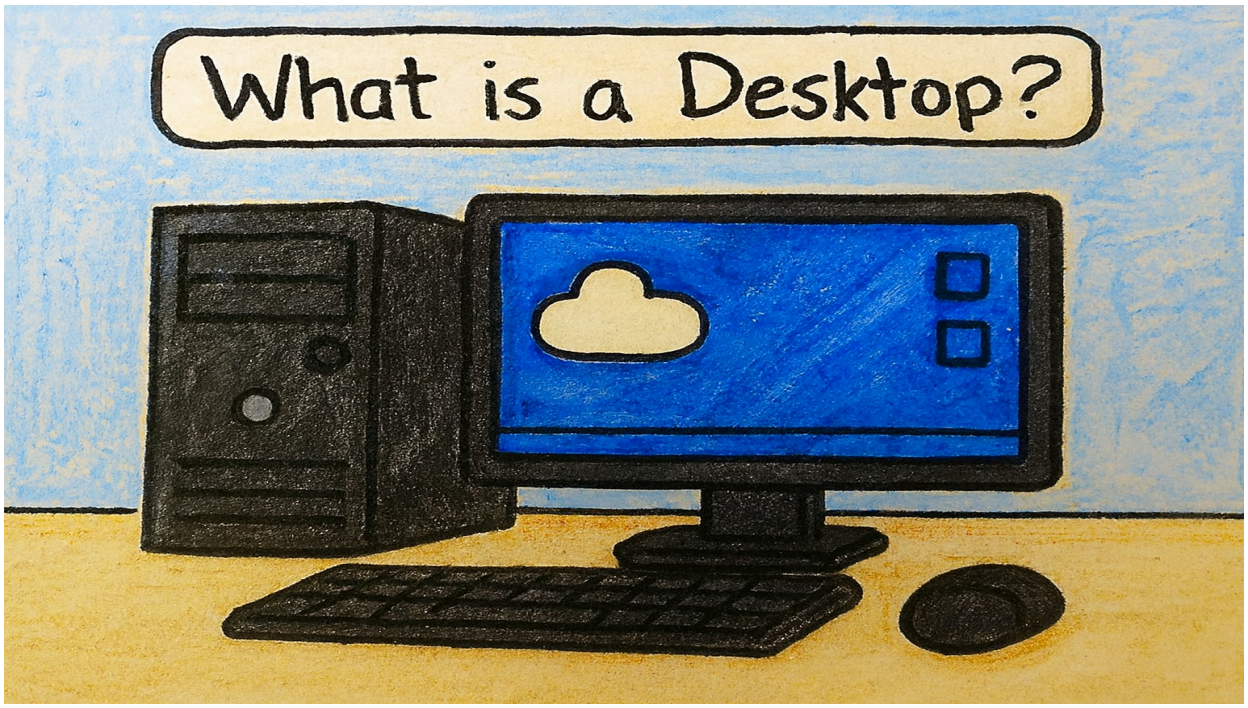


Class: 12th

Subject: Computer

Chapter 11: DECISION CONSTRUCTS

📍 Important MCQs:

1. What is the main purpose of control structures in C?

(a) To store data

(b) To increase program speed

(c) To control the flow of program execution ✅

(d) To remove syntax errors

2. How many basic types of control structures are used in C programs?

(a) Two

(b) Three ✅

(c) Four

(d) Five

3. Which control structure is known as the default flow of execution?

(a) Selection

(b) Repetition

(c) Sequence ✅

(d) Iteration

4. In a sequence control structure, instructions are executed:

(a) Randomly

(b) According to conditions

(c) Repeatedly

(d) In the order they are written ✅

5. A compound statement in C refers to:

(a) A single statement

(b) A comment block

(c) A group of statements enclosed in braces { } ✅

(d) A loop body only

6. Which of the following are the basic selection statements in C?

(a) for and while

(b) if and switch ✅

(c) do-while and for

(d) break and continue

7. The condition in an if statement evaluates to:

(a) Any integer value

(b) A character value

(c) True (1) or False (0) ✅

(d) A floating-point value

8. The statements inside a simple if block are executed when:

(a) The condition is false

(b) The program ends

(c) The condition is true ✅

(d) The condition is ignored

9. In a simple if statement, braces { } are:

(a) Always mandatory

(b) Never allowed

(c) Optional if there is only one statement ✅

(d) Required only for variables

10. What is the main function of a selection structure?

(a) To repeat statements

(b) To declare variables

(c) To choose one of multiple execution paths ✅

(d) To terminate the program

11. Which keyword is used with if to provide an alternative path of execution?

(a) then

(b) otherwise

(c) else ✅

(d) switch

12. In an if-else statement, which block is executed when the condition is false?

(a) if block

(b) main block

(c) else block ✅

(d) function block

13. How many execution paths are provided by an if-else statement?

(a) One

(b) Two ✅

(c) Three

(d) Unlimited

14. Why are braces { } used in the body of an if statement?

(a) To end the program

(b) To improve speed

(c) To represent a compound statement with multiple statements ✅

(d) To declare variables

15. What error message appears if braces are omitted from a compound if block?

(a) Syntax error

(b) Undefined reference

(c) Illegal else without matching if ✅

(d) Missing semicolon

16. If braces are omitted in the else part containing multiple statements, then:

(a) All statements execute conditionally

(b) No statement executes

(c) Only the first statement is treated as part of else ✅

(d) Program does not compile

17. A nested if statement means:

(a) Multiple if statements at the same level

(b) An if statement inside another if statement ✅

(c) An if without else

(d) An if inside a loop only

18. What happens when the level of nesting in nested if statements increases?

(a) Program becomes faster

(b) Program becomes simpler

(c) Program complexity increases ✅

(d) Errors are reduced

19. In a nested if statement, once a true condition is found:

(a) All remaining conditions are also checked

(b) Only else part runs

(c) Remaining conditions are skipped ✅

(d) Program terminates

20. Why is a nested if preferred over a sequence of if statements in some cases?

(a) It increases CPU usage

(b) It tests all conditions

(c) It avoids unnecessary condition checking and saves CPU time ✅

(d) It makes the program longer

21. Why is the if-else if statement preferred over deeply nested if statements?

(a) It increases program length

(b) It makes logic more complex

(c) It simplifies program structure and improves readability ✅

(d) It removes the need for conditions

22. In an if-else if structure, the conditions are evaluated:

(a) Randomly

(b) From bottom to top

(c) Sequentially from top to bottom ✅

(d) Only once

23. What happens when a true condition is found in an if-else if statement?

(a) All remaining conditions are also checked

(b) Program terminates

(c) The corresponding block executes and the rest are skipped ✅

(d) Control moves to else

24. If all conditions in an if-else if statement are false, then:

(a) No statement executes

(b) Only the first statement executes

(c) The statement following the final else executes ✅

(d) Program gives an error

25. Which logical operator is used to combine two or more conditions that must all be true?

(a) ||

(b) !

(c) && ✅

(d) ==

26. Which of the following logical operators represents OR in C?

(a) &&

(b) |

(c) || ✅

(d) !

27. The switch statement is mainly used when selection is based on:

(a) Floating-point values

(b) Logical expressions

(c) The value of a variable or expression ✅

(d) Complex conditions only

28. Which data type is NOT allowed in a switch expression?

(a) int

(b) char

(c) float ✅

(d) integer constant

29. What is the function of the break statement in a switch case?

(a) To terminate the program

(b) To skip the current case

(c) To exit the switch statement ✅

(d) To execute the default case

30. What happens if all break statements are omitted in a switch statement?

(a) Only one case executes

(b) Program gives an error

(c) Control exits the switch immediately

(d) All statements from the matched case to the end execute sequentially ✅

31. The conditional operator in C is also known as:

(a) Binary operator

(b) Unary operator

(c) Ternary operator ✅

(d) Logical operator

32. How many operands are required by the conditional operator?

(a) One

(b) Two

(c) Three ✅

(d) Four

33. Which of the following is the correct general form of the conditional operator?

(a) condition : true : false

(b) condition ? true : false ✅

(c) condition ? false : true

(d) condition ; true : false

34. The conditional operator is mainly used as an alternative to:

(a) switch statement

(b) loop statement

(c) if-else statement ✅

(d) break statement

35. In a conditional operator, which statement executes when the expression is false?

(a) True-case statement

(b) First operand

(c) False-case statement ✅

(d) No statement

36. In the coordinate plane case study, where does a point lie if x = 0 and y ≠ 0?

(a) On x-axis

(b) On y-axis ✅

(c) In first quadrant

(d) At origin

37. A point lies at the origin when:

(a) x > 0, y > 0

(b) x = 0, y ≠ 0

(c) x ≠ 0, y = 0

(d) x = 0, y = 0 ✅

38. If x > 0 and y > 0, the point lies in which quadrant?

(a) Second quadrant

(b) Third quadrant

(c) Fourth quadrant

(d) First quadrant ✅

39. In which quadrant does a point lie if x < 0 and y < 0?

(a) First quadrant

(b) Second quadrant

(c) Third quadrant ✅

(d) Fourth quadrant

40. If x > 0 and y < 0, the point lies in:

(a) First quadrant

(b) Second quadrant

(c) Third quadrant

(d) Fourth quadrant ✅

📍 Important Short Questions:

1. What are control structures in C?

Answer:

👉 Control structures are statements used to control the flow of execution of a program. They decide the order in which instructions are executed.

2. Name the three basic control structures used in C.

Answer:

👉 The three basic control structures used in C are:

• Sequence

• Selection

• Repetition (Loop)

3. What is meant by sequence (default) control structure?

Answer:

👉 Sequence control structure is the default flow of execution in which program statements are executed one after another in the order they are written.

4. What is a compound statement?

Answer:

👉 A compound statement is a group of two or more statements enclosed within braces { } and treated as a single block of code.

Example:

```
{
  statement1;
  statement2;
}
```

5. What is a simple if statement?

Answer:

👉 A simple if statement is a decision-making statement that executes a statement or a block of statements only when a given condition evaluates to true.

6. What is an if-else statement in C?

Answer:

👉 An if-else statement is a decision-making statement in C that allows the program to choose between two paths of

execution. The if block executes when the condition is true, and the else block executes when the condition is false.

Example:

C

```
if (x >= 0)
  printf("Positive");
else
  printf("Negative");
```

7. When is the else block executed in an if-else statement?

Answer:

👉 The else block is executed when the condition in the if statement evaluates to false.

Example:

C

```
if (x > 0)
  printf("Positive");
else
```

```
  printf("Zero or Negative"); // executed if x <= 0
```

8. Why are braces { } used in the if block of an if-else statement?

Answer:

👉 Braces { } are used to group multiple statements into a single block so that all statements are executed together when the condition is true.

Without braces, only the first statement is considered part of the if block.

9. What is a nested if statement?

Answer:

👉 A nested if statement is an if statement placed inside another if statement. It allows multiple levels of decision-making based on different conditions.

Example:

```
if (x > 0)

{

  if (x % 2 == 0)
```

```
    printf("Positive Even");
  else
    printf("Positive Odd");
}
```

10. Write one difference between a nested if statement and a sequence of if statements.

Answer:

👉 In a nested if statement, once a true condition is found, the remaining conditions are skipped, saving CPU time.

👉 In a sequence of if statements, all conditions are checked regardless of the result of the first true condition.

11. What is an if-else if statement in C?

Answer:

👉 An if-else if statement is used when there are multiple alternatives. It allows the program to choose one path from many based on different conditions.

Example:

```
if (num > 0)
  printf("Positive");
else if (num < 0)
  printf("Negative");
else
  printf("Zero");
```

12. How are multiple conditions evaluated in an if-else if statement?

Answer:

👉 Conditions in an if-else if statement are evaluated sequentially from top to bottom. As soon as a true condition is found, its block executes, and the rest are skipped. If all conditions are false, the final else block executes.

13. What are logical operators in C? Name them.

Answer:

👉 Logical operators are used to combine or modify conditions in decision-making statements.

They are:

• AND (&&) ✅

• OR (||) ✅

• NOT (!) ✅

14. What is a switch statement in C?

Answer:

👉 A switch statement is a conditional statement that allows selection of one path from many alternatives based on the value of an expression.

Example:

```
switch(ch)
{
  case 'a': printf("Vowel"); break;
  default: printf("Not a vowel");
}
```

15. What is the purpose of the break statement in a switch case?

Answer:

👉 The break statement is used to exit the switch after executing a case. Without break, the program continues to execute all following cases sequentially until the end of the switch.

📍 Important long questions:

🌟 Q.1: Explain the if statement in C with syntax, example, and flow of execution

❖ Definition of Simple if Statement

👉 The if statement in C is a decision-making statement that allows a program to execute a block of code only if a specific condition is true.

- It is the simplest form of decision-making in C.
- If the condition evaluates to false, the statements inside the if block are skipped.
- It is mainly used to control the flow of the program based on a certain condition.

Key Points:

- The condition must be a logical or relational expression (e.g., >, <, ==).
- The if statement can control one statement or a block of statements.
- A block of statements is enclosed in braces { }.

Syntax of Simple if Statement

```
if (condition)

{

   statement1;

   statement2;

   ...

}
```

- **condition:** An expression that evaluates to true (1) or false (0).
- **statement(s):** Code that will be executed only if the condition is true.

Notes on Braces { }:

- If there is only one statement, braces are optional.

- If there are multiple statements, braces are required to treat them as a single block.

How the Condition is Evaluated

1. The program checks the condition inside the parentheses.

2. If the condition is true:

- The program executes the statements inside the if block.

3. If the condition is false:

- The program skips the if block and continues with the rest of the program.

Example of Condition:

num > 0 // true if num is greater than 0, false otherwise

Short Program Example

Problem: Check whether a number entered by the user is positive.

```
#include <stdio.h>

void main()
```

```
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);   // input a number
    if (num > 0)         // check if number is positive
    {
        printf("The number is positive.\n");
    }
}
```

Explanation of the Program:

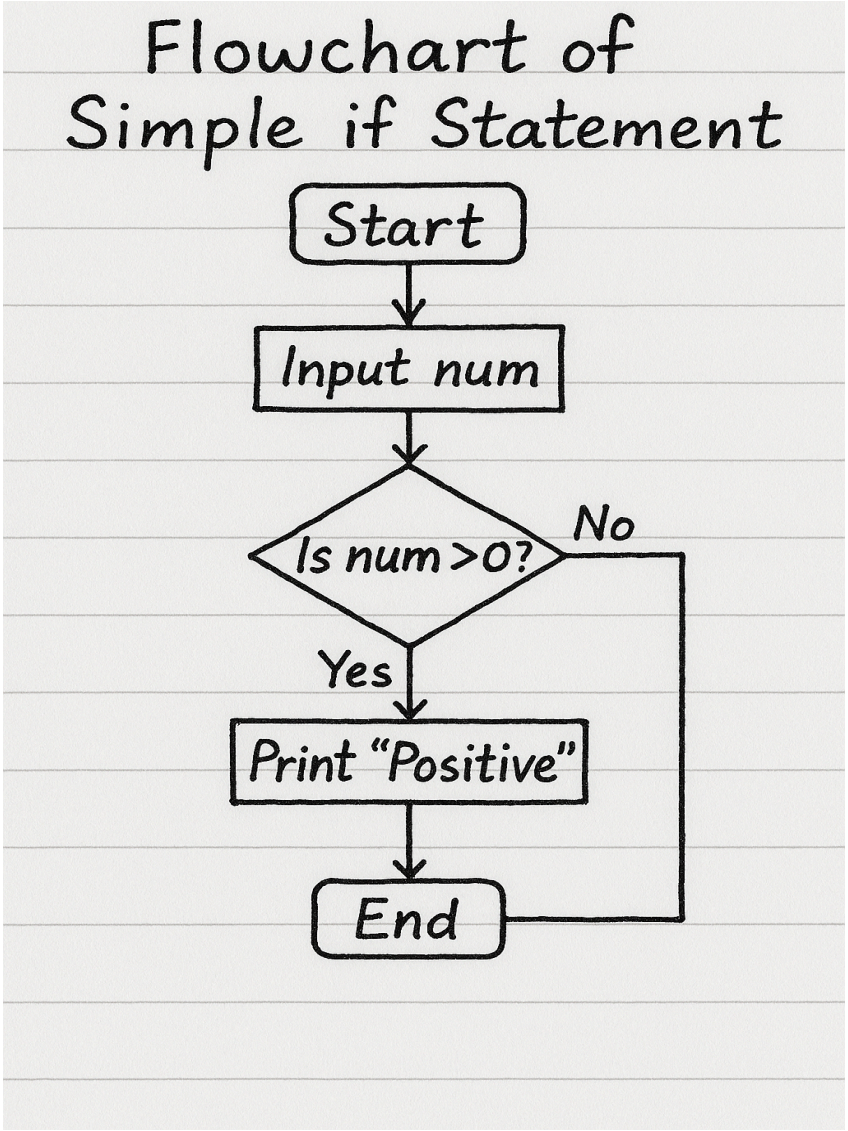
1. The program starts and asks the user to input a number.
2. The condition num > 0 is evaluated:
    - If the number is greatuer than 0 ⟶ the statement inside if block is executed ⟶ prints “The number is positive.”
    - If the number is less than or equal to 0 ⟶ the statement inside if block is skipped ⟶ nothing is printed.

Important:

This program does not print anything if the number is zero or negative.

This shows the selective execution of the if statement.

Flowchart of Simple if Statement



Explanation of Flowchart:

- Program starts and the user inputs a number.
- The condition num > 0 is checked.
- If the condition is true ⟶ the program executes the print statement.

1. If the condition is false ⟶ the program skips the print statement.
2. Program ends after executing/skipping the if block.

Key Advantages of Using if Statement

- Allows conditional execution of code.
- Helps in controlling program flow based on user input or other conditions.
- Reduces unnecessary execution of statements.

🌟 Q.2: Explain the if-else statement in C with syntax, example, and flowchart

❖ Definition of if-else Statement

👉 The if-else statement in C is a decision-making statement that allows the program to choose between two alternatives based on a condition.

- If the condition is true, the program executes the if block.
- If the condition is false, the program executes the else block.

Use of if-else statement:

- It is used when two mutually exclusive paths exist in a program.
- Ensures one block of code is always executed depending on the condition.

Syntax of if-else Statement

```
if (condition)
{
    // statements executed if condition is true
}
else
{
    // statements executed if condition is false
}
```

Explanation of Syntax:

1. condition: A relational or logical expression that evaluates to true (1) or false (0).

2. if block: Executed when the condition is true.

3. else block: Executed when the condition is false.

4. Braces {}:

- Required if more than one statement is inside a block.
- Optional for a single statement, but it is good practice to always use braces for clarity and to avoid errors.

Program Example

Problem: Calculate the square root of a number if it is non-negative; otherwise, display an error message.

```
#include <stdio.h>

#include <math.h>

void main()

{

   double x, square_root;
```

```
    printf("Enter a number: ");
    scanf("%lf", &x);
    if (x >= 0)      // condition to check non-negative number
    {
        square_root = sqrt(x);
        printf("Square root of %lf is %lf\n", x, square_root);
    }
    else
    {
        printf("Square root cannot be calculated for negative numbers.\n");
    }
}
```

Explanation of Program:

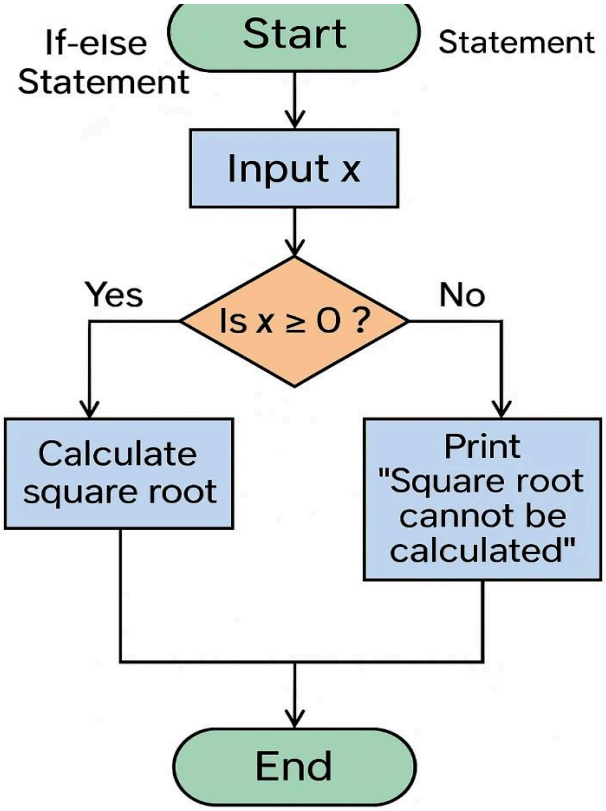
1. The program inputs a number from the user.
2. Condition x >= 0 is evaluated:

- **True:** executes the if block ⟶ calculates and prints the square root.
- **False:** executes the else block ⟶ prints error message.

3. The use of braces {} ensures that multiple statements in the if block are treated as one logical unit.

4. If braces were omitted and multiple statements were present, compilation errors such as Illegal else without matching if may occur.

Flowchart of if-else Statement



Flowchart Explanation:

1. Program starts and asks for input x.
2. Condition x >= 0 is checked.
3. If true, square root is calculated and printed.
4. If false, an error message is printed.
5. Program ends after executing the appropriate block.

Key Points:

- Ensures one of the two blocks is executed depending on the condition.
- Prevents unnecessary execution of irrelevant statements.
- Using braces {} avoids syntax errors and improves readability.

🌟 Q.3: Explain Nested if Statement in C. Give an Example and Compare it with Sequence of if Statements

❖ Definition of Nested if Statement

👉 A nested if statement is an if statement placed inside another if statement.

- It is used when a program needs to make decisions based on multiple conditions, where the second decision depends on the first.
- Nesting can be done to any level, but higher levels increase the complexity of the program.

How it works:

1. The outer if condition is evaluated first.
2. If the outer condition is true, the inner if is evaluated.
3. If the outer condition is false, the inner if is skipped.
4. Else blocks are optional and can be used with either outer or inner if statements.

Syntax of Nested if Statement

```
if (condition1)        // outer if
{
   if (condition2)     // inner if
   {
      // statements executed if both conditions are true
   }
```

```
    else
    {
        // statements executed if outer is true but inner is false
    }
}
else
{
    // statements executed if outer condition is false
}
```

Program Example

Problem: Determine whether a number is positive, negative, or zero using nested if statements.

```
#include <stdio.h>
void main()
{
    int num;
```

```
printf("Enter a number: ");
scanf("%d", &num);
if (num != 0)          // outer if
{
    if (num > 0)       // inner if
    {
        printf("The number is positive.\n");
    }
    else
    {
        printf("The number is negative.\n");
    }
}
else
{
    printf("The number is zero.\n");
```

```
    }
}
```

Explanation of Program:

1. The outer if checks whether the number is not zero (num != 0).
2. If true, the inner if checks whether the number is greater than 0.
    - If true ⟶ prints “The number is positive.”
    - If false ⟶ prints “The number is negative.”

3. If the outer condition is false ⟶ prints “The number is zero.”

4. Using braces {} ensures multiple statements in each block are treated as one logical unit, preventing errors.

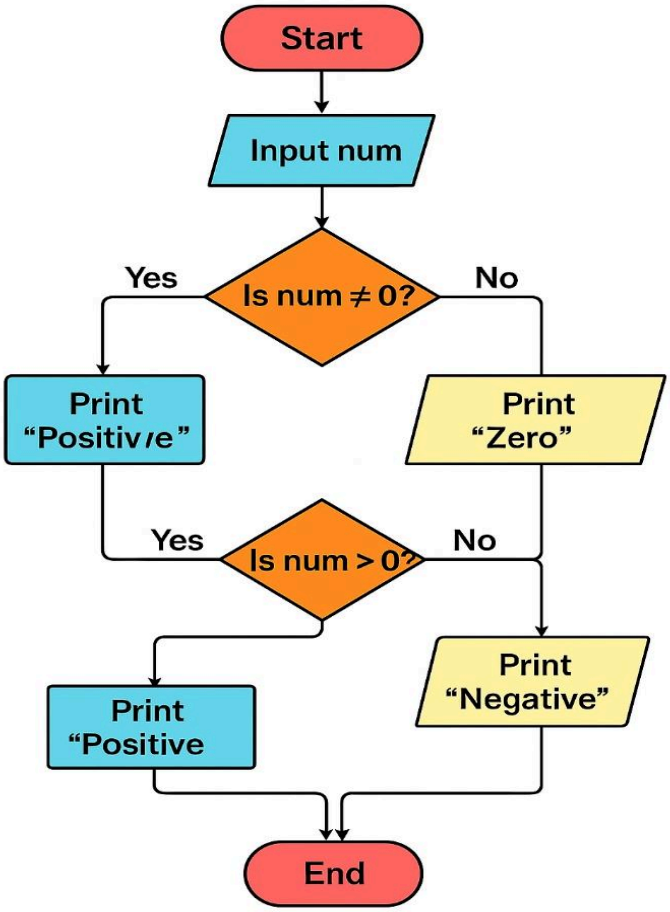
Comparison with Sequence of if Statements

- In nested if, once a true condition is found, the rest of the conditions are skipped, making it efficient.
- In sequence of if statements, all conditions are tested independently, even if the answer is already determined, which may waste CPU time.
- **Example:** If the number is 5,

- Nested if ⟶ checks num != 0 ⟶ true ⟶ checks num > 0 ⟶ true ⟶ prints positive. Done.
- Sequence of if ⟶ checks num > 0 ⟶ true ⟶ checks num < 0 ⟶ false ⟶ checks num == 0 ⟶ false. Extra checks are wasted.

- **Efficiency:** Nested if is more efficient, while sequence of if statements is simpler and easier for beginners.

Flowchart of Nested if Statement



Flowchart Explanation:

1. Program starts and inputs a number.
2. Outer condition num != 0 is checked.
3. If true ⟶ inner condition num > 0 is evaluated.
4. Depending on the inner condition ⟶ prints “positive” or “negative.”
5. If outer condition is false ⟶ prints “zero.”
6. Program ends after executing the appropriate block.

✦ Summary:

- Nested if is used for multiple dependent decisions.
- It is more efficient than sequence of if statements because unnecessary checks are avoided.
- Braces {} are important to ensure proper execution of multiple statements in each block.
- Flowchart helps to visualize the decision-making process.

🌟 Q.4: Explain the if-else if Statement in C. Include Syntax, Example, and Advantages over Nested if

❖ Definition of if-else if Statement

👉 The if-else if statement in C is a decision-making statement used to handle multiple alternatives more efficiently than nested if statements.

- It is used when a program needs to choose one path from several possible paths.
- Conditions are tested sequentially, and once a true condition is found, the corresponding block is executed, and the rest are skipped.

Use:

- Useful for more than two alternatives, where using nested if statements may become complex.

Syntax of if-else if Statement

```
if (condition1)

{

   // statements executed if condition1 is true

}

else if (condition2)

{
```

```
    // statements executed if condition2 is true
}
else if (condition3)
{
    // statements executed if condition3 is true
}
...
else
{
    // statements executed if all conditions are false
}
```

Explanation:

- Conditions are checked one by one from top to bottom.
- First true condition ⟶ executes its block.
- All remaining conditions are skipped after a true condition.

- Else block is optional; it executes when all conditions are false.

Program Example

Problem: Check whether a number is positive, negative, or zero using if-else if statement.

```
#include <stdio.h>

void main()

{

   int num;

   printf("Enter a number: ");

   scanf("%d", &num);

   if (num > 0)

   {

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

   }

   else if (num < 0)

   {
```

```
        printf("The number is negative.\n");
    }
    else
    {
        printf("The number is zero.\n");
    }
}
```

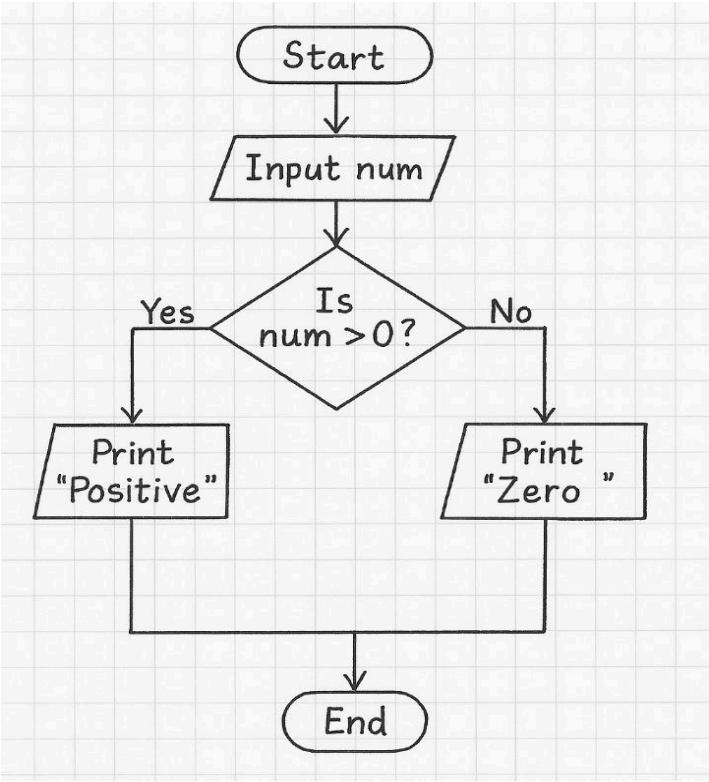
Explanation of Program:

1. Program inputs a number from the user.
2. The first condition num > 0 is evaluated:
    - If true ⟶ prints “positive” and skips remaining conditions.
3. If false ⟶ the second condition num < 0 is evaluated:
    - If true prints “negative” and skips remaining conditions.
4. If both conditions are false ⟶ the else block executes ⟶ prints “zero.”
5. This sequential evaluation simplifies the program compared to nested if.

Advantages of if-else if over Nested if

1. **Simpler and cleaner syntax:** Easy to read compared to multiple nested if statements.
2. **Efficient execution:** Once a true condition is found, remaining conditions are skipped, saving CPU time.
3. **Reduces complexity:** Avoids deep nesting, making the program more understandable.
4. **Better maintenance:** Easier to add or remove conditions without affecting the logic of other parts.

Flowchart of if-else if Statement



Flowchart Explanation:

- Program starts and inputs a number.
- Conditions are tested sequentially: num > 0, num < 0.
- First true condition executes its block.
- If no condition is true ⟶ else block executes.
- Program ends after work executing the appropriate statement.

✦ Summary:

1. The if-else if statement is ideal for multiple alternatives.
2. Evaluates conditions sequentially until a true one is found.
3. More efficient and readable than nested if.
4. Ensures only one block of code executes, avoiding unnecessary checks.

🌟 Q.5: Explain the Use of Logical Operators (AND, OR, NOT) in Decision-Making in C.

❖ Definition of Logical Operators

👉 Logical operators are used to combine or modify conditions in decision-making statements such as if, if-else, or if-else if.

Common logical operators in C:

1. **AND** (&&) – True if both conditions are true.
2. **OR** (||) – True if at least one condition is true.
3. **NOT** (!) – Reverses the truth value of a condition.

Use of Logical Operators in if Statements

- Logical operators help to test multiple conditions together.
- The result of a logical expression is either true (1) or false (0).
- These operators are especially useful when decisions depend on more than one condition.

Examples of Logical Operators

1. AND Operator (&&)

- Executes a block only if both conditions are true.

```
#include <stdio.h>

void main()

{

   int age;
```

```
    int hasID;
    printf("Enter age: ");
    scanf("%d", &age);
    printf("Do you have ID? (1 for Yes, 0 for No): ");
    scanf("%d", &hasID);
    if (age >= 18 && hasID == 1)
    {
        printf("You can vote.\n");
    }
    else
    {
        printf("You cannot vote.\n");
    }
}
```

Explanation:

- Condition age >= 18 && hasID == 1 requires both age ≥ 18 and ID = 1 to be true.
- If any one condition is false prints “You cannot vote.”

2. OR Operator (||)

- Executes a block if at least one condition is true.

```
#include <stdio.h>

void main()

{

   int day;


   printf("Enter day number (1-7): ");

   scanf("%d", &day);

   if (day == 6 || day == 7)

   {

      printf("It's a weekend.\n");

   }

   else
```

```
    {
        printf("It's a weekday.\n");
    }
}
```

Explanation:

- Condition day == 6 || day == 7 is true if day is 6 OR 7.
- Only if both are false ⟶ prints “It's a weekday.”

3. NOT Operator (!)

- Reverses the truth value of a condition.

```
#include <stdio.h>
void main()
{
    int light;
    printf("Is the light on? (1 for Yes, 0 for No): ");
    scanf("%d", &light);
    if (!light)
```

```
    {
        printf("Turn on the light.\n");
    }
    else
    {
        printf("Light is already on.\n");
    }
}
```

Explanation:

- !light is true when light is 0 (false).
- If light = 0 ⟶ prints “Turn on the light.”
- If light = 1 ⟶ prints “Light is already on.”

Truth Table for Logical Operators

- **AND (&&) ⟶ True only if both conditions are true:**
    - T && T ⟶ T
    - T && F ⟶ F
    - F && T ⟶ F
    - F && F ⟶ F

- **OR (||) ⟶ True if at least one condition is true:**
  - T || T ⟶ T
  - T || F ⟶ T
  - F || T ⟶ T
  - F || F ⟶ F
- **NOT (!) ⟶ Reverses the value:**
  - !T ⟶ F
  - !F ⟶ T

✦ **Summary:**

- Logical operators allow complex decisions in programs.
- && ensures all conditions are true, || allows any condition to be true, and ! reverses a condition.
- Using logical operators reduces the need for nested if statements, making programs more efficient and readable.

🌟 **Q.6: Explain the Switch Statement in C with Syntax, Example, and Flowchart**

❖ **Definition of Switch Statement**

👉 The switch statement in C is a decision-making statement used to select one among many alternatives.

- It is similar to if statements but is more efficient when checking a single variable against multiple constant values.
- **Purpose:** To execute different blocks of code based on the value of an expression.

Syntax of Switch Statement

```
switch (expression)
{
   case value1:
      // statements executed if expression == value1
      break;
   case value2:
      // statements executed if expression == value2
      break;
   case value3:
      // statements executed if expression == value3
      break;
```

```
    ...
    default:
        // statements executed if none of the cases match
        break;
}
```

Explanation:

1. **Expression:** Must be of type integer or character (not float or double).
2. **Case labels:** Constant values that the expression is compared against.
3. **Break statement:** Stops execution of the switch after a case is executed; prevents fall-through to the next case.
4. **Default case:** Optional; executes when no case matches the expression.

Program Example

Problem: Check whether a character is a vowel or not.

```
#include <stdio.h>
void main()
```

```
{
    char ch;
    printf("Enter a character: ");
    scanf("%c", &ch);
    switch (ch)
    {
        case 'a':
        case 'A':
        case 'e':
        case 'E':
        case 'i':
        case 'I':
        case 'o':
        case 'O':
        case 'u':
        case 'U':
```

```
            printf("The character is a vowel.\n");
            break;
        default:
            printf("The character is not a vowel.\n");
            break;
    }
}
```

Explanation of Program:

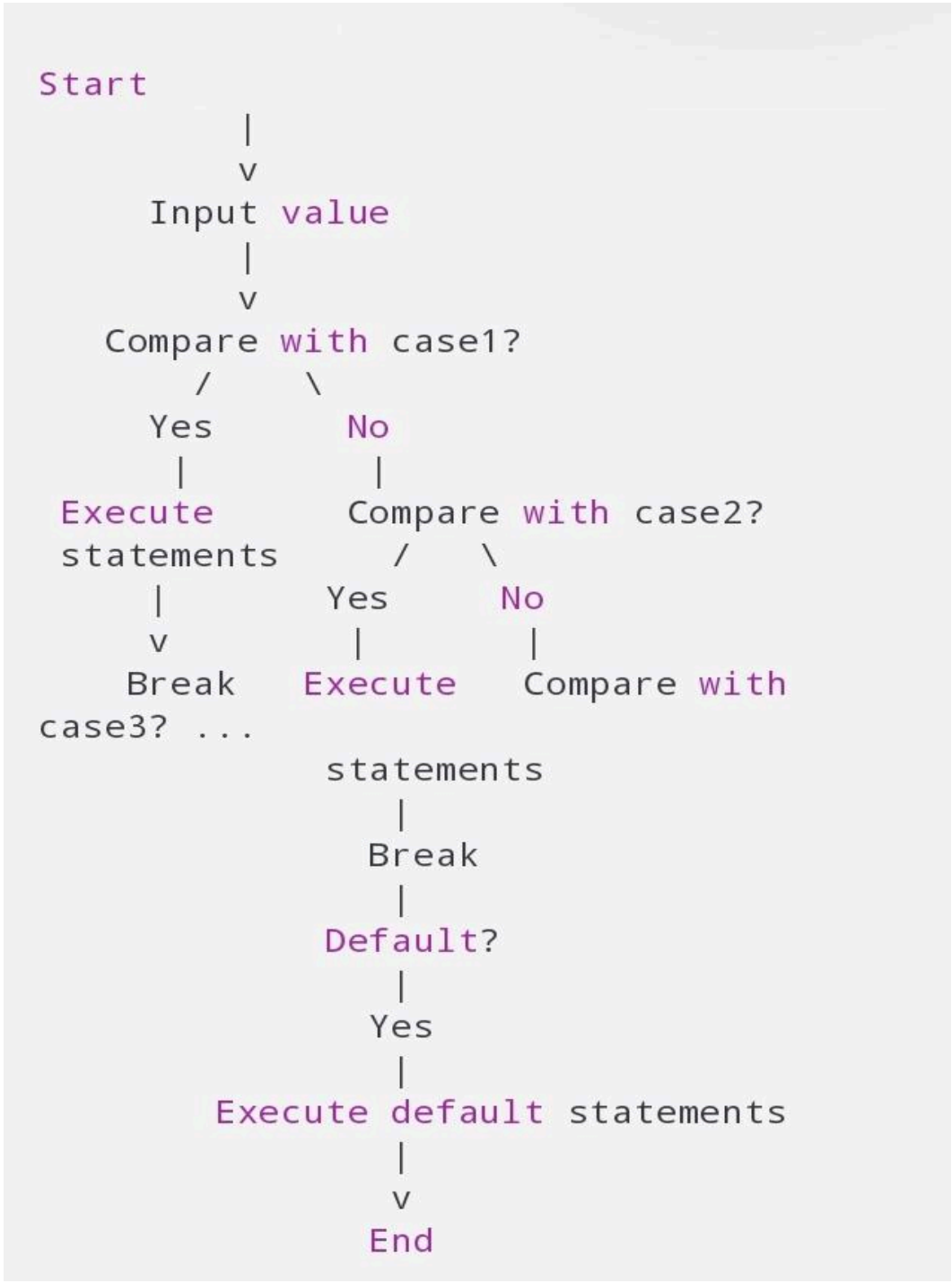
- Input a character from the user.
- The switch statement compares the character with each case label.
- If a match is found ⟶ executes the statements under that case.
- The break statement prevents execution of the following cases.
- If no match is found ⟶ the default case executes ⟶ prints “not a vowel.”

Note: Multiple cases can be combined without a break if they execute the same block of code.

Importance of Break Statements

- Without break, C will execute all subsequent cases (fall-through) until it finds a break or reaches the end.
- Using break ensures that only the matched case executes.
- The default case is optional but is recommended for handling unmatched values.

Flowchart of Switch Statement



Flowchart Explanation:

- Program starts and inputs a value.

- The value is compared sequentially with each case.
- If a match is found ⟶ executes statements ⟶ break ⟶ ends switch.
- If no match ⟶ default executes.
- Program ends after executing the matched case or default.

✦ **Summary:**

1. switch is efficient for multiple alternatives compared to multiple if-else if.
2. Uses cases, break, and optional default to control execution.
3. Prevents unnecessary comparisons and simplifies code when dealing with single variables and multiple constant values.
4. Break statements are essential to stop fall-through.

🌟 **Q.7: Write a program to check whether a number is positive, negative, or zero using**

(a) sequence of if statements,

(b) if-else if statement, and explain which method is more efficient.

❖ **Answer:**

A number can be positive (>0), negative (<0), or zero (=0). In C, we can determine this using two main methods: sequence of if statements and if-else if statements.

(a) Using Sequence of If Statements

Definition:

A sequence of if statements is a series of independent if statements where each condition is checked separately, regardless of whether previous conditions are true or false.

Program Example:

```
#include <stdio.h>

void main() {

    int num;

    printf("Enter a number: ");

    scanf("%d", &num);

    if (num > 0)

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

```
    if (num < 0)
        printf("The number is negative.\n");
    if (num == 0)
        printf("The number is zero.\n");
}
```

Control Flow Explanation:

1. Program reads a number from the user.
2. First if checks if num > 0.
    - If true ⟶ prints “positive”.
    - If false ⟶ moves to next if.
3. Second if checks if num < 0.
    - If true ⟶ prints “negative”.
4. Third if checks if num == 0.
    - If true ⟶ prints “zero”.

Observation:

- Even if the number is positive, the program still checks the other two conditions.
- Inefficient because unnecessary comparisons are done.

(b) Using If-Else If Statement

Definition:

- The if-else if statement is a decision-making structure that evaluates conditions sequentially.
- As soon as a true condition is found, the remaining conditions are skipped, making it efficient.

Program Example:

```
#include <stdio.h>

void main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num > 0)
        printf("The number is positive.\n");
    else if (num < 0)
        printf("The number is negative.\n");
    else
        printf("The number is zero.\n");
```

```
}
```

Control Flow Explanation:

1. Program reads a number.
2. First if checks num > 0.
    - If true ⟶ prints “positive” and skips the remaining checks.
3. If false ⟶ else if checks num < 0.
    - If true ⟶ prints “negative” and skips the else.
4. If both are false ⟶ else executes ⟶ prints “zero”.

Observation:

- Only the necessary condition is evaluated.
- CPU time is saved, especially in programs with many alternatives.

Comparison: Sequence of If vs If-Else If

Feature	**Sequence of If**	**If-Else If**
Evaluation	All conditions are evaluated	Stops at first true condition
Efficiency	Less efficient	More efficient
Readability	Can be cluttered for multiple conditions	Cleaner and easier to read
Use Case	When all conditions need checking independently	When conditions are mutually exclusive

Which Method is More Efficient?

✅ If-Else If Statement is more efficient because:

- Reduces unnecessary checks.
- Makes the program faster and cleaner.
- Ideal for mutually exclusive choices, like positive, negative, zero.

Flow of Execution (Conceptual)

Sequence of If:

Start ⟶ Input number ⟶ Check if >0 ⟶ Check if <0 ⟶ Check if ==0 ⟶ End

If-Else If:

Start ⟶ Input number ⟶ Check if >0?

⟶ Yes ⟶ Print positive ⟶ End

⟶ No ⟶ Check if <0?

⟶ Yes ⟶ Print negative ⟶ End

⟶ No ⟶ Print zero ⟶ End

✦ **Summary:**

- **Sequence of if:** Each condition is independent, may waste CPU time.
- **If-else if:** Stops checking after the first true condition, more efficient and readable.

Best practice: Use if-else if for mutually exclusive conditions.

Note:

This chapter is designed to provide a solid foundation of knowledge, with the goal of deepening understanding and encouraging further exploration of the subject. The content has been carefully selected to support effective learning and inspire students to engage with the topic more deeply.

Author: Muhammad Asghar

Purpose: To contribute to education by offering insightful, valuable content that enhances learning and understanding.