**Modeule 2 Introduction to Programming**

1. **Overview of C Programming**

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

C programming language is one of the most influential and widely-used programming languages in the history of computing. It was developed in the early 1970s and has had a profound impact on many areas of software development.

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

|  |  |
| --- | --- |
| **Application of C** | **Description and Examples** |
| Operating System Development | C is used to develop robust operating systems like UNIX and Linux, enabling efficient management of hardware and system resources. |
| Embedded Systems and IoT Devices | C powers embedded systems in devices such as automotive controllers and smart home gadgets, ensuring real-time performance and precise hardware control. |
| Compilers and Interpreters | Many compilers, including the GNU Compiler Collection (GCC), are written in C, facilitating the translation of high-level code into machine-executable instructions. |
| Database Systems | High-performance databases like MySQL and Oracle utilize C for efficient data management, query processing, and transaction handling. |
| Game Development and Graphics | C is used in developing game engines for titles like Doom and Quake, enabling high-performance graphics rendering and real-time processing. |
| Network Programming and Drivers | C is employed to implement network protocols and develop device drivers, ensuring efficient data transmission and seamless hardware interactions. |

1. **Setting Up Environment**

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

**Steps to Install a C Compiler & Set up an IDE**

**Steps to Install a C Compiler & Set up DevC++**

1. **Download DevC++**
   * Go to the official website or trusted sources (e.g., SourceForge) and download DevC**++** setup file.
   * Choose the version that includes the **TDM-GCC compiler** (this comes with GCC built-in).
2. **Install DevC++**
   * Run the setup file.
   * Follow the installation wizard, keep default options, and complete the installation.
3. **First-Time Setup**
   * Open DevC++.
   * If prompted, select **TDM-GCC** as the compiler.
   * Go to **Tools → Compiler Options** to confirm GCC is selected.
4. **Verify Compiler**
   * Click **File → New → Source File**, type a simple program, and compile it.
   * If it runs without errors, your DevC++ and GCC are ready.

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

**Write and Run First Program in DevC++**

1. **Open DevC++** and create a new file:
   * File → New → Source File
2. **Write Code**:

#include <stdio.h>

int main() {

printf("Hello, World!");

return 0;

}

1. **Save the File**

File → Save As → give it a name like hello.c.

**3. Basic Structure of a C Program**

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

**1. Header Files**

* Contain predefined functions and macros.
* Example:

#include <stdio.h> // for input/output functions

#include <math.h> // for math functions

**2. Main Function**

* Every C program must have a main() function — it’s the starting point of execution.
* Example:

int main() {

// program code

return 0; // indicates successful execution

}

**3. Comments**

* Used to explain code and improve readability.
* **Single-line**: // comment text
* **Multi-line**:

/\* This is

a multi-line comment \*/

**4. Data Types**

int 🡪 integers (e.x., 1,2,3,-5,-3)

float 🡪 Decimal Numbers (eg. 14.12,-0.5 )

char 🡪 single character (e.g. ‘P’, ‘R’, )

**Example Basic Structure**

#include <stdio.h>

Int main()

{

Printf(“Hello World”);

}

**LAB EXERCISE: o Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values. (Code name : Basic Structure of c program \_ 2)**

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

#include<stdio.h>

#define pi 3.14 // Constant Using marco

int main()

{

int a = 10; // Integer Variable

float b = 10.12; // Character Variable

char c = 'P'; // Float Variavble

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

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

printf("c = %c\n",c);

return 0;

}

**4. Operators in C**

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

**1. Arithmatic Operation**

🡪 Addition (+)

🡪 Subtraction (-)

🡪Multiplication (\*)

🡪Division (/)

🡪Modulo (%)

**Example:** a + b, a % b

**2. Relational Operators –** Compare two values and return 1 (true) or 0 (false):

🡪 == (Equal to)

🡪 != (Not equal to)

🡪 > (Greater than)

🡪 < (Less than)

🡪 >= (Greater than or equal to)

🡪 <= (Less than or equal to)

**Example**: a > b

**3. Logical Operators** – Used for logical conditions:

🡪 && (Logical AND) – true if both are true

🡪 || (Logical OR) – true if at least one is true

🡪 ! (Logical NOT) – reverses the result

**Example**: (a > b) && (b > 0)

4. **Assignment Operators** – Assign values to variables:

🡪 = (Assign)

🡪 += (Add and assign)

🡪 -= (Subtract and assign)

🡪 \*= (Multiply and assign)

🡪 /= (Divide and assign)

🡪 %= (Modulus and assign)

**Example:** a += 5 (same as a = a + 5)

**5. Increment/Decrement Operators** – Increase or decrease value by 1:

🡪 ++ (Increment) – ++a or a++

🡪 -- (Decrement) – --a or a—

**6. Bitwise Operators – Work on bits:**

🡪 & (Bitwise AND)

🡪 | (Bitwise OR)

🡪 ^ (Bitwise XOR)

🡪 ~ (Bitwise NOT)

🡪 << (Left shift)

🡪 >> (Right shift)

Example: a & b

**LAB EXERCISE: Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.**

**(Code Nmae : Operation in c)**

// Write a C program that accepts two integers from the user and performs arithmetic ,relational, and logical operations on them. Display the results.

**Code:**

#include <stdio.h>

int main()

{

int n1;

int n2;

printf("Enter thr Number 1: ");

scanf("%d",&n1);

printf("Enter the number 2: ");

scanf("%d",&n2);

printf("\n Arithmatic Operator\n");

printf("%d + %d = %d \n", n1, n2, n1 + n2);

printf("%d - %d = %d \n", n1, n2, n1 - n2);

printf("%d \* %d = %d \n", n1, n2, n1 \* n2);

printf("%d / %d = %d \n", n1, n2, n1 / n2);

printf("%d % %d = %d \n", n1, n2, n1 % n2);

printf("\n Relational Operator \n");

printf("%d == %d : %d\n",n1, n2, n1 == n2 );

printf("%d != %d : %d\n",n1, n2, n1 != n2 );

printf("%d > %d : %d\n",n1, n2, n1 > n2 );

printf("%d < %d : %d\n",n1, n2, n1 < n2 );

printf("LogicaL Operator\n");

printf("(%d > 0) && (%d > 0) : %d\n", n1, n2, (n1 > 0) && (n2 > 0));

printf("(%d > 0) || (%d > 0) : %d\n", n1, n2, (n1 > 0) || (n2 > 0));

printf("! (%d > 0) : %d\n", n1, !(n1 > 0));

return 0;

}

**5. Control Flow Statements in C**

**1. if Statement**

Executes a block of code only if the condition is true.

**Syntax:**

if (condition) {

// code if condition is true

}

**Example:**

if (age >= 18) {

printf("You are eligible to vote.");

}

**2. if-else Statement**

Executes one block if condition is true, otherwise executes another block.  
**Syntax:**

if (condition) {

// code if condition is true

} else {

// code if condition is false

}

**Example :**

if (num %2 == 0){

printf(“The number is Even”);

}

else{

printf(“The Number is Odd”);

}

**3. Nested if-else Statement :** An if or else contains another if-else.

**Syntax:**

if (condition1) {

if (condition2) {

// code if both true

} else {

// code if condition1 true but condition2 false

}

} else {

// code if condition1 is false

}

**Example :**

if (marks >= 50) {

if (marks >= 90) {

printf("Excellent");

} else {

printf("Pass");

}

} else {

printf("Fail");

}

**4. switch Statement**

**Selects one case to execute based on the value of an expression.**

**Syntax:**

switch (expression) {

case value1:

// code

break;

case value2:

// code

break;

default:

// code if no case matches

}

**Example:**

switch (expression) {

case value1:

// code

break;

case value2:

// code

break;

default:

// code if no case matches

}

**LAB EXERCISE**:

**Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the month name based on the user’s input (1 for January, 2 for February, etc.).**

**Code:**

//Explain decision-making statements in C (if, else, nested if-else, switch).

//Provide examples of each.

#include <stdio.h>

int main()

{

int num;

int month;

printf("Enter the Number: ");

scanf("%d",&num);

if (num % 2 == 0)

{

printf("The Number is Even\n");

}

else

{

printf("The number is odd\n");

}

printf("Enter the Month (1-12): ");

scanf("%d",&month);

switch (month)

{

case 1 : printf("January");

break;

case 2 : printf("Fabuary");

break;

case 3 : printf("March");

break;

case 4 : printf("April");

break;

case 5 : printf("May");

break;

case 6 : printf("June");

break;

case 7 : printf("July");

break;

case 8 : printf("August");

break;

case 9 : printf("September");

break;

case 10 : printf("Octomber");

break;

case 11 : printf("November");

break;

case 12 : printf("December");

break;

default: printf("Please Enter the Valid Input From (1 to 12)");

}

return 0;

}

**6. Looping in C**

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

1. While Loop:

* It checks the condition before running the loop.
* If the condition is false at the start, the loop does not run even once.
* Use it when you don’t know how many times you need to repeat something, and you want to keep going only while a condition is true.
* Example: Keep asking for a password until it is correct.

2. For Loop:

* It also checks the condition before running the loop.
* It has a starting point, a condition, and a step (like counting up or down).
* Use it when you know exactly how many times you want to repeat something.
* Example: Print numbers from 1 to 10.

3. Do-While Loop:

* It runs the loop once first, then checks the condition.
* This means the loop will always run at least one time.
* Use it when you want to run some code first and then decide if it should run again.
* Example: Show a menu, let the user pick something, then ask if they want to do it again.

**LAB EXERCISE: Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).**

**1. for loop**

#include <stdio.h>

int main()

{ int n;

printf("Enter the Number ");

scanf("%d",&n);

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

{

printf("Using For Loop Print 1 to %d numbers = %d\n",n,i);

}

}

**2. while loop**

**Code:**

#include <stdio.h>

int main()

{

int n;

int i;

printf("Enter the Number");

scanf("%d",&n);

i = 1;

while(i<=n)

{

printf("Using While Loop Print The 1 to %d = %d \n" ,i, n);

i++;

}

}

**3. do while**

**Code:**

#include<stdio.h>

int main()

{

int num;

num = 1;

do

{

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

num++;

}

while(num<=10);

printf("\n");

}

**7. Loop Control Statements**

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

**1. break Statement :** Immediately terminates the loop and transfers control to the first statement after the loop.

**Example:**

#include <stdio.h>

int main() {

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

if (i == 5) {

break; // loop stops when i = 5

}

printf("%d ", i);

}

return 0;

}

**2. continue Statement :** Skips the rest of the code in the current loop iteration and moves to the next iteration.

**Example:**

#include <stdio.h>

int main() {

for (int i = 1; i <= 5; i++) {

if (i == 3) {

continue; // skip printing 3

}

printf("%d ", i);

}

return 0;

}

**3.** **goto Statement :**  Transfers control to a labeled statement in the same function.

**Example :**

#include<stdio.h>

Int main()

{

int i;

i = 1;

start:

if(i<=10)

{

printf(“%d”,i);

i++;

goto start;

}

Return 0;

}

**LAB EXERCISE: Write a C program that uses the break statement to stop printing numbers when it reaches 5. Modify the program to skip printing the number 3 using the continue statement.**

**Code:**

**// break Statement**

#include<stdio.h>

int main()

{

int n;

int i;

printf("Enter the number ");

scanf("%d",&n);

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

{

if(i==6)

{

break;

}

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

}

}

// OutPut : Stops the number 6 // 1 2 3 4 5

**// continue statement ..**

#include<stdio.h>

int main()

{

int num;

int i;

printf("Enter the number ");

scanf("%d",&num);

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

{

if(i==10)

{

continue;

}

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

}

}

**8. Functions in C**

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

**What are Functions in C?**

A function in C is a block of code that performs a specific task. Functions help in code reusability, modularity, and better readability.

**Parts of a Function**

**1.** **Function Declaration**

Tells the compiler the function name, return type, and parameters before it is used.

**Example:**

int add(int a, int b);

**2.Function Definition**

Contains the actual body of the function.

**Syntax:**

return\_type function\_name(parameter\_list) {

// code

}

**Example:**

int add(int a, int b) {

return a + b;

}

**3. Function Call**

**Syn**

**Example**

**LAB EXERCISE: Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.**

**Code:**

#include<stdio.h>

int fact(int n);

int main()

{

int n;

printf("Enter the number");

scanf("%d",&n);

printf("%d",fact(n));

return 0;

}

int fact(int n)

{

if (n == 0)

return 1;

else

{

int minus\_num = fact(n-1);

int real\_num = n \* minus\_num;

return real\_num;

} }

**9. Arrays in C**

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

**Concept of Arrays in C**

* An array is a collection of elements of the same data type stored in contiguous memory locations.
* Each element is accessed using an index, starting from 0.
* Arrays help store multiple values in a single variable instead of declaring separate variables.

**Difference Between the one dimensional and two dimensional array**

|  |  |  |
| --- | --- | --- |
| **Feature** | **One-Dimensional Array (1D)** | **Two-Dimensional Array (2D)** |
| **Definition** | A linear list of elements | An array of arrays (matrix-like structure) |
| **Structure** | Single row or column | Rows and columns (table-like structure) |
| **Indexing** | Uses a single index (e.g., arr[i]) | Uses two indices (e.g., arr[i][j]) |
| **Example** | [10, 20, 30, 40] | [[10, 20], [30, 40]] |
| **Memory Layout** | Continuous memory block for elements | Group of continuous memory blocks |
| **Use Case** | Storing simple lists or vectors | Storing matrices, tables, or grids |
| **Access Complexity** | Simpler access with one index | Slightly more complex access with two indices |
| **Visualization** | Line or list | Grid or matrix |
| **Declaration (in C)** | int arr[5]; | int arr[3][4]; |
| **Typical Application** | Storing scores, names, etc. | Storing game boards, spreadsheets, etc. |

**LAB EXERCISE: Write a C program that stores 5 integers in a one-dimensional array and prints them. Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.**

**Code:**

#include<stdio.h>

int main()

{

int i;

int j;

int sum = 0;

int arr[3][3];

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

printf("please enter the matrix [%d][%d]: ",i,j);

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

sum = sum + arr[i][j];

}

}

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

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

}

printf("\n");

}

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

printf("Sum of the total number is = %d",sum);

}

**10. Pointers in C**

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

**What is Pointer?**

🡪 A pointer is a variable that stores the memory address of another variable.

🡪 Instead of storing a direct value, a pointer holds the location where the value is stored in memory.

**Pointer Declaration & Initialization**

**1. Declaration**

data\_type \*pointer\_name;

**Example**

int \*ptr; // pointer to an integer

float \*fptr; // pointer to a float

**2. Initialization**

int num = 10;

int \*ptr = &num; // store address of num in ptr

**3. Accessing Values with Pointers**

& → Address-of operator (gets memory address)

\* → Dereference operator (access value stored at the address)

**Example:**

printf("%d", \*ptr); // prints value of num

**LAB EXERCISE: Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.**

**Code:**

#include<stdio.h>

int main()

{

int num = 50;

int \*ptr = &num;

// Before any modification in pointer

printf("The value of num = %d\n",num);

printf("Address of num = %d\n",ptr);

printf("Value stored in pointer variable= %d \n",\*ptr);

printf("\n \n");

printf("After change the ptr address\n");

\*ptr = 40;

printf("After Modification in pointer value of num = %d\n",num);

printf("After modification in pointer value of address of num = %d\n",ptr);

printf("Ater the modification in pointeer variable = %d",\*ptr);

}

**11. Strings in C**

**THEORY EXERCISE: Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.**

**What is String?**

🡪A string in C is a sequence of characters terminated by a null character ('\0').

🡪Strings are stored as character arrays.

**Common String Handling Functions in C**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Example Use Case** |
| **strlen()** | Returns the length of a string (excluding '\0) | strlen("Hello") → 5 |
| **strcpy()** | Copies one string into another | Copying user input into a buffer |
| **strcat()** | Appends one string at the end of another | Joining first name and last name |
| **strcmp()** | Compares two strings (returns 0 if equal) | Password or input validation |
| **strchr()** | Returns a pointer to the first occurrence of a character in a string. | Searching if a character exists in a string |

**LAB EXERCISE: Write a C program that takes two strings from the user and concatenates them using strcat(). Display the concatenated string and its length using strlen().**

**Code:**

#include <stdio.h>

#include <string.h> // Before Use Any String Library Function Used This Library;

int main() {

**char str1[100], str2[100];**

**// Input two strings**

printf("Enter first string: ");

gets(str1); // Note: unsafe, better to use fgets() in real projects

printf("Enter second string: ");

gets(str2);

**// Concatenate str2 to str1**

strcat(str1, str2);

**// Display result**

printf("\nConcatenated String: %s", str1);

printf("\nLength of Concatenated String: %d\n", strlen(str1));

// Length of str2

printf("Length of the Concated String: %d\n",strlen(str2));

return 0;

}

**12. Structures in C**

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

**What is String ?**

🡪 A structure in C is a user-defined data type that allows grouping variables of different data types under a single name.

🡪 It is useful to represent real-world entities like students, employees, books, etc.

**How to Declare a Structure**

struct student {

char name[20];

int roll\_no;

float marks;

};

**How to Initialize**

struct student CE[3];

struct student IT[3];

OR

struct student s1 = {“Prince” , 1 , 60.23};

**access structure members**

printf(“Student Name = %s”,s1.name);

printf(“Student Roll No = %d”,s1.roll\_no);

printf(“Student marks = %f”,s1.marks);

**LAB EXERCISE: Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3 students and print them.**

#include<stdio.h>

struct student{

int rollNo;

char name[20];

float marks;

};

int main()

{

struct student CE[3];

struct student IT[3];

int i;

for(i=0;i<3;i++)

{

printf("Enter the CE Student Info\n");

printf("Student Roll no: ");

scanf("%d",&CE[i].rollNo);

printf("Student Name: ");

scanf("%s",&CE[i].name);

printf("Student Marks: ");

scanf("%f",&CE[i].marks);

printf("\n");

}

printf("\n \n");

for(i=0;i<3;i++)

{

printf("CE The Roll No is %d\n",CE[i].rollNo );

printf("CE Student name is %s\n",CE[i].name );

printf("CE Student Marks %f\n",CE[i].marks );

printf("\n");

}

}

**13. File Handling in C**

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

**What is File Handling?**

File handling in C allows a program to store data permanently on storage devices like hard disks by using files.

It helps in:

* Saving data for future use
* Reading large input from files
* Generating reports/logs

**Basic File Operations**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Function Used** | **Description** |
| Open a file | fopen() | Opens a file in a given mode |
| Write to file | fprintf() / fputs() / fwrite() | Writes data |
| Read from file | fscanf() / fgets() / fread() | Reads data |
| Close file | fclose() | Closes the file |

**Common File Modes in fopen()**

|  |  |
| --- | --- |
| **Mode** | **Meaning** |
| **"w"** | Write only (creates new or overwrites) |
| **"r"** | Read only (file must exist) |
| **"a"** | Append (add data at end) |
| **"w+"** | Read + Write |
| **"r+"** | Read + Write (file must exist) |

**LAB EXERCISE: Write a C program to create a file, write a string into it, close the file, then open the file again to read and display its contents.**

#include <stdio.h>

int main() {

FILE \*fptr;

char str[100];

// Step 1: Create and write to file

fptr = fopen("myfile.txt", "w"); // open in write mode

if (fptr == NULL) {

printf("Error opening file!\n");

return 1;

}

printf("Enter a string to write to file: ");

gets(str);

fprintf(fptr, "%s", str); // write string to file

fclose(fptr); // close file

// Step 2: Read and display from file

fptr = fopen("myfile.txt", "r"); // open in read mode

if (fptr == NULL) {

printf("Error opening file!\n");

return 1;

}

printf("\nContents of the file:\n");

while (fgets(str, sizeof(str), fptr)) {

printf("%s", str); // read and print content

}

fclose(fptr); // close file

return 0;

}