**INTRODUCTORY PROGRAMMING LANGUAGE USING C**

**BY**

**OJO ABOSEDE I. (MCPN, MNCS, MIAENG)**

**DEDICATION**

This book is dedicated to God Almighty, my immediate family and my extended family.

**ACKNOWLEDGEMENT**

I am grateful to all those who have contributed to the achievement of this book. I also acknowledge other Lecturers in the Department of Computer Science, Ogun State Institute of Technology, Igbesa, Ogun State, who have given an unquantifiable support; I pray that my good God will bless all of you in Jesus name.

My acknowledgement also goes to all the Authors that their books have contributed immensely to this book.

**TABLE OF CONTENT**

**CONTENT PAGES**

Front Page i

Dedication ii

Acknowledgement iii

Table of Contents iv

**CHAPTER ONE - UNDERSTAND BASIC CONCEPTS OF C PROGRAMMING LANGUAGE**

1.1 Introduction

1.2 Quality of C Programming

1.3 Reasons for C Popularity

1.4 Features of C Programming Language

1.5 Advantages of C

1.6 Disadvantages of C

1.7 The Limitations of C Programming Languages

1.8 History of Programming Languages and C

1.9 Programming Languages Developed Before C And After

1.10 List of C Compilers for Windows OS

1.11 C Compiler Installation on Windows

1.12 C Compiler Installation on Unix/Linux

1.13 C Compiler Installation on Mac

**CHAPTER TWO - UNDERSTAND DATA TYPES, CONSTANTS, VARIABLES AND PROGRAMMING PROCEDURE**

2.1 What is Variable?

2.1.1 Rules for Forming Variable Name

2.2 C Token

2.2.1 The C Keywords

2.2.2 The C Identifiers

2.2.3 Strings in C

2.2.4 Operators in C

2.2.5 Constant in C

2.2.6 Special Characters in C

**CHAPTER THREE - UNDERSTAND STORAGE CLASSES, OPERATORS AND TYPE CASTING**

3.1 What is Operator?

3.2 Types of Operators in C

3.2.1 Arithmetic Operators

3.2.2 Increment and Decrement Operators

3.2.3 Relational Operators

3.2.4 Logical Operators

3.2.5 Assignment Operators

3.2.6 Conditional Operator

3.2.7 Special Operators

3.3 Type Casting In C

3.3.1 Without Type Casting

3.3.2 After Type Casting

3.4 Basic Structure of C Program

**CHAPTER FOUR - UNDERSTAND STANDARD INPUT AND OUTPUT OPERATIONS**

4.1 Managing Input/Output

4.2 Reading Character in C

4.3 Writing Character in C

4.4 Formatted Input

4.5 Reading and Writing Strings in C

4.6 C Format Specifiers

4.6.1 List of Format Specifiers in C

**CHAPTER FIVE - UNDERSTAND CONTROL STRUCTURES (DECISION MAKING AND LOOPS)**

5.1 What is Control Structure in C?

5.2 Sequence Structure (Straight Line Paths)

5.3 Selection Structure (One or Many Branches) C or Switch or If-Else-If Ladder

5.3.1 Simple If Statements in C

5.3.2 If Else Statements in C

5.3.3 Nested If-Else Statements in C

5.3.4 Else-If Statements in C

5.3.5 C Switch Statement

5.4 Loop Structure (Repetition of a Set of Activities)

5.4.1 While Loop Control Statements

5.4.2 Do-While Loop 5.4.3 for Loops

5.4.3 for Loops

5.5 GOTO STATEMENT IN C

**CHAPTER SIX - UNDERSTAND THE FUNCTIONS AND SCOPE RULES**

6.1 Functions in C

6.2 Parts of Function

6.2.1 Function Prototype

6.2.2 Calling a Function in C

6.3 C Library Functions

6.4 Benefits of Using the Function in C

6.5 Scope of Variables

6.5.1 Local Variables

6.5.2 Global Variables

6.6 What is Recursion?

6.7 Storage Classes Types in C

**CHAPTER SEVEN -UNDERSTAND ARRAYS AND STRINGS**

7.1 Define an Array In C

7.2 Initialize an Array In C

7.3 Accessing Array Elements in C

7.4 Strings Declaration in C

7.5 Strings Initialization in C

**CHAPTER EIGHT - UNDERSTAND POINTER OPERATIONS**

8.1 Pointer Definition in C

8.2 Benefits of Using Pointers in C

8.3 How to Use Pointers in C

**CHAPTER NINE - UNDERSTAND STRUCTURES AND UNION DATA TYPES**

9.1 Defining a Structure in C

9.2 Accessing Structure Members in C

9.3 Unions Data Type in C

9.3.1 Defining a Union in C

9.3.2 Accessing Union Members in C

9.4 C Enums

9.4.1 Enumerated Type Declaration

9.4.2 Why Enums Are Used?

9.4.3 Why Enums Are Used?

9.4.4 How to Use Enums for Flags?

**CHAPTER TEN - UNDERSTAND FILE INPUT/OUTPUT (I/O) OPERATIONS**

10.1 C File Operations

10.2 Steps for Processing A File

10.3 Different File Input/Output and Their Functions

**CHAPTER ELEVEN - UNDERSTAND PREPROCESSORS AND HEADER FILES**

11.1 What is Preprocssor?

11.2 Lists All Preprocessor Directives.

11.3 What Are the Header Files In C?

11.4 How Include Works

11.5 Writing of Single and Multiple Uses of Header Files

C Codes Examples

References

**COURSE CONTENTS**

* Understand Basic Concepts of C Programming Language
* Understand Data types, Constants, Variables and programming procedure
* Understand Storage Classes, Operators and Type Casting
* Understand Standard Inputs and Output Operations
* Understand Control Structures (Decision Making and Loops)
* Understand the Functions and Scope Rules
* Understand Arrays and Strings
* Understand Pointer operations
* Understand Structures and Union data types
* Understand File Input/output (I/O) Operations
* Understand Pre-processors and Header Files

**CHAPTER ONE**

**UNDERSTAND BASIC CONCEPTS OF C PROGRAMMING LANGUAGE**

**1.1 INTRODUCTION**

**A compiler** is a computer program that transforms human-readable (programming language) source code into another computer language (binary) code. In simple terms, Compiler takes the code that you wrote and turned in to the binary code that the computer can understand.

C Language is a powerful general-purpose programming language. It can be used to develop software like operating systems, databases, compilers, and so on. C programming is an excellent language to learn to program for beginners. C is an imperative language (consists of commands that direct computer on what to do, it tells the computer how to do something) designed to compile relatively straightforwardly and provides low-level memory access. C language and its compiler have become available on a wide range of platforms from embedded microcontrollers to supercomputers.

The C compiler is a software application that transforms the human-readable C program code to machine-readable code. The process of transforming the code from High-Level Language to Machine Level Language is called **Compilation**. The human-readable code is the C program that consists of digits letters, special symbols, etc. which is understood by human beings. On the other hand, machine language is dependent on the processor and processor understands zeroes and ones (binary) only. All C program execution is based on a processor which is available in the CPU; that is why entire C source code needs to be converted to the binary system by the compiler.

**1.2 QUALITY OF C PROGRAMMING**

* **Procedural Language** - Instructions in a C program are executed step by step. A procedural language is a computer programming language that follows, in order, a set of commands.
* **Portable** - You can move C programs from one platform to another machines without any or minimal changes or modification.
* **Speed** - C programming is faster than most programming languages like Java, Python, etc.
* **General Purpose** - C programming can be used to develop operating systems, embedded systems, databases, and so on.

**1.3 REASONS FOR C POPULARITY**

* One of the early programming languages.
* One of the best programming language to learn quickly.
* C language is reliable, simple, and easy to use.
* C language is a structured language.
* Modern programming concepts are based on C.
* It can be compiled on a variety of computer platforms.
* Programs written in C are efficient and fast.
* Universities / polytechnic preferred to add C programming in their courseware.

**1.4 FEATURES OF C PROGRAMMING LANGUAGE**

* C is a robust language with a rich set of built-in functions and operators.
* C is a collection of C library functions; we can also create our function and add it to the C library.
* Structured Programming
* Popular system programming language
* Supports variety of platforms
* Efficient and also handle low-level activities.
* As fast as assembly language and hence used as system development language.

**1.5 ADVANTAGES OF C**

* C is the building block for many other programming languages.
* Several standard functions are there (like in-built) that can be used to develop programs.
* C programs are collections of C library functions, and it's also easy to add functions to the C library.
* The modular structure makes code debugging, maintenance, and testing easier.

**1.6 DISADVANTAGES of C**

* Oriented C does not provide Object Programming (OOP) concepts.
* C does not provide binding or wrapping (libraries that bridge two programminglanguagesso that a library written for one language can be used in another language) up of data in a single unit.
* C does not provide Constructor and Destructor. A constructor is something that initializes objects, and destructors are to destroy that initialization.

**1.7 THE LIMITATIONS OF C PROGRAMMING LANGUAGES**

* C allows a lot of freedom in writing code, and that is why you can put an empty line or white space anywhere in the program. And because there is no fixed place to start or end the line, so it isn't easy to read and understand the program.
* C compilers can only identify errors and are incapable of handling exceptions (run-time errors).
* C provides no data protection.
* It also doesn't feature the reusability of source code extensively.
* It does not provide strict data type checking (for example, an integer value can be passed for floating datatype).

**1.8 HISTORY OF PROGRAMMING LANGUAGES AND C**

* In 1988, the American National Standards Institute (ANSI) had formalized the C language.
* C was invented to write UNIX operating system.
* C is a successor of 'Basic Combined Programming Language' (BCPL) called B language.
* Linux OS, PHP, and MySQL are written in C.
* C has been written in assembly language.

**1.9 PROGRAMMING LANGUAGES DEVELOPED BEFORE C AND AFTER**

|  |  |
| --- | --- |
| **PROGRAMMING LANGUAGE** | **DEVELOPMENT YEAR** |
| Regional Assembly Language | 1951 |
| Autocode | 1952 |
| IPL (forerunner to LISP) | 1954 |
| FLOW-MATIC (led by COBOL) | 1955 |
| FORTRAN (First compiler) | 1957 |
| COMTRAN (precursor to COBOL) | 1957 |
| LISP | 1958 |
| ALGOL 58 | 1958 |
| FACT (forerunner to COBOL) | 1959 |
| COBOL | 1959 |
| RPG | 1959 |
| APL | 1962 |
| Simula | 1962 |
| SNOBOL | 1962 |
| CPL (forerunner to C) | 1963 |
| Speakeasy (computational environment) | 1964 |
| BASIC | 1964 |
| PL/I | 1964 |
| JOSS | 1966 |
| BCPL (forerunner to C) | 1967 |

**1.10 LIST OF C COMPILERS FOR WINDOWS OS**

Since there are various compilers available into the online market, here are the lists of some of the frequently used ones:

* CCS C Compiler
* Auto C combines Microsoft ActiveX Control Pad with SuperEdi
* Turbo C
* Minimalist GNU for Windows (MinGW)
* Portable C Compiler
* Clang C++
* Digital Mars C++ Compiler
* Intel C++
* IBM C++
* Visual C++ : Express Edition
* Oracle C++

All of these above compilers for C are free to download, but there are some other paid C compilers also available, or programmers can get it for trial version:

* Embarcadero C++
* Edison Design Group C++
* Green Hills C++
* HP C++ for Unix
* Intel C++ for Windows, Linux, and some embedded systems.
* Microsoft C++
* Paradigm C++

**1.11 C COMPILER INSTALLATION ON WINDOWS**

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

* You can download a 90-day trial version of Visual Studio
* You can download Dev-C++ IDE to develop C and C++ application.
* You can install MinGW

**1.12 C COMPILER INSTALLATION ON UNIX/LINUX**

If you are using UNIX / Linux, then most probably C compiler called GCC is already in your system. To check if you have it installed, you can type **cc** or **gcc** at the command prompt.

$ gcc -v

If for some reason it is not installed on your system, you can download it from gcc.gnu.org/install.

**1.13 C COMPILER INSTALLATION ON MAC**

Xcode development environment came with GNU C/C++ compiler; you can install it from Apple's website. You can download Xcode from developer.apple.com/technologies/tools.

**CHAPTER TWO**

**UNDERSTAND DATA TYPES, CONSTANTS, VARIABLES AND PROGRAMMING PROCEDURE**

**2.1 WHAT IS VARIABLE?**

Variables are memory locations (storage area) in the C programming language. The primary purpose of variables is to store data in memory for later use. Unlike constants which do not change during the program execution, variables value may change during execution. If you declare a variable in C, that means you are asking the operating system to reserve a piece of memory with that variable name.

**Variable Declaration in C**

Syntax:

type variable\_name;

or

type variable\_name, variable\_name, variable\_name;

**Variable Declaration and Initialization**

**Example:**

int width, height=5;

char letter='A';

float age, area;

double d;

/\* actual initialization \*/width = 10;

age = 26.5;

**Variable Assignment**

A variable assignment is a process of assigning a value to a variable.

**Example:**

int width = 60;

int age = 31;

**2.1.1 RULES FOR FORMING VARIABLE NAME**

There are some rules on choosing variable names

* A variable name can only consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and the underscore character.
* The first character must be a letter or underscore.
* Blank spaces cannot be used in variable names.
* Special characters like #, $ are not allowed.
* C keywords cannot be used as variable names.
* Variable names are case sensitive.
* Values of the variables can be numeric or alphabetic.
* Variable type can be char, int, float, double, or void.

**Example: C Program to Print Value of a Variable**

#include<stdio.h>

void main()

{

/\* c program to print value of a variable \*/ int age = 33;

printf("I am %d years old.\n", age);

}

**Program Output:**

I am 33 years old.

**2.2 C TOKEN**

Tokens in C language is the most important concept used in developing a C program. We can say the token in the C language is the smallest individual part. They are the building block of C programming language. C Supports 6 Types of Tokens

* Keywords
* Identifiers
* Strings
* Operators
* Constants
* Special Symbols

**2.2.1 THE C KEYWORDS**

Keywords in C language are predefined or reserved words used to expose the behavior of the data. These are reserved words that are known by the compiler, there are 32 keywords in C. Each keyword has its functionality to do. It must be part of program because you cannot use them as a variable name. You cannot use a keyword as an identifier in your C programs, its reserved words in C library and used to perform an internal operation. The meaning and working of these keywords are already known to the compiler.

**C KEYWORDS LIST**

A list of 32 reserved keywords in c language is given below:

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

**HOW TO USED KEYWORDS IN A PROGRAM**

Example:

#include<stdio.h>

main()

{

float a, b;

printf("Showing how keywords are used.");

return 0;

}

In the above program, float and return are keywords. The float is used to declare variables, and return is used to return an integer type value in this program.

**2.2.2 THE C IDENTIFIERS**

Identifier in C language is used for naming functions, variables, constants, structures, unions, arrays, etc. The identifier is user-defined words. These identifiers can be composed of uppercase, lowercase , underscore. Identifiers never used for keywords

**Example:**

int amount;

double totalbalance;

In the above example, amount and totalbalance are identifiers while, int, and double are keywords.

**Rules to construct identifiers is below**

* The first character should be either alphabet or underscore and then followed by any character, digit.
* Identifiers are case sensitive as there is **A** and **a** treated as different.
* Commas and blank space are not allowed
* Keywords can’t be used for identifiers.
* The length of the identifiers should not be more than 31 characters.
* Naming convention should understandable to the user.

**Syntax:**

dataType \_abc1= Valid

dataType 123abcZ=Invalid

dataType int=Invalid

dataType abc, ap=Invalid

**2.2.3 STRINGS IN C**

Strings in C is an array of characters having null character ‘\0’ at the end of the string. Strings in C are enclosed in double-quotes(“”) and Characters are enclosed in single quotes(”).

**Syntax:**

char a[10]={'1','2','3'};

char a[]="Amardeep";

char a[10]="Paramesh";

**2.2.4 OPERATORS IN C**

This is used to perform special operations on data.

* **Unary Operator:**Applied with a single operand.
* **Binary Operator:**Applied between 2 operands.

**Operators’ Types**

* Arithmetic Operators
* Relational Operators
* Shift Operators
* Logical Operators
* Bitwise Operators
* Conditional Operators
* Assignment Operator
* Misc Operator

**2.2.5 CONSTANT IN C**

Constants are like a variable, except that their value never changes during execution once defined. C Constants is the most fundamental and essential part of the C programming language. Constants in C are the fixed values that are used in a program, and its value remains the same during the entire execution of the program.

* Constants are also called literals.
* Constants can be any of the data types.
* It is considered best practice to define constants using only upper-case names.

Syntax:

const type constant\_name;

const keyword defines a constant in C.

Example:

#include<stdio.h>

main()

{

  const int SIDE = 10;

  int area;

  area = SIDE\*SIDE;

  printf("The area of the square with side: %d is: %d sq. units"

  , SIDE, area);

}

**Program Output:**



Putting const either before or after the type is possible.

int const SIDE = 10;

or

const int SIDE = 10;

**CONSTANT TYPES IN C**

Constants are categorized into two basic types, and each of these types has its subtypes/categories. These are:

**Primary Constants**

The following are the sub-category of primary constants.

* Numeric Constants
* Character Constants

**Numeric Constants**: This comprises of Integer Constants and Real Constants

**1. Integer Constant:** This refer to a sequence of digits. Integers are of three types which are

* Decimal Integer
* Octal Integer
* Hexadecimal Integer

**Example: 15, -265, 0, 99818, +25, 045, 0X6**

**2. Real constant:** The numbers containing fractional parts like 99.25 are called real or floating points constant.

**Character Constants:** This comprises are Single Character Constants, String Constants and Backslash Character Constants

* **Single Character Constants:** It simply contains a single character enclosed within ' and ' (a pair of single quote). It is to be noted that the character '**8**' is not the same as **8**. Character constants have a specific set of integer values known as ASCII values (American Standard Code for Information Interchange).

**Example: 'X', '5', ';'**

* **String Constants:** These are a sequence of characters enclosed in double quotes, and they may include letters, digits, special characters, and blank spaces. It is again to be noted that "**G**" and '**G**' are different - because "G" represents a string as it is enclosed within a pair of double quotes whereas 'G' represents a single character.

**Example: "Hello!", "2015", "2+1"**

* **Backslash character constant:** C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning which is known to the compiler. They are also termed as "Escape Sequence".

**For Example: \t is used to give a tab; \n is used to give a new line**

|  |  |
| --- | --- |
| Constants | Meaning |
| \a | beep sound |
| \b | backspace |
| \f | form feed |
| \n | new line |
| \r | carriage return |
| \t | horizontal tab |
| \v | vertical tab |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \0 | null |

**Secondary Constant:** This includes;

* Array
* Pointer
* Structure
* Union
* Enum

**2.2.6 SPECIAL SYMBOLS**

* **Square brackets [ ]:**Used for single and multi-dimensional arrays.
* **Simple brackets ( ):**Used for function declaration.
* **Curly braces { }:**Used for opening and closing the code.
* **The comma (,):**Used to separate variables.
* **Hash/pre-processor (#):** Used for the header file.
* **Asterisk (\*):**Used for Pointers.
* **Tilde (~):**Used for destructing the memory.
* **Period (.):**Used for accessing union members.

**CHAPTER THREE**

**UNDERSTAND STORAGE CLASSES, OPERATORS AND TYPE CASTING**

**3.1 WHAT IS OPERATOR?**

We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands.

Code sample

int c ;

c = a;

printf("Line 1 - = Operator Example, Value of c = %d\n", c );

c += a;

printf("Line 2 - += Operator Example, Value of c = %d\n", c );...

**3.2 TYPES OF OPERATORS IN C**

C programming language offers various types of operators having different functioning capabilities.

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Assignment Operators
* Increment and Decrement Operators
* Conditional Operator
* Bitwise Operators
* Special Operators

**3.2.1 ARITHMETIC OPERATORS:** Arithmetic Operators are used to performing mathematical calculations like addition (+), subtraction (-), multiplication (\*), division (/) and modulus (%).

|  |  |
| --- | --- |
| Operator | Description |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |

C Program to Add Two Numbers

Example:

#include <stdio.h>

void main()

{

int i=3,j=7,k; /\* Variables Defining and Assign values \*/

k=i+j;

printf("sum of two numbers is %d\n", k);

}

**Program Output:**



**3.2.2 INCREMENT AND DECREMENT OPERATORS:** Increment and Decrement Operators are useful operators generally used to minimize the calculation, i.e. ++x and x++ means x=x+1 or -x and x−−means x=x-1. But there is a slight difference between ++ or −− written before or after the operand. Applying the pre-increment first add one to the operand and then the result is assigned to the variable on the left whereas post-increment first assigns the value to the variable on the left and then increment the operand.

|  |  |
| --- | --- |
| Operator | Description |
| ++ | Increment |
| −− | Decrement |

**Example: To Demonstrate prefix and postfix modes.**

#include <stdio.h>

//stdio.h is a header file used for input.output purpose.

void main()

{

//set a and b both equal to 5.

int a=5, b=5;

//Print them and decrementing each time.

//Use postfix mode for a and prefix mode for b.

printf("\n%d %d",a--,--b);

printf("\n%d %d",a--,--b);

printf("\n%d %d",a--,--b);

printf("\n%d %d",a--,--b);

printf("\n%d %d",a--,--b);

}

**Program Output:**

5 4

4 3

3 2

2 1

1 0

**3.2.3 RELATIONAL OPERATORS:** Relational operators are used to comparing two quantities or values.

|  |  |
| --- | --- |
| Operator | Description |
| == | Is equal to |
| != | Is not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

**3.2.4 LOGICAL OPERATORS:** C provides three logical operators when we test more than one condition to make decisions. These are: && (meaning logical AND), || (meaning logical OR) and! (Meaning logical NOT).

|  |  |
| --- | --- |
| Operator | Description |
| && | And operator. It performs logical conjunction of two expressions. (if both expressions evaluate to True, result is True. If either expression evaluates to False, the result is False) |
| || | Or operator. It performs a logical disjunction on two expressions. (if either or both expressions evaluate to True, the result is True) |
| ! | Not operator. It performs logical negation on an expression. |

**Bitwise Operators:** C provides a special operator for bit operation between two variables.

|  |  |
| --- | --- |
| Operator | Description |
| << | Binary Left Shift Operator |
| >> | Binary Right Shift Operator |
| ~ | Binary Ones Complement Operator |
| & | Binary AND Operator |
| ^ | Binary XOR Operator |
| | | Binary OR Operator |

**3.2.5 ASSIGNMENT OPERATORS:** Assignment operators applied to assign the result of an expression to a variable. C has a collection of shorthand assignment operators.

|  |  |
| --- | --- |
| Operator | Description |
| = | Assign |
| += | Increments then assign |
| -= | Decrements then assign |
| \*= | Multiplies then assign |
| /= | Divides then assign |
| %= | Modulus then assign |
| <<= | Left shift and assign |
| >>= | Right shift and assign |
| &= | Bitwise AND assign |
| ^= | Bitwise exclusive OR and assign |
| |= | Bitwise inclusive OR and assign |

**3.2.6 CONDITIONAL OPERATOR:** C offers a ternary operator which is the conditional operator (?: in combination) to construct conditional expressions.

|  |  |
| --- | --- |
| Operator | Description |
| ? : | Conditional Expression |

**3.2.7 SPECIAL OPERATORS:** C supports some special operators

|  |  |
| --- | --- |
| Operator | Description |
| sizeof() | Returns the size of a memory location. |
| & | Returns the address of a memory location. |
| \* | Pointer to a variable. |

**Example: Program to demonstrate the use of sizeof operator**

#include <stdio.h>

void main()

{

int i=10; /\* Variables Defining and Assign values \*/

printf("integer: %d\n", sizeof(i));

}

**Program Output:**



**3.3 TYPE CASTING IN C**

Type Casting in C is used to convert a variable from one data type to another data type, and after type casting compiler treats the variable as of the new data type.

**3.3.1 WITHOUT TYPE CASTING**

**Example:**

#include <stdio.h>

main ()

{

int a;

a = 15/6;

printf("%d",a);

**}**

**Program Output:**

In the above C program, 15/6 alone will produce integer value as 2.



**3.3.2 AFTER TYPE CASTING**

#include <stdio.h>

main ()

{

float a;

a = (float) 15/6;

printf("%f",a);

}

**Program Output:**

After type cast is done before division to retain float value 2.500000.

**EXAMPLE: Writes the words "Hello, World!" on the screen \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

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

return;

}

**Program Output:**



The above example has been used to print Hello, World! Text on the screen.

|  |  |
| --- | --- |
| /\* Comments \*/ | Comments are a way of explaining what makes a program. The compiler ignores comments and used by others to understand the code. **Or** This is a comment block, which is ignored by the compiler. Comment can be used anywhere in the program to add info about the program or code block, which will be helpful for developers to understand the existing code in the future easily. |
| #include<stdio.h> | stdio is standard for input/output, this allows us to use some commands which includes a file called stdio.h.  **Or This** is a preprocessor command. That notifies the compiler to include the header file stdio.h in the program before compiling the source-code. |
| int/void main() | int/void is a return value, which will be explained in a while. |
| main() | The main () is the main function where program execution begins. Every C program must contain only one main function.  **Or** This is the main function, which is the default entry point for every C program and the void in front of it indicates that it does not return a value. |
| Braces | Two curly brackets "{...}" are used to group all statements**. or**  Curly braces which shows how much the main () function has its scope. |
| printf() | It is a function in C, which prints text on the screen. **Or** This is another pre-defined function of C which is used to be displayed text string in the screen. |
| return 0 | At the end of the main function returns value 0. |

**Let's look into various parts of the above C program.**

**3.4 BASIC STRUCTURE OF C PROGRAM**

The Documentation section usually contains the collection of comment lines that involves the name of the program, author's or programmer's name and few other details. The second part is the link-section which instructs the compiler to connect to the various functions from the system library. The Definition section describes all the symbolic-constants. The global declaration section is used to define those variables that are used globally within the entire program and is used in more than one function. This section also declares all the user-defined functions. Then comes the main (). All C programs must have a main () which contains two parts:

* Declaration part
* Execution part

**The declaration part** is used to declare all variables that will be used within the program. There needs to be at least one statement in the executable part, and these two parts are declared within the opening and closing curly braces of the main ().

**The execution of the program** begins at the opening brace '{' and ends with the closing brace '}'. Also, it has to be noted that all the statements of these two parts need to be terminated with a semi-colon.

The sub-program section deals with all user-defined functions that are called from the main(). These user-defined functions are declared and usually defined after the main() function.

As we all know the three essential functions of a computer are reading, processing and writing data. Majority of the programs take data as input, and then after processing the processed data is being displayed which is called information. In C programming you can use scanf() and printf() predefined function to read and print data.

**Example:**

#include<stdio.h>

void main()

{

int a,b,c;

printf("Please enter any two numbers: \n");

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

c = a + b;

printf("The addition of two number is: %d", c);

}

**Output:**

Please enter any two numbers:

12

3

The addition of two number is:15

The above program scanf() is used to take input from the user, and respectively printf() is used to display output result on the screen.

**CHAPTER FOUR**

**UNDERSTAND STANDARD INPUT AND OUTPUT OPERATIONS**

**4.1 MANAGING INPUT/OUTPUT**

I/O operations are useful for a program to interact with users. stdlib is the standard C library for input-output operations. While dealing with input-output operations in C, two important streams play their role. These are:

* Standard Input (stdin)
* Standard Output (stdout)

**Standard input** **or stdin** is used for taking input from devices such as the keyboard as a data stream.

**Standard output or stdout** is used for giving output to a device such as a monitor. For using I/O functionality, programmers must include stdio header-file within the program.

**4.2 READING CHARACTER IN C**

The easiest and simplest of all I/O operations are taking a character as input by reading that character from standard input (keyboard). getchar() function can be used to read a single character. This function is alternate to scanf() function.

**Syntax:**

var\_name = getchar();

**Example:**

#include<stdio.h>

void main()

{

char title;

title = getchar();

}

There is another function to do that task for files: getc which is used to accept a character from standard input.

**Syntax:**

int getc(FILE \*stream);

**4.3 WRITING CHARACTER IN C**

Similar to getchar() there is another function which is used to write characters, but one at a time.

**Syntax:**

putchar(var\_name);

**Example:**

#include<stdio.h>

void main()

{

char result = 'P';

putchar(result);

putchar('\n');

}

Similarly, there is another function putc which is used for sending a single character to the standard output.

**Syntax:**

int putc(int c, FILE \*stream);

**4.4 FORMATTED INPUT**

It refers to an input data which has been arranged in a specific format. This is possible in C using scanf(). We have already encountered this and familiar with this function.

**Syntax:**

scanf("control string", arg1, arg2, ..., argn);

The field specification for reading integer inputted number is:

%w sd

Here the % sign denotes the conversion specification; w signifies the integer number that defines the field width of the number to be read. d defines the number to be read in integer format.

**Example:**

#include<stdio.h>

void main()

{

int var1= 60;

int var2= 1234;

scanf("%2d %5d", &var1, &var2);

}

Input data items should have to be separated by spaces, tabs or new-line and the punctuation marks are not counted as separators.

**4.5 READING AND WRITING STRINGS IN C**

There are two popular library functions gets() and puts() provides to deal with strings in C.

**gets**: The char \*gets(char \*str) reads a line from stdin and keeps the string pointed to by the str and is terminated when the new line is read or EOF is reached. The declaration of gets() function is:

**Syntax:**

char \*gets(char \*str);

Where str is a pointer to an array of characters where C strings are stored.

**puts**: The function - int puts(const char \*str) is used to write a string to stdout, but it does not include null characters. A new line character needs to be appended to the output. The declaration is:

**Syntax:**

int puts(const char \*str)

where str is the string to be written in C

**4.6 C FORMAT SPECIFIERS**

Format specifiers can be defined as the operators which are used in association with printf() function for printing the data that is referred by any object or any variable. When a value is stored in a particular variable, then you cannot print the value stored in the variable straightforwardly without using the format specifiers. You can retrieve the data that are stored in the variables and can print them onto the console screen by implementing these format specifiers in a printf() function.

Format specifiers start with a percentage % operator and followed by a special character for identifying the type of data. There are mostly six types of format specifiers that are available in C.

**4.6.1 LIST OF FORMAT SPECIFIERS IN C**

|  |  |
| --- | --- |
| Format Specifier | Description |
| %d | Integer Format Specifier |
| %f | Float Format Specifier |
| %c | Character Format Specifier |
| %s | String Format Specifier |
| %u | Unsigned Integer Format Specifier |
| %ld | Long Int Format Specifier |

* **Integer Format Specifier %D:**The %d format specifier is implemented for representing integer values. This is used with printf() function for printing the integer value stored in the variable.

Syntax:

printf("%d",<variable name>);

* **Float Format Specifier %f:** The %f format specifier is implemented for representing fractional values. This is implemented within printf() function for printing the fractional or floating value stored in the variable. Whenever you need to print any fractional or floating data, you have to use %f format specifier.

**Syntax:**

printf("%f", <variable name>);

* **Character Format Specifier %c:** The %c format specifier is implemented for representing characters. This is used with printf() function for printing the character stored in a variable. When you want to print a character data, you should incorporate the %c format specifier.

**Syntax:**

printf("%c",<variable name>);

* **String Format Specifier %s:** The %s format specifier is implemented for representing strings. This is used in printf() function for printing a string stored in the character array variable. When you have to print a string, you should implement the %s format specifier.

**Syntax:**

printf("%s",<variable name>);

* **Unsigned Integer Format Specifier %u:** The %u format specifier is implemented for fetching values from the address of a variable having unsigned decimal integer stored in the memory. This is used within printf() function for printing the unsigned integer variable.

**Syntax:**

printf("%u",<variable name>);

* **Long Int Format Specifier %ld:** The %ld format specifier is implemented for representing long integer values. This is implemented with printf() function for printing the long integer value stored in the variable.

**Syntax:**

printf("%ld",<variable name>);

**CHAPTER FIVE**

**UNDERSTAND CONTROL STRUCTURES (DECISION MAKING AND LOOPS)**

**5.1 WHAT IS CONTROL** **STRUCTURE** **IN** **C**?

Control Structure defines how the statements in the program are going to execute.

A statement is a single line or instruction in a program. Control structure in c also known as control flow tells the order at which each statement is going to execute. Control structures are used to alter the flow of execution of the program.

Why do we need to alter the program flow? The reason is because of the **decision making***.* We use control structures to make decisions and alter the direction of program flow in one or the other path(s) available. There are three types of control structures available in c

* sequence structure (straight line paths)
* selection structure (one or many branches)
* loop structure (repetitionof a set of activities)

**5.2 SEQUENCE STRUCTURE (STRAIGHT LINE PATHS)**

A set of statements in a C program gets executed sequentially in the order in which they are written and appear, this occurs when there is no jump based statements or repetitions of certain calculations. This sequential structure follow step by step execution of program one after another as serial operations. This can be illustrated by the following flowchart: In fact, most processes, even of very complex problems will usually follow this elementary sequence structure or logic.

**5.3 SELECTION STRUCTURE (ONE OR MANY BRANCHES)** **C OR SWITCH OR IF-ELSE-IF LADDER**

The selection structure uses conditions and based on the decision taken by the computer after comparison data, one of the available alternatives is selected. This structure comes in different forms of If Statement in C and it can be used in various forms depending on the situation and complexity.

There are four different types of if statement in C. These are:

* Simple if Statement
* if-else Statement
* Nested if-else Statement
* else-if Ladder

**5.3.1 SIMPLE IF STATEMENTS IN C**

In C is used to control the program flow based on some condition, it's used to execute some statement code block if the expression is evaluated to true. Otherwise, it will get skipped. This is the simplest way to modify the control flow of the program.

**The basic format of if statement is:**

Syntax:

if(test\_expression)

{

statement 1;

statement 2;

...

}

'Statement n' can be a statement or a set of statements, and if the test expression is evaluated to true, the statement block will get executed, or it will get skipped.



**Figure - Flowchart of if Statement**

**Example of a C Program to Demonstrate if Statement**

**Example:**

#include<stdio.h>

main()

{

int a = 15, b = 20;

if (b & gt; a) {

printf("b is greater");

}

}

**Program Output:**



**Example:**

#include<stdio.h>

main()

{

int number;

printf( & quot; Type a number: & quot;);

scanf( & quot; % d & quot;, & amp; number);

/\* check whether the number is negative number \*/ if (number & lt; 0) {

/\* If it is a negative then convert it into positive. \*/

number = -number;

printf( & quot; The absolute value is % d\ n & quot;, number);

}

grtch();

}

**Program Output:**



**5.3.2 IF ELSE STATEMENTS IN C**

If else statements in C is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise execute else statement code block.

**The basic format of if else statement is:**

**Syntax:**

if(test\_expression)

{

//execute your code

}

else

{

//execute your code

}

**OR,**

Whenever you want to perform a set of operations based on a condition IF-ELSE is used.

**if**(conditional-expression) {

*// code*

} **else** {

*// code*

}



**Figure - Flowchart of if-else Statement**

**Example of a C Program to demonstrate if-else Statement**

**Example:**

#include<stdio.h>

main()

{

int a, b;

printf("Please enter the value for a:");

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

printf("\nPlease enter the value for b:");

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

if (a & gt; b) {

printf("\n a is greater");

} else {

printf("\n b is greater");

}

}

**Program Output:**



**Example:**

#include<stdio.h>

main() {

int num;

printf("Enter the number:");

scanf("%d", num);

/\* check whether the number is negative number \*/ if (num < 0)

printf("The number is negative.");

else

printf("The number is positive.");

}

**Program Output:**



**5.3.3 NESTED IF-ELSE STATEMENTS IN C**

Nested if else statements play an important role in C programming, it means you can use conditional statements inside another conditional statement.

**The basic format of Nested if else statement is:**

**Syntax:**

if(test\_expression one)

{

if(test\_expression two) {

//Statement block Executes when the boolean test expression two is true.

}

}

else

{

//else statement block

}

**Example of a C Program to Demonstrate Nested if-else Statement**

**Example:**

#include<stdio.h>

main()

{

int x=20,y=30;

if(x==20)

{

if(y==30)

{

printf("value of x is 20, and value of y is 30.");

}

}

}

Execution of the above code produces the following result.

**Output:**

value of x is 20, and value of y is 30.

**5.3.4 ELSE-IF STATEMENTS IN C**

Is like another if condition, it's used in a program when if statement having multiple decisions. But some situations may arise where we may have to change the order of execution of statements depending on some specific conditions. This involves a kind of decision making from a set of calculations. This type of structure requires that the programmers indicate several conditions for evaluation within a program. The statement(s) will get executed only if the condition becomes true and optionally, alternative statement or set of statements will get executed if the condition becomes false.

**Example of a C Program to Demonstrate else if Ladder Statement**

**#include<stdio.h>**

main()

{

int a, b;

printf("Please enter the value for a:");

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

printf("\nPlease enter the value for b:");

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

if (a & gt; b)

{

printf("\n a is greater than b");

}

else if (b & gt; a)

{

printf("\n b is greater than a");

}

else

{

printf("\n Both are equal");

}

}

**Program Output:**





**5.3.5 C SWITCH STATEMENT**

C switch statement is used when you have multiple possibilities for the if statement. Switch case will allow you to choose from multiple options. When we compare it to a general electric switchboard, you will have many switches in the switchboard but you will only select the required switch, similarly, the switch case allows you to set the necessary statements for the user.

**The basic format of the switch statement is:**

switch(variable)

{

case 1:

//execute your code

break;

case n:

//execute your code

break;

default:

//execute your code

break;

}

After the end of each block it is necessary to insert a break statement because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from every case onwards after matching the case block.

**Example of a C Program to Demonstrate Switch Statement**

#include<stdio.h>

main()

{

int a;

printf("Please enter a no between 1 and 5: ");

scanf("%d",&a);

switch(a)

{

case 1:

printf("You chose One");

break;

case 2:

printf("You chose Two");

break;

case 3:

printf("You chose Three");

break;

case 4:

printf("You chose Four");

break;

case 5:

printf("You chose Five.");

break;

default :

printf("Invalid Choice. Enter a no between 1 and 5");

break;

}

}

**Program Output:**



When none of the cases is evaluated to true, the default case will be executed, and break statement is not required for default statement

**EXAMPLE**

**switch**(conditional-expression) {

**case** value1:

*// code*

**break**; *// optional*

**case** value2:

*// code*

**break**; *// optional*

...

**default**:

*// code to be executed when all the above cases are not matched;*

}

**The basic format of else if statement is:**

**Syntax:**

if(test\_expression)

{

//execute your code

}

else if(test\_expression n)

{

//execute your code

}

else

{

//execute your code

}

**5.4 LOOP STRUCTURE (REPETITION OF A SET OF ACTIVITIES)**

Loop control statements are used to change the normal sequence of execution of the loop. This type of control structure allows a condition to be performed repetitively.

A computer is the most suitable machine for performing repetitive tasks, and it can perform a task thousands of times. Every programming language has the feature to instruct to do such repetitive tasks with the help of certain statements. Sometimes it is necessary for the program to execute the statement several times. A loop executes a block of commands a specified number of times until a condition is met. The process of repeatedly executing a collection of statements is called looping. The statements get executed many numbers of times based on the condition. But if the condition is given in such logic that the repetition continues any number of times with no fixed condition to stop looping those statements, then this type of looping is called infinite looping. C supports the following types of loops:

* while loops
* do-while loops
* for loops

All are slightly different and provide loops for different situations.

|  |  |  |
| --- | --- | --- |
| Statement | Syntax | Description |
| break statement | break; | It is used to terminate loop or switch statements. |
| continue statement | continue; | It is used to suspend the current loop iteration and transfer control to the loop for the next iteration. |
| goto statement | goto labelName;  labelName: statement; | It transfers the current program execution sequence to some other part of the program. |



**Figure - Flowchart of Looping**

**5.4.1 WHILE LOOP CONTROL STATEMENTS**

while loop is a most basic loop in C programming. While loop has one control condition, and executes as long the condition is true.  The condition of the loop is tested before the body of the loop is executed, hence it is called an entry-controlled loop. While is also used to iterate a set of statements based on a condition and allows to repeatedly run the same block of code until a condition is met. Usually while is preferred when number of iterations is not known in advance.

**while**(condition) {

*// code*

}

**The basic format of while loop statement is:**

While (condition)

{

statement(s);

Incrementation;

}



**Figure - Flowchart of while loop**

**Example of a C Program to Demonstrate while loop**

Example:

#include<stdio.h>

int main ()

{

/\* local variable Initialization \*/ int n = 1,times=5;

/\* while loops execution \*/ while( n <= times )

{

printf("C while loops: %d\n", n);

n++;

}

return 0;

}

**Program Output:**



**5.4.2 Do-While LOOP**

C do while loops are very similar to the while loops, but it always executes the code block at least once and furthermore as long as the condition remains true. This is an **exit-controlled loop**.Do-while is also used to iterate a set of statements based on a condition. It is mostly used when you need to execute the statements at least once.

**do** {

*// code*

} **while** (condition);

**The basic format of do while loop statement is:**

Syntax:

do

{

statement(s);

}while( condition );



**Figure - Flowchart of do while loop**

**Example of a C Program to demonstrate do while loop**

Example:

#include<stdio.h>

int main ()

{

/\* local variable Initialization \*/ int n = 1,times=5;

/\* do loops execution \*/ do

{

printf("C do while loops: %d\n", n);

n = n + 1;

}while( n <= times );

return 0;

}

**Program Output:**



**5.4.3 FOR LOOPS**

C for loops is very similar to a while loops in that it continues to process a block of code until a statement becomes false, and everything is defined in a single line. The for loop is also **entry-controlled** loop.

**The basic format of for loop statement is:**

Syntax:

for ( init; condition; increment )

{

statement(s);

}



**Figure - Flowchart of for loop**

**Example of a C Program to Demonstrate for loop**

#include<stdio.h>

int main ()

{

  /\* local variable Initialization \*/  int n,times=5;;

/\* for loops execution \*/ for( n = 1; n <= times; n = n + 1 )

{

printf("C for loops: %d\n", n);

}

return 0;

}

**Program Output:**



**5.5 GOTO STATEMENT IN C**

C supports a unique form of a statement that is the **goto** Statement which is used to branch unconditionally within a program from one point to another. Although it is not a good habit to use the goto statement in C, there may be some situations where the use of the goto statement might be desirable.

The goto statement is used by programmers to change the sequence of execution of a C program by shifting the control to a different part of the same program.

**The general form of the goto statement is:**

**goto label;**

A label is an identifier required for goto statement to a place where the branch is to be made. A label is a valid variable name which is followed by a colon and is put immediately before the statement where the control needs to be jumped/transferred unconditionally.

**Syntax:**

goto label;

- - -- -   -

- - - - - - - -

label:

statement - X;

/\* This the forward jump of goto statement \*/

or

label:

- - -- -   -

- - - - - - - -

goto label;

/\*This is the backward jump of goto statement \*/

**An Example of a C Program to Demonstrate goto Statement**

#include<stdio.h>

void main()

{

int age;

g: //label name

printf("you are Eligible\n");

s: //label name

printf("you are not Eligible");

printf("Enter you age:");

scanf("%d", &age);

if(age>=18)

goto g; //goto label g

else

goto s; //goto label s

getch();

}

**CHAPTER SIX**

**UNDERSTAND THE FUNCTIONS AND SCOPE RULES**

**6.1 FUNCTIONS IN C**

Function is a sub-routine which contains set of statements. Usually functions are written when multiple calls are required to same set of statements which increases re-usuability and modularity. C function is a self-contained block of statements that can be executed repeatedly whenever we need it. There are two types of functions are present in C

* Library Functions
* User defined functions:

**Library Functions:** Library functions are the in-built functions which are declared in header files like printf(),scanf(),puts(),gets() etc.,

* The system provided these functions and stored in the library. Therefore it is also called Library Function .g. scanf(), printf(), strcpy, strlwr, strcmp, strlen, strcat etc.
* To use these functions, you just need to include the appropriate C header files.

**User defined functions:** User defined functions are the ones which are written by the programmer based on the requirement. These functions are defined by the user at the time of writing the program.

**How to Declare a Function**

return\_type function\_name(parameters);

**How to call a Function**

function\_name (parameters)

**How to define a Function**

return\_type function\_name(parameters)

{

*//code*

}

**6.2 PARTS OF FUNCTION**

* Function Prototype (function declaration)
* Function Definition
* Function Call

**6.2.1 FUNCTION PROTOTYPE**

Syntax:

dataType functionName (Parameter List)

**Example:**

int addition();

**FUNCTION DEFINITION**

**Syntax:**

returnType functionName(Function arguments){

//body of the function

}

**Example:**

int addition()

{

}

**6.2.2 CALLING A FUNCTION IN C**

Program to illustrate the Addition of Two Numbers using User Defined Function

**Example:**

#include<stdio.h>

/\* function declaration \*/int addition();

int main()

{

/\* local variable definition \*/ int answer;

/\* calling a function to get addition value \*/ answer = addition();

printf("The addition of the two numbers is: %d\n",answer);

return 0;

}

/\* function returning the addition of two numbers \*/int addition()

{

/\* local variable definition \*/ int num1 = 10, num2 = 5;

return num1+num2;

}

**Program Output:**

The addition of the two numbers is: 15

**6.3 C LIBRARY FUNCTIONS**

The C library functions are provided by the system and stored in the library. The C library function is also called an inbuilt function in C programming. To use Inbuilt Function in C, you must include their respective header files, which contain prototypes and data definitions of the function.

**C Program to Demonstrate the Use of Library Functions**

Example:

#include<stdio.h>

#include<ctype.h>

#include<math.h>

void main()

{

int i = -10, e = 2, d = 10; /\* Variables Defining and Assign values \*/ float rad = 1.43;

double d1 = 3.0, d2 = 4.0;

printf("%d\n", abs(i));

printf("%f\n", sin(rad));

printf("%f\n", cos(rad));

printf("%f\n", exp(e));

printf("%d\n", log(d));

printf("%f\n", pow(d1, d2));

}

**Program Output:**



**6.4 BENEFITS OF USING THE FUNCTION IN C**

* The function provides modularity.
* The function provides reusable code.
* In large programs, debugging and editing tasks is easy with the use of functions.
* The program can be modularized into smaller parts.
* Separate function independently can be developed according to the needs.

**6.5 SCOPE OF VARIABLES**

A scope is a region of the program, and the scope of variables refers to the area of the program where the variables can be accessed after its declaration. In C every variable defined in scope. You can define scope as the section or region of a program where a variable has its existence; moreover, that variable cannot be used or accessed beyond that region.

In C programming, variable declared within a function is different from a variable declared outside of a function. The variable can be declared in three places. These are:

|  |  |
| --- | --- |
| Position | Type |
| Inside a function or a block. | Local variables |
| Out of all functions. | Global variables |
| In the function parameters. | Formal parameters |

**6.5.1 Local Variables:** Variables that are declared within the function block and can be used only within the function is called local variables.

**Local Scope or Block Scope:** A local scope or block is collective program statements put in and declared within a function or block (a specific region enclosed with curly braces) and variables lying inside such blocks are termed as local variables. All these locally scoped statements are written and enclosed within left ({) and right braces (}) curly braces. There's a provision for nested blocks also in C which means there can be a block or a function within another block or function. So it can be said that variable(s) that are declared within a block can be accessed within that specific block and all other inner blocks of that block, but those variables cannot be accessed outside the block.

**Example:**

#include <stdio.h>

int main ()

{

/\* local variable definition and initialization \*/ int x,y,z;

/\* actual initialization \*/ x = 20;

y = 30;

z = x + y;

printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

return 0;

}

**6.5.2 Global Variables:** Variables that are declared outside of a function block and can be accessed inside the function is called global variables.

**Global Scope:** Global variables are defined outside a function or any specific block, in most of the case, on the top of the C program. These variables hold their values all through the end of the program and are accessible within any of the functions defined in your program. Any function can access variables defined within the global scope, i.e., its availability stays for the entire program after being declared.

**Example:**

#include <stdio.h>

/\* global variable definition \*/int z;

int main ()

{

/\* local variable definition and initialization \*/ int x,y;

/\* actual initialization \*/ x = 20;

y = 30;

z = x + y;

printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

return 0;

}

**Global Variable Initialization:** After defining a local variable, the system or the compiler won't be initializing any value to it. You have to initialize it by yourself. It is considered good programming practice to initialize variables before using. Whereas in contrast, global variables get initialized automatically by the compiler as and when defined. Here's how based on datatype; global variables are defined.

|  |  |
| --- | --- |
| datatype | Initial Default Value |
| int | 0 |
| char | '\0' |
| float | 0 |
| double | 0 |
| pointer | NULL |

**6.6 WHAT IS RECURSION?**

C is a powerful programming language having capabilities like an iteration of a set of statements 'n' number of times. The same concepts can be done using functions also. Recursion can be defined as the technique of replicating or doing again an activity in a self-similar way calling itself again and again, and the process continues till specific condition reaches. In the world of programming, when your program lets you call that specific function from inside that function, then this concept of calling the function from itself can be termed as recursion, and the function in which makes this possible is called recursive function.

**Example Syntax:**

void rec\_prog(void) {

rec\_prog(); /\* function calls itself \*/}

int main(void) {

rec\_prog();

return 0;

}

C program allows you to do such calling of function within another function, i.e., recursion. But when you implement this recursion concept, you have to be cautious in defining an exit or terminating condition from this recursive function, or else it will continue to an infinite loop, so make sure that the condition is set within your program.

**Example:** Factorial Program

#include<stdio.h>

#include<conio.h>

int fact(int f) {

if (f & lt; = 1) {

printf("Calculated Factorial");

return 1;

}

return f \* fact(f - 1);

}

int main(void) {

int f = 12;

clrscr();

printf("The factorial of %d is %d \n", f, fact(f));

getch();

return 0;

}

Fibonacci Program

**Example:**

#include<stdio.h>

#include<conio.h>

int fibo(int g) {

if (g == 0) {

return 0;

}

if (g == 1) {

return 1;

}

return fibo(g - 1) + fibo(g - 2);

}

int main(void) {

int g;

clrscr();

for (g = 0; g & lt; 10; g++) {

printf("\nNumbers are: %d \t ", fibonacci(g));

}

getch();

return 0;

}

**6.7 STORAGE CLASSES TYPES IN C**

Storage Classes are associated with variables for describing the features of any variable or function in the C program. These storage classes deal with features such as scope, lifetime and visibility which helps programmers to define a particular variable during the program's runtime. These storage classes are preceded by the data type which they had to modify. There are four storage classes types in C:

* auto
* register
* static
* extern

**Auto Storage Class:** auto comes by default with all local variables as its storage class. The keyword auto is used to define this storage class explicitly

**Syntax:**

int roll; // contains auto by default

**is the same as:**

auto int roll;    // in addition, we can use auto keyword

The above example has a variable name roll with auto as a storage class. This storage class can only be implemented with the local variables.

**Register Storage Class:** This storage class is implemented for classifying local variables whose value needs to be saved in a register in place of RAM (Random Access Memory). This is implemented when you want your variable the maximum size equivalent to the size of the register. It uses the keyword register.

**Syntax:**

register int  counter;

Register variables are used when implementing looping in counter variables to make program execution fast. Register variables work faster than variables stored in RAM (primary memory).

**Example:**

for(register int counter=0; counter<=9; counter++)

{

// loop body

}

static storage class

This storage class uses static variables that are used popularly for writing programs in C language. Static variables preserve the value of a variable even when the scope limit exceeds. Static storage class has its scope local to the function in which it is defined.

On the other hand, global static variables can be accessed in any part of your program. The default value assigned is '0' by the C compiler. The keyword used to define this storage class is static.

**Example:**

static int var = 6;

extern Storage class

The extern storage class is used to feature a variable to be used from within different blocks of the same program. Mainly, a value is set to that variable which is in a different block or function and can be overwritten or altered from within another block as well. Hence it can be said that an extern variable is a global variable which is assigned with a value that can be accessed and used within the entire program. Moreover, a global variable can be explicitly made an extern variable by implementing the keyword 'extern' preceded the variable name.

**Example:**

#include <stdio.h>

int val;

extern void funcExtern();

main()

{

val = 10;

funcExtern();

}

**Example:**

#include <stdio.h>

extern int val; // now the variable val can be accessed and used from anywhere

// within the program

void funcExtern()

{

printf("Value is: %d\n", val);

}

**CHAPTER SEVEN**

**UNDERSTAND ARRAYS AND STRINGS**

**7.1 DEFINE AN ARRAY IN C**

Array is a collection of similar data which is stored in continuous memory addresses. Array values can be fetched using index. Index starts from 0 to size-1. Is a data structure in C programming, which can store a fixed-size sequential collection of elements of the same data type. For example, if you want to store ten numbers, it is easier to define an array of 10 lengths, instead of defining ten variables. In the C programming language, an array can be**One-Dimensional**, **Two-Dimensional,** and **Multidimensional**.

**A One-Dimensional Array:** This is called a one-dimensional array. An array type can be any valid C data types, and array size must be an integer constant greater than zero.

**Syntax**

**One dimentional Array:**

data-type array-name[size];

**Two dimensional array:**

data-type array-name[size][size];

OR

double amount[5];

**7.2 INITIALIZE AN ARRAY IN C**

Arrays can be initialized at declaration time:

int age[5]={22,25,30,32,35};

Initializing each element separately in a loop:

int myArray[5];

int n = 0;

// Initializing elements of array seperately

for(n=0;n<sizeof(myArray)/sizeof(myArray[0]);n++)

{

myArray[n] = n;

}

**A Pictorial Representation of the Array:**



**7.3 ACCESSING ARRAY ELEMENTS IN C**

**Example:**

int myArray[5];

int n = 0;

// Initializing elements of array seperately

for(n=0;n<sizeof(myArray)/sizeof(myArray[0]);n++)

{

myArray[n] = n;

}

int a = myArray[3]; // Assigning 3rd element of array value to integer 'a'.

**7.4 STRINGS DECLARATION IN C**

In C programming, the one-dimensional array of characters are called strings, which is terminated by a null character '\0'. here are two ways to declare a string in C programming:

**Example:**Through an array of characters.

char name[6];

Through pointers.

char \*name;

**7.5 STRINGS INITIALIZATION IN C**

**Example:**

char name[6] = {'C', 'l', 'o', 'u', 'd', '\0'};

or

char name[] = "Cloud";

Memory Representation of Above Defined String in C



**Example:**

#include<stdio.h>

int main ()

{

char name[6] = {'C', 'l', 'o', 'u', 'd', '\0'};

printf("Classwork%s\n", name );

return 0;

}

**Program Output:**

ClassworkCloud

**CHAPTER EIGHT**

**UNDERSTAND POINTER OPERATIONS**

**8.1 POINTER DEFINITION IN C**

A pointer is a variable in C, and pointers value is the address of a memory location.

Syntax:

type \*variable\_name;

**Example:**

int  \*width;

char  \*letter;

**8.2 BENEFITS OF USING POINTERS IN C**

* Pointers allow passing of arrays and strings to functions more efficiently.
* Pointers make it possible to return more than one value from the function.
* Pointers reduce the length and complexity of a program.
* Pointers increase the processing speed.
* Pointers save the memory.

**8.3 HOW TO USE POINTERS IN C**

**Example:**

#include<stdio.h>

int main ()

{

int n = 20, \*pntr;  /\* actual and pointer variable declaration \*/

pntr = &n;  /\* store address of n in pointer variable\*/

printf("Address of n variable: %x\n", &n );

/\* address stored in pointer variable \*/ printf("Address stored in pntr variable: %x\n", pntr );

/\* access the value using the pointer \*/ printf("Value of \*pntr variable: %d\n", \*pntr );

return 0;

}

Address of n variable: 2cb60f04

Address stored in pntr variable: 2cb60f04

Value of \*pntr variable: 20

**CHAPTER NINE**

**UNDERSTAND STRUCTURES AND UNION DATA TYPES**

**9.1 DEFINING A STRUCTURE IN C**

The structure is a user-defined data type in C, which is used to store a collection of different kinds of data.

* The structure is something similar to an array; the only difference is array is used to store the same data types.
* **struct** keyword is used to declare the structure in C.
* Variables inside the structure are called **members of the structure**.

**Syntax:**

struct structureName

{

//member definitions

};

**Example:**

struct Courses

{

char WebSite[50];

char Subject[50];

int Price;

};

**9.2 ACCESSING STRUCTURE MEMBERS IN C**

**Example:**

#include<stdio.h>

#include<string.h>

struct Courses

{

char WebSite[50];

char Subject[50];

int Price;

};

void main( )

{

struct Courses C;

//Initialization

strcpy( C.WebSite, "ogitech.in");

strcpy( C.Subject, "The C Programming Language");

C.Price = 200;

//Print

printf( "WebSite : %s\n", C.WebSite);

printf( "Book Author : %s\n", C.Subject);

printf( "Book Price : %d\n", C.Price);

}

**Program Output:**

WebSite : ogitech.in

Book Author: The C Programming Language

Book Price : 200

**9.3 UNIONS DATA TYPE IN C**

Unions are user-defined data type in C, which is used to store a collection of different kinds of data, just like a structure. However, with unions, you can only store information in one field at any one time.

* Unions are like structures except it used less memory.
* The keyword **union** is used to declare the structure in C.
* Variables inside the union are called **members of the union**.

**9.3.1 DEFINING A UNION IN C**

**Syntax:**

union unionName

{

//member definitions

};

**Example:**

union Courses

{

char WebSite[50];

char Subject[50];

int Price;

};

**9.3.2 ACCESSING UNION MEMBERS IN C**

**Example:**

#include<stdio.h>

#include<string.h>

union Courses

{

char WebSite[50];

char Subject[50];

int Price;

};

void main( )

{

union Courses C;

strcpy( C.WebSite, "w3schools.in");

printf( "WebSite : %s\n", C.WebSite);

strcpy( C.Subject, "The C Programming Language");

printf( "Book Author : %s\n", C.Subject);

C.Price = 0;

printf( "Book Price : %d\n", C.Price);

}

**Program Output:**

WebSite : w3schools.in

Book Author: The C Programming Language

C files I/O functions handle data on a secondary storage device, such as a hard disk.

C can handle files as **Stream-oriented data (Text) files**, and **System oriented data (Binary)** files.

|  |  |
| --- | --- |
| Stream-oriented data files | The data is stored in the same manner as it appears on the screen. The I/O operations like buffering, data conversions, etc. take place automatically. |
| System-oriented data files | System-oriented data files are more closely associated with the OS and data stored in memory without converting into text format. |

**9.4 C ENUMS**

In this tutorial, you will learn about enum (enumeration) in C programming with the help of examples. In C programming, an enumeration type (also called enum) is a data type that consists of integral constants. To define enums, the enum keyword is used.

enum flag {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).

// Changing default values of enum constants

enum suit {

club = 0,

diamonds = 10,

hearts = 20,

spades = 3,

};

**9.4.1 Enumerated Type Declaration**

When you define an enum type, the blueprint for the variable is created. Here's how you can create variables of enum types.

enum boolean {false, true};

enum boolean check; // declaring an enum variable

Here, a variable check of the type enum boolean is created.

You can also declare enum variables like this.

enum boolean {false, true} check; num boolean {false, true} check;

Here, the value of false is equal to 0 and the value of true is equal to 1.

**Example: Enumeration Type**

#include <stdio.h>

enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

int main()

{

// creating today variable of enum week type

enum week today;

today = Wednesday;

printf("Day %d",today+1);

return 0;

}

**Output**

Day 4

**9.4.2 WHY ENUMS ARE USED?**

An enum variable can take only one value. Here is an example to demonstrate it,

#include <stdio.h>

enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

int main()

{

// creating today variable of enum week type

enum week today;

today = Wednesday;

printf("Day %d",today+1);

return 0;

}

**Output**

Day 4

**9.4.3 WHY ENUMS ARE USED?**

An enum variable can take only one value. Here is an example to demonstrate it,

#include <stdio.h>

enum suit {

club = 0,

diamonds = 10,

hearts = 20,

spades = 3

} card;

int main()

{

card = club;

printf("Size of enum variable = %d bytes", sizeof(card));

return 0;

}

**Output**

Size of enum variable = 4 bytes

Here, we are getting 4 because the size of int is 4 bytes.

This makes enum a good choice to work with flags.

**9.4.4 HOW TO USE ENUMS FOR FLAGS?**

Let us take an example,

enum designFlags {

ITALICS = 1,

BOLD = 2,

UNDERLINE = 4

} button;

Suppose you are designing a button for Windows application. You can set flags ITALICS, BOLD and UNDERLINE to work with text.

There is a reason why all the integral constants are a power of 2 in the above pseudocode.

// In binary

ITALICS = 00000001

BOLD = 00000010

UNDERLINE = 00000100

Since the integral constants are a power of 2, you can combine two or more flags at once without overlapping using bitwise OR | operator. This allows you to choose two or more flags at once. For example,

#include <stdio.h>

enum designFlags {

BOLD = 1,

ITALICS = 2,

UNDERLINE = 4

};

int main() {

int myDesign = BOLD | UNDERLINE;

// 00000001

// | 00000100

// \_\_\_\_\_\_\_\_\_\_\_

// 00000101

printf("%d", myDesign);

return 0;

}

**Output**

5

When the output is 5, you always know that bold and underline is used.

Also, you can add flags according to your requirements.

if (myDesign & ITALICS) {

// code for italics

}

Here, we have added italics to our design. Note, only code for italics is written inside the if statement.

You can accomplish almost anything in C programming without using enumerations. However, they can be pretty handy in certain situations.

**CHAPTER TEN**

**UNDERSTAND FILE INPUT/OUTPUT (I/O) OPERATIONS**

**10.1 C FILE OPERATIONS**

Cfiles I/O functions handle data on a secondary storage device, such as a hard disk. C can handle files as Stream-oriented data (Text) files, and System oriented data (Binary) files.

F**ile Handling** is the storing of data in a file using a program. In C programming language, the programs store results, and other data of the program to a file usingfile handlingin C. Also, we can extract/fetch data from a file to work with it in the program.

Five major operations can be performed on file are:

* Creation of a new file.
* Opening an existing file.
* Reading data from a file.
* Writing data in a file.
* Moving data to a specific location on the file
* Closing a file.

**10.2 STEPS FOR PROCESSING A FILE**

* Declare a file pointer variable.
* Open a file using fopen() function.
* Process the file using the suitable function.
* Close the file using fclose() function.

To handle files in C, file **input/output** functions available in the **stdio** library are:

**10.3 DIFFERENT FILE INPUT/OUTPUT AND THEIR FUNCTIONS**

|  |  |
| --- | --- |
| **Function** | **Uses/Purpose** |
| fopen | Opens a file. |
| fclose | Closes a file. |
| getc | Reads a character from a file |
| putc | Writes a character to a file |
| getw | Read integer |
| putw | Write an integer |
| fprintf | Prints formatted output to a file |
| fscanf | Reads formatted input from a file |
| fgets | Read string of characters from a file |
| fputs | Write string of characters to file |
| [feof](https://www.w3schools.in/c-tutorial/file-handling/feof/) | Detects end-of-file marker in a file |

**C fopen**: a C library function used to open an existing file or create a new file. The basic format of fopen is:

Syntax:

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

**Parameters**

* **filePath**: The first argument is a pointer to a string containing the name of the file to be opened.
* **mode**: The second argument is an access mode.

**C fopen() access mode can be one of the following values:**

|  |  |
| --- | --- |
| Mode | Description |
| r | Opens an existing text file. |
| w | Opens a text file for writing if the file doesn't exist then a new file is created. |
| a | Opens a text file for appending (writing at the end of existing file) and create the file if it does not exist. |
| r+ | Opens a text file for reading and writing. |
| w+ | Open for reading and writing and create the file if it does not exist. If the file exists then make it blank. |
| a+ | Open for reading and appending and create the file if it does not exist. The reading will start from the beginning, but writing can only be appended. |

**Return Value**

C fopen function returns **NULL** in case of a failure and returns a **FILE stream pointer** on success.

**Example:**

#include<stdio.h>

int main()

{

FILE \*fp;

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

return 0;

}

* The above example will create a file called fileName.txt.
* The **w** means that the file is being opened for writing, and if the file does not exist then the new file will be created.

**fclose()** function is C library function and it's used to releases the memory stream, opened by fopen() function. The basic format of fclose is:

Syntax:

int fclose( FILE \* stream );

**Return Value**

C fclose returns EOF in case of failure and returns 0 on success.

Example:

#include<stdio.h>

int main()

{

FILE \*fp;

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

fprintf(fp, "%s", "Sample Texts");

fclose(fp);

return 0;

}

* The above example will create a file called fileName.txt.
* The **w** means that the file is being opened for writing, and if the file does not exist then the new file will be created.
* The **fprintf** function writes Sample Texts text to the file.
* The **fclose** function closes the file and releases the memory stream.

**getc()** function is C library function, and it's used to read a character from a file that has been opened in read mode by **fopen()** function.

Syntax:

int getc( FILE \* stream );

**Return Value**

* **getc()** function returns next requested object from the stream on success.
* Character values are returned as an unsigned char cast to an int or EOF on end of file or error.
* The function **feof()** and **ferror()** to distinguish between end-of-file and error must be used.

**Example:**

#include<stdio.h>

int main()

{

FILE \*fp = fopen("fileName.txt", "r");

int ch = getc(fp);

while (ch != EOF)

{

/\* To display the contents of the file on the screen \*/ putchar(ch);

ch = getc(fp);

}

if (feof(fp))

printf("\n Reached the end of file.");

else

printf("\n Something gone wrong.");

fclose(fp);

getchar();

return 0;

}

**putc()** function is C library function, and it's used to write a character to the file. This function is used for writing a single character in a stream along with that it moves forward the indicator's position.

**Syntax:**

int putc( int c, FILE \* stream );

Example:

int main (void)

{

FILE \* fileName;

char ch;

fileName = fopen("anything.txt","wt");

for (ch = 'D' ; ch <= 'S' ; ch++) {

putc (ch , fileName);

}

fclose (fileName);

return 0;

}

**C fscanf function** reads formatted input from a file. This function is implemented in file related programs for reading formatted data from any file that is specified in the program.

**Syntax:**

int fscanf(FILE \*stream, const char \*format, ...)

Its return the number of variables that are assigned values, or EOF if no assignments could be made.

**Example:**

int main()

{

char str1[10], str2[10];

int yr;

FILE\* fileName;

fileName = fopen("anything.txt", "w+");

fputs("Welcome to", fileName);

rewind(fileName);

fscanf(fileName, "%s %s %d", str1, str2, &yr);

printf("----------------------------------------------- \n");

printf("1st word %s \t", str1);

printf("2nd word %s \t", str2);

printf("Year-Name %d \t", yr);

fclose(fileName);

return (0);

}

**CHAPTER ELEVEN**

**UNDERSTAND PREPROCESSORS AND HEADER FILES**

**11.1 WHAT IS PREPROCSSOR?**

The preprocessor is a program invoked by the compiler that modifies the source code before the actual composition takes place. To use any preprocessor directives, first, we have to prefix them with pound symbol

**11.2 LISTS ALL PREPROCESSOR DIRECTIVES.**

|  |  |  |
| --- | --- | --- |
| **Category** | **Directive** | **Description** |
| Macro substitution division | #include | File include |
| #define #undif | Macro define, Macro undefine |
| #ifdef #ifndef | If macro defined, If macro not defined |
| File inclusion division | #if #elif #else #endif | If, Else, ifElse, End if |
| Compiler control division | #line #error #pragma | Set line number, Abort compilation, Set compiler option |

**C Preprocessors Examples**

**Syntax:**

#include <stdio.h>

/\* #define macro\_name character\_sequence \*/

#define LIMIT 10

int main()

{

int counter;

for(counter =1; counter <=LIMIT; counter++)

{

        printf("%d\n",counter);

}

return 0;

}

In the above example for loop will run ten times.

#include <stdio.h>

#include "header.h"

#include <stdio.h> tell the compiler to add stdio.h file from System Libraries to the current source file, and #include "header.h" tells the compiler to get header.h from the local directory.

#undef  LIMIT

#define  LIMIT 20

This tells the compiler to undefine existing LIMIT and set it as 20.

#ifndef LIMIT

#define LIMIT 50

#endif

This tells the compiler to define LIMIT, only if LIMIT isn't already defined.

#ifdef LIMIT

/\* Your statements here \*/#endif

This tells the compiler to do the process the statements enclosed if LIMIT is defined.

**11.3 WHAT ARE THE HEADER FILES IN C?**

C language is famous for its different libraries and the predefined functions pre-written within it. These make programmer's effort a lot easier. Header files are helping file of your C program which holds the definitions of various functions and their associated variables that needs to be imported into your C program with the help of pre-processor #include statement. All the header file have a '.h' an extension that contains C function declaration and macro definitions. In other words, the header files can be requested using the preprocessor directive #include. The default header file that comes with the C compiler is the stdio.h.

Including a header file means that using the content of header file in your source program. A straightforward practice while programming in C or C++ programs is that you can keep every macro, global variables, constants, and other function prototypes in the header files. The basic syntax of using these header files is:

**Syntax:**

#include <file>

or

#include "file"

This kind of file inclusion is implemented for including system oriented header files. This technique (with angular braces) searches for your file-name in the standard list of system directories or within the compiler's directory of header files. Whereas, the second kind of header file is used for user-defined header files or other external files for your program. This technique is used to search for the file(s) within the directory that contains the current file.

**11.4 HOW INCLUDE WORKS**

The C's #include preprocessor directive statement exertions by going through the C preprocessors for scanning any specific file like that of input before abiding by the rest of your existing source file.  Let us take an example where you may think of having a header file **karl.h** having the following statement:

**Example:**

char \*example (void);

then, you have a main C source program which seems something like this:

**Example:**

#include<stdio.h>

int x;

#include "karl.h"

int main ()

{

printf("Program done");

return 0;

}

So, the compiler will see the entire C program and token stream as:

Example:

#include<stdio.h>

int x;

char \* example (void);

int main ()

{

printf("Program done");

return 0;

}

**11.5 WRITING OF SINGLE AND MULTIPLE USES OF HEADER FILES**

You can use various header files based on some conditions. In case, when a header file needs to be included twice within your program, your compiler will be going to process the contents inside it - twice which will eventually lead to an error in your program. So to eliminate this, you have to use conditional preprocessor directives. Here's the syntax:

**Syntax:**

#ifndef HEADER\_FILE\_NAME

#define HEADER\_FILE\_NAME

the entire header file

#endif

Again, sometimes it's essential for selecting several diverse header files based on some requirement to be incorporated into your program. For this also multiple conditional preprocessors can be used like this:

**Syntax:**

#if FIRST\_SYSTEM

#include "sys.h"

#elif SEC\_SYSTEM

#include "sys2.h"

#elif THRID\_SYSTEM

....

#endif

You can create your **custom header files in C**; it helps you to manage user-defined methods, global variables, and structures in a separate file, which can be used in different modules.

**A process to Create Custom Header File in C**

For example, I am calling an external function named **swap** in my **main.c** file.

**Example:**

#include<stdio.h>

#include"swap.h"

void main()

{

int a=20;

int b=30;

swap (&a,&b);

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

printf ("b=%d\n",b);

}

Swap method is defined in **swap.h** file, which is used to swap two numbers by using a temporary variable.

Example:

void swap (int\* a, int\* b)

{

int tmp;

tmp = \*a;

\*a = \*b;

\*b = tmp;

}

**Note:**

* header file name must have a **.h** file extension.
* In this example, I have named **swap.h** header file.
* Instead of writing <swap.h> use this terminology **swap.h** for include custom header file.
* Both files **swap.h** and **main.c** must be in the same folder.

**C CODES EXAMPLES**

**1. C program to read name and marks of n number of students and store them in a file.**

#include <stdio.h>

int main()

{

char name[50];

int marks, i, num;

printf("Enter number of students: ");

scanf("%d", &num);

FILE \*fptr;

fptr = (fopen("C:\\student.txt", "w"));

if(fptr == NULL)

{

printf("Error!");

exit(1);

}

for(i = 0; i < num; ++i)

{

printf("For student%d\nEnter name: ", i+1);

scanf("%s", name);

printf("Enter marks: ");

scanf("%d", &marks);

fprintf(fptr,"\nName: %s \nMarks=%d \n", name, marks);

}

fclose(fptr);

return 0;

**2. C program to read name and marks of n number of students from keyboard and store them in a file. If the file previously exits, add the information to the file.**

#include <stdio.h>

int main()

{

char name[50];

int marks, i, num;

printf("Enter number of students: ");

scanf("%d", &num);

FILE \*fptr;

fptr = (fopen("C:\\student.txt", "a"));

if(fptr == NULL)

{

printf("Error!");

exit(1);

}

for(i = 0; i < num; ++i)

{

printf("For student%d\nEnter name: ", i+1);

scanf("%s", name);

printf("Enter marks: ");

scanf("%d", &marks);

fprintf(fptr,"\nName: %s \nMarks=%d \n", name, marks);

}

fclose(fptr);

return 0;

}

**3. C program to write all the members of an array of structures to a file using fwrite(). Read the array from the file and display on the screen.**

#include <stdio.h>

struct student

{

#include <stdio.h>

struct student

{

char name[50];

int height;

};

int main(){

struct student stud1[5], stud2[5];

FILE \*fptr;

int i;

fptr = fopen("file.txt","wb");

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

{

fflush(stdin);

printf("Enter name: ");

gets(stud1[i].name);

printf("Enter height: ");

scanf("%d", &stud1[i].height);

}

fwrite(stud1, sizeof(stud1), 1, fptr);

fclose(fptr);

fptr = fopen("file.txt", "rb");

fread(stud2, sizeof(stud2), 1, fptr);

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

{

printf(

**4.** **PROGRAM TO TAKE INPUT OF VARIOUS DATAT YPES IN C**

Below is a program to explain how to take input from user for different datatypes available in C language. The different datatypes are int(integer values), float(decimal values) and char(character values).

* printf() is used to display text onto the screen & is used to assign the input value to the variable and store it at that particular location.
* scanf() is use to take input from the user using format specifier %d and %i, both are used to take numbers as input from the user.
* %f is the format specifier to take float as input from the user
* %s is the format specifier to take character as input from the user

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int num1, num2;

float fraction;

char character;

printf("Enter two numbers number\n");

// Taking integer as input from user

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

printf("\n\nThe two numbers You have entered are %d and %i\n\n", num1, num2);

// Taking Character as input from the user

printf("\n\nEnter a Character\n");

scanf("%c",&character);

printf("\n\nThe character that you have entered is %c", character); printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

**Output:**



**5. HOW TO USE GETS () FUNCTION**

Some of the important points about scanf() and gets() are:

* scanf() and gets() both are used to take input from the user.
* scanf() can only take input until it encounters a space. The words after space are ignored by it.
* gets() is used to take a single input at a time but can be used to input a complete sentence with spaces unlike scanf().

**Below is a program on use of gets().**

gets() takes only a single line at a time i.e all the words before hitting \n(enter key).

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

char str[50]; // char array of size 50

printf("Enter your complete name:\n\n\n");

gets(str);

printf("\n\nWelcome to Studytonight %s\n\n\n", str);

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

**OUTPUT**



**6. Program to find average of N Numbers**

Below is a program to calculate average of n numbers.

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n, i;

float sum = 0, x;

printf("Enter number of elements: ");

scanf("%d", &n);

printf("\n\n\nEnter %d elements\n\n", n);

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

{

scanf("%f", &x);

sum += x;

}

printf("\n\n\nAverage of the entered numbers is = %f", (sum/n));

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**OUTPUT**



**7. Checking for Odd and Even Numbers using Bitwise Operator**

**Below is a program to find whether a number is even or odd using bitwise operator.**

x&1 returns true if the LSB(Least significant Bit) of binary representation of an integer x is 1. It returns false if the LSB or the Right most bit in a binary sequence is 0.

In binary representation of an integer, if LSB is 1 then it is odd and if LSB is 0 then it is even.

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int x;

for(x = 0; x <= 10; x++)

{

if(x&1) // if number is odd

printf("\t\t\t%d is odd\n",x);

else if(!(x&1)) // ! is used inside if to reverse the boolean value

printf("\t\t\t%d is even\n",x);

}

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



We have used a simple for loop to input numbers and show how to use the Bitwise operator. You can take input from user using scanf() and use th same logic to find if the input number is odd or even.

**8. Checking if inout number is Odd or Even without using %(Mod) Operator**

**Below is a program to find whether a number is even or odd without using %(Mod) operator.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n;

printf("Enter a number: ");

scanf("%d",&n);

if((n/2)\*2 == n)

printf("\n\n\t\t %d is Even\n", n);

else

printf("\n\n\t\t %d is Odd\n", n);

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**9. Program to find Factors of a Number. Below is a program to find factors of a number.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int num, i;

printf("Enter the number to find the factors of : ");

scanf("%d",&num);

printf("\n\n\nFactors of %d are \n\n", num);

for(i = 1; i <= num/2; i++)

{

if(num%i == 0)

printf("\t\t\t%d\n", i);

}

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**10. Program to find Sum of N input Numbers in C - Below is a program on sum of n numbers.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n,sum=0,c,value;

printf("\n\nEnter the number of integers you want to add: ");

scanf("%d", &n);

printf("Enter %d integers\n\n",n);

for(c = 1; c <= n; c++)

{

scanf("%d", &value);

/\*

need to initialise sum before using otherwise

garbage value will get printed

\*/

sum += value;

}

printf("\n\n\nsum of entered numbers = %d", sum);

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

**Output:**



**11. Program to find first N Prime Numbers - Below is a program to find first n prime numbers using nested for loops, where the value of n is input by the user.**

Before you continue with program, check this topics to understand the program:

* Loops in C
* For Loop Programs in C
* Nested For Loop Programs in C

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n,i = 3, count, c;

printf("\nEnter the number of prime numbers required : ");

scanf("%d", &n);

if(n >= 1)

{

printf("\n\nFirst %d prime numbers are : ", n);

printf("2 ");

}

// iteration for n prime numbers

// i is the number to be checked in each iteration starting from 3

for(count = 2; count <= n; i++)

{

// iteration to check c is prime or not

for(c = 2; c < i; c++)

{

if(i%c == 0)

break;

}

if(c == i) // c is prime

{

printf("%d ", i);

count++; // increment the count of prime numbers

}

}

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

**Output**



**12. Program to find the Largest number among n input Numbers - Below is a program to find largest number among n user input numbers.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n,i;

float c,big;

printf("\n\nEnter the number of elements you wish to find the greatest element of: ");

scanf("%d", &n);

printf("\n\nEnter %d numbers :\n", n);

printf("\n\n\t\t\tElement 1: ");

//Important step- always initialize big to the first element

scanf("%f", &big);

for(i = 2; i <= n; i++)

{

printf("\n\t\t\tElement %d : ", i);

scanf("%f", &c);

/\*

if input number is larger than the

current largest number

\*/

if(big < c)

big = c; // update big to the larger value

}

printf("\n\n\nThe largest of the %d numbers is %f ", n, big);

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**

**13. Program to find exponential without using pow() method - Below is a program to find exponential without using pow() method.**

long long int is of double the size of long int.

%lld is the format specifier for long long int.

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n, exp, exp1;

long long int value = 1;

printf("Enter the number and its exponential:\n\n");

scanf("%d%d",&n, &exp);

exp1 = exp; // storing original value for future use

// same as while((--exp)!=-1)

while(exp-- > 0)

{

value \*= n; // multiply n to itself exp times

}

printf("\n\n %d^%d = %lld\n\n", n, exp1, value);

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**14. Program to check if input Number is int or float - is a program to check whether the user input number is of integer or float datatype.**

strlen() does not count the null character '\0'.

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

char number[10];

int flag = 0;

int length, i = 0;

printf("\n\nEnter a number: ");

scanf("%s", number);

length = strlen(number);

// till string does not end

while(number[i++] != '\0') // same as while(length-->0)

{

if(number[i] == '.') // decimal point is present

{

flag = 1;

break;

}

}

// if(0) is same as if(false)

if(flag)

printf("\n\n\n\tEntered Number is a Floating point Number\n\n");

else

printf("\n\n\n\tEntered Number is a integer Number\n\n");

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**15. Program to print the Multiplication Table of any Number - Below is a program to print the multiplication table of any user input number.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n,i;

printf("Enter an integer you need to print the table of: ");

scanf("%d", &n);

printf("\n\n\n");

for(i = 1; i <= 10; i++)

{

printf("\n\t\t\t%d \* %d = %d \n", n, i, n\*i);

}

printf("\n\n\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**16. Program to print the reverse of an Array - Below is a simple program to reverse an array**.

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int c, d, n, a[100], b[100];

printf("\n\nEnter number of elements in array :");

scanf("%d", &n);

printf("\n\nEnter %d elements\n", n);

for(c = 0; c < n; c++)

scanf("%d", &a[c]);

/\*

temporarily storing elements into array b

starting from end of array a

\*/

for(c = n-1, d = 0; c >= 0; c--, d++)

b[d] = a[c];

/\*

copying reversed array into original.

Here we are modifying original array to reverse it.

\*/

for(c = 0; c < n; c++)

a[c] = b[c];

printf("\n\n Resultant array is: ");

for(c = 0; c < n; c++)

printf("%d", a[c]);

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**17. Program to insert an element in an Array - Below is a simple program to insert an element in an array.**

Here is the C language tutorial explaining Arrays → Arrays in C

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int array[100], position, c, n, value;

printf("\n\nEnter number of elements in array:");

scanf("%d", &n);

printf("\n\nEnter %d elements\n", n);

for(c = 0; c < n; c++)

scanf("%d", &array[c]);

printf("\n\nEnter the location where you want to insert new element: ");

scanf("%d", &position);

printf("\n\nEnter the value to insert: ");

scanf("%d", &value);

// shifting the elements from (position to n) to right

for(c = n-1; c >= position-1; c--)

array[c+1] = array[c];

array[position - 1] = value; // inserting the given value

printf("\n\nResultant array is: ");

/\*

the array size gets increased by 1

after insertion of the element

\*/

for(c = 0; c <= n; c++)

printf("%d ", array[c]);

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

In the above program we take an array as user input and then ask the user for a new number that they wish to add to the original array, and the position where they want to add the new number.

The we shift the existing numbers from the index position to the end of the array one position to the right, therby vacating a space for the new element. And then we add the new number at the user specified position index.

Output:



**18. Program to Delete an Element from Array in C - Below is a simple program to delete an element from array, where the position of element to be deleted is given by user.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int array[100], position, c, n;

printf("\n\nEnter number of elements in array:");

scanf("%d", &n);

printf("\n\nEnter %d elements\n", n);

for(c = 0; c < n; c++)

scanf("%d", &array[c]);

printf("\n\nEnter the location where you want to delete element from: ");

scanf("%d", &position);

if(position >= n+1)

printf("\n\nDeletion not possible\n\n");

else

// updating the locations with next elements

for(c = position-1; c < n-1; c++)

array[c] = array[c+1];

printf("\n\nResultant array is: ");

/\*

the array size gets reduced by 1

after deletion of the element

\*/

for(c = 0; c < n-1; c++)

printf("%d ", array[c]);

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**19. Program to Delete an element from array based on value - Below is a simple program to delete an element from array, where the element to be deleted is given by user:**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int array[10], element, c, n, pos;

/\*

initialization as garbage value is

stored by default in c variables

\*/

int found = 0;

printf("\n\nEnter number of elements in array:");

scanf("%d", &n);

printf("\n\nEnter %d elements\n", n);

for(c = 0; c < n; c++)

scanf("%d", &array[c]);

printf("\n\nThe input array is: ");

for(c = 0; c < n; c++)

printf("%d", array[c]);

printf("\n\nEnter the element to be deleted: ");

scanf("%d", &element);

// check the element to be deleted is in array or not

for(c = 0; c < n; c++)

{

if(array[c] == element)

{

found = 1;

pos = c;

break; // terminate the loop

}

}

if(found == 1) // the element to be deleted exists in the array

{

for(c = pos; c < n-1; c++)

array[c] = array[c+1];

}

else

printf("\n\nElement %d is not found in the array\n\n", element);

printf("\n\nResultant array is: ");

/\*

the array size gets reduced by 1

after deletion of the element

\*/

for(c = 0; c < n-1; c++)

printf("%d ",array[c]);

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**20 Program to find Largest and Smallest Element in an Array - Below is a program to find the largest and smallest elements in array.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int a[50], size, i, big, small;

printf("\nEnter the size of the array: ");

scanf("%d", &size);

printf("\n\nEnter the %d elements of the array: \n\n", size);

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

scanf("%d", &a[i]);

big = a[0]; // initializing

/\*

from 2nd element to the last element

find the bigger element than big and

update the value of big

\*/

for(i = 1; i < size; i++)

{

if(big < a[i]) // if larger value is encountered

{

big = a[i]; // update the value of big

}

}

printf("\n\nThe largest element is: %d", big);

small = a[0]; // initializing

/\*

from 2nd element to the last element

find the smaller element than small and

update the value of small

\*/

for(i = 1; i < size; i++)

{

if(small>a[i]) // if smaller value is encountered

{

small = a[i]; // update the value of small

}

}

printf("\n\nThe smallest element is: %d", small);

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**21. Program to find Sum of N input Numbers using Array - Below is a program to find and print the sum of n numbers using arrays.**

#include<stdio.h>

int main()

{

printf("\n\n\t\tStudytonight - Best place to learn\n\n\n");

int n, sum = 0, c, array[100];

printf("Enter the number of integers you want to add: ");

scanf("%d", &n);

printf("\n\nEnter %d integers \n\n", n);

for(c = 0; c < n; c++)

{

scanf("%d", &array[c]);

sum += array[c]; // same as sum = sum + array[c]

}

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

printf("\n\n\t\t\tCoding is Fun !\n\n\n");

return 0;

}

Copy

**Output:**



**22. Simple Program to Sort Array elements - Below is a program to sort array elements in an array.**

#include<stdio.h>

#include<conio.h>

void sorting(int \*x, int y);

void main()

{

int a[20], i, c, n;

clrscr();

printf("Enter number of elements you want to sort: ");

scanf("%d", &n);

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

scanf("%d", &a[i]);

sorting(a, n);

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

printf("%d\t", a[i]);

getch();

}

void sorting(int \*x, int y)

{

int i, j, temp;

for(i = 1; i <= y-1; i++)

{

for(j = 0; j < y-i; j++)

{

if(\*(x+j) > \*(x+j+1))

{

temp = \*(x+j);

\*(x+j) = \*(x+j+1);

\*(x+j+1) = temp;

}

}

}

}

Copy

Enter number of elements you want to sort: 6

5 3 4 2 1 6

1 2 3 5 6

**23. PROGRAM EXAMPLES**

**Hello World Program - C Language**

Below is a simple program printing Hello World in C language.

printf() is a system defined function under the header file **stdio.h**, used to print data onto the screen

\n is used to move the control onto the next line

\t is used to give a horizontal tab i.e. continuous five spaces

**Writes the words "Hello World" on the screen \*/**

#include<stdio.h>

int main()

{

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

getch(); //Use to get one character input from user, and it will not be printed on screen.

return 0;

}

**Program Output:**



The above example has been used to print Hello, World! Text on the screen.

**REFERENCES**

W3schools.In(2021). C Tutorial - Learn C Programming - W3schools. Available at

https://www.w3schools.in/c-tutorial on 11/07/2021

Onecompiler.com (2021). C - OneCompiler - Write, run and share C code online. Available at

<https://onecompiler.com/> on 11/07/2021

Educba.com (2020). Tokens in C | Complete Guide to Top 6 Types of Tokens in C.

https://www.educba.com/tokens-in-c on 11/07/2021

**ABOUT THE AUTHOR**

**OJO ABOSEDE IBIRONKE** is from Lagos State, Nigeria. She had her ND and HND in Computer Science from Kwara State Polytechnic, Ilorin, PGD in Computer Maintenance Engineering from University of Ife, Ile-Ife and Master in Computer Systems from University of Ibadan. She also obtained her Bachelor Degree in Computer Science from National Open University of Nigeria and another Master degree from Lagos State University .

She has written some Publications and Textbooks, She is a member of Nigeria Computer Society (NCS), Computer Professional of Nigeria (CPN) and International Association of Engineers (IAENG).

She is a Lecturer at Department of Computer Science, Ogun State Institute of Technology, Igbesa, Ogun State. She is happily married with children.

**ABOUT THE BOOK**

**INTRODUCTORY PROGRAMMING LANGUAGE USING C** is a programming language book for mostly people willing to study computer science or to have knowledge on how to write a computer program (code).

C Language is a powerful general-purpose programming language that can be used to develop software like operating systems, databases, compilers, and so on. It is an excellent language to learn to program for beginners. C is designed to provide low-level memory access. The C compiler is a software application that transforms the human-readable C program code to machine-readable code. This process of transforming the code from High-Level Language to Machine Level Language is called **Compilation**. C language and its compiler have become available on a wide range of platforms from embedded microcontrollers to supercomputers.

Do have a remarkable experience as you explore the book.