MODULE 2 - Introduction to Programming

1. Overview of C Programming

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

- C programming, developed in the early 1970s by Dennis Ritchie at Bell Labs, is one of the most influential languages in computer science. It was originally created to build the UNIX operating system and quickly became popular because of its speed, portability, and ability to interact closely with hardware. Over time, C evolved through standards like ANSI C, C99, and C11, improving its features while keeping its simple and powerful structure.

  C remains important today because it forms the foundation of many modern technologies. Operating systems, embedded systems, device drivers, and even popular languages like C++, Java, and Python are built using concepts from C. Its efficiency and reliability make it essential for applications where performance matters. Even after 50 years, C is still trusted in industries worldwide, proving its timeless value.

2. Setting Up Environment

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

-

3. Basic Structure of a C Program

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

- C program follows a standard structure that includes header files, the main () function, comments, data types, and variables.

▪ *Header Files:*

  Header files contain predefined functions that your program can use.

  E.g. #include<stdio.h>

▪ *Comments:*
  Comments explain the code. They are ignored by the compiler.
  E.g. a. Single Line comment - // this is a single-line comment

b. Multi-line comment - /* this is a multi-line comment */

- *main() Function:*

  Every C program must have a main () function. This is where the program starts running.

  ```
  main () {
     // code goes here
  }
  ```
- *Data Types:*

  Data types specify what type of data a variable can hold.

  E.g. int, float, char, double
- *Variables:*

  Variables are used to store data. You must declare a variable before using it.

  ```
  E.g. int age;
       float price;
       char grade;
  ```

4. Operators in C

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

- Operators are symbols that perform operations on variables and values in a C program. They are grouped into several types:

Arithmetic Operators = Used for basic mathematical calculations.

Operator	Meaning	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (remainder)	a % b

Relational Operators = Used to compare two values. Result is always true (1) or false (0).

Operator	Meaning	Example
==	Equal to	a == b
!=	Not equal	a != b
>	Greater than	a > b
<	Less than	a < b

Operator	Meaning	Example
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

Logical Operators = Used to combine multiple conditions. Result is true or false.

Operator	Meaning	Example
&&	Logical AND	(a > b) && (a > c)
!	Logical NOT	!(a > b)

Assignment Operators = Used to assign values to variables.

Operator	Meaning	Example
=	Simple assignment	a = 5
+=	Add and assign	a += 3; // a = a + 3
-=	Subtract and assign	a -= 3
*=	Multiply and assign	a *= 3
/=	Divide and assign	a /= 3
%=	Modulus and assign	a %= 3

Increment and Decrement Operators = Used to increase or decrease a variable by 1.

Operator	Meaning	Example
++	Increment	a++; ++a
--	Decrement	a--; --a

Two types:

- **Pre-increment**: ++a (increase then use)
- **Post-increment**: a++ (use then increase)

Bitwise Operators = Used to perform operations on **bits** (0s and 1s).

Operator	Meaning	Example
&	Bitwise AND	a & b
^	Bitwise XOR	a ^ b

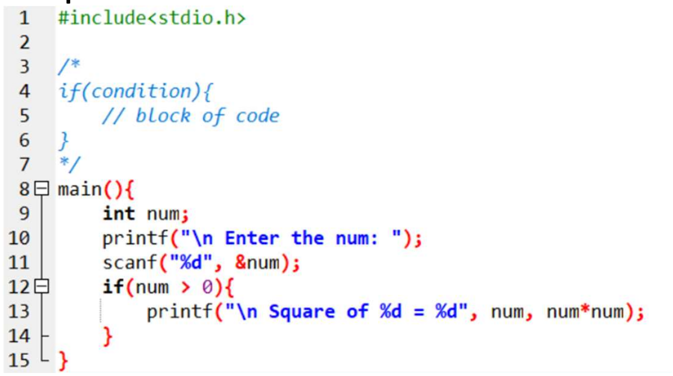
Operator	Meaning	Example
~	Bitwise NOT	~a
<<	Left shift	a << 2
>>	Right shift	a >> 2

5. Control Flow Statements in C

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

- Decision-making statements allow a program to choose different actions based on conditions.
  The main decision statements are: if, else, nested if-else, and switch.

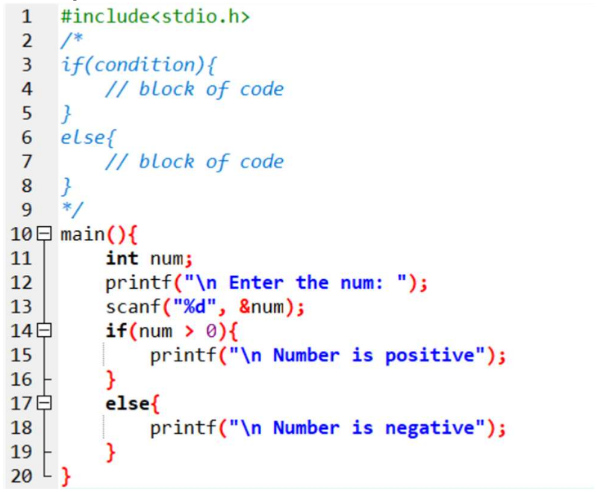
  if Statement = The if statement checks a condition. If the condition is true, the block of code runs.
  Example:



```c
#include<stdio.h>

/*
if(condition){
    // block of code
}
*/
main(){
    int num;
    printf("\n Enter the num: ");
    scanf("%d", &num);
    if(num > 0){
        printf("\n Square of %d = %d", num, num*num);
    }
}
```

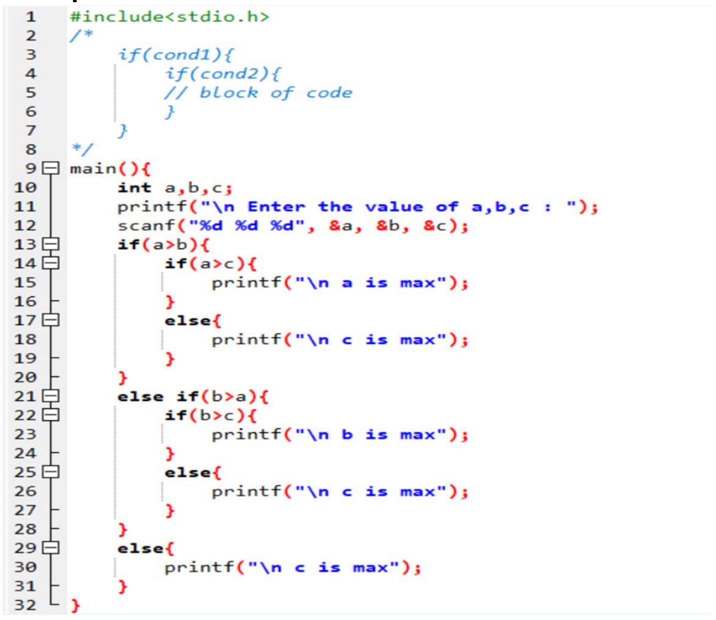
  If-else Statement = Runs one block if the condition is true, otherwise runs the else block.
  Example:



```c
#include<stdio.h>
/*
if(condition){
    // block of code
}
else{
    // block of code
}
*/
main(){
    int num;
    printf("\n Enter the num: ");
    scanf("%d", &num);
    if(num > 0){
        printf("\n Number is positive");
    }
    else{
        printf("\n Number is negative");
    }
}
```

Nested if-else = When one if-else is written inside another if-else. Used for multiple conditions.

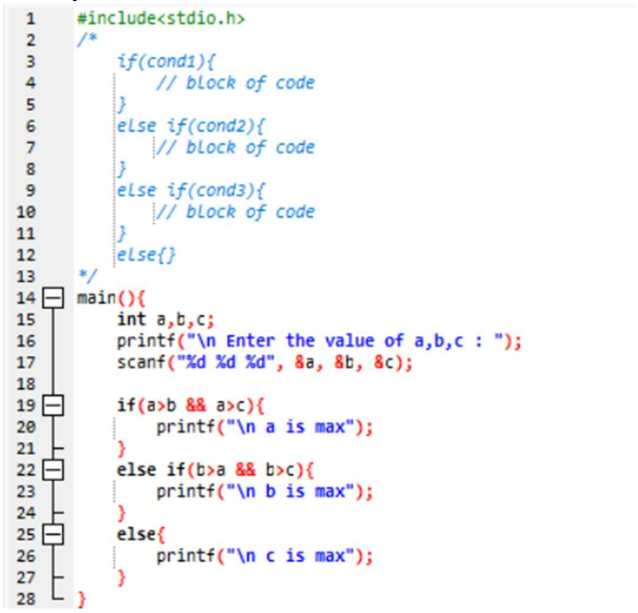
Example:



```c
#include<stdio.h>
/*
    if(cond1){
        if(cond2){
        // block of code
        }
    }
*/
main(){
    int a,b,c;
    printf("\n Enter the value of a,b,c : ");
    scanf("%d %d %d", &a, &b, &c);
    if(a>b){
        if(a>c){
            printf("\n a is max");
        }
        else{
            printf("\n c is max");
        }
    }
    else if(b>a){
        if(b>c){
            printf("\n b is max");
        }
        else{
            printf("\n c is max");
        }
    }
    else{
        printf("\n c is max");
    }
}
```

Else-if Ladder = Used to check many conditions one by one.

Example:

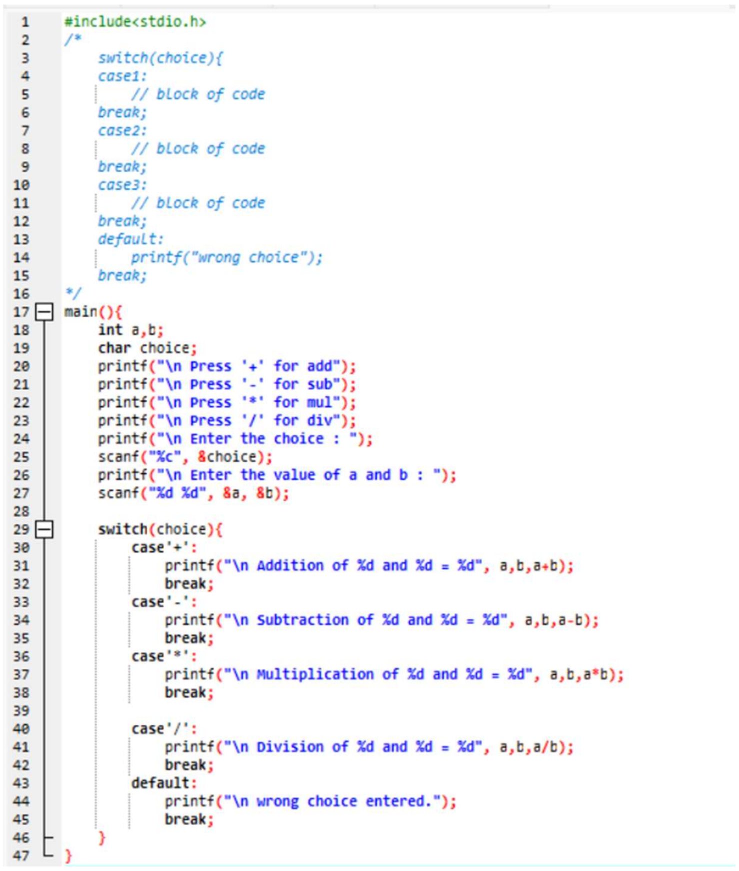


```c
#include<stdio.h>
/*
    if(cond1){
        // block of code
    }
    else if(cond2){
        // block of code
    }
    else if(cond3){
        // block of code
    }
    else{}
*/
main(){
    int a,b,c;
    printf("\n Enter the value of a,b,c : ");
    scanf("%d %d %d", &a, &b, &c);

    if(a>b && a>c){
        printf("\n a is max");
    }
    else if(b>a && b>c){
        printf("\n b is max");
    }
    else{
        printf("\n c is max");
    }
}
```

Switch Statement = the switch statement chooses a block based on the value of a variable. Useful when there are many constant choices.

Example:



```c
#include<stdio.h>
/*
    switch(choice){
    case1:
        // block of code
    break;
    case2:
        // block of code
    break;
    case3:
        // block of code
    break;
    default:
        printf("wrong choice");
    break;
*/
main(){
    int a,b;
    char choice;
    printf("\n Press '+' for add");
    printf("\n Press '-' for sub");
    printf("\n Press '*' for mul");
    printf("\n Press '/' for div");
    printf("\n Enter the choice : ");
    scanf("%c", &choice);
    printf("\n Enter the value of a and b : ");
    scanf("%d %d", &a, &b);

    switch(choice){
        case'+':
            printf("\n Addition of %d and %d = %d", a,b,a+b);
            break;
        case'-':
            printf("\n Subtraction of %d and %d = %d", a,b,a-b);
            break;
        case'*':
            printf("\n Multiplication of %d and %d = %d", a,b,a*b);
            break;

        case'/':
            printf("\n Division of %d and %d = %d", a,b,a/b);
            break;
        default:
            printf("\n wrong choice entered.");
            break;
    }
}
```

6. Looping in C

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

- Comparison of while, for, and do-while Loops in C

Feature	**while Loop**	**for Loop**	**do-while Loop**
Condition check	Before loop	Before loop	After loop
Executes at least once?	No	No	Yes
Use case	Unknown iterations	Known iterations	Must run once
Structure	Condition-only loop	Compact and controlled	Post-condition loop
Common use	Reading data, waiting	Counting, arrays	Menus, user input

7. Loop Control Statements

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

- Here are clear notes on break, continue, and goto statements in C, along with simple examples.

  Break Statement = Used to exit a loop (for, while, do-while) immediately. Also used to exit a switch case.

  How it works: When break is executed, the control jumps out of the loop or out of the switch.

  Example in a loop:

```
for (int i = 1; i <= 10; i++) {
   if (i == 5) {
      break;     // Loop ends when i = 5
   }
   printf("%d ", i);
}
```

  Continue Statement = Skips the current iteration of a loop. Moves directly to the next iteration.

  How it works:

  In for loop → jump to increment step

  In while/do-while → jump to condition checking

  Example:

```
for (int i = 1; i <= 5; i++) {
   if (i == 3) {
      continue;  // skips printing 3
   }
   printf("%d ", i);
}
```

  Goto Statement = Used to jump to a labelled statement anywhere in the program. Rarely used today because it can make code hard to read.

  Example:

```
int i = 1;
start:
printf("%d ", i);
i++;

if (i <= 5) {
    goto start;  // jumps back to label
}
```

8. Functions in C

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

- A function is a set of statements that take inputs, do some specific computation and produces output.
  E.g. main (), sum (), swap ().

  1. Function Declaration =Tells the compiler the name, return type, and parameters of the function before it is used.

  Example:
  ```
  int add(int, int);   // function declaration
  ```

  2. Function Definition = This is where the function's actual code (logic) is written.

  Example:
  ```
  int add(int a, int b) {
     return a + b;
  }
  ```

  3. Calling a Function = You call/use the function in main() or another function by writing its name and passing arguments.

  Example:
  ```
  #include <stdio.h>

  int add(int, int);   // declaration

  int main() {
     int result = add(5, 3);   // function call
     printf("Sum = %d", result);
  }

  int add(int a, int b) {      // definition
     return a + b;
  }
  ```

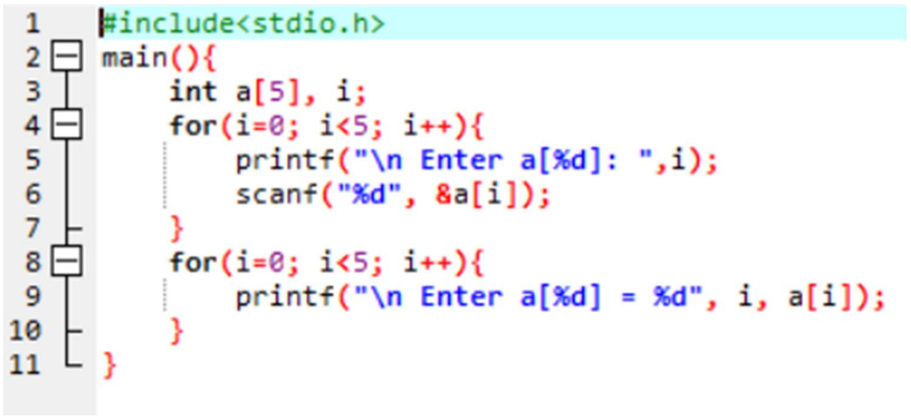
9. Arrays in C

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

- An array is used to store a collection of data having same data type, and it is often used as a collection variables of same data type. They consist of contiguous memory.

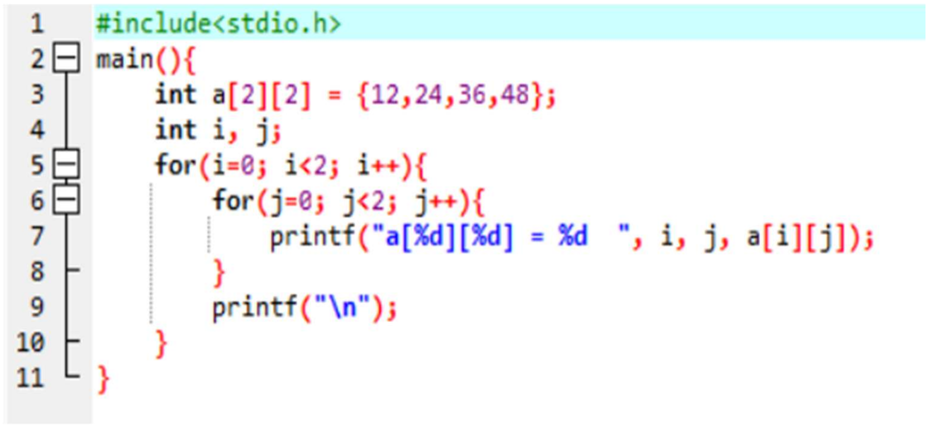
  Below is the example of arrays showing difference between 1D and 2D arrays-

  One-dimensional array:



```c
#include<stdio.h>
main(){
    int a[5], i;
    for(i=0; i<5; i++){
        printf("\n Enter a[%d]: ",i);
        scanf("%d", &a[i]);
    }
    for(i=0; i<5; i++){
        printf("\n Enter a[%d] = %d", i, a[i]);
    }
}
```

  Two-dimensional array:



```c
#include<stdio.h>
main(){
    int a[2][2] = {12,24,36,48};
    int i, j;
    for(i=0; i<2; i++){
        for(j=0; j<2; j++){
            printf("a[%d][%d] = %d  ", i, j, a[i][j]);
        }
        printf("\n");
    }
}
```

10. Pointers in C

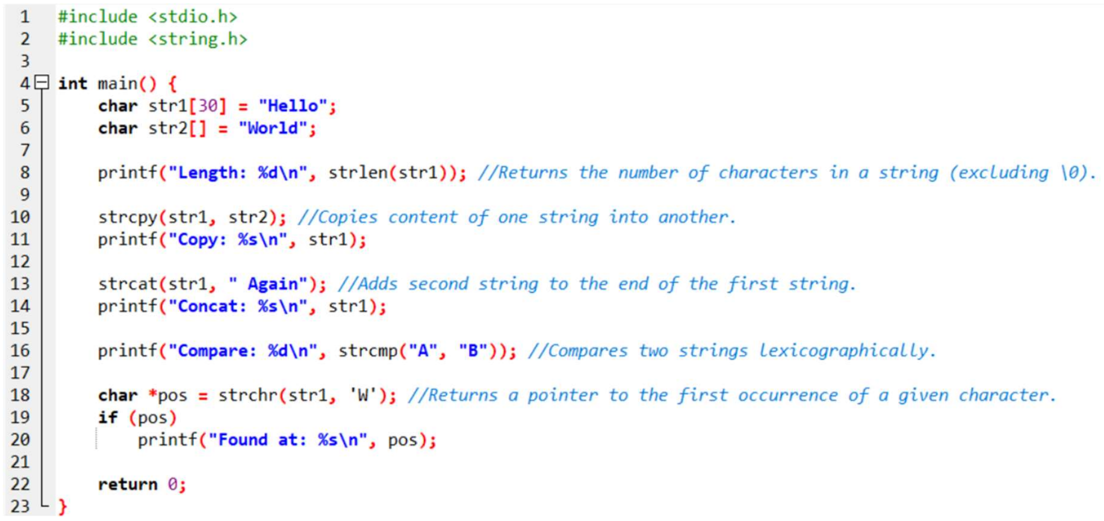
Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

- A pointer is a variable that stores the memory address of another variable. Instead of storing a value directly, it stores where the value is located in memory.
  It is declared using an asterisk (*) operator.
  The pointer is initialized using an address of operator-&.

11. Strings in C

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

- Below image explains all the functions.



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

int main() {
    char str1[30] = "Hello";
    char str2[] = "World";

    printf("Length: %d\n", strlen(str1)); //Returns the number of characters in a string (excluding \0).

    strcpy(str1, str2); //Copies content of one string into another.
    printf("Copy: %s\n", str1);

    strcat(str1, " Again"); //Adds second string to the end of the first string.
    printf("Concat: %s\n", str1);

    printf("Compare: %d\n", strcmp("A", "B")); //Compares two strings lexicographically.

    char *pos = strchr(str1, 'W'); //Returns a pointer to the first occurrence of a given character.
    if (pos)
        printf("Found at: %s\n", pos);

    return 0;
}
```

12. Structures in C

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

- In C, a structure is a user-defined data type that allows grouping different types of data under a single name. Each element is a member and they can be of different types, such as integer, float or array. They are used to represent complex entities such as database. They are defined as ‘struct’ keyword or dot operator ’.’

13. File Handling in C

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

- File handling allows a C program to store data permanently on disk.
  Basic file handling operations are:
  - Opening a file = Use fopen() to open a file.

- Writing a file =
  fprintf(fp, "Hello World");
  fputs("Hello C", fp);
  fputc('A', fp);

- Reading a file =
  fscanf(fp, "%d", &num);
  fgets(str, 50, fp);
  ch = fgetc(fp);

- Closing a file = fclose(fp);