Fundamentals of Programming

Lab Manual 4

Exploring Decision Control Structures in Programming: Ternary Operators, Nested If-Else, and Switch Case Statements

SCHOOL OF INTERDISCIPLINARY ENGINEERING AND SCIENCES NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY (NUST)

Objective:

To gain proficiency in utilizing various decision control structures in programming, including ternary operators, nested if-else statements, and switch-case statements, through practical exercises and experimentation.

What are Decision Control Structures?

Decision control structures, also known as conditional statements, are fundamental constructs in programming that allow a program to execute different sequences of instructions based on specified conditions. These structures enable programmers to make decisions within their code, leading to dynamic and flexible program behavior.

Importance of Decision Control Structures:

- Decision control structures are essential for creating programs that can adapt and respond to different situations or inputs.
- They enable programs to perform different actions based on varying conditions, enhancing the program's versatility and usefulness.
- Without decision control structures, programs would execute linearly, lacking the ability to make intelligent decisions or respond dynamically to changing circumstances.

Types of Decision Control Structures:

There are several types of decision control structures commonly used in programming languages:

- Ternary Conditional Operator (? :): Provides a concise way to make decisions based on a condition.
- Nested if-else Statements: Allows for multiple levels of decision-making by nesting if-else blocks within each other.
- Switch-Case Statements: Provides a structured way to compare a single value against multiple possible cases and execute different blocks of code accordingly

Ternary Conditional Operator (? :):

The ternary conditional operator, often denoted as **condition ? expression1 : expression2**, is a compact way to express conditional statements in C programming. It evaluates a condition and returns one of two expressions based on whether the condition is true or false. Here's an explanation along with a code example:

Code Example:

```
#include <stdio.h>

int main() {
    int num = 10;
    char* result;

    // Using ternary conditional operator to assign a value to 'result' based on the condition
    result = (num % 2 == 0) ? "Even" : "Odd";

    // Output the result
    printf("The number %d is %s.\n", num, result);

    return 0;
}
```

Explanation:

- In this example, we have an integer variable num initialized to 10.
- We want to determine if num is even or odd.
- The ternary conditional operator (num % 2 == 0) ? "Even" : "Odd" is used to evaluate the condition num % 2 == 0.

- If the condition num % 2 == 0 is true (meaning num is even), the expression "Even" is returned.
- If the condition is false (meaning num is odd), the expression "Odd" is returned.
- The result of the ternary operator is then assigned to the variable result.
- Finally, we print out the result which indicates whether the number is even or odd.

Output:

The number 10 is Even.

In summary, the ternary conditional operator provides a concise and readable way to make decisions based on a condition, making the code more compact and expressive.

Task 1:

Write a program that takes three integer inputs from the user representing the lengths of the sides of a triangle. Determine and display whether the triangle is equilateral, isosceles, or scalene using the ternary conditional operator.

Instructions:

1. Prompt the user to enter the lengths of the three sides of the triangle.
2. Read the inputs from the user and store them in separate variables.
3. Use the ternary conditional operator to determine the type of triangle based on the lengths of its sides:
   - If all three sides are equal, output "Equilateral triangle".
   - If two sides are equal, output "Isosceles triangle".
   - If no sides are equal, output "Scalene triangle".
4. Display the type of triangle according to the determination made in step 3.

Enter the lengths of the three sides of the triangle:

5

5

5

The triangle is an Equilateral triangle.

Note: Make sure to validate the input to ensure that the lengths entered by the user can form a valid triangle. If the lengths do not satisfy the triangle inequality theorem, prompt the user to re-enter valid lengths.

Nested if-else statements:

Nested if-else statements are conditional statements within other conditional statements. They allow for multiple levels of decision-making by nesting one if-else statement inside another. This allows for more complex logic to be implemented, where certain conditions need to be checked only if the outer condition is true.

Code Example:

```
#include <stdio.h>
int main() {
    int num;
    // Prompt the user to enter a number
    printf("Enter a number: ");
    scanf("%d", &num);
    // Check if the number is positive, negative, or zero
    if (num > 0) {
        printf("The number is positive.\n");
        // Nested if-else to check if the number is even or odd
        if (num % 2 == 0) {
            printf("And it is even.\n");
        } else {
            printf("And it is odd.\n");
        }
    } else if (num < 0) {
        printf("The number is negative.\n");
    } else {
        printf("The number is zero.\n");
    }
```

```
    return 0;
}
```

Explanation:

- The program prompts the user to enter a number and reads the input into the variable num.
- It then checks if the number is positive, negative, or zero using an outer if-else statement.
- If the number is positive, it enters into the nested if-else statement to determine if it's even or odd.
- Inside the nested if-else, if the number modulo 2 equals 0, it's even; otherwise, it's odd.
- If the number is negative, it prints that the number is negative.
- If the number is zero, it prints that the number is zero.
- This example demonstrates how nested if-else statements allow for hierarchical decision-making based on multiple conditions.

TASK 2:

You are tasked with developing a program that calculates the Body Mass Index (BMI) of an individual based on their weight and height inputs and classifies their weight status according to their BMI. Additionally, the program provides specific recommendations based on the weight status category and severity of the BMI.

Requirements:

1. **Input:** Prompt the user to input their weight in kilograms and height in meters.
2. **BMI Calculation:** Calculate the BMI using the formula: BMI = weight / (height * height).
3. **Weight Status Classification:**
    - If BMI < 18.5, classify as "Underweight".
    - If BMI is between 18.5 and 24.9, classify as "Normal weight".
    - If BMI is between 25 and 29.9, classify as "Overweight".
    - If BMI is 30 or greater, classify as "Obesity".

4. **Recommendations:**
    - For "Underweight":
        - If BMI is less than 16.0, recommend immediate medical attention.
        - If BMI is between 16.0 and 16.9, recommend consulting a healthcare professional.
        - If BMI is 17.0 or greater, recommend focusing on a balanced diet and regular exercise routine.
    - For "Normal weight": Recommend maintaining a healthy lifestyle with balanced nutrition and regular physical activity.
    - For "Overweight": Recommend adopting healthier eating habits and increasing physical activity.
    - For "Obesity":
        - If BMI is less than 35, recommend consulting with a healthcare professional to discuss weight management strategies.
        - If BMI is between 35 and 39.9, recommend immediate intervention to address obesity-related health risks.
        - If BMI is 40 or greater, recommend urgent medical attention due to severe obesity-related health complications.
5. **Output:** Display the calculated BMI, weight status category, and specific recommendations based on the weight status and severity of the BMI.

Example Output:

Enter your weight in kilograms: 75

Enter your height in meters: 1.75

BMI: 24.49

Weight Status: Normal weight

Recommendation: Maintain a healthy lifestyle with balanced nutrition and regular physical activity.

Switch-Case Statements:

Switch-case statements are control flow statements used in programming to perform different actions based on the value of a variable or expression. They provide a cleaner alternative to using multiple nested if-else statements when you have many conditions to check.

The “break” Statement:

The break statement is used to exit from the body of the switch structure (or loop structure). If it is not used then the statements of other cases that come after the matching case will also be executed.

Code Example:

```
#include <stdio.h>
int main() {
    int choice;
    // Prompt the user to select an option
    printf("Choose an option:\n");
    printf("1. Option 1\n");
    printf("2. Option 2\n");
    printf("3. Option 3\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    // Switch-case statement to perform different actions based on the 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;
}
```

Explanation:

- In this example, the user is prompted to select an option by entering a number.
- The value entered by the user is stored in the variable choice.
- The switch-case statement evaluates the value of choice and performs different actions based on its value.
- If the choice is 1, it prints "You chose Option 1." and then breaks and exits the switch statement.
- If the choice is 2, it prints "You chose Option 2." and exits the switch statement.
- If the choice is 3, it prints "You chose Option 3." and exits the switch statement.
- If the choice doesn't match any of the cases (i.e., it's not 1, 2, or 3), the default case is executed, printing "Invalid choice."

Switch-case statements are particularly useful when you have a variable or expression that can take on a discrete set of values, and you need to perform different actions based on those values.

TASK 3:

Design and implement a simple ATM (Automated Teller Machine) simulator program that allows users to perform basic banking transactions. The program should utilize switch-case statements to handle different transaction options selected by the user.

Requirements:

1. Display a menu of available transactions to the user, including options for:
    1. Checking balance
    2. Depositing funds
    3. Withdrawing funds
    4. Quitting the program
2. Prompt the user to enter their choice from the menu.
3. Based on the user's choice, perform the corresponding action using switch-case statements:
    - If the user chooses to check balance, display their current account balance.
    - If the user chooses to deposit funds, prompt them to enter the amount to deposit and update their account balance accordingly.
    - If the user chooses to withdraw funds, prompt them to enter the amount to withdraw, ensuring that they do not withdraw more than their current balance, and update their account balance accordingly.
    - If the user chooses to quit the program, terminate the program.
4. After completing each transaction, display a message indicating the result of the transaction or any relevant information.

Guidelines:

1. Ensure error handling for invalid inputs, such as non-numeric values or negative amounts for deposits and withdrawals.
2. Provide a clear and user-friendly interface with appropriate prompts and messages.