeated way is defined as a loop.

There are 4 types of loop statements in C.

* While loop
* For Loop
* Do…While Loop
* Nested Loop

1. What is a nested loop?

Answer :A loop that runs within another loop is referred to as a nested loop. The first loop is called the Outer Loop and the inside loop is called the Inner Loop. The inner loop executes the number of times defined in an outer loop.

1. What is the general form of function in C?

Answer : The function definition in C contains four main sections.

return\_type function\_name( parameter list )

{

      body of the function

}

* Return Type: Data type of the return value of the function.
* Function Name: The name of the function and it is important to have a meaningful name that dscribes the activity of the function.
* Parameters: The input values for the function that are used to perform the required action.
* Function Body: Collection of statements that performs the required action.

1. What is a pointer on a pointer in C programming language?

Answer : A pointer variable that contains the address of another pointer variable is called pointer on a pointer. This concept de-refers twice to point to the data held by a pointer variable.

1. What are the valid places to have keyword “Break”?

Answer : The purpose of the Break keyword is to bring the control out of the code block which is executing. It can appear only in looping or switch statements.

1. What is the behavioral difference when the header file is included in double-quotes (“”) and angular braces (<>)?

Answer :When the Header file is included within double quotes (“ ”), compiler search first in the working directory for the particular header file. If not found, then it searches the file in the include path. But when the Header file is included within angular braces (<>), the compiler only searches in the working directory for the particular header file.

1. What is a sequential access file?

Answer :  General programs store data into files and retrieve existing data from files. With the sequential access file, such data are saved in a sequential pattern. When retrieving data from such files each data is read one by one until the required information is found.

1. Select the incorrect operator form in the following list(== , <> , >= , <=) and what is the reason for the answer?

Answer : Incorrect operator is ‘<>'. This format is correct when writing conditional statements, but it is not the correct operation to indicate not equal in C programming. It gives a compilation error as follows.

1. Is it possible to use curly brackets ({}) to enclose a single line code in C program?

Answer : Yes, it works without any error. Some programmers like to use this to organize the code. But the main purpose of curly brackets is to group several lines of codes.

1. Describe the modifier in C?

Answer : Modifier is a prefix to the basic data type which is used to indicate the modification for storage space allocation to a variable.

Example - In a 32-bit processor, storage space for the int data type is 4.When we use it with modifier the storage space change as follows:

* Long int: Storage space is 8 bit
* Short int: Storage space is 2 bit

1. What are the modifiers available in C programming language?

Answer : There are 5 modifiers available in the C programming language as follows:

* Short
* Long
* Signed
* Unsigned
* long long

1. Is that possible to store 32768 in an int data type variable?

Answer : Int data type is only capable of storing values between - 32768 to 32767. To store 32768 a modifier needs to used with the int data type. Long Int can use and also if there are no negative values, unsigned int is also possible to use.

1. Is there any possibility to create a customized header file with C programming language?

Answer : Yes, it is possible and easy to create a new header file. Create a file with function prototypes that are used inside the program. Include the file in the ‘#include' section from its name.

1. Describe dynamic data structure in C programming language?

Answer : Dynamic data structure is more efficient to memory. The memory access occurs as needed by the program.

1. Is that possible to add pointers to each other?

Answer : There is no possibility to add pointers together. Since pointer contains address details there is no way to retrieve the value from this operation.

1. What is indirection?

Answer : If you have defined a pointer to a variable or any memory object, there is no direct reference to the value of the variable. This is called the indirect reference. But when we declare a variable, it has a direct reference to the value.

1. What are the ways to a null pointer that can be used in the C programming language?

Answer : Null pointers are possible to use in three ways.

* As an error value.
* As a sentinel value.
* To terminate indirection in the recursive data structure.

1. What is the explanation for modular programming?

Answer : The process of dividing the main program into executable subsection is called module programming. This concept promotes reusability.

1. Which of the following double, float, char, array is not a basic data type in C language.

Answer : array

1. Distinguish between malloc() and calloc() memory allocation.

Answer : Both allocates memory from heap area/dynamic memory. By default calloc fills the allocated memory with 0’s.

1. What is keyword auto for ?

Answer : By default every local variable of the function is automatic (auto). In the below function both the variables ‘i’ and ‘j’ are automatic variables.

void f() {

int i;

auto int j;

}

NOTE − A global variable can’t be an automatic variable.

1. Explain the syntax for for loop.

Answer :

for(expression-1;expression-2;expression-3) {

//set of statements

}

When control reaches for expression-1 is executed first. Then following expression-2, and if expression-2 evaluates to non-zero ‘set of statements’ and expression-3 is executed, follows expression-2.

1. How a negative integer is stored.

Answer : Get the two’s compliment of the same positive integer. Eg: 1011 (-5)

Step-1 − One’s compliment of 5 : 1010

Step-2 − Add 1 to above, giving 1011, which is -5

1. Explain the purpose of the sprintf().

Answer : Prints the formatted output onto the character array.

1. What is the meaning of base address of the array?

Answer : The starting address of the array is called as the base address of the array.

1. When should we use the register storage specifier?

Answer : If a variable is used most frequently then it should be declared using register storage specifier, then possibly the compiler gives CPU register for its storage to speed up the look up of the variable.

1. [S++ or S = S+1, which can be recommended to increment the value by 1 and why?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : S++, as it is single machine instruction (INC) internally.

1. [What is a dangling pointer?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A pointer initially holding valid address, but later the held address is released or freed. Then such a pointer is called as dangling pointer.

1. [What is lvalue and rvalue?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : The expression appearing on right side of the assignment operator is called as rvalue. Rvalue is assigned to lvalue, which appears on left side of the assignment operator. The lvalue should designate to a variable not a constant.

1. [What is the difference between actual and formal parameters?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : The parameters sent to the function at calling end are called as actual parameters while at the receiving of the function definition called as formal parameters.

1. [Can a program be compiled without main() function?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Yes, it can be but cannot be executed, as the execution requires main() function definition.

1. [What is the advantage of declaring void pointers?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : When we do not know what type of the memory address the pointer variable is going to hold, then we declare a void pointer for such.

1. [Where an automatic variable is stored?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Every local variable by default being an auto variable is stored in stack memory.

1. [What is a nested structure?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A structure containing an element of another structure as its member is referred so.

1. [What is the difference between variable declaration and variable definition?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Declaration associates type to the variable whereas definition gives the value to the variable.

1. [What is a self-referential structure?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A structure containing the same structure pointer variable as its element is called as self-referential structure.

1. [Does a built-in header file contains built-in function definition?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : No, the header file only declares function. The definition is in library which is linked by the linker.

1. [What is a token?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol.

1. [What is a preprocessor?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Preprocessor is a directive to the compiler to perform certain things before the actual compilation process begins.

1. [Explain the use of %i format specifier w.r.t scanf().](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Can be used to input integer in all the supported format.

1. [Does a break is required by default case in switch statement?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Yes, if it is not appearing as the last case and if we do not want the control to flow to the following case after default if any.

1. [When to use -> (arrow) operator.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : If the structure/union variable is a pointer variable, to access structure/union elements the arrow operator is used.

1. [What are bit fields?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : We can create integer structure members of differing size apart from non-standard size using bit fields. Such structure size is automatically adjusted with the multiple of integer size of the machine.

1. [What are command line arguments?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : The arguments which we pass to the main() function while executing the program are called as command line arguments.

1. [What are the different ways of passing parameters to the functions? Which to use when?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer :

Call by value − We send only values to the function as parameters. We choose this if we do not want the actual parameters to be modified with formal parameters but just used.

Call by reference − We send address of the actual parameters instead of values. We choose this if we do want the actual parameters to be modified with formal parameters.

1. [What is the purpose of built-in stricmp() function.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : It compares two strings by ignoring the case.

1. [Describe the file opening mode “w+”.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Opens a file both for reading and writing. If a file is not existing it creates one, else if the file is existing it will be over written.

1. [Where the address of operator (&) cannot be used?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : It cannot be used on constants.

It cannot be used on variable which are declared using register storage class.

1. [Is FILE a built-in data type?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : No, it is a structure defined in stdio.h.

1. [What is reminder for 5.0 % 2?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Error, It is invalid that either of the operands for the modulus operator (%) is a real number.

1. [How many operators are there under the category of ternary operators?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : There is only one operator and is conditional operator (? : ).

1. [Which key word is used to perform unconditional branching?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : goto

1. Explain the use of comma operator (,).

Answer : Comma operator can be used to separate two or more expressions.

Eg: printf(“hi”) , printf(“Hello”);

1. [Which operator is used to continue the definition of macro in the next line?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Backward slash (\) is used.

E.g. #define MESSAGE "Hi, \

Welcome to C"

1. [Which operator is used to receive the variable number of arguments for a function?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Ellipses (…) is used for the same.

1. [Which built-in library function can be used to re-size the allocated dynamic memory?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : realloc().

1. [Define an array.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Array is collection of similar data items under a common name.

1. [What are enumerations?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Enumerations are list of integer constants with name. Enumerators are defined with the keyword enum.

1. [Which built-in function can be used to move the file pointer internally?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : fseek()

1. [What is a variable?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A variable is the name storage.

1. [C is successor of which programming language?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : B

1. [What is the full form of ANSI?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : American National Standards Institute.

1. [Which operator can be used to determine the size of a data type or variable?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : sizeof

1. [Can we assign a float variable to a long integer variable?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Yes, with loss of fractional part.

1. [Is 059 a valid octal number?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : No, it contains invalid octal digits.

1. [What is the return value of a relational operator if it returns any?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Return a value 1 if the relation between the expressions is true, else 0.

1. [How does bitwise operator XOR works.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : If both the corresponding bits are same it gives 0 else 1.

1. [What is an infinite loop?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A loop executing repeatedly as the loop-expression always evaluates to true such as

while(0 == 0) {

}

1. [Can variables belonging to different scope have same name? If so show an example.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Variables belonging to different scope can have same name as in the following code snippet.

int var;

void f() {

int var; }

main() {

int var; }

1. [What is the default value of local and global variables?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Local variables get garbage value and global variables get a value 0 by default.

1. [Can a pointer access the array?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Pointer by holding array’s base address can access the array.

1. [What are valid operations on pointers?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : The only two permitted operations on pointers are

1. Comparision ii) Addition/Substraction (excluding void pointers)
2. [What is a string length?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : It is the count of character excluding the ‘\0’ character.

1. [What is the built-in function to append one string to another?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : strcat() form the header string.h

1. [Which operator can be used to access union elements if union variable is a pointer variable?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Arrow (->) operator.

1. [Explain about ‘stdin’.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : stdin in a pointer variable which is by default opened for standard input device.

1. [Name a function which can be used to close the file stream.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : fclose().

1. [What is the purpose of #undef preprocessor?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : It be used to undefine an existing macro definition.

1. [Define a structure.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A structure can be defined of collection of heterogeneous data items.

1. [What is typecasting?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Typecasting is a way to convert a variable/constant from one type to another type.

1. [What is recursion?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Function calling itself is called as recursion.

1. [Which function can be used to release the dynamic allocated memory?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : free().

1. [What is the first string in the argument vector w.r.t command line arguments?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Program name.

1. [How can we determine whether a file is successfully opened or not using fopen() function?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : On failure fopen() returns NULL, otherwise opened successfully.

1. [What is the output file generated by the linker.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Linker generates the executable file.

1. [What is the default function call method?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : By default the functions are called by value.

1. [Functions must and should be declared. Comment on this.](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Function declaration is optional if the same is invoked after its definition.

1. [When the macros gets expanded?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : At the time of preprocessing.

1. [Can a function return multiple values to the caller using return reserved word?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : No, only one value can be returned to the caller.

1. [What is a constant pointer?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : A pointer which is not allowed to be altered to hold another address after it is holding one.

1. [To make pointer generic for which date type it need to be declared?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Void

1. [Can the structure variable be initialized as soon as it is declared?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : Yes, w.r.t the order of structure elements only.

1. [Is there a way to compare two structure variables?](https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm)

Answer : There is no such. We need to compare element by element of the structure variables.

1. What is constant?

Answer : A value which cannot be modified is called constant. Such variable are qualified with keyword const.