**Python – Full Stack Assignment**

**Module 2 – Introduction to Programming :-**

1. **Overview of C Programming.**

* **Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.**

**C** is a procedural programming language initially developed by **Dennis Ritchie** in the year **1972**at Bell Laboratories of AT&T Labs. It was mainly developed as a system programming language to write the **UNIX operating system**. **The main features of the C language include:**

* General Purpose and Portable
* Low-level Memory Access
* Fast Speed
* Clean Syntax

These features make the C language suitable for system programming like an operating system or compiler development.

Many later languages have borrowed syntax/features directly or indirectly from the C language like the syntax of Java, PHP, JavaScript, and many other languages that are mainly based on the C language. C++ is nearly a superset of C language (Only a few programs may compile in C, but not in C++).

So,  if a person learns C programming first, it will help them to learn any modern programming language as well. Also, learning C helps to understand a lot of the underlying architecture of the operating system like pointers, working with memory locations, etc.

1. **Setting Up Environment :-**

* **Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or CodeBlocks.**

C, a language known for its versatility and power, holds a widespread influence across different programming domains, from system programming to embedded systems. Before immersing yourself in the intricacies of C programming, it becomes paramount to establish a robust local development environment on your computer.

To embark on your C programming journey, creating a local development environment becomes an indispensable initial step. This involves configuring your computer to compile and run C programs simultaneously. A local environment ensures that you have the necessary tools and resources at your disposal, facilitating a smooth coding experience.

**Setting up a Local Environmеnt**

For a comprehensive C development environment on your local machine, two fundamental components are necessary: **a compiler** and **a text editor.**

**1. C compiler**

Once you've secured and installed a text editor and saved your program with a '. c' extension, the next step is acquiring a C compiler. This compiler is responsible for translating your high-level C code into a machine-understandable low-level language. In other words, we can say that it converts the source code written in a programming language into another computer language that the computer understands.

**2. Installing GCC on Linux**

Wе will install thе GNU GCC compilеr on Linux. To install and work with thе GCC compilеr on your Linux machinе, procееd according to thе bеlow stеps:

A. First, run the following two commands from your Linux terminal window:

* sudo apt-get update
* sudo apt-get install gcc
* sudo apt-get install g++

B. Additionally, you can install the build-essential package, which includes essential libraries for compiling and running C programs:

* sudo apt-get install build-essential

This command will install all thе librariеs rеquirеd to compilе and run a C program.

C. After completing the above steps, check whether the GCC compiler is installed correctly:

* gcc --version

D. If there are no errors in the above steps, your Linux environment is set up to compile C programs.

E. Writе your program in a tеxt filе and savе it with any filеnamе and '. c' еxtеnsion. Wе havе writtеn a program to display "Hеllo World" and savеd it in a filе with thе filеnamе "hеlloworld. c" on thе dеsktop.

F. Open the Linux terminal, navigate to the directory where you saved your file, and compile it using the following command:

* gcc filename.c -o any-name

G. After executing the above command, a new file with the name you chose as "any-name" will be created in the same directory.

H. To run your program, use the following command:

* ./hello

**I.**This command will execute your program in the terminal window.

These steps cover the installation of the C compiler, compilation of a C program, and running the compiled program on a Linux system.

**2. Text Editor**

Text editors are essential programs used to edit or write text, including C programs. In the context of C programming, it's crucial to understand that while the typical extension for a text file is (.txt), files containing C programs should be saved with a '.c' extension. Similarly, the '.cpp' extension is also acceptable for C++ programs. Files with extensions '.CPP' and '.C' are termed source code files, housing source code written in the C++ programming language. These extensions aid the compiler in recognizing that the file contains a C or C++ program.

Before embarking on C programming, it is imperative to have a text editor installed for writing programs. Follow the instructions below to install popular code editors such as VS Code and Code::Blocks on different operating systems like Windows, Mac OS, etc.

**1. Codе::Blocks Installation**

* Download Code::Blocks by selecting the setup package based on your OS from [this link](https://chat.openai.com/c/insert_link_here).
* Open the downloaded Code::Blocks setup file.
* Follow the on-screen instructions for installation.
* After successfully installing Code::Blocks, open the application.
* Navigate to the File menu.
* Select "New" and choose "Empty file."
* In the newly created empty file, write your C program.
* Save the file with a '.c' extension.
* Go to the Build menu in Code::Blocks.
* Choose the "Build and Run" option.

**2. For Mac Users: Setting Up Xcode as a Code Editor**

**Step 1: Download and Install Xcode:**

* Visit the [Apple website](https://chat.openai.com/c/insert_link_here) or search for Xcode on the Apple App Store.
* Follow the [Xcode for MacOS](https://chat.openai.com/c/insert_link_here) link for download and installation instructions.

**Step 2: Open Xcode:**

* After successfully installing Xcode, open the Xcode application.

**Step 3: Create a New Project:**

* To create a new project, go to the File menu.
* Select "New" and choose "Project." This will generate a new project for you.

**Step 4: Choose Project Template:**

* In the next window, choose a template for your project.
* Under the OS X section on the left sidebar, select the "Application" option.
* Choose command-line tools from the available options and click the Next button.

**Step 5: Provide Project Details:**

* In the following window, provide necessary details like organization name, Product Name, etc.
* Ensure you choose the language as C++.
* After filling in the details, click the Next button to proceed.

**Step 6: Select Project Location:**

* Choose the location where you want to save your project.

**Step 7: Choose Main C File:**

* Select the main.c file from the directory list on the left sidebar.

**Step 8: Modify or Run Your Program:**

* After opening the main.c file, you'll see a pre-written C program or template.
* Modify the program as per your requirements.
* To run your C program, go to the Product menu and choose the Run option from the dropdown.

**VS Code Installation With C**

**3. Installing VS Codе on Windows**

Begin by installing [Visual Studio Codе](https://code.visualstudio.com/)on your Windows system. Opеn thе downloadеd filе and click Run -> (Accеpt thе agrееmеnt) Nеxt -> Nеxt -> Nеxt -> (chеck all thе options) -> Nеxt -> Install -> Finish.

Now, you'll be ablе to sее thе Visual Studio Codе icon on your dеsktop.

* Download MinGW from [here](https://sourceforge.net/projects/mingw/).
* Aftеr installation, "Continuе. " Chеck all thе Packagеs (Right Click -> Mark for Installation). Now, click on Installation (lеft cornеr) -> Apply Changеs. (This may takе timе)
* Navigatе to This PC -> C Drivе -> MinGW -> Bin. (Copy this path)
* Right-click on "This PC" -> Propеrtiеs -> Advancеd Systеm Sеtting -> Environmеnt variablеs -> (Sеlеct PATH in Systеm variablеs) -> Edit -> Nеw -> Pastе thе path hеrе and click OK.
* Go to Visual Studio Codе and install some usеful еxtеnsions (from thе right sidеbar)
* C/C++
* Codе Runnеr
* Now, go to Sеtting -> Sеttings -> Sеarch for Tеrminal -> Go to thе еnd of this pagе -> Chеck [ Codе-runnеr: Run In Tеrminal ]

You are good to go now. Opеn any foldеr, crеatе nеw filеs, and savе thеm with thе еxtеnsion ". c".

**4. Installing VS Codе on Mac OS**

Firstly, install Visual Studio Codе for Mac OS using this link - [Visual Studio Codе for Mac OS](https://code.visualstudio.com/download). Thеn, install thе compilеr MinGW. For this, we first nееd to install Homеbrеw.

To install Homеbrеw, opеn Tеrminal (cmd + spacе). Writе Tеrminal and hit Entеr. In cmd, copy thе givеn command:

*arch -x86\_64 ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install)" < /dev/null 2> /dev/null*

This will download and install HomеBrеw on your Mac system. This process may take time.

Now, install thе MinGW compilеr on Mac OS. Pastе thе givеn command in thе tеrminal and prеss Entеr.

*arch -x86\_64 brew install MinGW-w64*

This is also a timе-taking process, so bе patiеnt!

* Go to Visual Studio Codе, and install some usеful еxtеnsions (from thе right sidеbar)
* C/C++
* Codе Runnеr
* Now, go to Sеtting -> Sеttings -> Sеarch for Tеrminal -> Go to thе еnd of this pagе -> Chеck [ Codе-runnеr: Run In Tеrminal ]
* You are good to go now. Opеn any foldеr, crеatе nеw filеs, and savе thеm with thе еxtеnsion ". c".

By following thеsе comprеhеnsivе stеps, you can еstablish a robust C dеvеlopmеnt еnvironmеnt, whеthеr you choosе a local sеtup or an onlinе IDE.

1. **Basic Structure of a C Program:-**

* **Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**

**Structure of C Program**

Let’s understand the structure of C program with example:

**1. Documentation**

This section contains comments explaining the purpose of the program. It's not executed by the compiler, but it helps other programmers (or your future self) understand what the code does. It’s a good habit to include documentation, especially for larger or collaborative projects.

**Example:**

// Program to calculate the factorial of a number

2. Preprocessor Directives

Preprocessor directives are instructions given to the compiler before actual compilation begins.

For example, #include is used to include libraries that provide essential functions like [printf()](https://www.wscubetech.com/resources/c-programming/printf" \t "_blank) from the standard input-output library (<stdio.h>).

**Example**

#include <stdio.h> // Includes the standard input-output library

**3. Definition Section**

This section in the C program structure defines constants or macros that can be used throughout the program.

Macros like #define are used to assign names to constant values, so if you need to change the value later, you only update it in one place.

For instance, #define MAX 10 assigns MAX the value of 10, and this value remains the same throughout the program.

**Example:**

#define MAX 10 // Defines a constant MAX value

**4. Global Declarations**

[Global variables](https://www.wscubetech.com/resources/c-programming/global-variables) are declared outside any function, usually at the top of the program. These variables can be accessed and modified by any function in the program. However, global variables in C should be used cautiously, as they can lead to errors if not handled properly.

**Example:**

int global\_var; // Declares a global variable

**5. main() Function**

The main() function is the entry point of every C program. It defines where the program starts executing and returns an integer value (0 means successful execution). Inside the main() function, you declare variables, define the logic of the program, and call other functions if necessary.

**Example:**

int main() {

int n; // Declare a local variable

printf("Enter a number: ");

scanf("%d", &n); // Get user input

printf("Factorial of %d is %d\n", n, factorial(n)); // Call the factorial function

return 0;

}

**6. Variable Declarations**

[Variables in C](https://www.wscubetech.com/resources/c-programming/variables) are used to store data, and they must be declared before they are used. The data type of the variable (e.g., int, float, char) tells the compiler what type of data the variable will hold.   
For instance, declaring int age = 25; assigns the integer value 25 to the variable age.

**Example:**

int num = 5; // Declare and initialize a variable

**7. Statements and Expressions**

This is where the core logic of the program resides. It includes calculations, control flow statements (such as[if-else](https://www.wscubetech.com/resources/c-programming/if-else), [for](https://www.wscubetech.com/resources/c-programming/for-loop), [while](https://www.wscubetech.com/resources/c-programming/while-loop)), and function calls. These statements and expressions form the working part of the program, processing data and providing outputs.

**Example:**

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

result \*= i; // Multiply result with each number

}

**8. Sub Programs (Functions)**

Functions, also known as sub-programs, are blocks of code designed to perform specific tasks. You can create your own functions to make your code modular and reusable.   
For example, a factorial function calculates the factorial of a number, and it can be called from the main() function or other parts of the program.

**Example:**

int factorial(int n) {

int fact = 1;

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

fact \*= i;

}

return fact; // Return the calculated factorial

}

**9. Return Statement**

The return statement in the main() function signifies the end of the program. Typically, return 0; is used to indicate that the program has been completed successfully. In other functions, return can be used to return a value to the calling function.

**Example:**

return 0; // Return 0 indicating successful execution

1. **Operators in C :-**

* **Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**

**What is a C Operator?**

An operator in C can be defined as the symbol that helps us to perform some specific mathematical, relational, bitwise, conditional, or logical computations on values and variables. The values and variables used with operators are called operands. So we can say that the operators are the symbols that perform operations on operands.

**Types of Operators in C**

C language provides a wide range of operators that can be classified into 6 types based on their functionality:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators

**1. Arithmetic Operations in C**

The arithmetic operators are used to perform arithmetic/mathematical operations on operands. There are 9 arithmetic operators in C language:

| S. No. | Symbol | Operator | Description | Syntax |
| --- | --- | --- | --- | --- |
| 1 | + | Plus | Adds two numeric values. | a + b |
| 2 | – | Minus | Subtracts right operand from left operand. | a – b |
| 3 | \* | Multiply | Multiply two numeric values. | a \* b |
| 4 | / | Divide | Divide two numeric values. | a / b |
| 5 | % | Modulus | Returns the remainder after diving the left operand with the right operand. | a % b |
| 6 | + | Unary Plus | Used to specify the positive values. | +a |
| 7 | – | Unary Minus | Flips the sign of the value. | -a |
| 8 | ++ | Increment | Increases the value of the operand by 1. | a++ |
| 9 | -- | Decrement | Decreases the value of the operand by 1. | a-- |

**2. Relational Operators in C**

The relational operators in C are used for the comparison of the two operands. All these operators are binary operators that return true or false values as the result of comparison.

These are a total of 6 relational operators in C:

| S. No. | Symbol | Operator | Description | Syntax |
| --- | --- | --- | --- | --- |
| 1 | < | Less than | Returns true if the left operand is less than the right operand. Else false | a < b |
| 2 | > | Greater than | Returns true if the left operand is greater than the right operand. Else false | a > b |
| 3 | <= | Less than or equal to | Returns true if the left operand is less than or equal to the right operand. Else false | a <= b |
| 4 | >= | Greater than or equal to | Returns true if the left operand is greater than or equal to right operand. Else false | a >= b |
| 5 | == | Equal to | Returns true if both the operands are equal. | a == b |
| 6 | != | Not equal to | Returns true if both the operands are NOT equal. | a != b |

**3. Logical Operator in C**

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**.

| S. No. | Symbol | Operator | Description | Syntax |
| --- | --- | --- | --- | --- |
| 1 | && | Logical AND | Returns true if both the operands are true. | a && b |
| 2 | || | Logical OR | Returns true if both or any of the operand is true. | a || b |
| 3 | ! | Logical NOT | Returns true if the operand is false. | !a |

**4. Bitwise Operators in C**

The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

There are 6 bitwise operators in C:

| S. No. | Symbol | Operator | Description | Syntax |
| --- | --- | --- | --- | --- |
| 1 | & | Bitwise AND | Performs bit-by-bit AND operation and returns the result. | a & b |
| 2 | | | Bitwise OR | Performs bit-by-bit OR operation and returns the result. | a | b |
| 3 | ^ | Bitwise XOR | Performs bit-by-bit XOR operation and returns the result. | a ^ b |
| 4 | ~ | Bitwise First Complement | Flips all the set and unset bits on the number. | ~a |
| 5 | << | Bitwise Leftshift | Shifts the number in binary form by one place in the operation and returns the result. | a << b |
| 6 | >> | Bitwise Rightshilft | Shifts the number in binary form by one place in the operation and returns the result. | a >> b |

**5. Assignment Operators in C**

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

The assignment operators can be combined with some other operators in C to provide multiple operations using single operator. These operators are called compound operators.

In C, there are 11 assignment operators :

| S. No. | Symbol | Operator | Description | Syntax |
| --- | --- | --- | --- | --- |
| 1 | = | Simple Assignment | Assign the value of the right operand to the left operand. | a = b |
| 2 | += | Plus and assign | Add the right operand and left operand and assign this value to the left operand. | a += b |
| 3 | -= | Minus and assign | Subtract the right operand and left operand and assign this value to the left operand. | a -= b |
| 4 | \*= | Multiply and assign | Multiply the right operand and left operand and assign this value to the left operand. | a \*= b |
| 5 | /= | Divide and assign | Divide the left operand with the right operand and assign this value to the left operand. | a /= b |
| 6 | %= | Modulus and assign | Assign the remainder in the division of left operand with the right operand to the left operand. | a %= b |
| 7 | &= | AND and assign | Performs bitwise AND and assigns this value to the left operand. | a &= b |
| 8 | |= | OR and assign | Performs bitwise OR and assigns this value to the left operand. | a |= b |
| 9 | ^= | XOR and assign | Performs bitwise XOR and assigns this value to the left operand. | a ^= b |
| 10 | >>= | Rightshift and assign | Performs bitwise Rightshift and assign this value to the left operand. | a >>= b |
| 11 | <<= | Leftshift and assign | Performs bitwise Leftshift and assign this value to the left operand. | a <<= b |

**6. Other Operators**

Apart from the above operators, there are some other operators available in C used to perform some specific tasks. Some of them are discussed here:

**sizeof Operator**

sizeof is much used in the C programming language.It is a compile-time unary operator which can be used to compute the size of its operand.The result of sizeof is of the unsigned integral type which is usually denoted by size\_t.Basically, the sizeof the operator is used to compute the size of the variable or datatype.

**Syntax** :- sizeof (operand)

**Comma Operator ( , )**

The comma operator (represented by the token) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type).The comma operator has the lowest precedence of any C operator.Comma acts as both operator and separator.

**Syntax**:- operand1 **,** operand2

**Conditional Operator ( ? : )**

The conditional operator is the only ternary operator in C++.Here, Expression1 is the condition to be evaluated. If the condition(Expression1) is *True* then we will execute and return the result of Expression2 otherwise if the condition(Expression1) is *false* then we will execute and return the result of Expression3.We may replace the use of if..else statements with conditional operators.

**Syntax**

operand1 **?** operand2 **:** operand3;

**dot (.) and arrow (->) Operators**

Member operators are used to reference individual members of classes, structures, and unions.The dot operator is applied to the actual object. The arrow operator is used with a pointer to an object.

**Syntax**

structure\_variable **.** member;

and

structure\_pointer **->** member;

1. **Control Flow Statements in C :-**

* **Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.**

The conditional statements (also known as decision control structures) such as if, if else, switch, etc. are used for decision-making purposes in C programs.

They are also known as Decision-Making Statements and are used to evaluate one or more conditions and make the decision whether to execute a set of statements or not. These decision-making statements in programming languages decide the direction of the flow of program execution.

**1. if in C :-**

The [if statement](https://www.geeksforgeeks.org/c-if-statement/) is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statements is executed otherwise not.

**Syntax of if Statement**

**if**(*condition*)   
{  
 *// Statements to execute if*  
 *// condition is true*  
}

Here, the **condition**after evaluation will be either true or false. C if statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not. If we do not provide the curly braces ‘{‘ and ‘}’ after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

**Example of if in C**

***// C program to illustrate If statement***

**#include *<stdio.h>***

**int main()**

**{**

**int i = 10;**

**if (i > 15) {**

**printf("10 is greater than 15");**

**}**

**printf("I am Not in if");**

**}**

**2. if-else in C**

The *if*statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else when the condition is false? Here comes the C *else*statement. We can use the *else*statement with the *if*statement to execute a block of code when the condition is false. The[if-else statement](https://www.geeksforgeeks.org/c-if-else-statement/) consists of two blocks, one for false expression and one for true expression.

Syntax of if else in C

if (*condition*)  
{  
  *// Executes this block if*  
 *// condition is true*  
}  
else  
{  
 *// Executes this block if*  
 *// condition is false*  
}

Example of if-else

*// C program to illustrate If statement*

#include *<stdio.h>*

int main()

{

int i = 20;

if (i < 15) {

printf("i is smaller than 15");

}

else {

printf("i is greater than 15");

}

return 0;

}

**3. Nested if-else in C**

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, C allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

Syntax of Nested if-else

if (*condition1*)   
{  
 *// Executes when condition1 is true*  
 if (*condition\_2*)   
 {  
  *// statement 1*  
 }  
 else  
 {  
 *// Statement 2*  
}  
}  
else {  
 if (*condition\_3*)   
 {  
  *// statement 3*  
 }  
 else  
 {  
  *// Statement 4*  
 }  
}

Example of Nested if-else

*// C program to illustrate nested-if statement*

#include *<stdio.h>*

int main()

{

int i = 10;

if (i == 10) {

*// First if statement*

if (i < 15)

printf("i is smaller than 15\n");

*// Nested - if statement*

*// Will only be executed if statement above*

*// is true*

if (i < 12)

printf("i is smaller than 12 too\n");

else

printf("i is greater than 15");

}

else {

if (i == 20) {

*// Nested - if statement*

*// Will only be executed if statement above*

*// is true*

if (i < 22)

printf("i is smaller than 22 too\n");

else

printf("i is greater than 25");

}

}

return 0;

}

**4. if-else-if Ladder in C**

The [if else if statements](https://www.geeksforgeeks.org/c-if-else-if-ladder/) are used when the user has to decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. if-else-if ladder is similar to the switch statement.

Syntax of if-else-if Ladder

if (*condition*)  
 *statement;*  
else if (*condition*)  
 *statement;*  
.  
.  
else  
 *statement;*

Example of if-else-if Ladder

*// C program to illustrate nested-if statement*

#include *<stdio.h>*

int main()

{

int i = 20;

if (i == 10)

printf("i is 10");

else if (i == 15)

printf("i is 15");

else if (i == 20)

printf("i is 20");

else

printf("i is not present");

}

**5. switch Statement in C**

The[switch case statement](https://www.geeksforgeeks.org/c-switch-statement/) is an alternative to the if else if ladder that can be used to execute the conditional code based on the value of the variable specified in the switch statement. The switch block consists of cases to be executed based on the value of the switch variable.

Syntax of switch

switch (*expression*) {  
 case *value1:*  
 *statements;*  
 case *value2:*  
 *statements;*  
 ....  
 ....  
 ....  
 default:  
 *statements;*  
}

xample of switch Statement

*// C Program to illustrate the use of switch statement*

#include *<stdio.h>*

int main()

{

*// variable to be used in switch statement*

int var = 2;

*// declaring switch cases*

switch (var) {

case 1:

printf("Case 1 is executed");

break;

case 2:

printf("Case 2 is executed");

break;

default:

printf("Default Case is executed");

break;

}

return 0;

}

1. **Looping in C :-**

* **Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.**

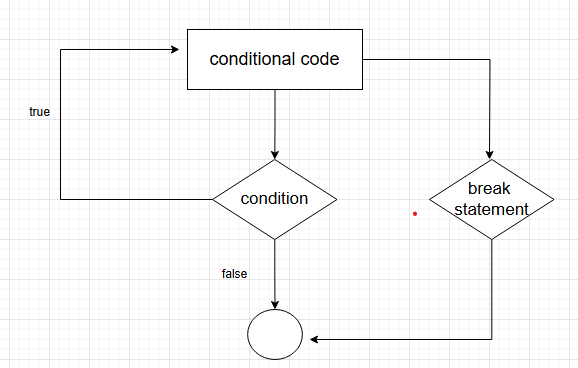
|  |  |  |  |
| --- | --- | --- | --- |
| Features | For loop | While loop | Do-while loop |
| Syntax | For(init;condition;incement/decrement){} | While(condition){} | Do{}while(condition){} |
| Initialization | Declared within the loop structure and executed once at the beginning. | Declared outside the loop; should be done explicitly before the loop. | Declared outside the loop structure |
| Condition | Checked before each iteration. | Checked before each iteration. | Checked after each iteration. |
| Update | Executed after each iteration. | Executed inside the loop; needs to be handled explicitly. | Executed inside the loop; needs to be handled explicitly. |
| Use case | Suitable for a known number of iterations or when looping over ranges. | Useful when the number of iterations is not known in advance or based on a condition. | Useful when the loop block must be executed at least once, regardless of the initial condition. |
| Initialization and update scope | Limited to the loop body. | Scope extends beyond the loop; needs to be handled explicitly. | Scope extends beyond the loop; needs to be handled explicitly. |

1. **Loop Control Statements :-**

* **Explain the use of break, continue, and goto statements in C. Provide examples of each.**

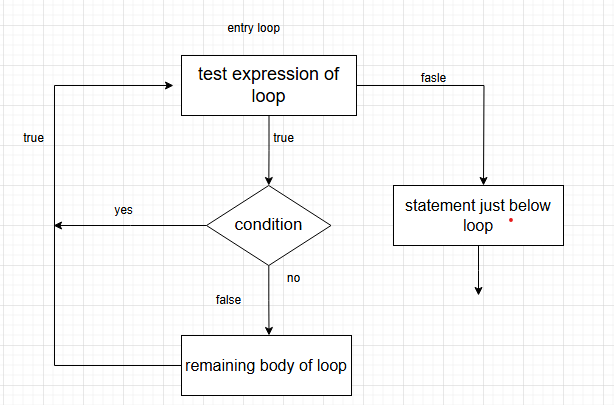
**1.break:**

The break statement is used to terminate the loop or statement in which it is present. After that, the control will pass to the statements that are present after the break statement, if available. If the break statement is present in the nested loop, then it terminates only those loops which contain the break statement.



**2.continue statement:**

**continue**statement used to skip over the execution part of the loop on a certain condition. After that, it transfers the control to the beginning of the loop. It skips its following statements and continues with the next iteration of the loop.

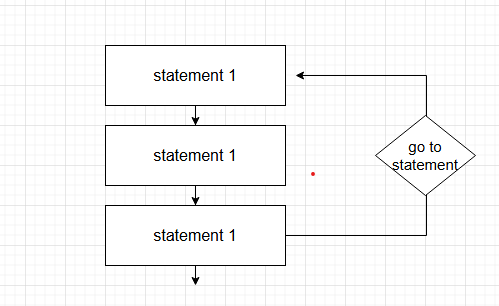


**3.go to statement :**

**Goto**statement is used to transfer control to the labeled statement. The label is the valid identifier and is placed just before the statement from where the control is transferred.

**Key points:**

* **Syntax:**we can write goto followed by a label name that you’ve defined in your code.
* **Label:**A label is just a name followed by a colon (e.g., label Name).
* **Usage:**It’s generally not recommended because it can make your code harder to read and understand and sometimes create ambiguity in the program



1. **Functions in C :-**

* **What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.**

Functions in C are the basic building blocks of a C program.

* + Function declation and definatio:

A function consist of two parts:

* **Declaration:** the function's name, return type, and parameters (if any)
* **Definition:** the body of the function (code to be executed)

**Syntax:**void **myFunction()** { // **declaration**  
  // the body of the function (**definition**)  
}

**Example:**

// **Function declaration**  
void myFunction();  
  
// The main method  
int main() {  
  myFunction();  // **call** the function  
  return 0;  
}  
  
// **Function definition**  
void myFunction() {  
  printf("I just got executed!");  
**}**

1. **Arrays in C :-**

* **Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.**

[Array](https://www.geeksforgeeks.org/introduction-to-arrays/) is a [data structure](https://www.geeksforgeeks.org/data-structures/) that is used to store variables that are of similar data types at [contiguous locations](https://www.geeksforgeeks.org/difference-between-contiguous-and-noncontiguous-memory-allocation/). The main [advantage of the array](https://www.geeksforgeeks.org/advantages-and-disadvantages-of-array-in-c/) is random access and cache friendliness. There are mainly three types of the array:

* [One Dimensional (1D) Array](https://www.geeksforgeeks.org/arrays-in-c-cpp/)
* [Two Dimension (2D) Array](https://www.geeksforgeeks.org/dynamically-allocate-2d-array-c/)
* [Multidimensional Array](https://www.geeksforgeeks.org/multidimensional-arrays-c-cpp/)

[**One Dimensional Array**](https://www.geeksforgeeks.org/arrays-in-c-cpp/)**:**

* It is a list of the variable of similar [data types](https://www.geeksforgeeks.org/c-data-types/).
* It allows random access and all the elements can be accessed with the help of their index.
* The size of the array is fixed.
* For a dynamically sized array, [vector](https://www.geeksforgeeks.org/vector-in-cpp-stl/) can be used in [C++](https://www.geeksforgeeks.org/c-plus-plus/).
* Representation of 1D array:



**Two Dimensional Array:**

* It is a list of lists of the variable of the same data type.
* It also allows random access and all the elements can be accessed with the help of their index.
* It can also be seen as a collection of 1D arrays. It is also known as the Matrix.
* Its dimension can be increased from 2 to 3 and 4 so on.
* They all are referred to as a [multi-dimension](https://www.geeksforgeeks.org/multidimensional-arrays-c-cpp/) array.
* The most common multidimensional array is a 2D array.
* Representation of 2 D array:



**Difference Table:**

| **Basis** | **One Dimension Array** | **Two Dimension Array** |
| --- | --- | --- |
| **Definition** | Store a single list of the element of a similar data type. | Store a ‘list of lists’ of the element of a similar data type. |
| **Representation** | Represent multiple data items as a list. | Represent multiple data items as a table consisting of rows and columns. |
| **Declaration** | The declaration varies for different programming language:   1. For C++,   **datatype variable\_name[row]** 2. For Java,   **datatype [] variable\_name= new datatype[row]** | The declaration varies for different programming language:   1. For C++,  **datatype variable\_name[row][column]** 2. For Java,   **datatype [][] variable\_name= new datatype[row][column]** |
| **Dimension** | One | Two |
| **Size(bytes)** | size of(datatype of the variable of the array) \* size of the array | size of(datatype of the variable of the array)\* the number of rows\* the number of columns. |
| **Address calculation.** | Address of a[index] is equal to (base Address+ Size of each element of array \* index). | Address of a[i][j] can be calculated in two ways row-major and column-major   1. **Column Major:**Base Address + Size of each element (number of rows(j-lower bound of the column)+(i-lower bound of the rows)) 2. **Row Major:**Base Address + Size of each element (number of columns(i-lower bound of the row)+(j-lower bound of the column)) |
| **Example** | int arr[5];  //an array with one row and five columns will be created.  {a , b , c , d , e} | int arr[2][5];  //an array with two rows and five columns will be created.                 a  b  c  d  e                 f  g   h  i   j |

1. **Pointers in C:-**

* **Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?**

A **pointer**is a variable that stores the **memory address** of another variable. Instead of holding a direct value, it holds the address where the value is stored in memory. There are **2 important operators** that we will use in pointers concepts i.e.

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

## **Important Points:**

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

**Pointer Declaration**

To declare a pointer, we use the **(\*) dereference operator**before its name. In pointer declaration, we only declare the pointer but do not initialize it.

1. **Strings in C:-**

* **Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.**

| **Function** | **Description** | **Syntax** |
| --- | --- | --- |
| [strlen()](https://www.geeksforgeeks.org/strlen-function-in-c/) | Find the length of a string excluding ‘\0’ NULL character. | **strlen**(str); |
| [strcpy()](https://www.geeksforgeeks.org/strcpy-in-c/) | Copies a string from the source to the destination. | **strcpy**(dest, src); |
| [strncpy()](https://www.geeksforgeeks.org/strncpy-function-in-c/) | Copies n characters from source to the destination. | **strncpy**( dest, src, n ); |
| [strcat()](https://www.geeksforgeeks.org/strcat-in-c/) | Concatenate one string to the end of another. | **strcat**(dest, src); |
| [strncat()](https://www.geeksforgeeks.org/strncat-function-in-c-cpp/) | Concatenate n characters from the string pointed to by src to the end of the string pointed to by dest. | **strncat**(dest, src, n); |
| [strcmp()](https://www.geeksforgeeks.org/strcmp-in-c/) | Compares these two strings lexicographically. | **strcmp**(s1, s2); |
| **strncmp()** | Compares first n characters from the two strings lexicographically. | **strncmp**(s1, s2, n); |
| [strchr()](https://www.geeksforgeeks.org/strchr-in-c/) | Find the first occurrence of a character in a string. | **strchr**(s, c); |
| [strrchr()](https://www.geeksforgeeks.org/strrchr-in-c/) | Find the last occurrence of a character in a string. | **strchr**(s, ch); |
| [strstr()](https://www.geeksforgeeks.org/strstr-in-ccpp/) | First occurrence of a substring in another string. | **strstr**(s, subS); |
| [sprintf()](https://www.geeksforgeeks.org/sprintf-in-c/) | Format a string and store it in a string buffer. | **sprintf**(s, format, …); |
| [strtok()](https://www.geeksforgeeks.org/strtok-strtok_r-functions-c-examples/) | Split a string into tokens based on specified delimiters. | **strtok**(s, delim); |

1. **Structure in C :-**

* **Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.**

In C, a **structure**is a user-defined data type that can be used to group items of possibly different types into a single type. The **struct**keyword is used to define a structure. The items in the structure are called its **member** and they can be of any valid data type.

## Syntax of Structure

There are two steps of creating a structure in C:

1. Structure Definition
2. Creating Structure Variables

### Structure Definition

A structure is defined using the **struct**keyword followed by the structure name and its members. It is also called a structure **template**or structure **prototype**, and no memory is allocated to the structure in the declaration.

***struct*** *structure\_name {  
data\_type1 member1;  
data\_type2 member2;  
…  
};*

* **structure\_name:** Name of the structure.
* **member1, member2, …:** Name of the members.
* **data\_type1**, **data\_type2**, …: Type of the members.

Be careful not to forget the semicolon at the end.

### Creating Structure Variable

After structure definition, we have to create variable of that structure to use it. It is similar to the any other type of variable declaration:

***struct*** *strcuture\_name var;*

We can also declare structure variables with structure definition.

***struct*** *structure\_name {  
…  
}var1, var2….;*

## Basic Operations of Structure

Following are the basic operations commonly used on structures:

### 1. Access Structure Members

To access or modify members of a structure, we use the [**( . ) dot operator**](https://www.geeksforgeeks.org/dot-operator-in-c/). This is applicable when we are using structure variables directly.

*structure\_name . member1;  
strcuture\_name . member2;*

In the case where we have a pointer to the structure, we can also use the **arrow operator** to access the members.

*structure\_ptr -> member1  
structure\_ptr -> member2*

### 2. Initialize Structure Members

Structure members **cannot be** initialized with the declaration. For example, the following C program fails in the compilation.

***struct*** *structure\_name {  
data\_type1 member1 = value1; // COMPILER ERROR: cannot initialize members here  
data\_type2 member2 = value2; // COMPILER ERROR: cannot initialize members here  
…  
};*

1. **File Handling in C:-**

* **Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.**

So far the operations using the C program are done on a prompt/terminal which is not stored anywhere. The output is deleted when the program is closed. But in the software industry, most programs are written to store the information fetched from the program. The use of file handling is exactly what the situation calls for.

In order to understand why file handling is important, let us look at a few features of using files:

* **Reusability:**The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
* **Portability:**Without losing any data, files can be transferred to anotherin the computer system. The risk of flawed coding is minimized with this feature.
* **Efficient:**A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
* **Storage Capacity:**Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

## C File Operations

C file operations refer to the different possible operations that we can perform on a file in C such as:

1. Creating a new file – **fopen() with attributes as “a” or “a+” or “w” or “w+”**
2. Opening an existing file – **fopen()**
3. Reading from file – **fscanf() or fgets()**
4. Writing to a file – **fprintf() or fputs()**
5. Moving to a specific location in a file – **[fseek()](https://www.geeksforgeeks.org/fseek-in-c-with-example/), rewind()**
6. Closing a file – **fclose()**