

Module 2 – Introduction to Programming

1. Overview of C Programming

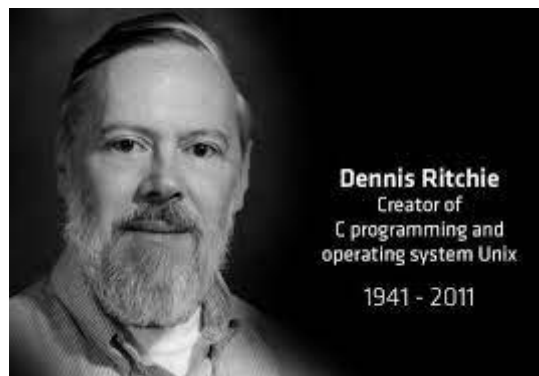
- **THEORY EXERCISE:**

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

C programming language was developed by Dennis Ritchie in **1972 at Bell Labs**. It was derived from the earlier language called B, which was developed by Ken Thompson. The purpose of C language was to create a more efficient and powerful language for writing operating systems. In the early days, C was used to write system software like the Unix operating system and the C compiler, which was used to compile the source code of C programs.

Over the years, C has become a very popular and versatile language, used for a variety of applications. It is used for writing device drivers, gaming applications, embedded systems, and operating systems. It is also used for creating graphical user interfaces, network programming, database management systems, web development, and many other tasks.

C has also been used to create various high-level programming languages such as C++, Java, and C#. These languages are based on the principles of C and provide a more user-friendly and modern syntax.



C language was developed in the **1970s** by **Dennis Ritchie**, who was an American computer scientist at **AT & T Bell Labs in USA**.

C language was created as a replacement for the 'B' language, which was used for writing operating systems.

C language was designed to be a general-purpose, high-level programming language that is both powerful and efficient.

C language is often used for system programming and embedded systems development, as it provides low-level access to memory and allows for efficient manipulation of hardware resources.

C language has become one of the most widely used programming languages, and it is used for a variety of applications, including operating systems, device drivers, and embedded systems.

- LAB EXERCISE:

Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.

C, the venerable programming language, continues to be a powerhouse in the world of software development. Its efficiency, portability, and low-level control make it an ideal choice for a wide range of applications. In this article, we'll explore real-life examples of C in action, demonstrating its practical applications across various domains.

Table of Contents

- [1. Embedded Systems: Smart Thermostat](#)
- [2. System Programming: Simple File Compression](#)
- [3. Networking: Simple Chat Server](#)

2. Setting Up Environment

- **THEORY EXERCISE:**

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

Install C in Your Computer

Installing C in your own computer requires downloading and installing two software that together forms the C development environment:

- **C Compiler:** A compiler is a software that converts your C code into executable programs.
- **Text Editor (IDE):** A compiler only compiles the C source code into executable code. We also need a code editor (or IDE) for editing and writing C code.

Install C Compiler

There are many C compilers provided by different vendors available. Below are the instructions for installing GCC (GNU Compiler Collection) on different operating systems.

Install GCC on Linux

To [install GCC on Linux](#) (Ubuntu, Debian, Linux Mint, and Kali Linux), you can use the **apt** package manager. Just enter and run the following commands one by one in the terminal:

```
sudo apt update
```

```
sudo apt install build-essential
```

The above command will download and install GCC in your system.

Install GCC on Windows

To **install GCC on Windows**, we can use MinGW (Minimalist GNU for Windows) installer. Follow these steps:

1. First [download MinGW](#).
2. Run the installer and select the GCC package.

3. Once installed, [add the MinGW\bin directory to the system's PATH](#) variable to be able to use GCC from command line.

Install GCC on Mac

On macOS, GCC can be installed using Homebrew package manager. First install Homebrew in macOS using the following command:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Then enter the following command to install GCC:

```
brew install gcc
```

Verify Installation

We can check if the GCC compiler is successfully installed on the operating system by using the following command in the command line interface:

```
gcc --version
```

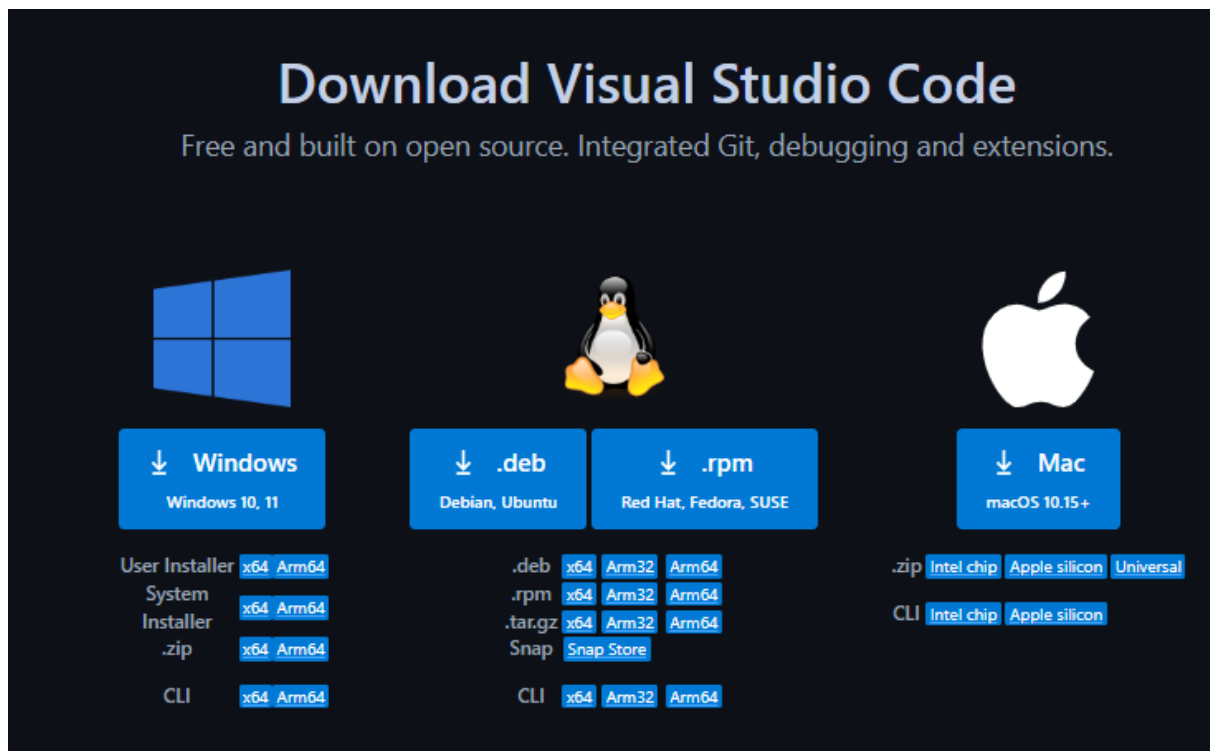
If the installation was successful, below text is shown:

```
C:\Users\ritik>gcc --version
gcc (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Install Code Editor (IDE)

There are many IDEs available on the internet for free use. One popular code editor is VS Code, which is an excellent and lightweight code editor. Follow the below steps to install VS code:

Step 1: Download the suitable VS Code installer according to your **operating system**.



Step 2: Once downloaded, open it and install the VS Code with recommended settings.

For installing different IDEs, you can refer to the articles given below:

- [Install Atom Text Editor in Linux](#)
- [Install Code Blocks on Windows](#)
- [Install Code Blocks on Linux](#)
- [Install Sublime Text in Windows](#)

- LAB EXERCISE:

Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.

Input:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("Hello World");
    getch();
}
```

3. Basic Structure of a C Program

• THEORY EXERCISE:

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

The basic structure of a C program is divided into 6 parts which makes it easy to read, modify, document, and understand in a particular format. C program must follow the below-mentioned outline in order to successfully compile and execute. Debugging is easier in a well-structured C program.

Sections of the C Program

There are 6 basic sections responsible for the proper execution of a program. Sections are mentioned below:

1. Documentation
2. Preprocessor Section
3. Definition
4. Global Declaration
5. Main() Function
6. Sub Programs

1. Documentation

This section consists of the description of the program, the name of the program, and the creation date and time of the program. It is specified at the start of the program in the form of comments. Documentation can be represented as:

```
// description, name of the program, programmer name, date, time etc.
```

or

```
/*  
    description, name of the program, programmer name, date, time etc.  
*/
```

Anything written as comments will be treated as documentation of the program and this will not interfere with the given code. Basically, it gives an overview to the reader of the program.

2. Preprocessor Section

All the header files of the program will be declared in the [preprocessor](#) section of the program. Header files help us to access other's improved code into our code. A copy of these multiple files is inserted into our program before the process of compilation.

Example:

```
#include<stdio.h>
#include<math.h>
```

3. Definition

Preprocessors are the programs that process our source code before the process of compilation. There are multiple steps which are involved in the writing and execution of the program. Preprocessor directives start with the '#' symbol. The #define preprocessor is used to create a constant throughout the program. Whenever this name is encountered by the compiler, it is replaced by the actual piece of defined code.

Example:

```
#define long long ll
```

4. Global Declaration

The global declaration section contains global variables, function declaration, and static variables. Variables and functions which are declared in this scope can be used anywhere in the program.

Example:

```
int num = 18;
```

5. Main() Function

Every C program must have a main function. The main() function of the program is written in this section. Operations like declaration and execution are performed inside the curly braces of the main program. The return type of the main() function can be int as well as void too. void() main tells the compiler that the program will not return any value. The int main() tells the compiler that the program will return an integer value.

Example:

```
void main()
```

or

```
int main()
```

6. Sub Programs

User-defined functions are called in this section of the program. The control of the program is shifted to the called function whenever they are called from the main or outside the main() function. These are specified as per the requirements of the programmer.

Structure of C Program with example

Example: Below C program to find the sum of 2 numbers:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    //addition
    int a=10,b=20,c;
    //data type and variable
    clrscr();
    c=a+b;
    printf("%d",c);
    getch();
}
```

- LAB EXERCISE:

Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    //declaring integer variable
    int a=10;
    //declare float variable
    float b=5.4;
    //declare character variable
    char c='A';
    //clear screen
    clrscr();
    //print all variable and contractor
    printf("Age: %d\n",a);
    printf("Grade: %c\n",c);
    printf("Height: %f\n",b);
    getch();
}
```

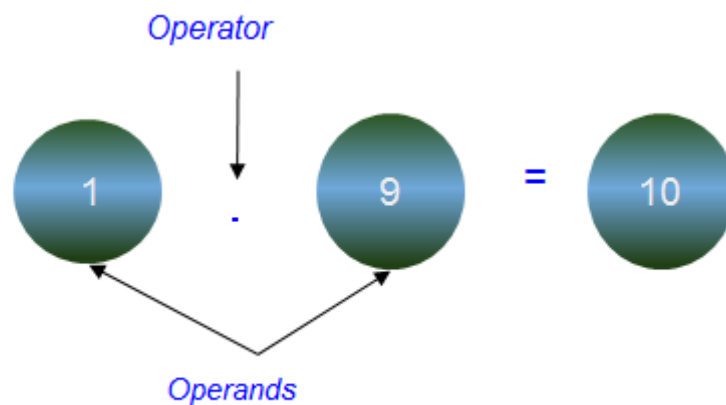
4. Operators in C

• THEORY EXERCISE:

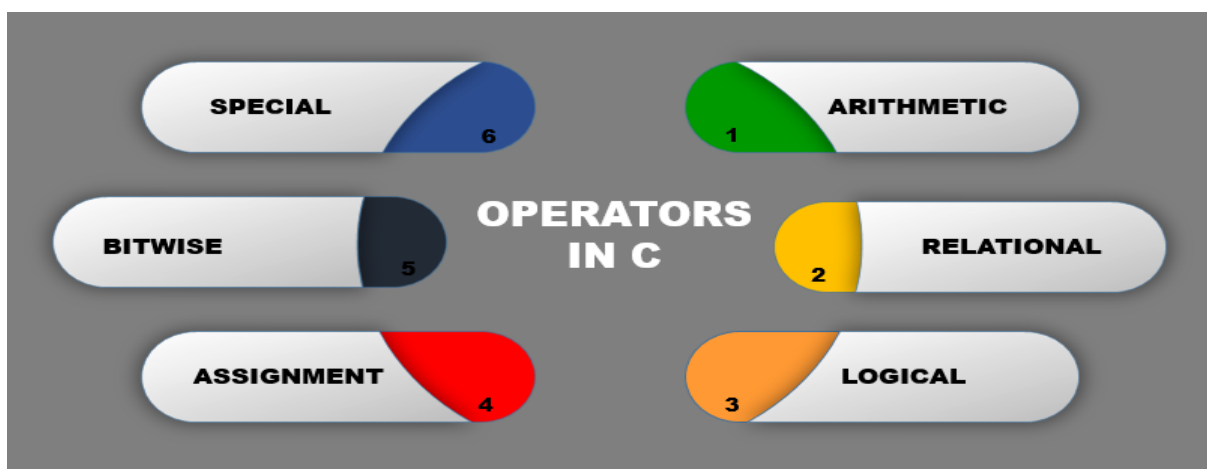
Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

In this tutorial you will grasp the skill to work with different operators used in C to perform logical and arithmetical calculations with the aid of simple & easy.

Operators are unique symbols that perform some sort of computation. The objects or values on which operators act are known as operands and the combination of operators and operands are termed as expressions in c.



C language support a wide range of built-in operators to manipulate data and values and hence is broadly categorised as follows:



Arithmetic Operators:

Like real life mathematics, arithmetic operators of C do the job of division, multiplication, addition, and subtraction. The involved operators are '/', '*', '+' and '-' respectively. Except these, there are other three operators modulus, increment and decrement operator. Modulus or '%' outputs the remainder of any division of numbers.

Operator	Meaning	Description	Example
+	Addition	Adds two operands or unary plus	$10+2=12$
-	Subtraction	Subtracts right operand from left operand or unary minus	$10-2=8$
*	Multiplication	Multiplies two operands	$10*2=20$
/	Division	Divides left operand by right operand	$10/2=5$
%	Modulus	Remainder after division	$10\%2=0$
++	Increment	increases value by one unit	<code>++a</code> or <code>a++</code> = $a+1$
--	Decrement	decreases value by one unit	<code>--b</code> or <code>b--</code> = $b-1$

Increment and Decrement Operators

Increment operator '++' increases the value of integer by one unit, whereas the decrement operator '--' decreases the same by one unit. These operators can be either prefixed or postfix with the operand and are used extensively in a different type of loops in C .

Relational Operators:

Relational operators do compare the data to give binary outputs i.e. True or False. Here are the six operators demonstrated using the two operands a and b.

Operator	Meaning	Description	Examples
==	Equal to	Returns True if two operands are equal	a==b
!=	Not Equal to	Returns True if two operands are not equal	a!=b
>	Greater than	Returns True if left operands is greater than the right	a>b
<	Less than	Returns True if left operand is less than the right	a=	Greater than or equal to	Returns True if left operands is greater than or equal to the right	a>=b
<=	Less than or equal to	Returns True if left operand is less than or equal to the right	a<=b

Logical Operators:

These operators perform binary operations to process data at machine level (logic gates like AND, OR, NOR, NAND etc.). If the result is true, it is denoted by returning '1'. The negative result is expressed by '0'. Here is the description of three basic logical operators in C which are extensively used in decision making.

Operator	Meaning	Description	Example
&&	Logical AND/ Conjunction	Returns True if and only if both statements are true	X and Y
	Logical OR / Disjunction	Returns True if any of the statement is true	X or Y
!	Logical NOT/ Negation	Returns true if operand is a negation	not X

For better understanding of Logical operators you should know about the truth table.

A	B	A && B	A B	~A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

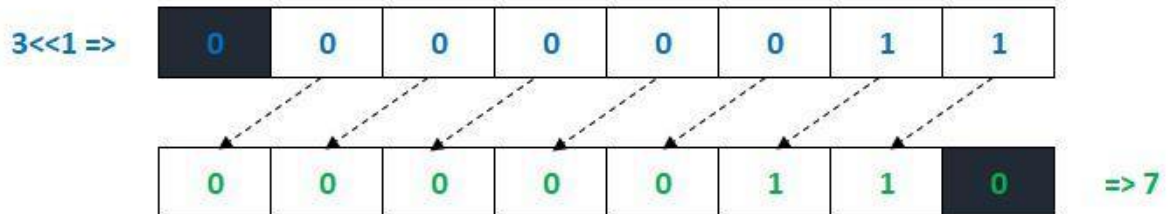
Truth Table for Logical Operators

The complement of AND is called NAND and OR is called NOR. They are used in conjunction with other operators like $A! \& B$, $A! = B$ etc.

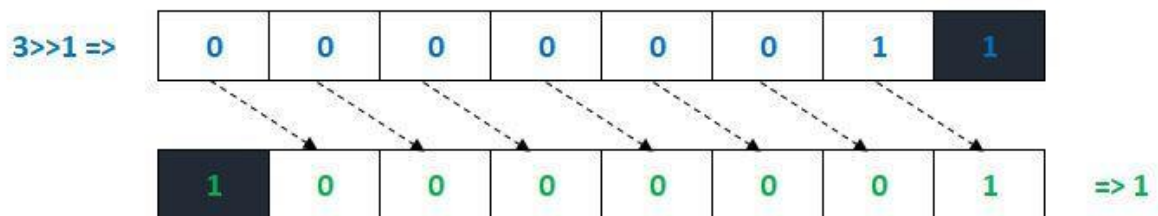
Operators	Meaning	Description
&	Binary AND	Result is 1 if both operands are true otherwise 0
	Binary OR	Result is 1 if any one operand is true otherwise 0
^	Binary XOR	Result is 1 if it's both operands are different and 0 if both operands are same
~	Binary Ones Complement	Result is the negation of the operand
<<	Binary Left Shift	Aligns the bits to the left
>>	Binary Right Shift	Aligns the bits to the right

Shift Operators in C

Bitwise Shift operators '<<' and '>>' are called binary left shift and the right shift operators respectively. The value of the operands is the left side of moved by the amount specified on the right-hand side of the operator.



Bitwise Shift Left Representation



Assignment Operators

As its name indicates, assignment operators are used to assign values to variables. '=' (equals) is the most basic type of them assigning the value at the right-hand side to the left-hand side. $C=A+B$ will impose the value of $(A+B)$ to C . Below table gives you the other assignment operators used in C to perform arithmetic operations.

Operators	Example	Meaning
=	$a = 10$	
+=	$a+=10$	$a=a+10$
-=	$a-=10$	$a=a-10$
=	$a=10$	$a=a*10$
/=	$a/=10$	$a=a/10$
%=	$a\%=10$	$a=a$

• LAB EXERCISE:

Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results. 5. Control Flow Statements in C

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("\nEnter value A:");
    scanf("%d",&a);
    printf("\nEnter value B:");
    scanf("%d",&b);
    c=a+b;
    printf("\nAddition of %d & %d is: %d ",a,b,c);
    c=a-b;
    printf("\nSubtraction of %d & %d is: %d ",a,b,c);
    c=a*b;
    printf("\nMultiplication of %d & %d is: %d ",a,b,c);
    c=a/b;
    printf("\ndivision of %d & %d is: %d ",a,b,c);
    c=a%b;
    printf("\nModule of %d & %d is: %d",a,b,c);
    getch();
}
```

5. Control Flow Statements in C

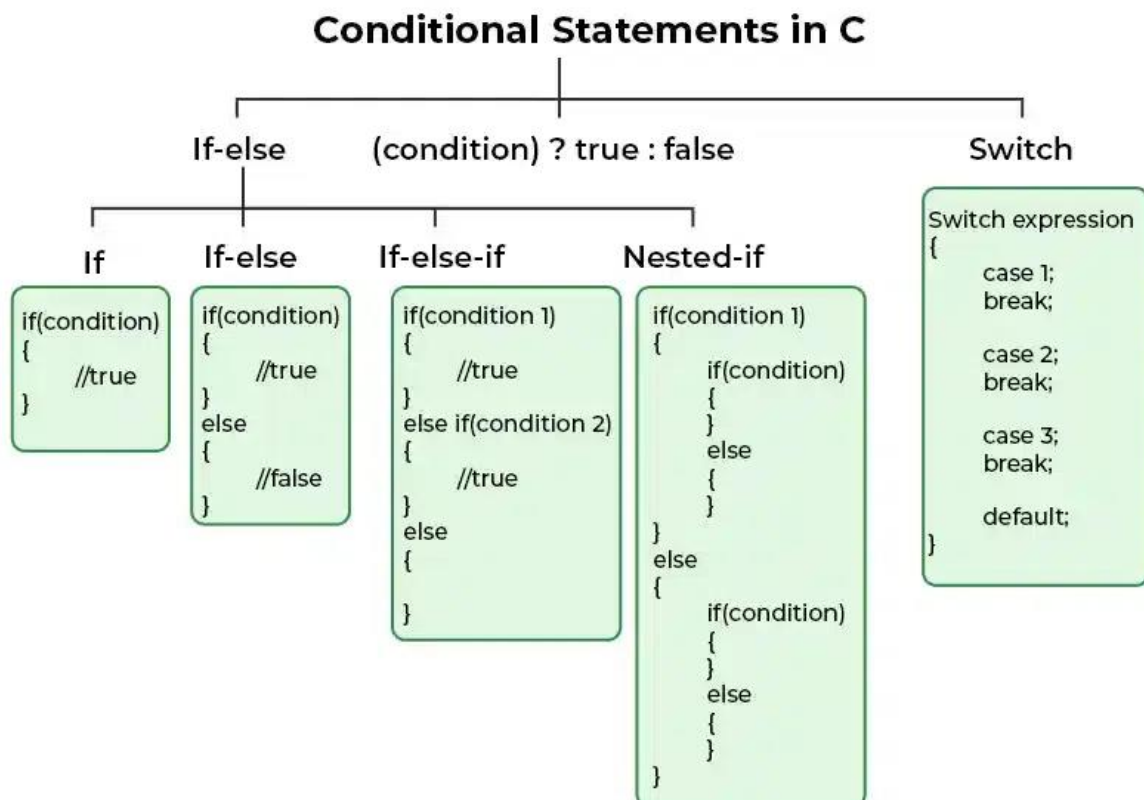
• THEORY EXERCISE:

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

In C, programs can choose which part of the code to execute based on some condition. This ability is called **decision making** and the statements used for it are called **conditional statements**. These statements evaluate one or more conditions and make the decision whether to execute a block of code or not.

Types of Conditional Statements in C

In the above program, we have used if statement, but there are many different types of conditional statements available in C language:



1. if in C

The [if statement](#) is the simplest 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.

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](#) consists of two blocks, one for false expression and one for true expression.

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.

4. if-else-if Ladder in C

The [if else if statements](#) 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.

5. switch Statement in C

The [switch case 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.

6. Conditional Operator in C

The [conditional operator](#) is used to add conditional code in our program. It is similar to the if-else statement. It is also known as the ternary operator as it works on three operands.

7. Jump Statements in C

These statements are used in C for the unconditional flow of control throughout the functions in a program. They support four types of jump statements:

A) break

This loop control statement is used to terminate the loop. As soon as the [break](#) statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

B) continue

This loop control statement is just like the break statement. The [continue statement](#) is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

C) return

The [return](#) in C returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and returns the control from where it was called. The return statement may or may not return anything for a void function, but for a non-void function, a return value must be returned.

6. Looping in C

- **THEORY EXERCISE:**

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

While Loop

Initialization is not the integral part. The loop continuation condition test is done at the beginning of the loop. The Loop variable is not the integral part of the loop. It should be handled explicitly.

The loop body is never executed if the condition is false. That is loop body is executed zero or more times.

Syntax:

```
while (testexpression) statement1;
```

Do- While Loop

Initialization is not the integral part. The loop continuation condition test is done at the end of the loop.

The loop variable is not the integral part of the loop. It should be handled explicitly.

The loop body is executed at least once if the condition is false. That is loop body is executed one or more times.

Syntax:

```
do{  
statement1;  
} while (test expression);
```

For Loop

Initialization is within loop constructor. The loop continuation condition test is done at the beginning of the loop. The loop variable is an integral part of the loop. It is handled within the loop construct.

The loop body is never executed if the condition is false. That is loop body is executed zero or more times.

Syntax:

```
for(expression1;expression2;expression3)
{
statement1;
}
```

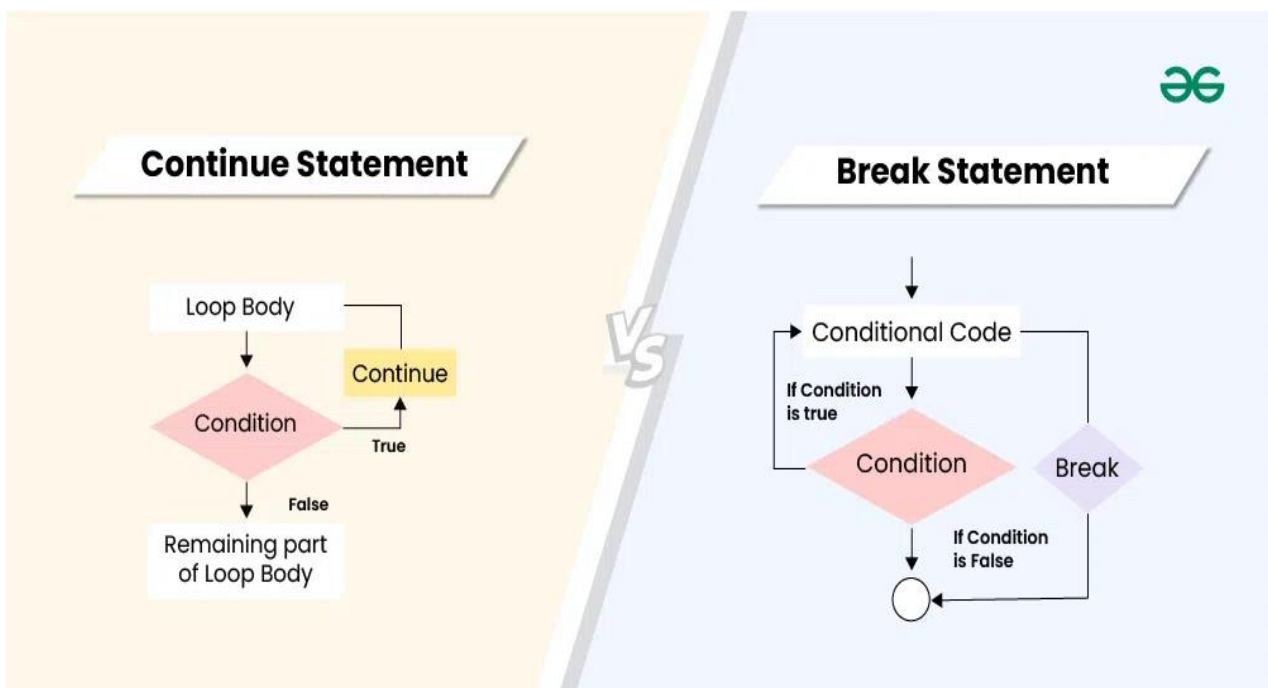

7. Loop Control Statements

- THEORY EXERCISE:

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

Break and **Continue** statements are the keywords that are used always within a loop. The purpose of a break and continue statement is to stop a running loop or to continue a particular iteration. In this article we will see what are the break and continue statements, what is their use and what are the differences between them.

Break vs Continue Statement



Break Statement:

A break statement is used when we want to terminate the running loop whenever any particular condition occurs. Whenever a break statement occurs loop breaks and stops executing.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    printf("break Statement:");
    for(i=0;i<10;i++)
    {
        if(i==5)
        {
            break;
        }
        printf("\n %d",i);
    }
    getch();
}
```

Continue Statement:

On the other side continue statement is used when we have to skip a particular iteration. Whenever we write continue statement the whole code after that statement is skipped and loop will go for next iteration.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    printf("continue Statement:");
    for(i=0;i<10;i++)
    {
        if(i==5)
        {
            continue;
        }
        printf("\n %d",i);
    }
    getch();
}
```

8. Functions in C

• THEORY EXERCISE:

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

A **function in C** is a set of statements that, when called, perform some specific tasks. It is the basic building block of a C program that provides modularity and code reusability. They are also called subroutines or procedures in other languages.

Function Definition

A function definition informs the compiler about the function's name, its return type, and what it does. It is compulsory to define a function before it can be called.

- **return_type:** type of value the function return.
- **name:** Name of the function
- **Body of function:** Statements inside **curly brackets { }** are executed when function call.

- **Example:**

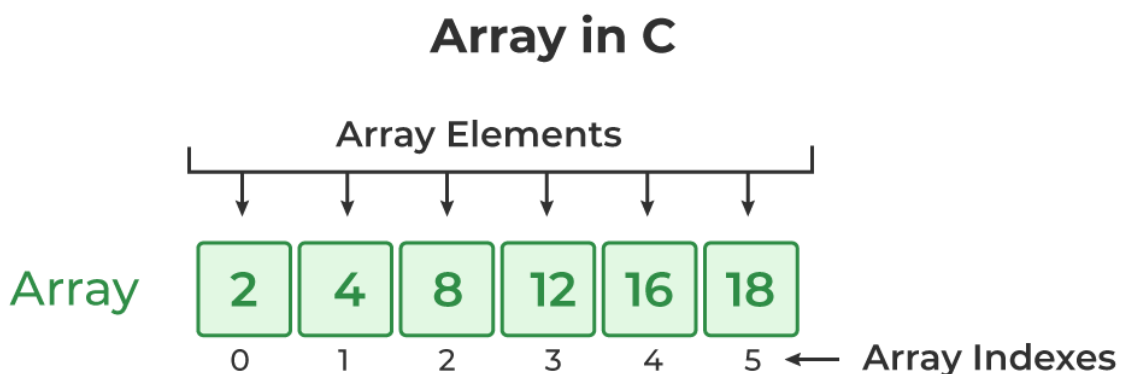
```
#include <stdio.h>
#include <conio.h>
main()
{
    clrscr();
    printf("hello world");
    return 0;
    getch();
}
```

9. Arrays in C

- **THEORY EXERCISE:**

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

An array in C is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., as well as derived and user-defined data types such as pointers, structures, etc.



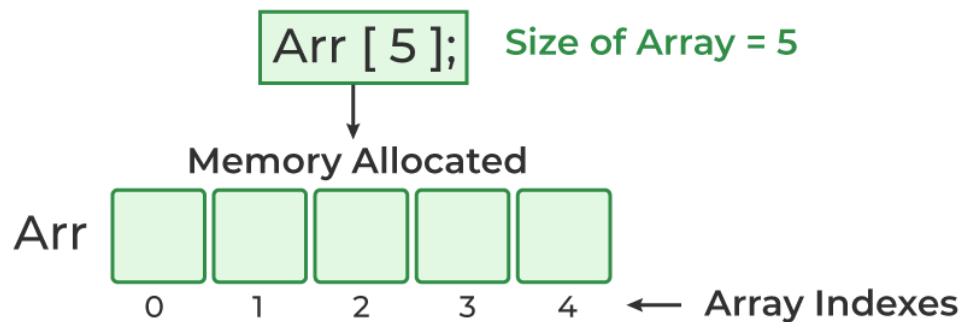
Creating an Array in C

The whole process of creating an array in C language can be divided into two primary sub processes i.e.

1. Array Declaration

Array declaration is the process of specifying the type, name, and size of the array. In C, we have to declare the array like any other variable before using it. When we declare an array in C, the compiler allocates the memory block of the specified size to the array name.

Array Declaration



2. Array Initialization

Initialization in C is the process to assign some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful values. Which can be done using initializer list, which is the list of values enclosed within braces `{ }` separated by a comma.

- One dimensional array

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5],i;
clrscr();
for(i=0;i<5;i++)
{
printf("\nEnter value:");
scanf("%d",&a[i]);
}
for(i=0;i<5;i++)
{
printf("\nA[%d]:",a[i]);
}
getch();
}
```


- Two dimensional:

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3],b[3][3],c[3][3],i,j;
    clrscr();
    printf("Enter 2D array A:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter value for A[%d][%d]:",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("\nEnter 2D array B:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter value for A[%d][%d]:",i,j);
            scanf("%d",&b[i][j]);
        }
    }
```

```
    }  
    for(i=0;i<3;i++)  
    {  
        for(j=0;j<3;j++)  
        {  
            c[i][j]=a[i][j]+b[i][j];  
        }  
    }  
    printf("\nAddition of two matrix:\n");  
    for(i=0;i<3;i++)  
    {  
        for(j=0;j<3;j++)  
        {  
            printf("\t%d",c[i][j]);  
        }  
        printf("\n");  
    }  
    getch();  
}
```