1. a) How can comments (remarks) be included within a C program? Where can comments be
placed? (5 Marks)

In C, comments are used to add explanatory notes within the code, making it easier to understand for programmers. Comments are ignored by the compiler, meaning they do not affect the execution of the program.

Comments in the program

In the C programming language, // and /* ... */ are two different ways to represent comments, which are annotations in the code that are ignored by the compiler.

The // syntax is used for single-line comments.

The /* ... */ syntax is used for multi-line comments.

Comments can be placed:

- **Before a code block** to describe its purpose.
- **Beside a statement** for quick explanations.
- **At the beginning of a program** to provide general information like the program's purpose, author, and date.
- **Inside functions** to explain logic.
- **Between lines of code** to describe complex sections.

b) Name and describe the four basic types of constants in C. (4 Marks)

Constants

Fixed values that do not change during the execution of a program. There are different types of constants in C programming

Integer Constant 10, 20, 450 etc. Real or

Floating-point Constant 10.3, 20.2, 450.6 etc.

Character Constant ’a’, ’b’, ’x’ etc.

String Constant ”c”, ”c program”.

2. Can a keyword be used as an identifier? Justify your answer. Also, list at least 5 C keywords and explain their purpose in C.(9marks)

No, a keyword **cannot** be used as an identifier in C.

Justification:

- **Keywords are reserved words** that have special meanings in the C language.
- They are predefined and serve specific purposes in program structure and syntax.
- Using a keyword as an identifier (e.g., a variable or function name) would cause a **compilation error** since the compiler expects the keyword to perform its designated function.

Eg) int return = 10;   // Error: 'return' is a keyword

This results in a **syntax error** because return is a reserved keyword used for returning values from functions.

Five C Keywords and Their Purpose:

int

Used to declare integer-type variables.a

Eg) int age = 25; // Declares an integer variable 'age'

return

- Used to return a value from a function.

```
Eg) int sum() {

    return 10 + 20;  // Returns the sum

}
```

if

- Used for conditional statements.

```
Eg)if (age > 18) {

    printf("Adult");

}
```

while

- Used for loop execution as long as a condition is true.

```
Eg)int i = 0;

while (i < 5) {
    printf("%d ", i);

    i++;

}
```

void

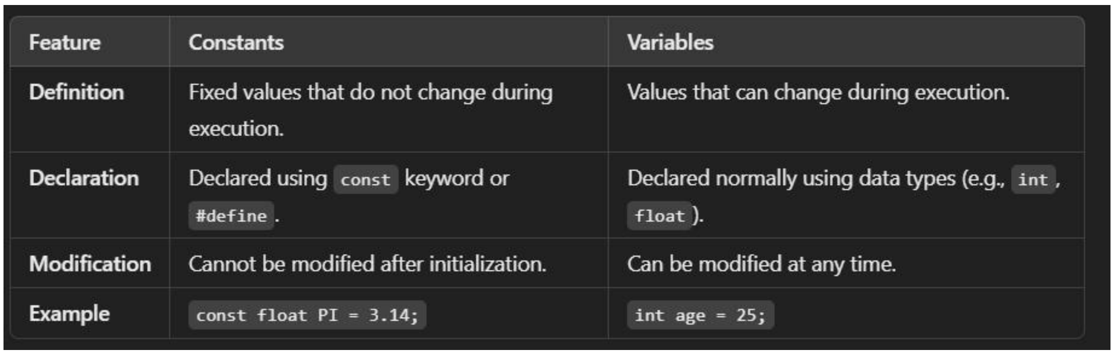
- Specifies that a function does not return a value.

```
Eg)void greet() {

    printf("Hello, World!");

}
```

3. a) Discuss the key features of C programming language. (6 Marks)

1. **Simple and Efficient** – Provides a structured approach with clear syntax, making it easy to understand and efficient in execution.
2. **Fast Execution** – Programs written in C run quickly due to direct interaction with system hardware.
3. **Portability** – Code written in C can be compiled and run on different operating systems with minimal modifications.
4. **Rich Library Support** – Includes standard libraries (stdio.h, math.h, etc.) for various functionalities.
5. **Memory Management** – Provides direct access to memory using pointers and functions like malloc() and free().
6. **Structured Programming** – Supports functions, loops, and conditional statements, allowing modular program design.

b) How do constants differ from variables in C? (3 Marks)



Feature	Constants	Variables
Definition	Fixed values that do not change during execution.	Values that can change during execution.
Declaration	Declared using `const` keyword or `#define`.	Declared normally using data types (e.g., `int`, `float`).
Modification	Cannot be modified after initialization.	Can be modified at any time.
Example	`const float PI = 3.14;`	`int age = 25;`

4. What are constants and keywords in C? Provide examples.

Constants – Fixed values that do not change during program execution.

```
    Eg)#define MAX 100

    const float PI = 3.1415;
```

Keywords – Reserved words with predefined meanings in C

```
  Eg)int num = 10;  // 'int' is a keyword

if (num > 5) {  // 'if' is a keyword

    return 1;  // 'return' is a keyword

}
```

5. How do variables and data types in C contribute to the efficiency and clarity of a program? Provide an example.

- **Variables store data dynamically**, making the program flexible.
- **Data types ensure memory efficiency** by allocating appropriate storage.
- **Improves readability** by clearly defining what kind of data is being handled.

Eg)

```
#include <stdio.h>

int main() {

    int age = 20;  // Integer variable

    float height = 5.8;  // Floating-point variable

    printf("Age: %d, Height: %.1f", age, height);

    return 0;

}
```

6.Explain identifiers. Give rules for declaring identifiers.

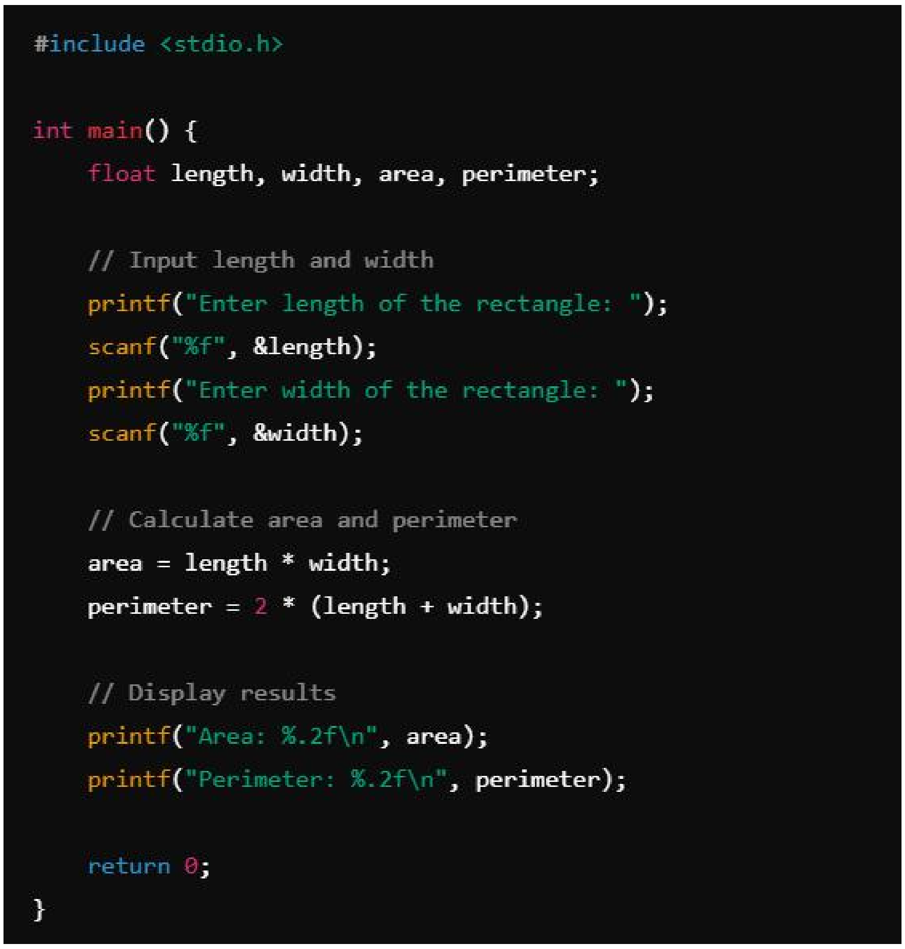
What are Identifiers?

Identifiers are names given to variables, functions, and other user-defined elements in a C program.

Rules for Declaring Identifiers:

1. Must start with a letter (A-Z or a-z) or an underscore (_).
2. Can contain letters, digits (0-9), and underscores.
3. Cannot be a C **keyword** (e.g., int, return).
4. **Case-sensitive** (Age and age are different).
5. Should be meaningful and descriptive.

7. Write a C program to calculate the area and perimeter of a rectangle. What are the appropriate data types for the inputs (length and width) and the calculated values (area and perimeter), and why?(9)



```c
#include <stdio.h>

int main() {
    float length, width, area, perimeter;

    // Input length and width
    printf("Enter length of the rectangle: ");
    scanf("%f", &length);
    printf("Enter width of the rectangle: ");
    scanf("%f", &width);

    // Calculate area and perimeter
    area = length * width;
    perimeter = 2 * (length + width);

    // Display results
    printf("Area: %.2f\n", area);
    printf("Perimeter: %.2f\n", perimeter);

    return 0;
}
```

Appropriate Data Types:

- float or double for **length and width**: These values may include decimals (e.g., 5.5, 10.2).
- float or double for **area and perimeter**: Since they are computed using length × width or 2 × (length + width), floating-point precision is needed.

Why float**?**

- Length and width can be decimal values.
- Area and perimeter calculations may result in decimal numbers.

8. a) What is the most suitable variable type to represent the area of a circle in square inches, and why? (3 Marks)

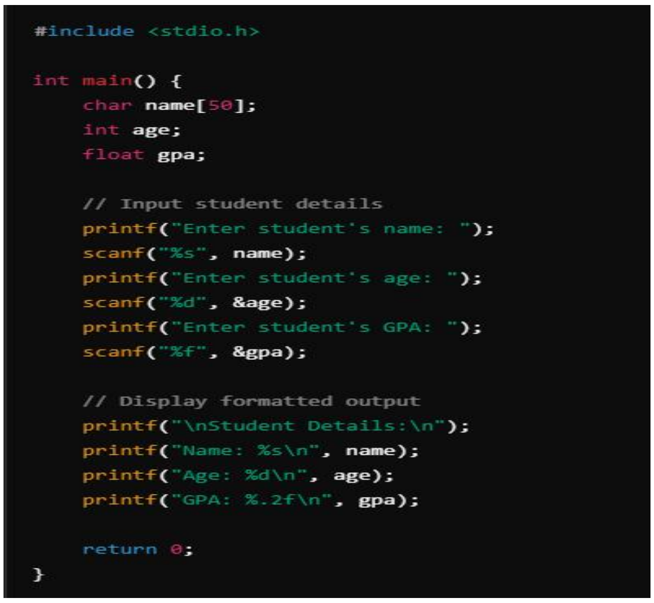
The most suitable data type is double.

- The area of a circle is calculated as **π × r²**, which involves floating-point calculations.
- double provides **higher precision** than float, reducing rounding errors.

Example declaration:

double area;

b) Write a C program that accepts the following inputs: a student's name (string), age (integer), and GPA (float). Then,display the student's details in a formatted output. What are the appropriate data types for each input, and why? (6 Marks)



```
#include <stdio.h>

int main() {
    char name[50];
    int age;
    float gpa;

    // Input student details
    printf("Enter student's name: ");
    scanf("%s", name);
    printf("Enter student's age: ");
    scanf("%d", &age);
    printf("Enter student's GPA: ");
    scanf("%f", &gpa);

    // Display formatted output
    printf("\nStudent Details:\n");
    printf("Name: %s\n", name);
    printf("Age: %d\n", age);
    printf("GPA: %.2f\n", gpa);

    return 0;
}
```

Appropriate Data Types:

- char[] for **name** – A string of characters.
- int for **age** – Whole numbers only.
- float for **GPA** – May include decimal values

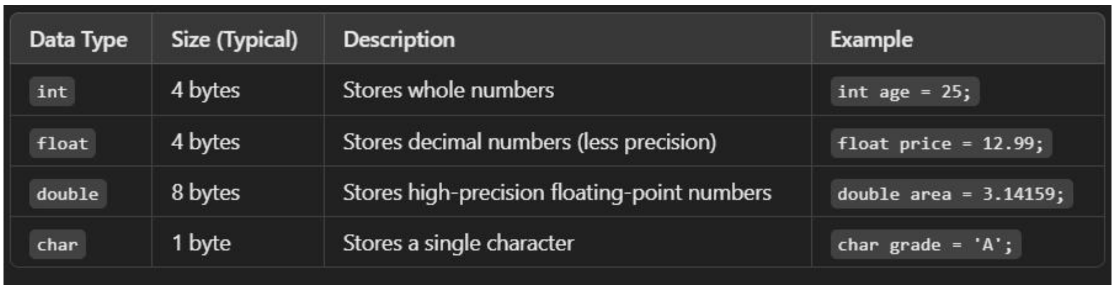
Why These Data Types?

- char name[50]: Holds a sequence of characters (name).
- int age: Stores whole numbers.
- float gpa: Allows decimal values.

9. a) Why is it important to initialize variables before using them in a program? (3 Marks)

· **Avoids Garbage Values** – Uninitialized variables may contain unpredictable values, leading to incorrect results.
· **Prevents Undefined Behavior** – Using an uninitialized variable can cause errors or crashes.
· **Improves Code Reliability** – Ensures predictable program execution.

b) Describe the fundamental data types in C. (6 Marks)



Data Type	Size (Typical)	Description	Example
`int`	4 bytes	Stores whole numbers	`int age = 25;`
`float`	4 bytes	Stores decimal numbers (less precision)	`float price = 12.99;`
`double`	8 bytes	Stores high-precision floating-point numbers	`double area = 3.14159;`
`char`	1 byte	Stores a single character	`char grade = 'A';`

10. What will be the output of the following C code and why?

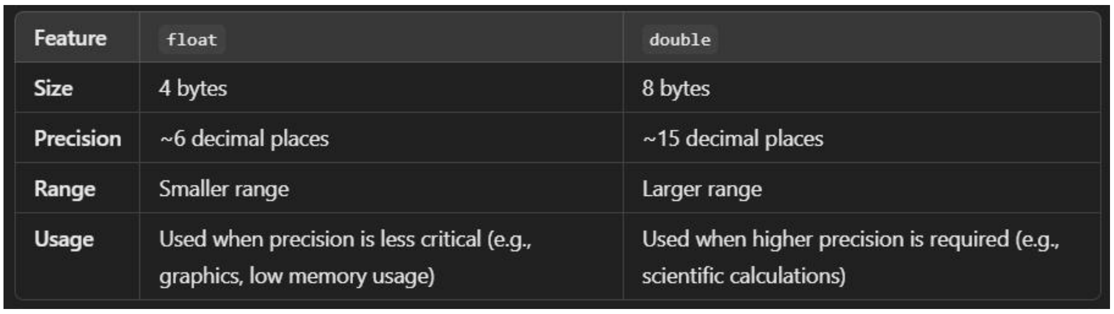
```
int a = 5, b = 2, c;
c = a / b;
printf("%d", c);
```

Output= 2

Explanation:

- a and b are both declared as int, meaning integer division is performed.
- 5 / 2 in **integer division** discards the decimal part and results in 2, not 2.5.
- Since c is also an int, it stores 2 as the final result.

11. Explain the difference between float and double data types in C.



Feature	`float`	`double`
Size	4 bytes	8 bytes
Precision	~6 decimal places	~15 decimal places
Range	Smaller range	Larger range
Usage	Used when precision is less critical (e.g., graphics, low memory usage)	Used when higher precision is required (e.g., scientific calculations)

12. What will be the value of x after execution of the following C code and why?

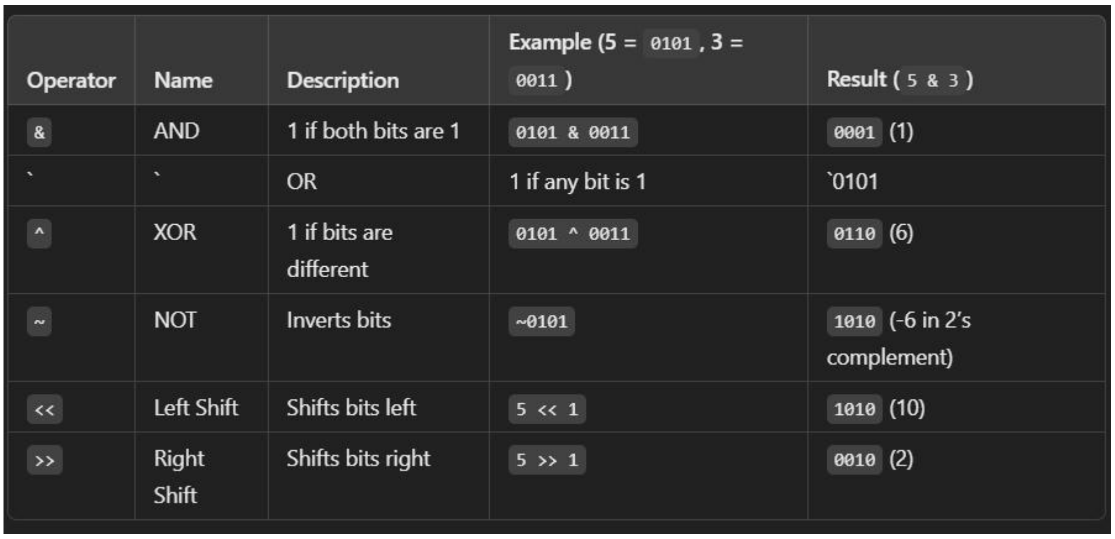
```
int x,y=10;
char z='a';
x=y+z;
```

107

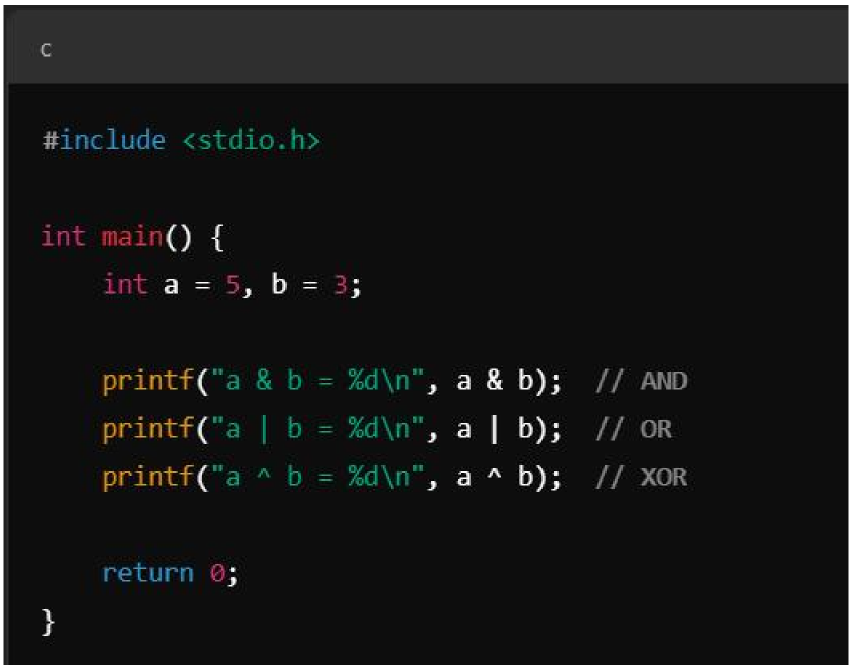
Explanation:

- char z = 'a' → The ASCII value of 'a' is **97**.
- y + z → 10 + 97 = 107.
- Since x is an int, it stores 107.

13. Explain the different bit-wise operators available in C. Write a C program that demonstrates the use of the following bit-wise operators: &, |, ^ on two integer variables. (9marks)



Operator	Name	Description	Example (5 = `0101`, 3 = `0011`)	Result (`5 & 3`)
`&`	AND	1 if both bits are 1	`0101 & 0011`	`0001` (1)
`	`	OR	1 if any bit is 1	`0101
`^`	XOR	1 if bits are different	`0101 ^ 0011`	`0110` (6)
`~`	NOT	Inverts bits	`~0101`	`1010` (-6 in 2's complement)
`<<`	Left Shift	Shifts bits left	`5 << 1`	`1010` (10)
`>>`	Right Shift	Shifts bits right	`5 >> 1`	`0010` (2)



c

```c
#include <stdio.h>

int main() {
    int a = 5, b = 3;

    printf("a & b = %d\n", a & b);  // AND
    printf("a | b = %d\n", a | b);  // OR
    printf("a ^ b = %d\n", a ^ b);  // XOR

    return 0;
}
```

<u>output</u>

a & b = 1

a | b = 7

a ^ b = 6

14. Describe the concept of operators in C and explain the various types with examples. (9 marks)

· **Arithmetic Operators** (For mathematical operations)

- +, -, *, /, %
- Example: int sum = 10 + 5;

· **Relational (Comparison) Operators** (For comparisons)

- ==, !=, >, <, >=, <=
- Example: if (a > b)

· **Logical Operators** (For boolean logic)

- && (AND), || (OR), ! (NOT)
- Example: if (x > 5 && y < 10)

· **Bitwise Operators** (Operate on bits)

- &, |, ^, ~, <<, >>
- Example: a & b

· **Assignment Operators** (Assign values)

- =, +=, -=, *=, /=, %=
- Example: a += 5;

· **Increment and Decrement Operators**

- ++ (Increment), -- (Decrement)
- Example: x++

· **Ternary Operator** (?:)

- Shorthand for if-else
- Example: int min = (a < b) ? a : b;

· **Sizeof Operator**

- Returns the size of a data type
- Example: sizeof(int)

· **Comma Operator**

- Used in loops and variable declarations
- Example: int a = (b = 5, c = 10);

15. a) How can modulus and division operators be used to manipulate numbers? (3 Marks)

- **Modulus (`%`)** – Returns the remainder of a division.
- **Division (`/`)** – Returns the quotient.

Eg)

```
int x = 10, y = 3;

printf("%d", x / y);  // Output: 3 (quotient)

printf("%d", x % y);  // Output: 1 (remainder)
```

Useful for:

- Checking if a number is even (`x % 2 == 0`).
- Extracting digits (`x % 10` gives last digit of a number).

b) What is the result of the following C code? Explain your answer based on operator precedence and the behavior of the increment operators. (6 Marks)
int x = 5,y=10,result; result = x++ + ++y;

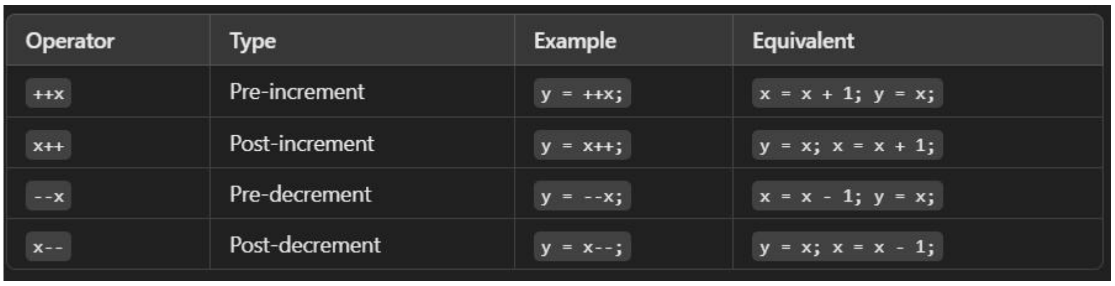
Step-by-Step Execution:

1. x++ → Uses x = 5, then increments x to 6.
2. ++y → Increments y to 11, then uses 11.
3. result = 5 + 11 = 16.

Final Values:

- x = 6
- y = 11
- result = 16

16. Explain the increment and decrement operators in C programming with examples. (3 marks)



Operator	Type	Example	Equivalent
`++x`	Pre-increment	`y = ++x;`	`x = x + 1; y = x;`
`x++`	Post-increment	`y = x++;`	`y = x; x = x + 1;`
`--x`	Pre-decrement	`y = --x;`	`x = x - 1; y = x;`
`x--`	Post-decrement	`y = x--;`	`y = x; x = x - 1;`

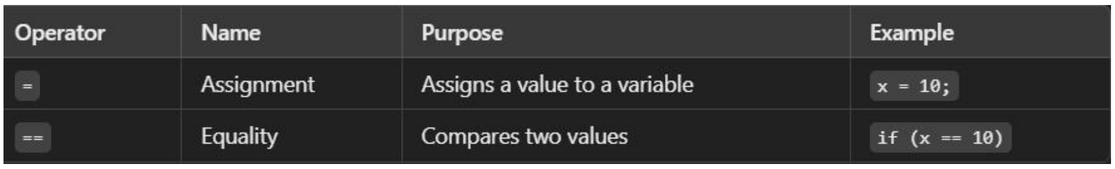
17. What is the significance of operator precedence in a C program, and how does it impact the order of execution in expressions? (3 marks)

Operator precedence determines the **order of evaluation** in expressions.

Impact:

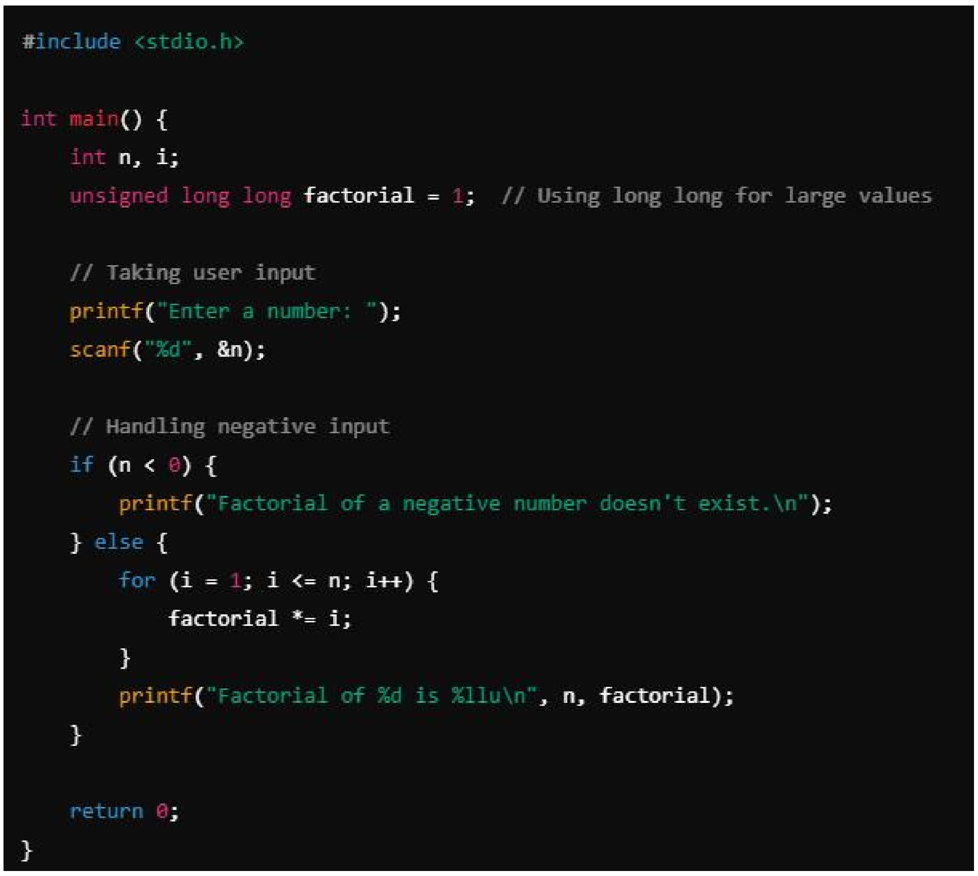
- Operators with higher precedence execute **first**.
- Parentheses () can **override precedence**.

18. Differentiate between the equality operator and the assignment operator with examples.(3 marks)



Operator	Name	Purpose	Example
`=`	Assignment	Assigns a value to a variable	`x = 10;`
`==`	Equality	Compares two values	`if (x == 10)`

18. Write a C program that prompts the user to enter a number and then computes and displays the factorial of that number.(9)



```c
#include <stdio.h>

int main() {
    int n, i;
    unsigned long long factorial = 1;  // Using long long for large values

    // Taking user input
    printf("Enter a number: ");
    scanf("%d", &n);

    // Handling negative input
    if (n < 0) {
        printf("Factorial of a negative number doesn't exist.\n");
    } else {
        for (i = 1; i <= n; i++) {
            factorial *= i;
        }
        printf("Factorial of %d is %llu\n", n, factorial);
    }

    return 0;
}
```

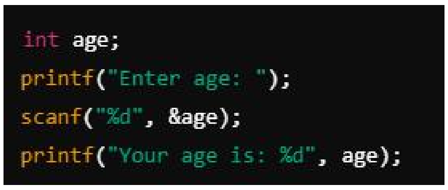
20. Explain the formatted and unformatted I/O functions of the C language. (9)

Formatted I/O Functions:

Used for structured input/output with format specifiers.

- printf(): Displays output in a formatted manner.
- scanf(): Reads formatted input from the user.

Eg)



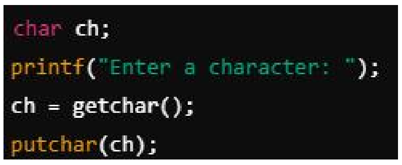
```c
int age;
printf("Enter age: ");
scanf("%d", &age);
printf("Your age is: %d", age);
```

Unformatted I/O Functions:

Work with raw input/output (no format specifiers).

- getchar(), putchar(): Read/write single characters.
- gets(), puts(): Read/write strings.

Eg)

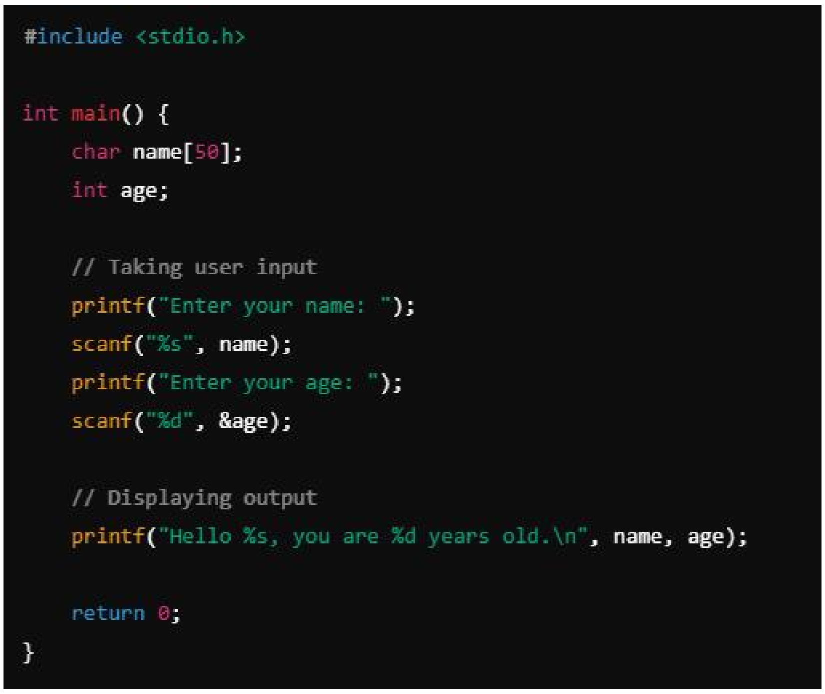


```c
char ch;
printf("Enter a character: ");
ch = getchar();
putchar(ch);
```

21. a) Why is the main() function considered the entry point of a C program? (3 Marks)

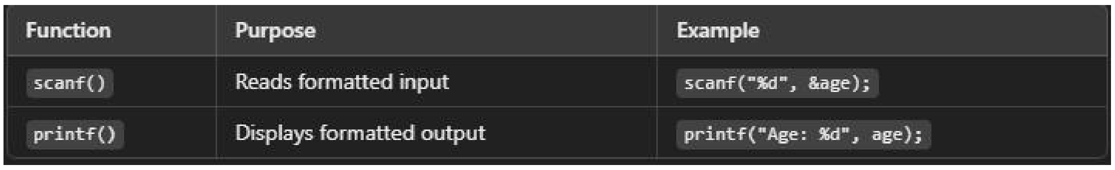
· **Execution starts from** main() – The operating system calls main() when a C program runs.
· **Controls program flow** – Other functions execute within main().
· **Standardization** – main() is required in every C program.

b) Write a C program that accepts the user's name and age as input and prints a message in the
format: Hello <name>, you are <age> years old. (6 Marks)



```c
#include <stdio.h>

int main() {
    char name[50];
    int age;

    // Taking user input
    printf("Enter your name: ");
    scanf("%s", name);
    printf("Enter your age: ");
    scanf("%d", &age);

    // Displaying output
    printf("Hello %s, you are %d years old.\n", name, age);

    return 0;
}
```

22. Describe the usage of the built-in functions scanf() for input and printf() for output formatting.(3)

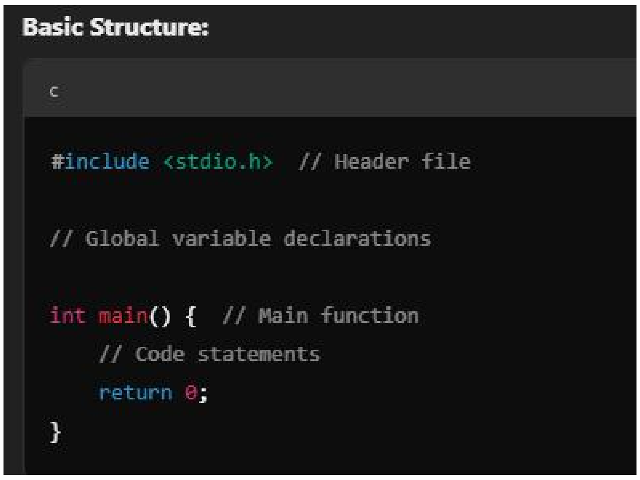


Function	Purpose	Example
`scanf()`	Reads formatted input	`scanf("%d", &age);`
`printf()`	Displays formatted output	`printf("Age: %d", age);`

23. What is the role of the documentation section in a C program?(3)

- **Provides comments about the program** (author, purpose, date).
- **Improves code readability** for other developers.
- **Uses** /*...*/ **or** // for documentation.

24. What is the basic structure of a C program, and why is it important to include elements like the main function and header files?(3)



Basic Structure:

```c
#include <stdio.h>  // Header file

// Global variable declarations

int main() {  // Main function
    // Code statements
    return 0;
}
```

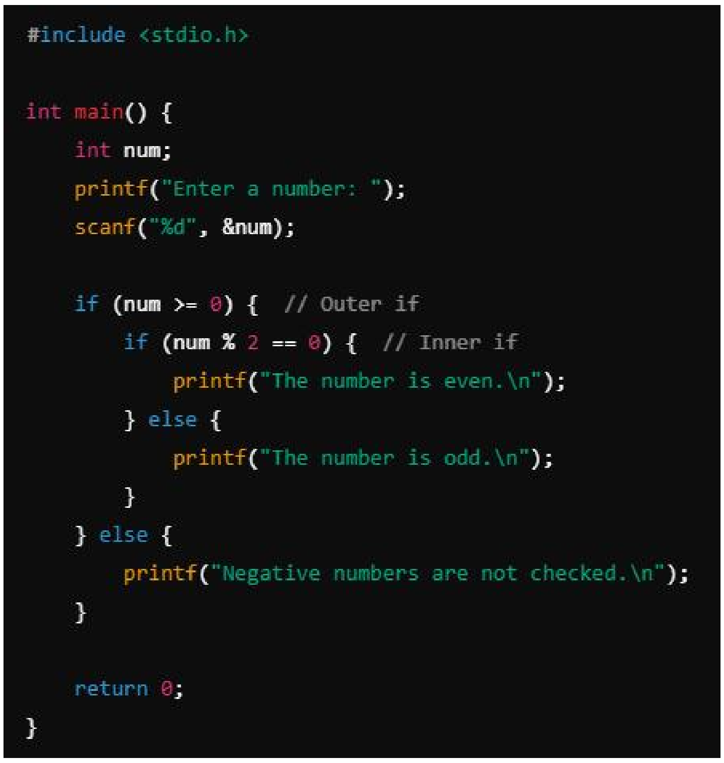
Why Important?

- **Header Files** (#include <stdio.h>) – Enable built-in functions.
- main() **Function** – Essential entry point.
- **Code Blocks** ({}) – Define execution scope.

Without main()**, a C program cannot execute!**

25. a) What are nested if statements? Explain their importance with an example. (4 Marks)

A **nested** if **statement** is an if condition inside another if condition. It is used when multiple conditions need to be checked sequentially.



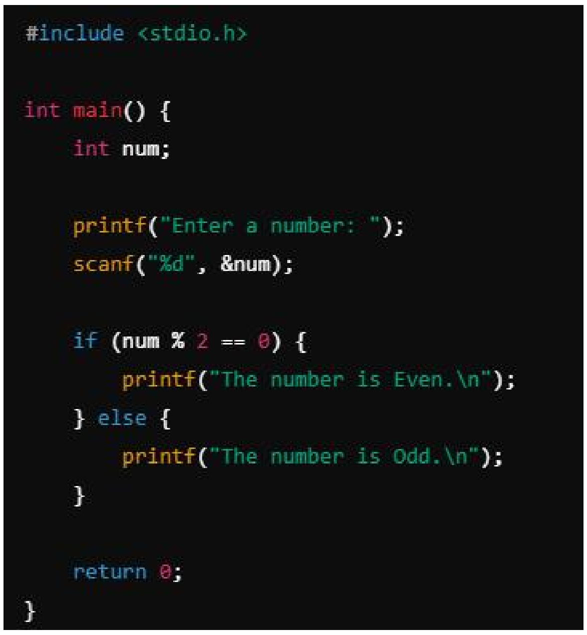
```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num >= 0) {  // Outer if
        if (num % 2 == 0) {  // Inner if
            printf("The number is even.\n");
        } else {
            printf("The number is odd.\n");
        }
    } else {
        printf("Negative numbers are not checked.\n");
    }

    return 0;
}
```

Importance:

- Allows checking multiple conditions step by step.

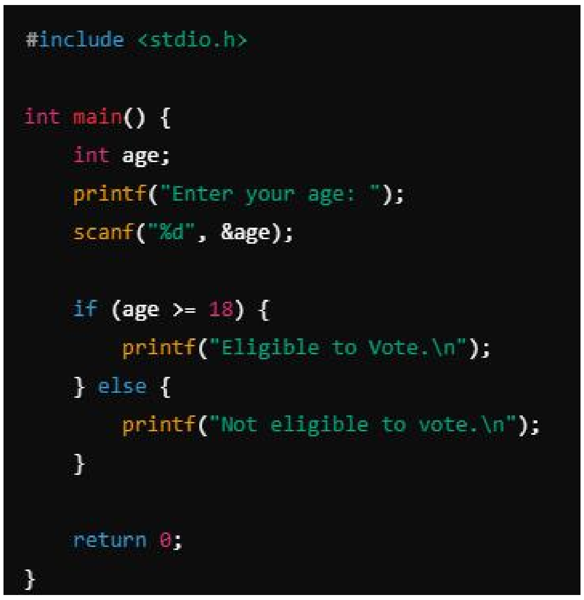
- Used when decisions depend on previous conditions.

b) Write a program to check whether a number is even or odd. (5 Marks)



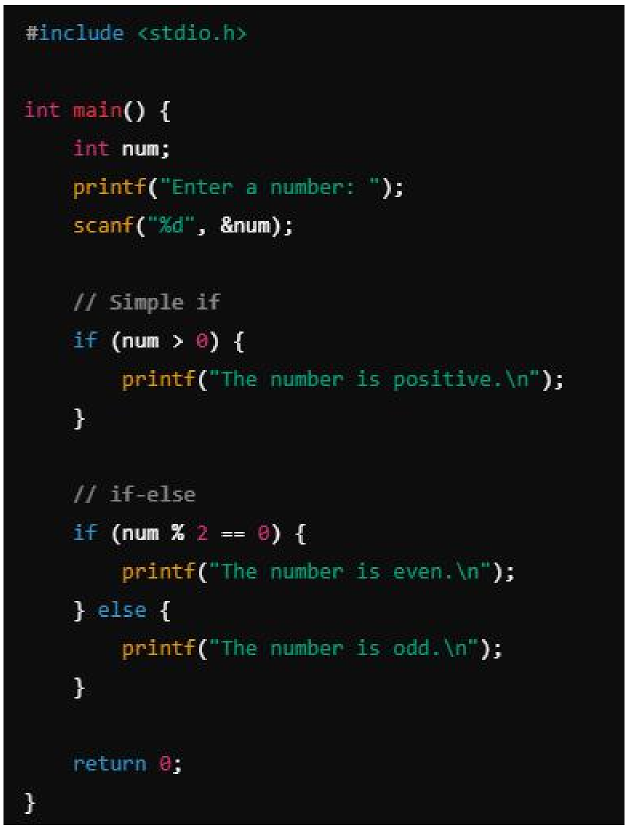
```c
#include <stdio.h>

int main() {
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num % 2 == 0) {
        printf("The number is Even.\n");
    } else {
        printf("The number is Odd.\n");
    }

    return 0;
}
```

26. a) Write a program to check whether the candidate's age is greater than 17. If yes, display "Eligible to Vote." (5 Marks)



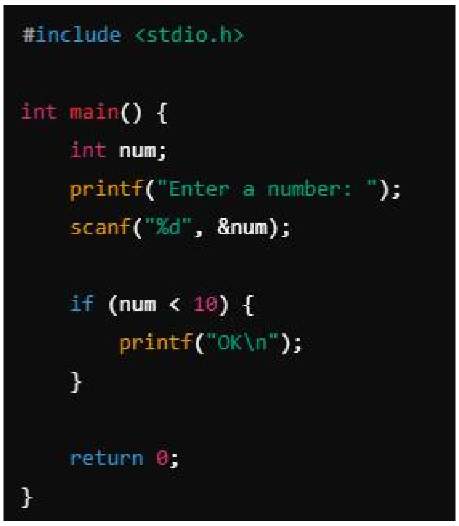
```c
#include <stdio.h>

int main() {
    int age;
    printf("Enter your age: ");
    scanf("%d", &age);

    if (age >= 18) {
        printf("Eligible to Vote.\n");
    } else {
        printf("Not eligible to vote.\n");
    }

    return 0;
}
```

b) Write a C program to demonstrate the use of if and if-else statements with examples. (4 Marks)



```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    // Simple if
    if (num > 0) {
        printf("The number is positive.\n");
    }

    // if-else
    if (num % 2 == 0) {
        printf("The number is even.\n");
    } else {
        printf("The number is odd.\n");
    }

    return 0;
}
```

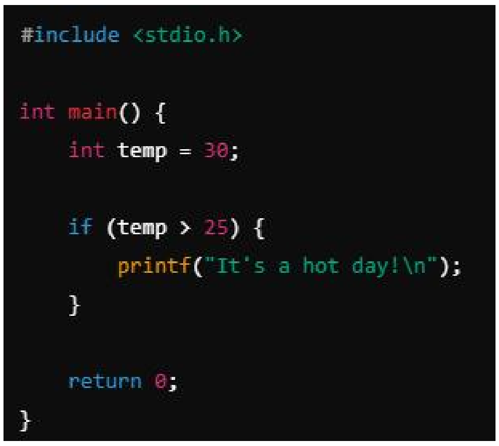
27. a) Write a program to check whether the entered number is less than 10. If yes, display the text “OK.” (5 Marks)



```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num < 10) {
        printf("OK\n");
    }

    return 0;
}
```

b) What is a simple if statement? Explain their importance with an example. (4 Marks)

A **simple** if **statement** executes a block of code **only if** the condition is true.

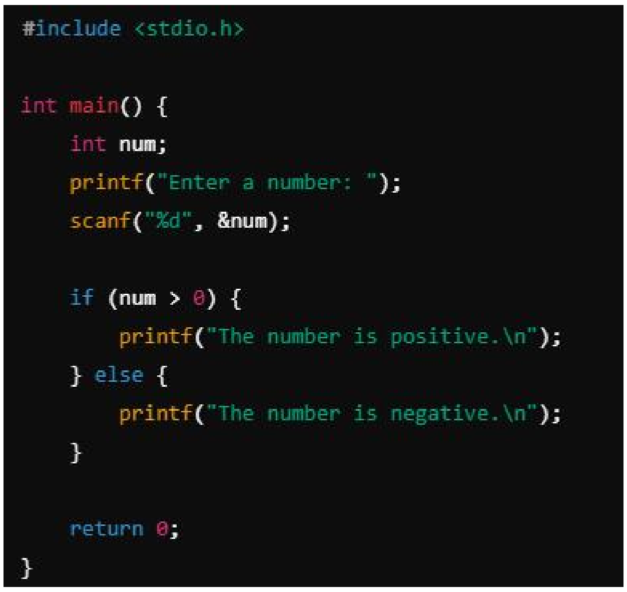


```c
#include <stdio.h>

int main() {
    int temp = 30;

    if (temp > 25) {
        printf("It's a hot day!\n");
    }

    return 0;
}
```

Importance:

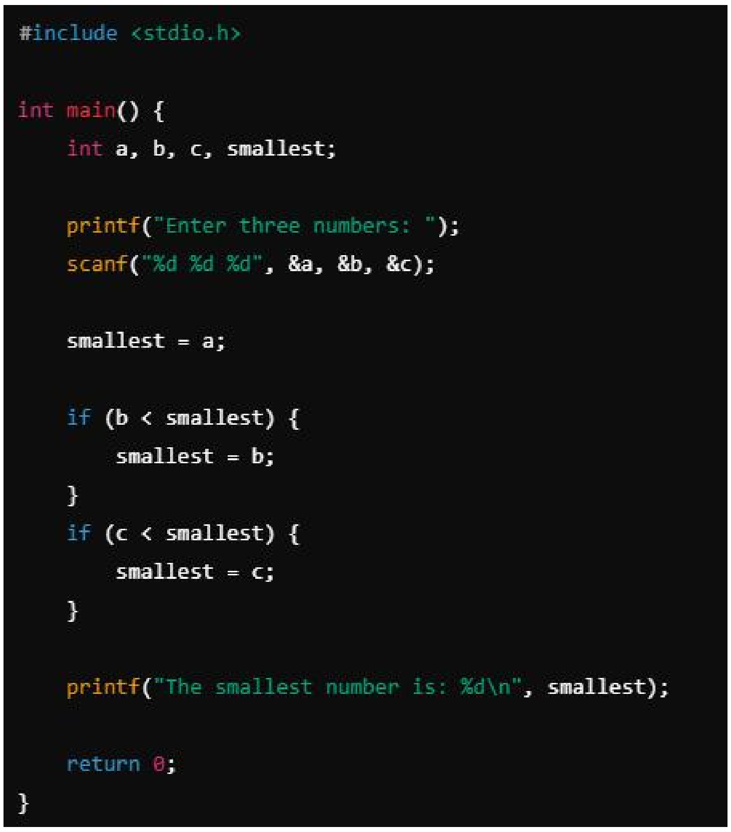
- Helps control the flow of execution.
- Ensures that only relevant code executes based on conditions.

28. Write a simple C program that checks whether a number is positive or negative using if statement.



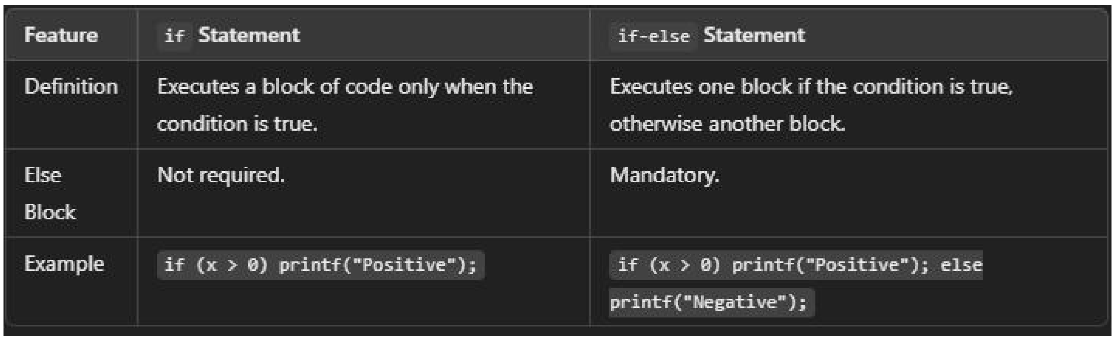
```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num > 0) {
        printf("The number is positive.\n");
    } else {
        printf("The number is negative.\n");
    }

    return 0;
}
```

29. Write a C program to find the smallest number among three numbers.



```c
#include <stdio.h>

int main() {
    int a, b, c, smallest;

    printf("Enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);

    smallest = a;

    if (b < smallest) {
        smallest = b;
    }
    if (c < smallest) {
        smallest = c;
    }

    printf("The smallest number is: %d\n", smallest);

    return 0;
}
```

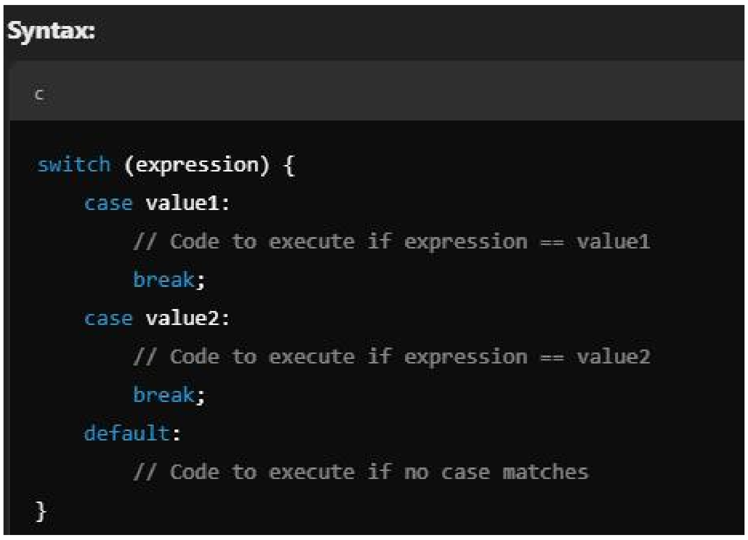
30. Explain the difference between if and if-else statements in C.



Feature	`if` Statement	`if-else` Statement
Definition	Executes a block of code only when the condition is true.	Executes one block if the condition is true, otherwise another block.
Else Block	Not required.	Mandatory.
Example	`if (x > 0) printf("Positive");`	`if (x > 0) printf("Positive"); else printf("Negative");`

31. Explain the role and functionality of the switch statement in C with an example.(9)

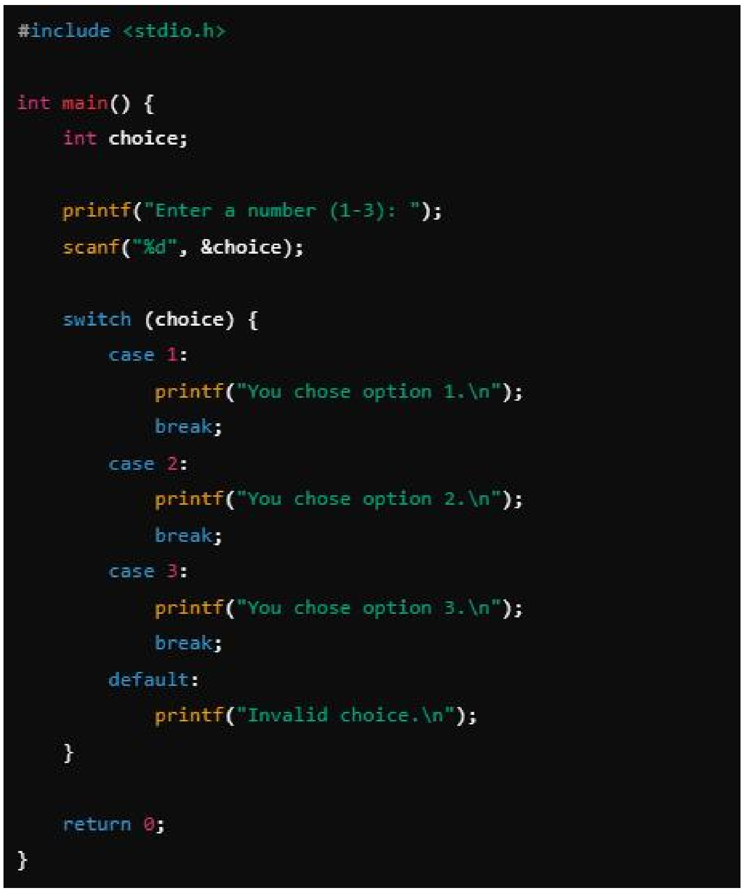
The switch statement in C is a control structure used to simplify decision-making based on the value of an expression.
It provides a more organized way to handle multiple conditions than a series of if...else statements.



Syntax:

```c
switch (expression) {
    case value1:
        // Code to execute if expression == value1
        break;
    case value2:
        // Code to execute if expression == value2
        break;
    default:
        // Code to execute if no case matches
}
```

Eg)

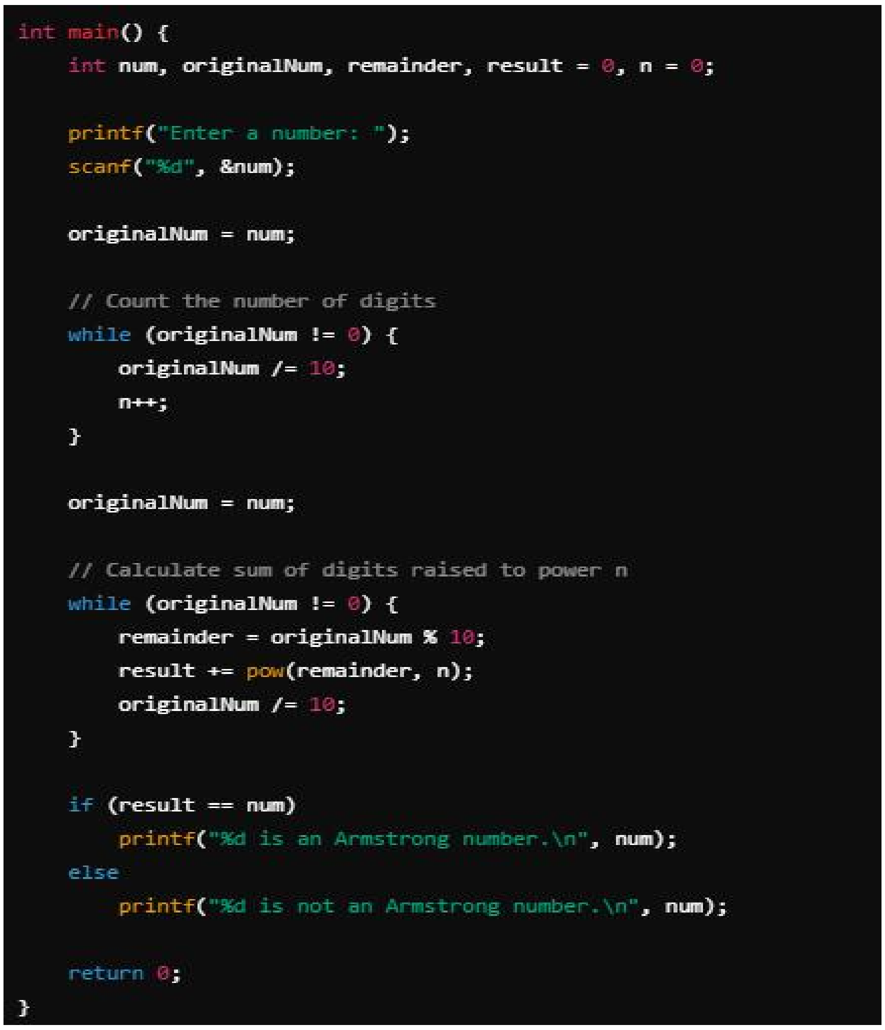


```c
#include <stdio.h>

int main() {
    int choice;

    printf("Enter a number (1-3): ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("You chose option 1.\n");
            break;
        case 2:
            printf("You chose option 2.\n");
            break;
        case 3:
            printf("You chose option 3.\n");
            break;
        default:
            printf("Invalid choice.\n");
    }

    return 0;
}
```

Importance:

- Simplifies decision-making for multiple cases.
- Improves performance over multiple if-else checks.
- Works with integer and character values.

32. Write a C program to check whether a given number is an Armstrong number.(9)



```c
int main() {
    int num, originalNum, remainder, result = 0, n = 0;

    printf("Enter a number: ");
    scanf("%d", &num);

    originalNum = num;

    // Count the number of digits
    while (originalNum != 0) {
        originalNum /= 10;
        n++;
    }

    originalNum = num;

    // Calculate sum of digits raised to power n
    while (originalNum != 0) {
        remainder = originalNum % 10;
        result += pow(remainder, n);
        originalNum /= 10;
    }

    if (result == num)
        printf("%d is an Armstrong number.\n", num);
    else
        printf("%d is not an Armstrong number.\n", num);

    return 0;
}
```

33. Write a menu-driven program to perform addition, subtraction, multiplication, and division.(9)

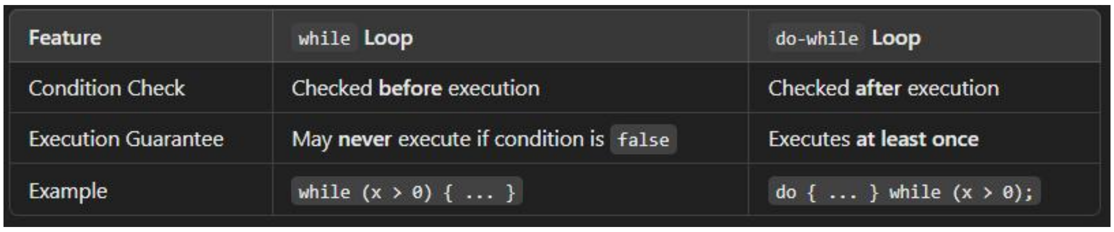
```c
#include <stdio.h>

int main() {
   int choice;
   float num1, num2, result;

   while (1) {
      printf("\nMenu:\n");
      printf("1. Addition\n2. Subtraction\n3. Multiplication\n4. Division\n5.
Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      if (choice == 5) {
            printf("Exiting program.\n");
            break;
        }

        printf("Enter two numbers: ");
        scanf("%f %f", &num1, &num2);

        switch (choice) {
            case 1:
                result = num1 + num2;
                printf("Result: %.2f\n", result);
                break;
            case 2:
                result = num1 - num2;
                printf("Result: %.2f\n", result);
                break;
            case 3:
                result = num1 * num2;
                printf("Result: %.2f\n", result);
                break;
            case 4:
                if (num2 != 0)
                    result = num1 / num2;
                else
                    printf("Error: Division by zero.\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

34. Can a switch statement work with float or double data types? Why or why not?(3)

No, the switch statement **cannot** work with float or double data types because:

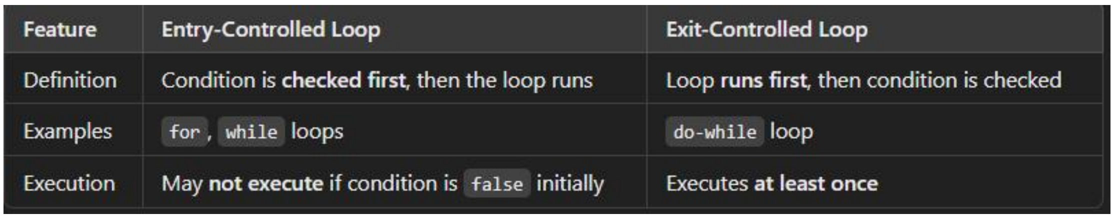
1. **Floating-point numbers are stored as approximations** (due to precision errors).
2. switch requires **discrete (exact match) case labels**, which floating-point values cannot reliably provide.

34. Differentiate between while and do-while statements.(3)

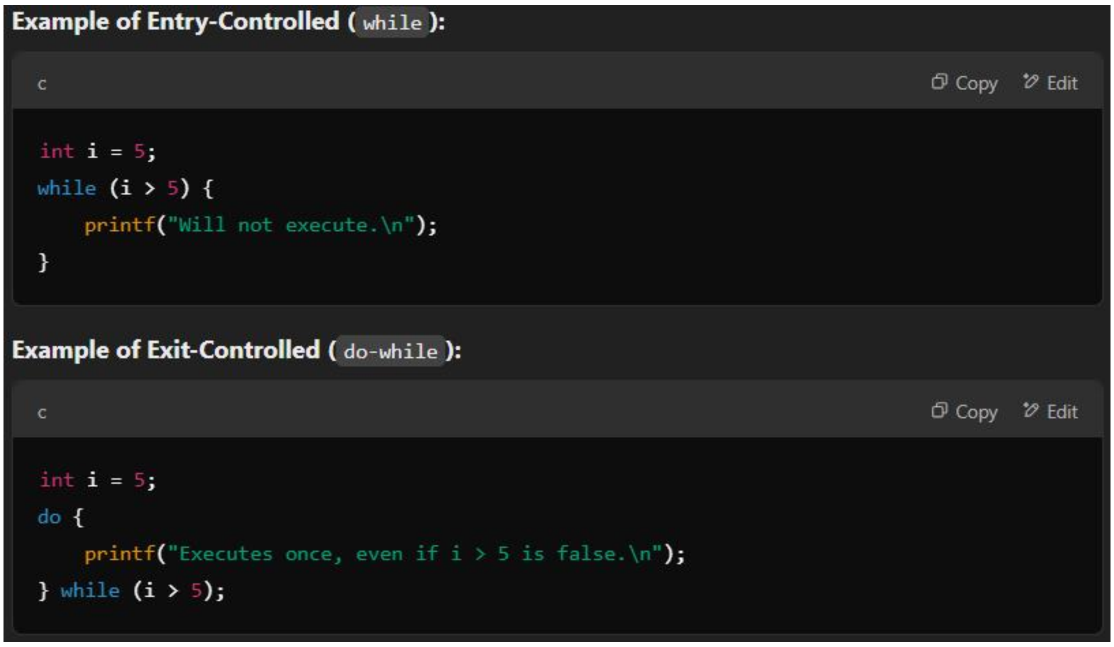


Feature	`while` Loop	`do-while` Loop
Condition Check	Checked **before** execution	Checked **after** execution
Execution Guarantee	May **never** execute if condition is `false`	Executes **at least once**
Example	`while (x > 0) { ... }`	`do { ... } while (x > 0);`

36. Compare an exit-controlled loop and an entry-controlled loop with suitable examples.(3)



Feature	Entry-Controlled Loop	Exit-Controlled Loop
Definition	Condition is **checked first**, then the loop runs	Loop **runs first**, then condition is checked
Examples	`for`, `while` loops	`do-while` loop
Execution	May **not execute** if condition is `false` initially	Executes **at least once**



Example of Entry-Controlled (`while`):

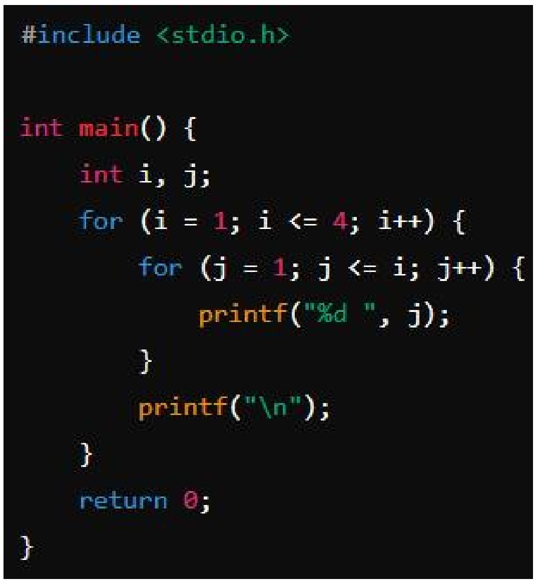
```c
int i = 5;
while (i > 5) {
    printf("Will not execute.\n");
}
```

Example of Exit-Controlled (`do-while`):

```c
int i = 5;
do {
    printf("Executes once, even if i > 5 is false.\n");
} while (i > 5);
```

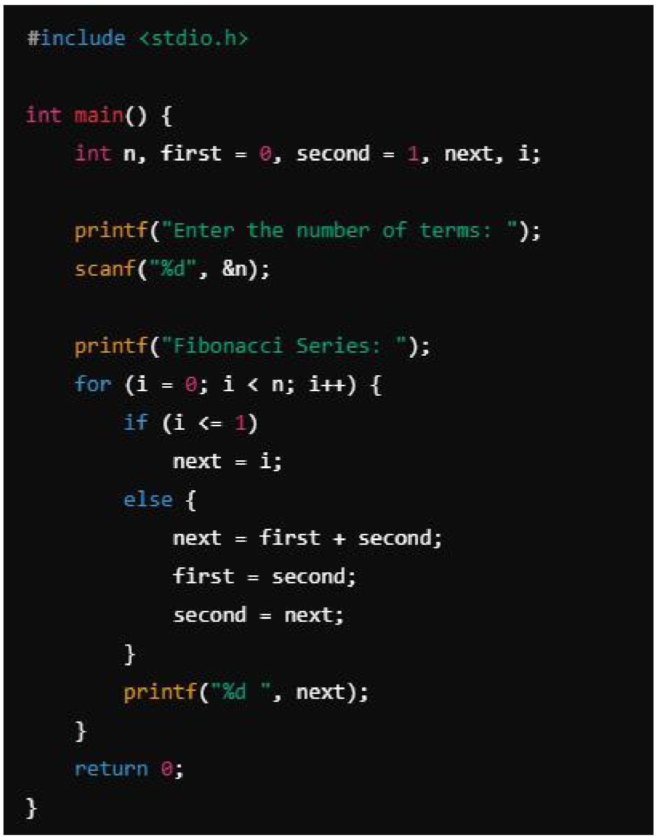
37. Write a program to generate the following pattern:(9)

```
1
1 2
1 2 3
1 2 3 4
```



```c
#include <stdio.h>

int main() {
    int i, j;
    for (i = 1; i <= 4; i++) {
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

37. a) Write a C program to display the Fibonacci series up to a given number of terms. (7 Marks)

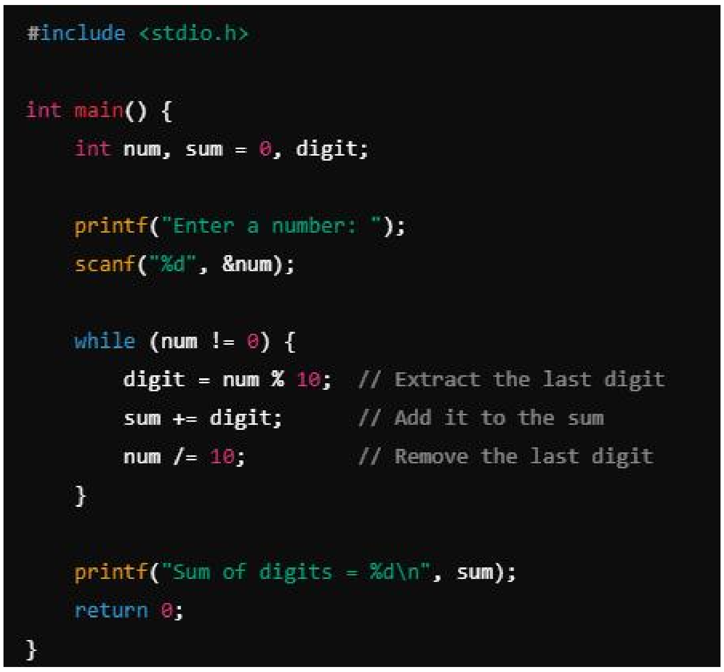


```c
#include <stdio.h>

int main() {
    int n, first = 0, second = 1, next, i;

    printf("Enter the number of terms: ");
    scanf("%d", &n);

    printf("Fibonacci Series: ");
    for (i = 0; i < n; i++) {
        if (i <= 1)
            next = i;
        else {
            next = first + second;
            first = second;
            second = next;
        }
        printf("%d ", next);
    }
    return 0;
}
```

b) Why is a loop necessary for generating the Fibonacci sequence in a program? (2 Marks)

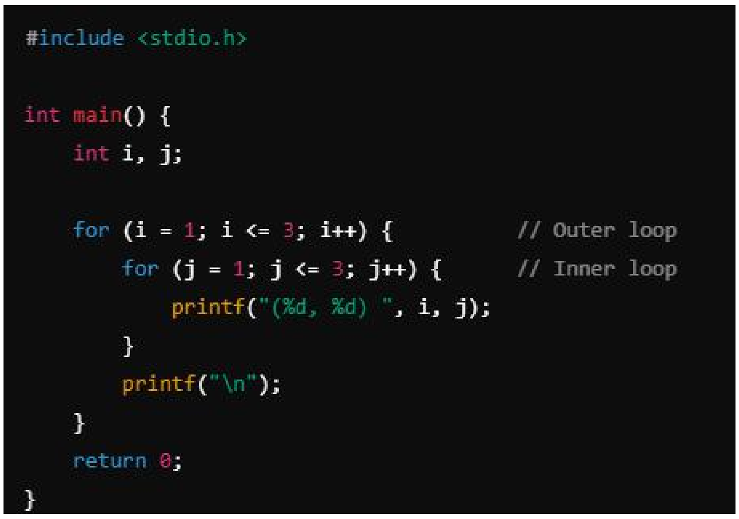
A loop is necessary because each new term in the Fibonacci sequence is the sum of the two preceding terms. This repetitive calculation is efficiently handled by loops, avoiding redundant code.

38. Write a program to find the sum of digits of a given number.(3)



```c
#include <stdio.h>

int main() {
    int num, sum = 0, digit;

    printf("Enter a number: ");
    scanf("%d", &num);

    while (num != 0) {
        digit = num % 10;  // Extract the last digit
        sum += digit;      // Add it to the sum
        num /= 10;         // Remove the last digit
    }

    printf("Sum of digits = %d\n", sum);
    return 0;
}
```

40. Demonstrate the use of nested loops in C with an example.(3)

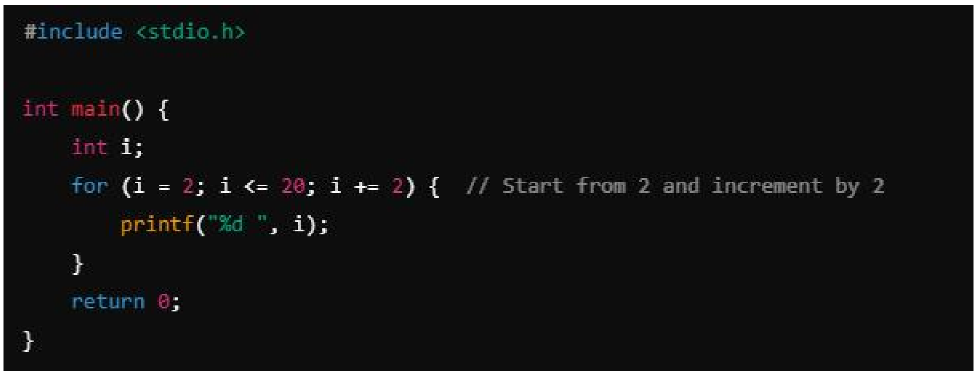


```c
#include <stdio.h>

int main() {
    int i, j;

    for (i = 1; i <= 3; i++) {          // Outer loop
        for (j = 1; j <= 3; j++) {      // Inner loop
            printf("(%d, %d) ", i, j);
        }
        printf("\n");
    }
    return 0;
}
```

Explanation:

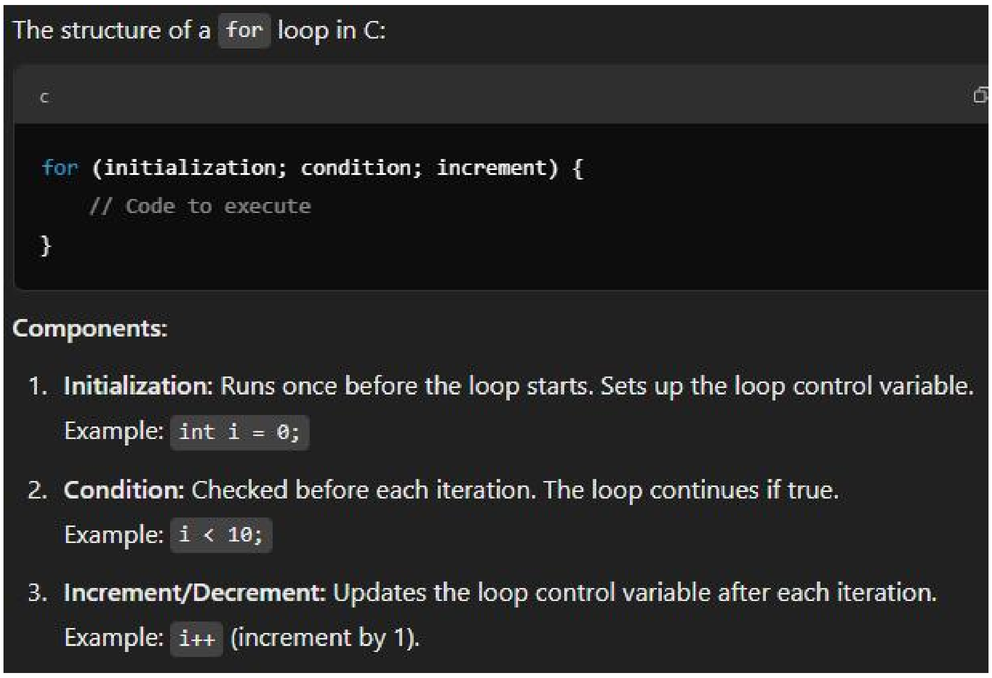
- The outer loop runs 3 times.
- For each iteration of the outer loop, the inner loop runs 3 times.
- This results in a grid of coordinate pairs.

41. Write a for loop to print all even numbers between 1 and 20.(3)



```c
#include <stdio.h>

int main() {
    int i;
    for (i = 2; i <= 20; i += 2) {  // Start from 2 and increment by 2
        printf("%d ", i);
    }
    return 0;
}
```

42. Explain the structure of a for loop and the significance of its three components.



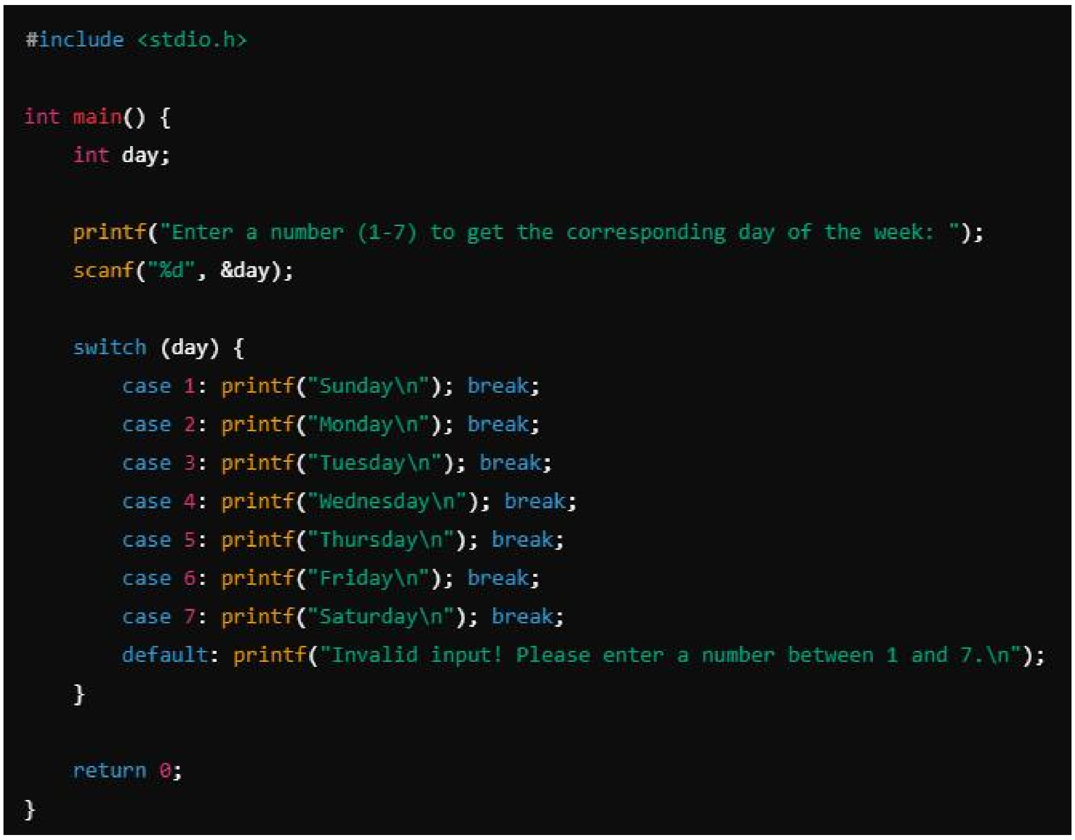
The structure of a `for` loop in C:

```c
for (initialization; condition; increment) {
    // Code to execute
}
```

Components:

1. **Initialization:** Runs once before the loop starts. Sets up the loop control variable.
   Example: `int i = 0;`
2. **Condition:** Checked before each iteration. The loop continues if true.
   Example: `i < 10;`
3. **Increment/Decrement:** Updates the loop control variable after each iteration.
   Example: `i++` (increment by 1).

43. a) Write a C program to display the corresponding day of the week using switch statements. (5 Marks)



```c
#include <stdio.h>

int main() {
    int day;

    printf("Enter a number (1-7) to get the corresponding day of the week: ");
    scanf("%d", &day);

    switch (day) {
        case 1: printf("Sunday\n"); break;
        case 2: printf("Monday\n"); break;
        case 3: printf("Tuesday\n"); break;
        case 4: printf("Wednesday\n"); break;
        case 5: printf("Thursday\n"); break;
        case 6: printf("Friday\n"); break;
        case 7: printf("Saturday\n"); break;
        default: printf("Invalid input! Please enter a number between 1 and 7.\n");
    }

    return 0;
}
```

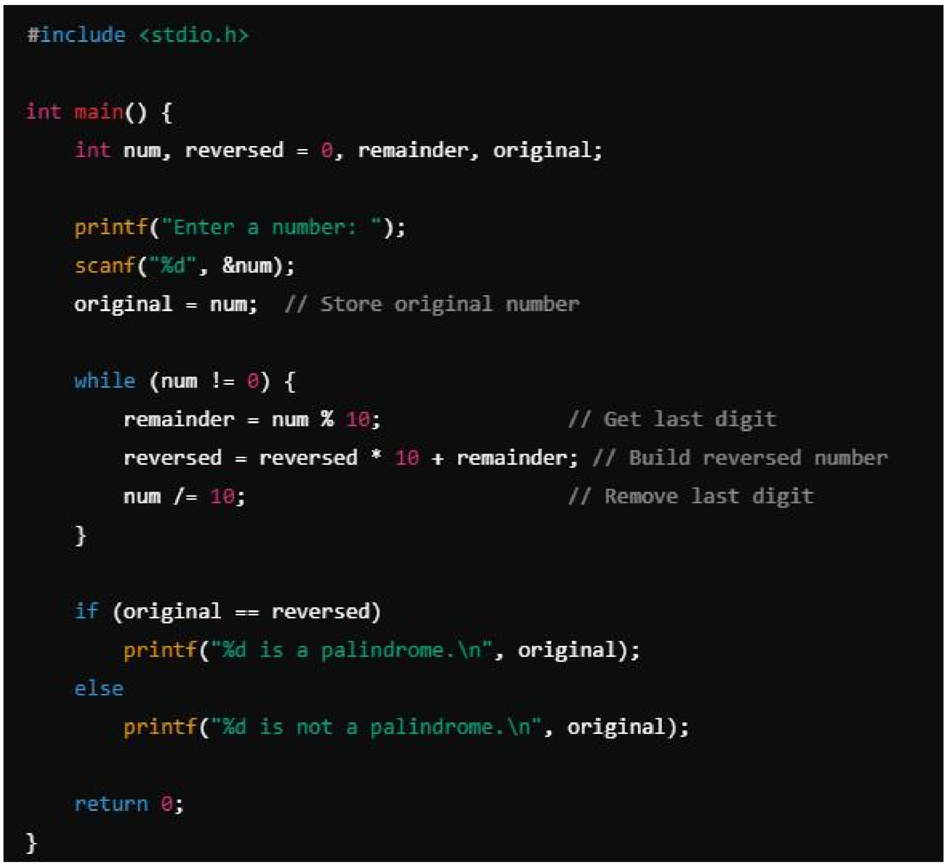
b) Explain how the break statement works in a switch statement. (4 Marks)

· The break statement **terminates the current case** in a switch block.

· Without break, the program will continue to execute the subsequent cases even if the condition is met (this is called **fall-through**).

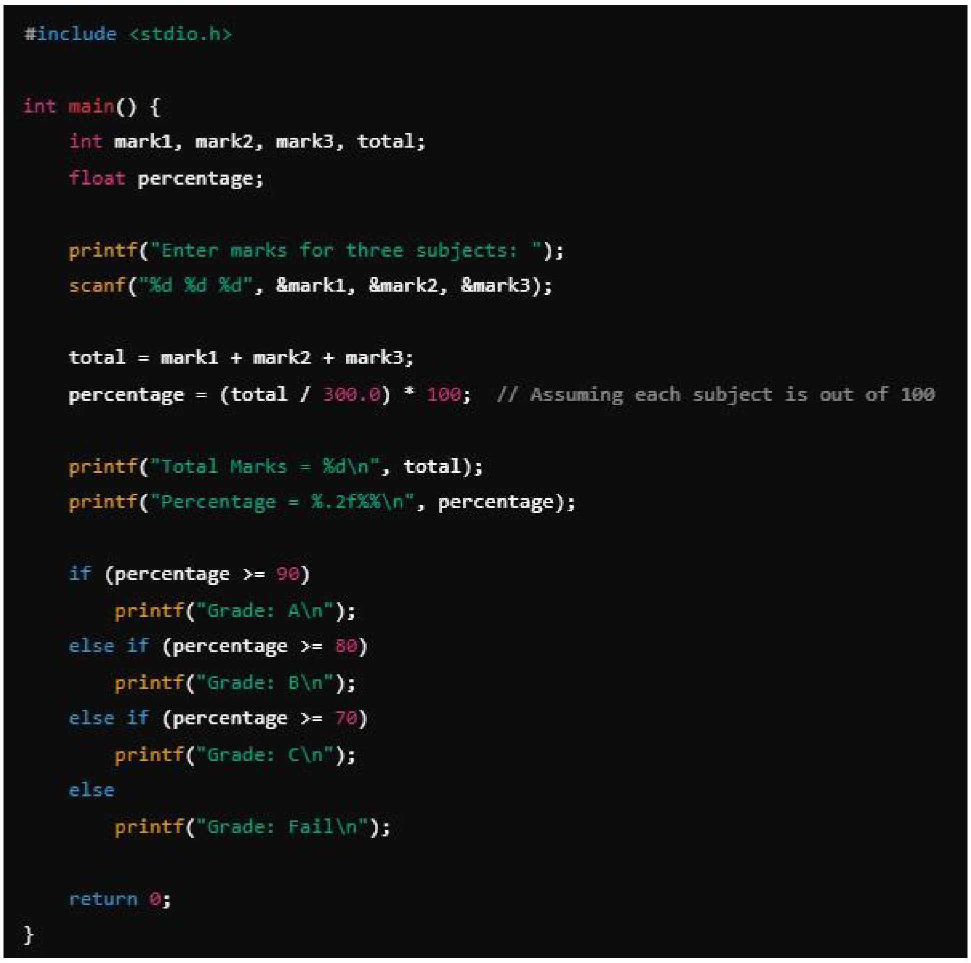
· It helps prevent unintended code execution after the desired case.

44. Write a C program to check whether a number is a palindrome.(9)



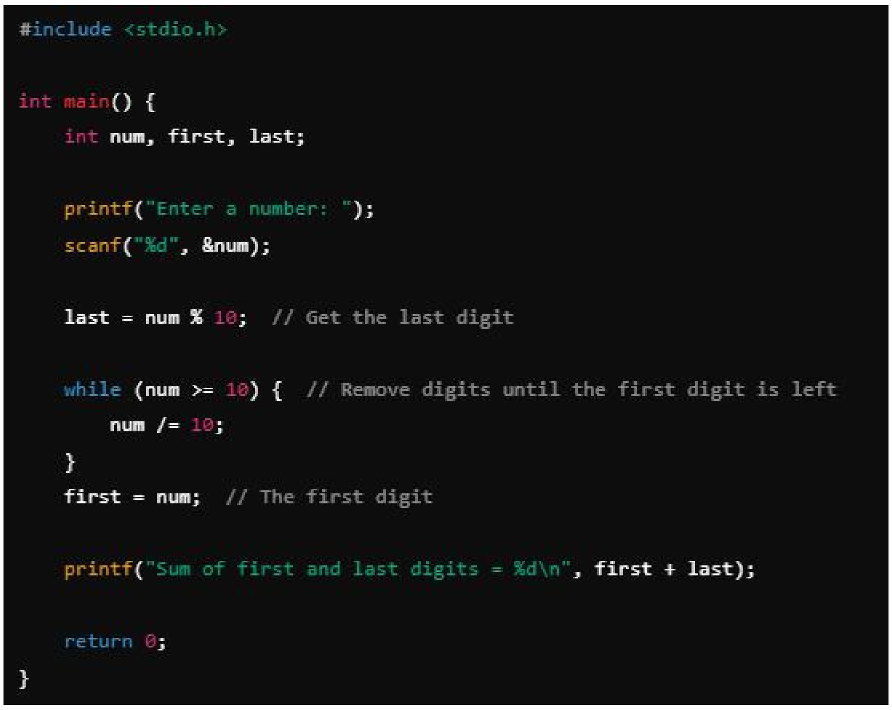
```c
#include <stdio.h>

int main() {
    int num, reversed = 0, remainder, original;

    printf("Enter a number: ");
    scanf("%d", &num);
    original = num;  // Store original number

    while (num != 0) {
        remainder = num % 10;                // Get last digit
        reversed = reversed * 10 + remainder; // Build reversed number
        num /= 10;                           // Remove last digit
    }

    if (original == reversed)
        printf("%d is a palindrome.\n", original);
    else
        printf("%d is not a palindrome.\n", original);

    return 0;
}
```

45. Write a program to input marks for three subjects, calculate the total percentage, and display grades according to the following criteria:(9)
Percentage >= 90: A grade
Percentage >= 80: B grade
Percentage >= 70: C grade
Percentage < 70: Fail.



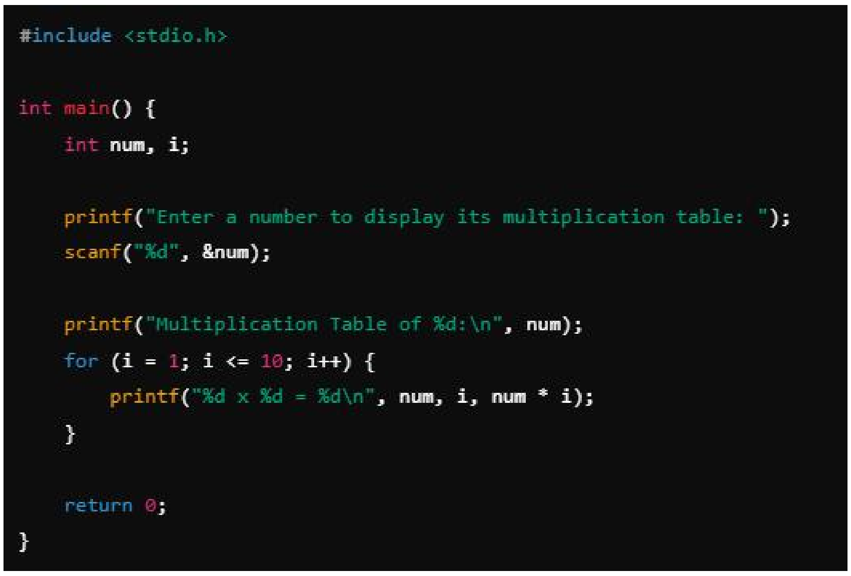
```c
#include <stdio.h>

int main() {
    int mark1, mark2, mark3, total;
    float percentage;

    printf("Enter marks for three subjects: ");
    scanf("%d %d %d", &mark1, &mark2, &mark3);

    total = mark1 + mark2 + mark3;
    percentage = (total / 300.0) * 100;  // Assuming each subject is out of 100

    printf("Total Marks = %d\n", total);
    printf("Percentage = %.2f%%\n", percentage);

    if (percentage >= 90)
        printf("Grade: A\n");
    else if (percentage >= 80)
        printf("Grade: B\n");
    else if (percentage >= 70)
        printf("Grade: C\n");
    else
        printf("Grade: Fail\n");

    return 0;
}
```

46. Write a C program to find the sum of the first and last digits of a number.(3)



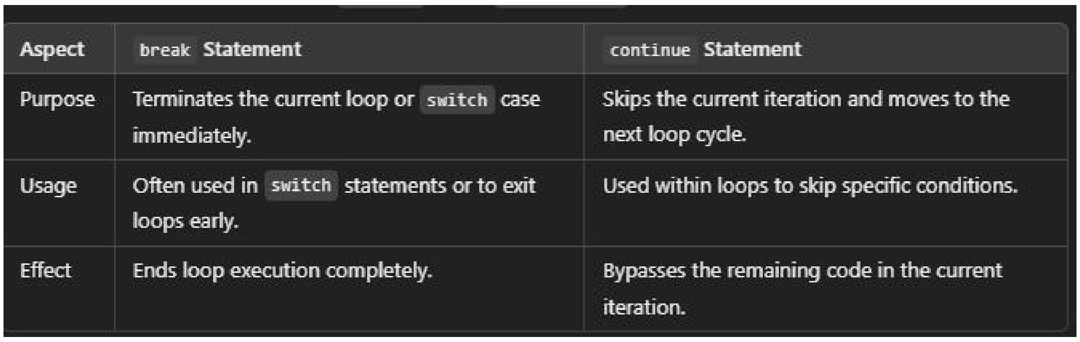
```c
#include <stdio.h>

int main() {
    int num, first, last;

    printf("Enter a number: ");
    scanf("%d", &num);

    last = num % 10;  // Get the last digit

    while (num >= 10) {  // Remove digits until the first digit is left
        num /= 10;
    }
    first = num;  // The first digit

    printf("Sum of first and last digits = %d\n", first + last);

    return 0;
}
```

47. Write a C program to display a multiplication table of a given number.(3)

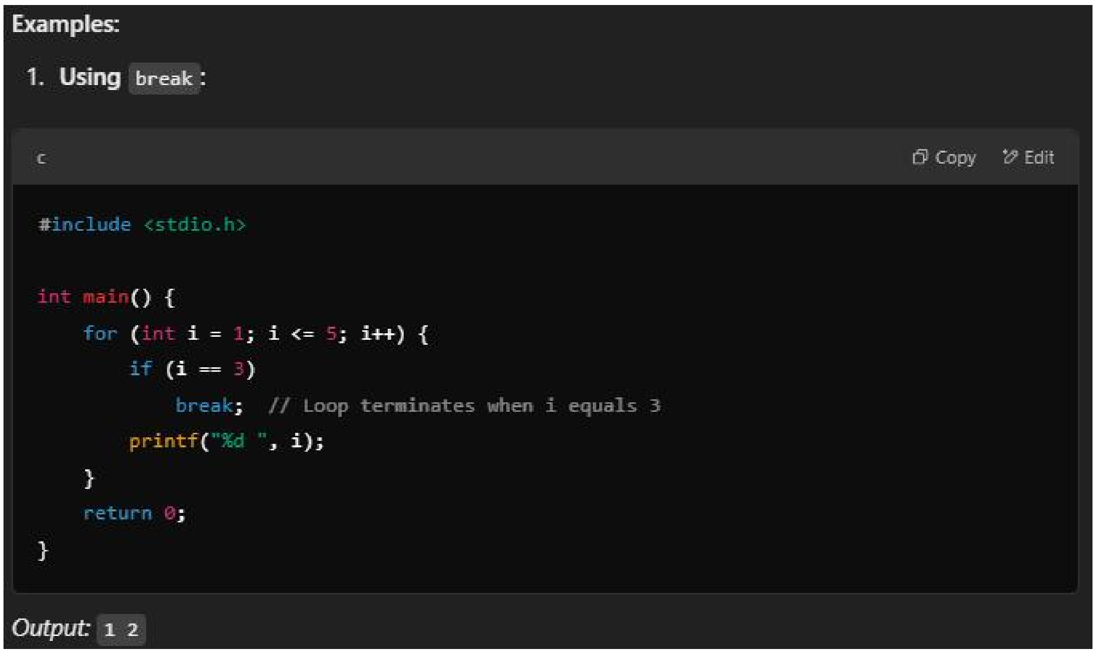


```c
#include <stdio.h>

int main() {
    int num, i;

    printf("Enter a number to display its multiplication table: ");
    scanf("%d", &num);

    printf("Multiplication Table of %d:\n", num);
    for (i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", num, i, num * i);
    }

    return 0;
}
```

48. Differentiate between break and continue statements with examples.(3)



Aspect	`break` Statement	`continue` Statement
Purpose	Terminates the current loop or `switch` case immediately.	Skips the current iteration and moves to the next loop cycle.
Usage	Often used in `switch` statements or to exit loops early.	Used within loops to skip specific conditions.
Effect	Ends loop execution completely.	Bypasses the remaining code in the current iteration.

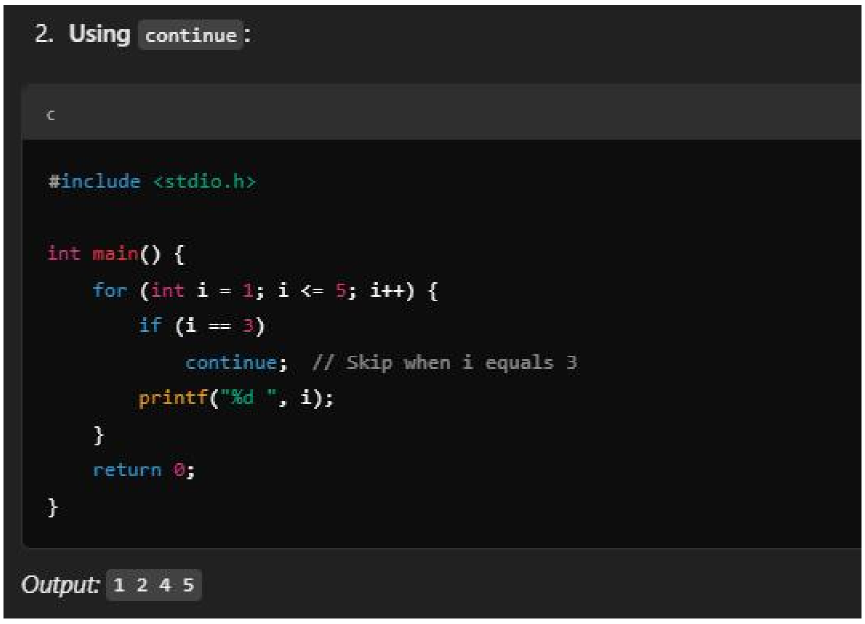


Examples:

1. **Using `break`:**

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3)
            break;  // Loop terminates when i equals 3
        printf("%d ", i);
    }
    return 0;
}
```

Output: `1 2`



2. **Using** `continue`:

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3)
            continue;  // Skip when i equals 3
        printf("%d ", i);
    }
    return 0;
}
```

Output: `1 2 4 5`