Module 2 – Introduction to programming:

1. Over of C Programming:

1.1 Write an essay covering the history and evolution of c programming

Explain its importance and why it is still used today.

Ans. C programming is a general-purpose, procedure- oriented programming language. It is both machine-independent and structured. C is high level programming language developed by Dennis Ritchie in the early 1970s. now it is a one of the most popular and influential programming language worldwide.

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

Ans. Embedded system for developing firmware and IoT device as it offers hardware-level control and efficient memory used.

Operating system is powering foundational software like UNIX and LINUX and providing the efficiency needed for kernel and driver development.

2. Setting Up Environment:

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

Ans. Download **MinGW** from [mingw-w64.org](https://www.mingw-w64.org/) or through a package manager like **MSYS2**.

Follow the installation wizard and ensure you select the GCC components.

Add the bin directory of MinGW to the system's PATH variable for terminal access.

Dev C++:

Download **Dev-C++** from a trusted source like [Bloodshed Dev-C++](https://bloodshed.net/) or its updated fork.

Install the software and ensure the integrated GCC compiler is selected.

Open Dev-C++, configure project settings, and start coding.

VS code:

Download **VS Code** from [code.visualstudio.com](https://code.visualstudio.com/).

Install the **C/C++ Extension** from the Extensions Marketplace.

Configure the environment:

Install GCC or MinGW.

Set up a tasks.json file for build automation.

Optionally, add launch.json for debugging.

2.2 Install a C compiler on your system and configure the IDE.

Write your first program to print "Hello, World!" and run it.

Ans. IDE install in my system.

#include<stdio.h>

Int main(){

Printf(“hello world”);

Return 0;

}

3. Basic Structure of a C Program.

3.1 explain the basic structure of C program, including headers, main function, comment, data type and variables. Provide example.

Ans. Header: #include<stdio.h>

Main function: int main()

Comments: // single line

/\* multiline line\*/

Data types: int, float, char.

int age = 25; // Integer variable

float height = 5.9; // Floating-point variable

char initial = 'J'; // Character variable

example:

#include <stdio.h> // Include standard I/O header

int main() {

// Declare and initialize variables

int age = 25; // Integer variable

float height = 5.9; // Floating-point variable

char initial = 'J'; // Character variable

// Print the values

printf("Age: %d\n", age); // %d is used for integers

printf("Height: %.1f\n", height); // %.1f is for one decimal float

printf("Initial: %c\n", initial); // %c is for characters

return 0; // Exit the program

}

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

Ans.

#include <stdio.h> // Standard input-output header

#define PI 3.14159 // Define a constant (value of π)

int main() {

// Variable declarations

int age = 25; // Integer variable to store age

float height = 5.8; // Float variable to store height in feet

char initial = 'A'; // Character variable to store the initial of a name

// Constant declaration

const int daysInWeek = 7; // Constant to represent days in a week

// Displaying values

printf("Age: %d\n", age); // Display integer value

printf("Height: %.1f feet\n", height); // Display float value with 1 decimal point

printf("Initial: %c\n", initial); // Display character

printf("Value of PI: %.5f\n", PI); // Display constant value of PI

printf("Days in a week: %d\n", daysInWeek); // Display constant integer

return 0; // Indicate successful program termination

}

4. Operators in C:

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

Ans. 1. Arithmetic Operators:

+ (addition)

- (subtraction)

\* (multiplication)

/ (division)

% (modulus).

2. Relational Operators:

== (equal),

!= (not equal),

> (greater than),

< (less than),

>= (greater or equal),

<= (less or equal).

3. Logical Operators:

&& (logical AND),

|| (logical OR),

! (logical NOT).

4. Assignment Operators:

= (assign),

+= (add and assign),

-= (subtract and assign),

\*= (multiply and assign),

/= (divide and assign).

5. Increment and Decrement Operators

++ (increment),

-- (decrement).

6. Bitwise Operators:

& (AND),

| (OR),

^ (XOR),

~ (NOT),

<< (left shift),

>> (right shift).

7. Conditional (Ternary) Operator:

condition ? expression1 : expression2

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

Ans.

#include <stdio.h>

int main() {

int num1, num2;

// Accept input from the user

printf("Enter the first integer: ");

scanf("%d", &num1);

printf("Enter the second integer: ");

scanf("%d", &num2);

// Arithmetic operations

printf("\nArithmetic Operations:\n");

printf("%d + %d = %d\n", num1, num2, num1 + num2);

printf("%d - %d = %d\n", num1, num2, num1 - num2);

printf("%d \* %d = %d\n", num1, num2, num1 \* num2);

printf("%d / %d = %d\n", num1, num2, num1 / num2);

printf("%d %% %d = %d\n", num1, num2, num1 % num2);

// Relational operations

printf("\nRelational Operations:\n");

printf("%d == %d : %d\n", num1, num2, num1 == num2);

printf("%d != %d : %d\n", num1, num2, num1 != num2);

printf("%d > %d : %d\n", num1, num2, num1 > num2);

printf("%d < %d : %d\n", num1, num2, num1 < num2);

printf("%d >= %d : %d\n", num1, num2, num1 >= num2);

printf("%d <= %d : %d\n", num1, num2, num1 <= num2);

// Logical operations

printf("\nLogical Operations:\n");

printf("(%d && %d) : %d\n", num1, num2, num1 && num2);

printf("(%d || %d) : %d\n", num1, num2, num1 || num2);

printf("!(%d) : %d\n", num1, !num1);

return 0;

}

Output:

Enter the first integer: 10

Enter the second integer: 5

Arithmetic Operations:

10 + 5 = 15

10 - 5 = 5

10 \* 5 = 50

10 / 5 = 2

10 % 5 = 0

Relational Operations:

10 == 5 : 0

10 != 5 : 1

10 > 5 : 1

10 < 5 : 0

10 >= 5 : 1

10 <= 5 : 0

Logical Operations:

(10 && 5) : 1

(10 || 5) : 1

!(10) : 0

5. Control Flow Statements in C:

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

If statement:

#include <stdio.h>

int main() {

int num = 10;

if (num > 0) {

printf("The number is positive.\n");

}

return 0;

}

If-else statement:

#include <stdio.h>

int main() {

int num = -5;

if (num > 0) {

printf("The number is positive.\n");

} else {

printf("The number is not positive.\n");

}

return 0;

}

Nested if-else statement:

#include <stdio.h>

int main() {

int num = 0;

if (num >= 0) {

if (num == 0) {

printf("The number is zero.\n");

} else {

printf("The number is positive.\n");

}

} else {

printf("The number is negative.\n");

}

return 0;

}

Switch case statement:

#include <stdio.h>

int main() {

int day = 3;

switch (day) {

case 1:

printf("Monday\n");

break;

case 2:

printf("Tuesday\n");

break;

case 3:

printf("Wednesday\n");

break;

case 4:

printf("Thursday\n");

break;

case 5:

printf("Friday\n");

break;

case 6:

printf("Saturday\n");

break;

case 7:

printf("Sunday\n");

break;

default:

printf("Invalid day\n");

}

return 0;

}

5.2 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.).

Number is even or odd:

#include<stdio.h>

int main(){

int num;

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

scanf("%d",&num);

if (num%2==0)

{

printf("number is even");

}

else{

printf("number is odd");

}

return 0;

}

Switch case :

#include<stdio.h>

int main(){

int month;

printf("enter a month:");

scanf("%d",&month);

switch (month){

case 1:

printf("January\n");

break;

case 2:

printf("February\n");

break;

case 3:

printf("March\n");

break;

case 4:

printf("April\n");

break;

case 5:

printf("May\n");

break;

case 6:

printf("June\n");

break;

case 7:

printf("July\n");

break;

case 8:

printf("August\n");

break;

case 9:

printf("September\n");

break;

case 10:

printf("October\n");

break;

case 11:

printf("November\n");

break;

case 12:

printf("December\n");

break;

default:

printf("Invalid\n");

}

return 0;

}

6. Looping in C:

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

Ans.

While loop:

1. Repeats while a condition is true.

2. Condition is evaluated **before** the first iteration.

3. May not execute if the condition is initially false.

For loop:

1. Repeats for a specified number of iterations or over a range.

2. Condition or range is generally defined at the beginning.

3. May not execute if the range or condition is invalid.

Do-while-loop:

1. Executes at least once and then repeats while a condition is true.

2. Condition is evaluated **after** the first iteration.

3. Always executes **at least once**, regardless of the condition.

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

While loop:

#include <stdio.h>

int main() {

int i;

printf("Using while loop:\n");

i = 1;

while (i <= 10) {

printf("%d ", i);

i++;

}

printf("\n");

return 0;

}

For loop:

#include <stdio.h>

int main() {

int i;

printf("Using for loop:\n");

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

printf("%d ", i);

}

printf("\n");

return 0;

}

Do-while-loop:

#include <stdio.h>

int main() {

int i;

printf("Using do-while loop:\n");

i = 1;

do {

printf("%d ", i);

i++;

} while (i <= 10);

printf("\n");

return 0;

}

7. Loop control statement:

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

Ans.

1. break statement:

The statement is used to terminate a loop or a switch-case statement prematurely. When break is executed, the control exits the loop or switch immediately.

#include <stdio.h>

int main() {

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

if (i == 5) {

break;

}

printf("%d ", i);

}

printf("\n");

return 0;

}

2. continue statement:

The statement skips the rest of the code in the current iteration of a loop and moves to the next iteration.

#include <stdio.h>

int main() {

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

if (i == 5) {

continue;

}

printf("%d ", i);

}

printf("\n");

return 0;

}

3. goto statement:

The statement transfers control to a labelled statement within the program. It is generally discouraged because it can make code difficult to read and debug.

#include <stdio.h>

int main() {

int i = 1;

while (i <= 10) {

if (i == 5) {

goto end;

}

printf("%d ", i);

i++;

}

end:

printf("\nExited loop using goto.\n");

return 0;

}

7.2 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.

Ans.

Break Statement:

#include <stdio.h>

int main() {

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

if (i == 5) {

break;

}

printf("%d ", i);

}

printf("\n");

return 0;

}

Continue Statement:

#include <stdio.h>

int main() {

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

if (i == 3) {

continue;

}

printf("%d ", i);

}

printf("\n");

return 0;

}

8. Functions in C

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

Ans.a function is a block of code that performs a specific task. Functions allow for code reusability, modularity, and organization. By breaking down your code into functions, it becomes easier to manage, debug, and test.

declaration: A function declaration informs the compiler about the function's name, return type, and parameters without providing the full body of the function. It is often placed before the main() function or in a header file.

definination: The function definition is where you provide the actual code for the function. It includes the return type, name, parameters, and the body of the function.

function call: A function call is used to invoke or call a function from another part of the program, usually from main() or another function. When calling a function, you pass the arguments in the function call.

example:

#include <stdio.h>

int add(int, int);

int main() {

int x = 3, y = 5;

int result = add(x, y);

printf("Sum: %d\n", result);

return 0;

}

int add(int a, int b) {

return a + b;

}

8.2 Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.

Ans. #include <stdio.h>

int factorial(int);

int main() {

int num, result;

printf("Enter a positive integer: ");

scanf("%d", &num);

if (num < 0) {

printf("Factorial is not defined for negative numbers.\n");

return 1;

}

result = factorial(num);

printf("Factorial of %d is %d\n", num, result);

return 0;

}

int factorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

9. Arrays in C

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

Ans. an array is a collection of elements of the same type, stored in contiguous memory locations. Arrays allow you to store multiple values in a single variable, making it easier to manage data. Instead of declaring multiple variables to store similar data, you can use arrays to group the data together.

Types of Arrays in C

Arrays in C can be of various types, but the most common ones are:

One-Dimensional Arrays (1D Arrays)

Multi-Dimensional Arrays (2D, 3D, etc.)

1. One-Dimensional Arrays (1D Arrays)

A one-dimensional array is essentially a list of elements of the same type. It is the simplest type of array where the elements are accessed by a single index.

type array\_name[array\_size];

#include <stdio.h>

int main() {

int arr[5] = {10, 20, 30, 40, 50}; // One-dimensional array

printf("Element at index 0: %d\n", arr[0]); // Output: 10

printf("Element at index 3: %d\n", arr[3]); // Output: 40

return 0;

}

2. Multi-Dimensional Arrays

A multi-dimensional array in C is an array of arrays. The most common type is a two-dimensional array (2D array), which can be thought of as a table or matrix (rows and columns). However, you can also have three-dimensional arrays (3D), and so on.

2D Arrays (Two-Dimensional Arrays)

A 2D array is like a grid or matrix where elements are accessed by two indices: row and column.

type array\_name[row\_size][column\_size];

#include <stdio.h>

int main() {

int arr[2][3] = {{1, 2, 3}, {4, 5, 6}}; // 2D array (2 rows, 3 columns)

printf("Element at [0][0]: %d\n", arr[0][0]); // Output: 1

printf("Element at [1][2]: %d\n", arr[1][2]); // Output: 6

return 0;

}

3D Arrays (Three-Dimensional Arrays)

A 3D array is an array of 2D arrays. It is often used to represent data in three dimensions, like a cube.

type array\_name[x\_size][y\_size][z\_size];

#include <stdio.h>

int main() {

int arr[2][2][2] = {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}; // 3D array

printf("Element at [1][0][1]: %d\n", arr[1][0][1]); // Output: 6

return 0;

}

9.2 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.

Ans.

#include <stdio.h>

int main() {

int arr[5] = {10, 20, 30, 40, 50};

printf("One-Dimensional Array (5 integers):\n");

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

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

}

printf("\n\n");

int matrix[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

printf("Two-Dimensional Array (3x3 Matrix):\n");

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

int sum = 0;

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

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

sum += matrix[i][j];

}

}

printf("\nSum of all elements in the 3x3 matrix: %d\n", sum);

return 0;

}

10. Pointers in C

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

Ans. a pointer is a variable that stores the memory address of another variable. Instead of holding the data directly, a pointer holds the location in memory where the data is stored. Pointers are a powerful feature of C and are extensively used to manage dynamic memory, handle arrays, pass large structures to functions efficiently, and create complex data structures like linked lists, trees, etc.

Declaration of Pointers

A pointer is declared by specifying the type of variable it points to, followed by an asterisk (\*), and then the pointer variable's name.

type \*pointer\_name;

Initialization of Pointers

Pointers are usually initialized by assigning them the memory address of another variable. This is done using the address-of operator (&).

pointer\_name = &variable\_name;

Why Pointers Are Important in C

Pointers are a fundamental and powerful feature in C, and they are important for several reasons:

Efficient Memory Management:

Pointers allow for dynamic memory allocation and deallocation, enabling the creation of data structures that can grow or shrink at runtime (e.g., linked lists, trees, etc.).

You can manage large arrays or structures efficiently by passing pointers to functions, avoiding unnecessary copying of large amounts of data.

Access to Hardware/Memory Addresses:

Pointers are essential for working with low-level operations, such as interacting directly with hardware or manipulating memory addresses. This is particularly useful in embedded systems programming.

Function Efficiency:

Pointers allow passing large structures and arrays to functions without copying the entire data. Instead, only the memory address (pointer) is passed, which is much more efficient in terms of memory and processing time.

Functions can modify variables passed by reference, which would not be possible using only variables (passed by value).

Creating Complex Data Structures:

Many complex data structures such as linked lists, stacks, queues, and trees are built using pointers, where each element stores the address of the next element in the list or tree.

Handling Arrays:

Arrays in C are essentially pointers. The name of an array is a pointer to its first element, and pointer arithmetic can be used to iterate through the array elements efficiently.

Memory Efficiency:

Pointers can help save memory by reusing memory locations rather than allocating new memory for every function call or operation.

10.2 Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.

Ans.

#include <stdio.h>

int main() {

int num = 10;

int \*ptr;

ptr = &num;

printf("Before modification:\n");

printf("Value of num: %d\n", num);

\*ptr = 20;

printf("After modification:\n");

printf("Value of num: %d\n", num);

return 0;

}

11. Strings in C

11.1 Explain string handling functions like strlen(), strcpy(), strcat(),

strcmp(), and strchr(). Provide examples of when these functions are

useful.

Ans.

1. strlen()

The strlen() function is used to find the length of a string (excluding the null terminator '\0').

size\_t strlen(const char \*str);

2. strcpy()

The strcpy() function copies the contents of one string to another.

char \*strcpy(char \*dest, const char \*src);

3. strcat()

The strcat() function concatenates (appends) one string to the end of another string.

char \*strcat(char \*dest, const char \*src);

4. strcmp()

The strcmp() function compares two strings lexicographically (character by character).

int strcmp(const char \*str1, const char \*str2);

5. strchr()

The strchr() function searches for the first occurrence of a specified character in a string.

char \*strchr(const char \*str, int c);

11.2 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().

Ans.

#include <stdio.h>

#include <string.h>

int main() {

char str1[100], str2[100];

printf("Enter the first string: ");

fgets(str1, sizeof(str1), stdin);

str1[strcspn(str1, "\n")] = '\0';

printf("Enter the second string: ");

fgets(str2, sizeof(str2), stdin);

str2[strcspn(str2, "\n")] = '\0';

strcat(str1, str2);

printf("Concatenated string: %s\n", str1);

printf("Length of the concatenated string: %zu\n", strlen(str1));

return 0;

}

12. Structures in C

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

Ans. structure in C is a user-defined data type that allows grouping different types of variables (called members or fields) under a single name. Each member in a structure can have a different data type (such as int, float, char, etc.). Structures are used to represent a collection of related data that belongs together, making them useful for creating complex data models.

Structure Declaration

A structure is declared using the struct keyword, followed by a name for the structure and the body containing the members.

struct structure\_name {

data\_type member1;

data\_type member2;

data\_type member3;

// more members

};

Structure Initialization

You can initialize a structure in two ways:

At the time of declaration (in one step).

Using individual assignments after declaring a structure variable.

struct Student student1 = {"kamlesh", 20, 85.5};

struct Student student1;

strcpy(student1.name, "kamlesh");

student1.age = 20;

student1.marks = 85.5;

Accessing Structure Members

To access the members of a structure, you use the dot operator (.) if you are dealing with a structure variable.

structure\_variable.member\_name

12.2 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

Ans.

#include <stdio.h>

#include <string.h>

struct Student {

char name[50];

int roll\_no;

float marks;

};

int main() {

struct Student students[3];

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

printf("Enter details for student %d:\n", i + 1);

printf("Name: ");

fgets(students[i].name, sizeof(students[i].name), stdin);

students[i].name[strcspn(students[i].name, "\n")] = '\0';

printf("Roll Number: ");

scanf("%d", &students[i].roll\_no);

printf("Marks: ");

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

getchar();

printf("\n");

}

printf("Student Details:\n");

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

printf("Student %d:\n", i + 1);

printf("Name: %s\n", students[i].name);

printf("Roll Number: %d\n", students[i].roll\_no);

printf("Marks: %.2f\n", students[i].marks);

printf("\n");

}

return 0;

}

13. File Handling in C

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

Ans. File handling in C is crucial because it allows a program to store and retrieve data from external storage (such as a hard drive or other storage devices) for long-term use. Without file handling, programs would only be able to work with data temporarily in memory, and any data processed during runtime would be lost once the program ends.

File Opening

To open a file in C, the function fopen() is used. This function requires two arguments:

File name (the path to the file).

Mode (the action you want to perform on the file).

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

Closing a File

After finishing all file operations, the file should be closed using the fclose() function to ensure that the file is properly saved and system resources are released.

int fclose(FILE \*file);