C Programming

Chapter 1

Introduction to C

What is C?

- C is a general-purpose, high-level programming language. It was developed by Dennis Ritchie at Bell Labs in the early 1970s.
- It is the most widely used programming language in the world.
- C is used for developing applications and games.
- Even if it's old, it is still a very popular programming language.

Why Learn C?

- It is one of the most popular programming language in the world.
- C is very fast, compared to other programming languages, like Java and Python.
- C programming can be compiled and executed on various platforms, making it highly portable.

First C Program

```c
#include <stdio.h>

int main ()
{
  printf( "Hello World!");
  return 0 ;
}
```

Output:

```
Hello World!
```

C Syntax Explanation

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

Explanation: This is a preprocessor that tells the compiler to include the stdio.h library, which contains functions for input and output, like printf.

```c
int main ()
```

Explanation: This is the definition of the main function. Every C program must have a main function as the entry point. The int means that the function returns an integer value, which is typically used to indicate whether the program executed successfully or encountered an error.

```c
{
```

Explanation: This is the opening brace that marks the beginning of the main body.

```c
printf("Hello World!");
```

Explanation: The printf function is used to print output to the console. Here, it prints the string "Hello World!" to the screen. The ; at the end marks the end of the statement.

```c
return 0;
```

Explanation: This statement ends the main function and returns the value 0 to the console. A return value of 0 usually signifies that the program executed successfully.

```c
}
```

Explanation: This is the closing brace that marks the end of the main function.

C Comments

Comments are ignored by the compiler and do not affect the execution of the program.

Single Line Comments

- Single-line comments start with two forward slashes (//).
- Any content between // and the end of the line is ignored by the compiler.

Example:

```c
#include <stdio.h>

int main()
{
    //This is a single line comment
    printf("Hello World!");
    return 0;
}
```

Multi-line comments

- Multi-line comments start with /* and ends with */.
- Any content between /* and */ will be ignored by the compiler.

Example:

```c
#include <stdio.h>

int main()
{
    /* This is a
    multi line
    comment */
    printf("Hello World!");
    return 0;
}
```

Chapter 2

C Variables

Variables are containers for storing data values.

In C, variables are classified into different types:

- **int:** an integer variable can store only an integer.
- **float:** a floating point variable can store only a floating point numbers.
- **char:** a character variable can store only a character.

Variable Declaration

Variable declaration is the process of specifying the name and type of a variable before using it in the program.

Example:

```
// Declare a variable
int myNum;

// Assign a value to the variable
myNum = 15;
```

Rules for Naming Variables

We can assign any name to the variable as long as it follows the following rules:

- A variable name can only contain alphabets, digits, and underscores.
- A variable cannot start with a digit.
- A variable cannot include any white space in its name.

Data Types

A data type in C specifies the type of data that a variable can store.

C provides several built-in data types, including:

Data Type	Size	Description
int	2 or 4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals.
char	1 byte	Stores a single character/letter/numbers
double	8 bytes	Stores fractional numbers, containing one or more decimals.

Example:

```c
#include <stdio.h>
int main()
{
    int integer = 20;
    float floating = 10.32;
    char character = 'B';

    printf("%d\n", integer);
    printf("%f\n", floating);
    printf("%c\n", character);
    return 0;
}
```

Output:

```
20
10.320000
B
```

C Constants

Constants in C are variables that do not change during program execution. Constants are also referred to as literals. There are several different types of constants, such as numeric constants, character constants, string constants.

The following are some typical C constant types:

- **Integer Constants:** The numbers with decimals (base 10), hexadecimals, binary, or octal representations are known as integer constants.
- **Floating-Point Constants:** Real values with an exponent or a decimal point are represented as floating-point constants.
- **String Constants:** String constants are enclosed in double quotes, e.g., "Hello, World".

Example:

```
const int minutesPerHour = 60;
const float PI = 3.14;
```

Chapter 3

C Operators

Operators are symbols that perform operations on operands.

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations such as addition, subtraction, multiplication, division, and modulus.

Operator	Description	Syntax
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus	a % b

Example:

```c
#include <stdio.h>

int main ()
{
  int a = 5;
  int b = 3;
  printf ("a + b = %d\n", a + b);
  printf ("a - b = %d\n", a - b);
  printf ("a * b = %d\n", a * b);
  printf ("a / b = %d\n", a / b);
  printf ("a % b = %d\n", a % b);
  return 0;
}
```

Output:

```
a + b = 8
a - b = 2
a * b = 15
a / b = 1.6
a % b = 0.15
```

Relational Operators

Relational operators are used to compare the relationship between two operands.

Operator	Description	Syntax
>	Greater than	a > b
<< /td>	Less than	a < b
>=	Greater than or equal to	a >= b
=<< /td>	Less than or equal to	a <= b
==	Equal to	a == b
!=	Not equal to	a != b

Example:

```c
#include <stdio.h>

int main ()
{
  int a = 5;
  int b = 3;
  printf ("a < b : %d\n", a < b);
  printf ("a > b : %d\n", a > b);
  printf ("a == b = %d\n", a == b);
  printf ("a != b : %d\n", a != b);
  return 0;
}
```

Output:

```
a < b : 0
a > b : 1
a == b = 0
a != b : 1
```

Logical Operators

Logical operators are used to perform logical operations on two or more conditions.

Operator	Description	Syntax
&&	AND Operator	a && b
||	OR Operator	a || b
!	NOT Operator	!a

Example:

```c
#include <stdio.h>

int main ()
{
  int a = 15;
  int b = 6;
  printf ("a && b : %d\n", a && b);
  printf ("a || b = %d\n", a || b);
  printf ("!a: %d\n", !a);
  return 0;
}
```

Output:

```
a && b : 1
a || b = 1
!a: 0
```

Assignment Operators

Assignment operators are used to assign values to a variable.

Operator	Description	Syntax
=	It assigns the right side operand value to the left side operand.	a = b
+=	It adds the right operand to the left operand and assigns the result to the left operand.	a += b
-=	It subtracts the right operand from the left operand and assigns the result to the left operand.	a -= b
*=	It multiplies the right operand with the left operand and assigns the result to the left operand.	a *= b
/=	It divides the left operand with the right operand and assigns the result to the left operand.	a /= b

Example:

```c
#include <stdio.h>

int main ()
{
  int a = 15;
  int b = 6;
  printf ("a = b = %d\n", a = b);
  printf ("a += b = %d\n", a += b);
  printf ("a -= b = %d\n", a -= b);
  printf ("a *= b = %d\n", a *= b);
  printf ("a /= b = %d\n", a /= b);
  return 0;
}
```

Output:

```
a = b = 6
a += b = 12
a -= b = 6
a *= b = 36
a /= b = 6
```

Chapter 4
C Conditional Statements

if...else Statements

The if-else statement in C is used to make decisions based on conditions. It allows you to execute a block of code if a specified condition is true, and another block of code if the condition is false.

The syntax of the if-else statement in C is:

Syntax:

```c
if (condition) {
  //code to be executed if the condition is true
} else {
  //code to be executed if the condition is false
}
```

Example:

```c
#include <stdio.h>

int main ()
{
  int num = 15;

  if (num > 0) {
      printf ("%d is a positive number.\n", num);
  } else {
      printf ("%d is a negative number.\n", num);
  }
  return 0;
}
```

Output:

```
15 is a positive number.
```

if-else Ladder

If we want to check the multiple conditions then if-else ladder can used.

Syntax:

```c
if (condition) {
  //code
}
else if (condition) {
  //code
}
else if (condition) {
  //code
}
else {
  //code
}
```

Example:

```c
#include <stdio.h>

int main ()
{
  int num = -6;

  if (num > 0) {
      printf ("%d is a positive number.\n", num);
  }
  else if (num < 0) {
      printf ("%d is a negative number.\n", num);
  }
  else {
      printf ("%d is a Zero.\n", num);
  }
  return 0;
}
```

Output:

```
-6 is a negative number.
```

Chapter 5
C Switch Statement

The control statement that allows us to make a decision from the number of choices is called a switch.

The syntax of the switch case statements:

```c
switch (integer expression)
{
case 1:
do this;

case 2:
do this;

case 3:
do this;

default:
do this;
}
```

Example:

```c
#include <stdio.h>

int main()
{
    int i = 3;

    switch (i)
    {
    case 1:
        printf("I am Statement 1");
        break;

    case 2:
        printf("I am Statement 2");
        break;

    case 3:
        printf("I am Statement 3");
        break;

    default:
        printf("I am default");
        break;
    }
    return 0;
}
```

Output:

```
I am Statement 3
```

Chapter 6

C Loops

Loops in C are used to execute a block of code until the specified condition is met.

Following are the three types of loops in C programming:

- for loop
- while loop
- do-while loop

for Loop

A for loop in C programming is a repetition control structure that allows programmers to write a loop that will be executed a specific number of times.

The for loop allows us to specify three things about the loop:

- Setting a loop counter to a initial value.
- Testing the loop counter to determine wheather its value has reached the number of repetitions desired.
- Increasing the value of loop counter each time the body of the loop has been executed.

While Loop

The while loop is used when you want to execute a block of code as long as a condition is true.

do-while Loop

The do-while loop is similar to the while loop. This loop would execute its statements at least once, even if the condition fails for the first time.

for Loop

A for loop in C programming is a repetition control structure that allows programmers to write a loop that will be executed a specific number of times.

Syntax

```
for (initialization; testExpression; increment/decrement)
{
    // block of code
}
```

Example:

```
#include <stdio.h>

int main()
{
    int i;
    for (i = 0; i <= 6; i++)
    {
        printf("%d", i);
    }
    return 0;
}
```

Output:

```
0 1 2 3 4 5 6
```

While Loop

The while loop is used when you want to execute a block of code as long as a condition is true.

Syntax:

```
while (condition)
{
    // block of code
}
```

Example:

```
#include <stdio.h>

int main()
{
    int i = 0;
    while (i < 8)
    {
        printf("%d", i);
        i++;
    }
    return 0;
}
```

Output:

```
0 1 2 3 4 5 6 7
```

do...while Loop

The do-while loop is similar to the while loop. This loop would execute its statements at least once, even if the condition fails for the first time.

Syntax:

```
do {
  // block of code
} while (condition);
```

Example:

```c
#include <stdio.h>

int main()
{
    int i = 0;

    do
    {
        printf("%d", i);
        i++;
    } while (i < 4);

    return 0;
}
```

Output:

```
0 1 2 3
```

Chapter 7

C Break and Continue

Break Statement

The break statement is used to break out of the loop in which it is encountered. The break statement is used inside loops or switch statements in C programming.

Example:

```c
#include <stdio.h>

int main () {
  int i;
  for (i = 0; i < 10; i++) {
    if (i == 6) {
      break;
    }
    printf ("%d", i);
  }
  return 0;
}
```

Output:

```
0 1 2 3 4 5
```

Continue Statement

The continue statement skips the loop's current iteration and proceeds to the next one.

Example:

```c
#include <stdio.h>

int main () {
  int i,j;
  for (i = 1; i <= 2; i++)
  {
    for (j = 1; j <= 2; j++) {
      if (i == j)
       continue;
      printf ("%d%d", i,j);
    }
  }
  return 0;
}
```

Output:

```
1 2 2 1
```

Chapter 8

C Arrays

An array is a collection of similar data items. It is stored at contiguous memory locations in arrays.

Array Declaration

Like other variables, an array needs to be declared so that the compiler will know what type of an array and how large an array we want.

```
data_type array_name [size];
```

Accessing Elements of an Array

The index number of an element in an array makes it easy to access that element. The index number starts from 0.

Example:

```
#include <stdio.h>

int main()
{
    int myArr[] = {2, 6, 8, 10};

    printf("%d", myArr[2]);
    return 0;
}
```

Output:

```
8
```

Change an Array Element

To modify the value of a given element, use the index number.

Example:

```c
#include <stdio.h>

int main()
{
    int myArr[] = {2, 6, 8, 10};
    myArr[1] = 5;

    printf("%d", myArr[1]);
    return 0;
}
```

Output:

```
5
```

Chapter 9

C Strings

Strings are used for storing text/characters. For example, "Hello World" is a string of characters.

Example:

```c
char name[] = "John";
```

String Declaration

Declaring a string is the same as declaring a one-dimensional array.

Syntax:

```c
char string_name[string_size];
```

Example:

```c
#include <stdio.h>

int main () {
  char str1[] = "Tutorials4Coding";
  printf("%s", str1);

  return 0;
}
```

Output:

```
Tutorials4Coding
```

Access Strings

To modify the value of a given element, use the index number.

Example:

```
#include <stdio.h>

int main () {
  char str1[] = "Tutorials4Coding";
  printf("%c", str1[2]);

  return 0;
}
```

Output:

```
t
```

String Functions

String functions are a collection of functions given by the C standard library for working with strings in C programming.
Let us understand these functions one by one.

strlen():

This function counts the number of characters present in a string.

Example:

```
#include <stdio.h>

int main () {
  char str1[] = "Tutorials4Coding";
  printf("%d", strlen(str1));

  return 0;
}
```

Output:

```
16
```

strcpy():

This function copies the contents of one string to another.

Example:

```c
#include <stdio.h>
#include <string.h>

int main () {
  char str1[20] = "Tutorials4Coding";
  char str2[20];

  strcpy(str2, str1);

  printf("%s", str2);

  return 0;
}
```

Output:

```
Tutorials4Coding
```

strcat():

This function concatenates the source string at the end of the target string. For example, "Hello" and "World" on concatenation would result in a string “HelloWorld”.

Example:

```c
#include <stdio.h>
#include <string.h>

int main () {
  char str1[20] = "Hello ";
  char str2[] = "World";

  strcat(str1, str2);

  printf("%s", str1);

  return 0;
}
```

Output:

```
Hello World
```

Chapter 10

C Functions

Functions in C are blocks of code that perform a specific task. They are intended to improve code readability, modularity, and reusability by dividing a program into parts.

Key points about functions

- Function Declaration
- Function Definition
- Function Calls

Function Declarations

Before using a function, it must first be declared. The declaration specifies the function's return type, name, and parameters.

Syntax:

```c
return_type function_name (para_1, para_2){
  // block of the function
}
```

Function Definition

Function definitions consist of the function's actual implementation.

Function Call

A function is called by its name, followed by parenthesis.

Example:

```c
#include <stdio.h>

// Function declaration
void func();

int main() {
  func(); // calling the function
  return 0;
}

// Function definition
void func() {
    printf("Execution Succesfull.");
}
```

Output:

```
Execution Succesfull.
```

Types of functions

- Library Functions
- User Defined Functions

Library Functions

In the C programming language, library functions are pre-defined functions. These functions are declared in header files.

Example: printf(), scanf(), etc

User Defined Functions

User-defined functions are functions that the programmer creates to do specific tasks. These functions are defined by the programmer based on their needs and can be reused several times throughout the program.

Example: Any function defined by the programmer.

Recursion

Recursion is a programming method in C that involves calling a function on itself to solve a problem.

Example:

```c
#include <stdio.h>
int fibo(int);
int main()
{
    int terms = 12, i, n = 0;
    for (i = 1; i <= terms; i++)
    {
        printf ("%d\t", fibo (n));
        n++;
    }
    return 0;
}

int fibo(int n)
{
    if (n == 0 || n == 1)
        return n;
    else
        return (fibo(n - 1) + fibo (n - 2));
}
```

Output:

```
0 1 1 2 3 5 8 13 21 34 55 89
```

Chapter 11

C Structures

Structures are used to group variables of various data types under a single name. For Example, a 'book' is a collection of items like title, author, publisher, number of pages, date of publication etc. for dealing with such data C provides a data type called structure.

Syntax:

```
struct MyStructure {
  dataType member1;
  dataType member2;
};
```

Example:

```
#include <stdio.h>

struct Books
{
    char title[40];
    char author[40];
    float price;
};

int main()
{
    struct Books book1 = {"Macbeth", "William Shakespeare", 100.00};

    printf("Title of the book is %s\n", book1.title);
    printf("Author of the book is %s\n", book1.author);
    printf("Price of the book is %f\n", book1.price);

    return 0;
}
```

Output:

```
Title of the book is Macbeth
Author of the book is William Shakespeare
Price of the book is 100.00
```

Chapter 12

C Files

File handling in C is a fundamental part of programming that lets you read and write files on the system. It consists of opening files, reading data from them, writing data to them, and then closing them once completed.

Types of Files in C

There are two types of files:

Text Files

A text files contains only textual information like alphabets, digits and special symbols. An example of a text file is a .txt file.

Binary Files

A binary files is merely a collection of bytes. An example of binary file is a .bin file.

C File Operations

There are different operations that can be perform on a file. These are:

- Creating of a new file
- Opening an existing file
- Reading from file
- Writing to a file
- Closing a file

Creating a File

To create a file in C, use the `fopen()` function with the proper mode.

Syntax

```
FILE *fptr;
fptr = fopen("filename.txt", "w");
```

Open a File

To open a file, use the `fopen()` function.

Syntax

```
FILE* fopen("fileopen","mode");
```

Reading From a File

Use the functions like `fscanf(), fgets()` to read data from the file.

Syntax

```
FILE *fptr;
fptr = fopen("filename.txt", "r");
```

Closing a File

When we have finished reading from the file, we need to close it. This is done using the fclose() function.

Syntax

```
fclose(fp);
```