**THEORY EXERCISES**

1. **History and Evolution of C Programming**
   * **Summary**: C was developed by Dennis Ritchie in the 1970s at Bell Labs to write the UNIX operating system. It evolved from the B language, offering a mix of low-level operations and high-level abstractions. C became widely popular for system programming and continues to be relevant due to its efficiency and ability to interact directly with hardware. It has been standardized over time (ANSI C, C99, C11).
2. **Setting Up Environment**
   * **Summary**: Installing a C compiler (e.g., GCC) involves downloading and configuring it on your system. On Windows, you can use MinGW or MSYS2. On Linux, GCC can be installed via sudo apt install gcc. For macOS, you can install it using xcode-select --install. An IDE (like DevC++, VS Code, or CodeBlocks) can also be set up to write and compile C programs.
3. **Basic Structure of a C Program**
   * **Summary**: A C program consists of headers (e.g., #include <stdio.h>), the main function (int main()), data types (e.g., int, char, float), variables, constants, and comments. The main function contains the program logic, and comments are used to explain code. The program starts from the main() function.
4. **Operators in C**
   * **Summary**: C has different operators:
     + **Arithmetic**: +, -, \*, /, %
     + **Relational**: ==, !=, >, <, >=, <=
     + **Logical**: &&, ||, !
     + **Assignment**: =, +=, -=, \*=, /=
     + **Increment/Decrement**: ++, --
     + **Bitwise**: &, |, ^, ~, <<, >>
     + **Conditional**: ?: (Ternary operator)

5. **Control Flow Statements in C**

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

**Example**:

#include <stdio.h>

int main() {

int num, month;

// Check if number is even or odd

printf("Enter a number: ");

scanf("%d", &num);

if (num % 2 == 0) {

printf("Even\n");

} else {

printf("Odd\n");

}

// Use switch to display the month name

printf("Enter month number (1-12): ");

scanf("%d", &month);

switch(month) {

case 1: printf("January\n"); break;

case 2: printf("February\n"); break;

// other months

default: printf("Invalid month\n");

}

return 0;

}

**6. Looping in C**

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

**Example**:

#include <stdio.h>

int main() {

// Using while loop

int i = 1;

while (i <= 10) {

printf("%d ", i);

i++;

}

printf("\n");

// Using for loop

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

printf("%d ", i);

}

printf("\n");

// Using do-while loop

i = 1;

do {

printf("%d ", i);

i++;

} while (i <= 10);

return 0;

}

**7. Loop Control Statements in C**

* **Lab Task**: Write a C program that:
  + Uses the break statement to stop printing numbers when it reaches 5.
  + Modifies the program to **skip** printing the number 3 using the continue statement.

**Example**:

#include <stdio.h>

int main() {

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

if (i == 5) {

break; // Exit the loop when i is 5

}

printf("%d ", i);

}

printf("\n");

// Using continue to skip printing 3

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

if (i == 3) {

continue; // Skip the number 3

}

printf("%d ", i);

}

return 0;

}

**9. Arrays in C**

* **Concept of Arrays**:  
  An **array** in C is a collection of elements, all of the same data type, stored in contiguous memory locations. It allows you to store multiple values in a single variable, rather than declaring separate variables for each value.
* **One-Dimensional Arrays**: A **one-dimensional array** is a list of elements, accessed using a single index. It can be thought of as a row of values.

**Example of a One-Dimensional Array**:

int arr[5] = {1, 2, 3, 4, 5};

printf("%d", arr[2]); // Outputs 3

* **Multi-Dimensional Arrays**: A **multi-dimensional array** is an array of arrays. The most common is a **two-dimensional array**, which is essentially a matrix (a table of rows and columns).

**Example of a Two-Dimensional Array**:

int arr[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

printf("%d", arr[1][1]); // Outputs 5

* **Key Differences**:
  + **One-Dimensional Array**: A single list of elements.
  + **Multi-Dimensional Array**: An array where each element is an array itself, often used to represent matrices or tables.

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

* A pointer is a variable that stores the memory address of another variable. In C, pointers allow you to manipulate data indirectly and work with memory directly. They are an essential feature that helps in dynamic memory allocation, arrays, and function arguments.
* To declare a pointer in C, you use the \* symbol along with the pointer's type.
* Pointers are important in C because they allow efficient memory management, dynamic memory allocation, and manipulation of large data structures (like arrays or linked lists). They enable functions to modify variables outside their scope and help in building efficient data structures and algorithms.

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

* strlen(): Calculates the length of a string (excluding the null character \0).

Example: strlen("Hello") returns 5.

* strcpy(): Copies a string into another string.

Example: strcpy(str, "Hello"); copies "Hello" to str.

* strcat(): Concatenates (appends) one string to the end of another.

Example: strcat(str, " World"); appends " World" to str.

* strcmp(): Compares two strings lexicographically.

Example: strcmp("apple", "banana") returns a negative value because "apple" is lexicographically smaller than "banana".

* strchr(): Searches for a character in a string and returns a pointer to its first occurrence.

Example: strchr("Hello", 'e') returns a pointer to 'e' in the string.

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

* A structure is a user-defined data type that allows grouping different types of variables (data members) under one name. Structures help in organizing complex data.
* To declare a structure, use the struct keyword:

struct Student {

char name[10];

int No;

};

* To initialize a structure:-

struct Student student1 = {"Drashti Dave", 13};

* Access structure members using the dot (.) operator:

printf("\nName: %s", s1.name);

printf("\nRoll No: %d", s1.No);

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

**Importance of File Handling in C:**

* File handling in C allows programs to interact with external files for reading and writing data. It is essential for persistent data storage and manipulation.

**File Operations:**

* Opening a File: Use fopen() to open a file in the desired mode ("r", "w", "a", etc.).
* Closing a File: Use fclose() to close a file when done.
* Reading and Writing: Use functions like fgetc(), fputc(), fgets(), and fputs() to read and write data.