Module-2
Control and Loop statements

School of Electronics Engineering (SENSE)
Vellore Institute of Technology
Chennai

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

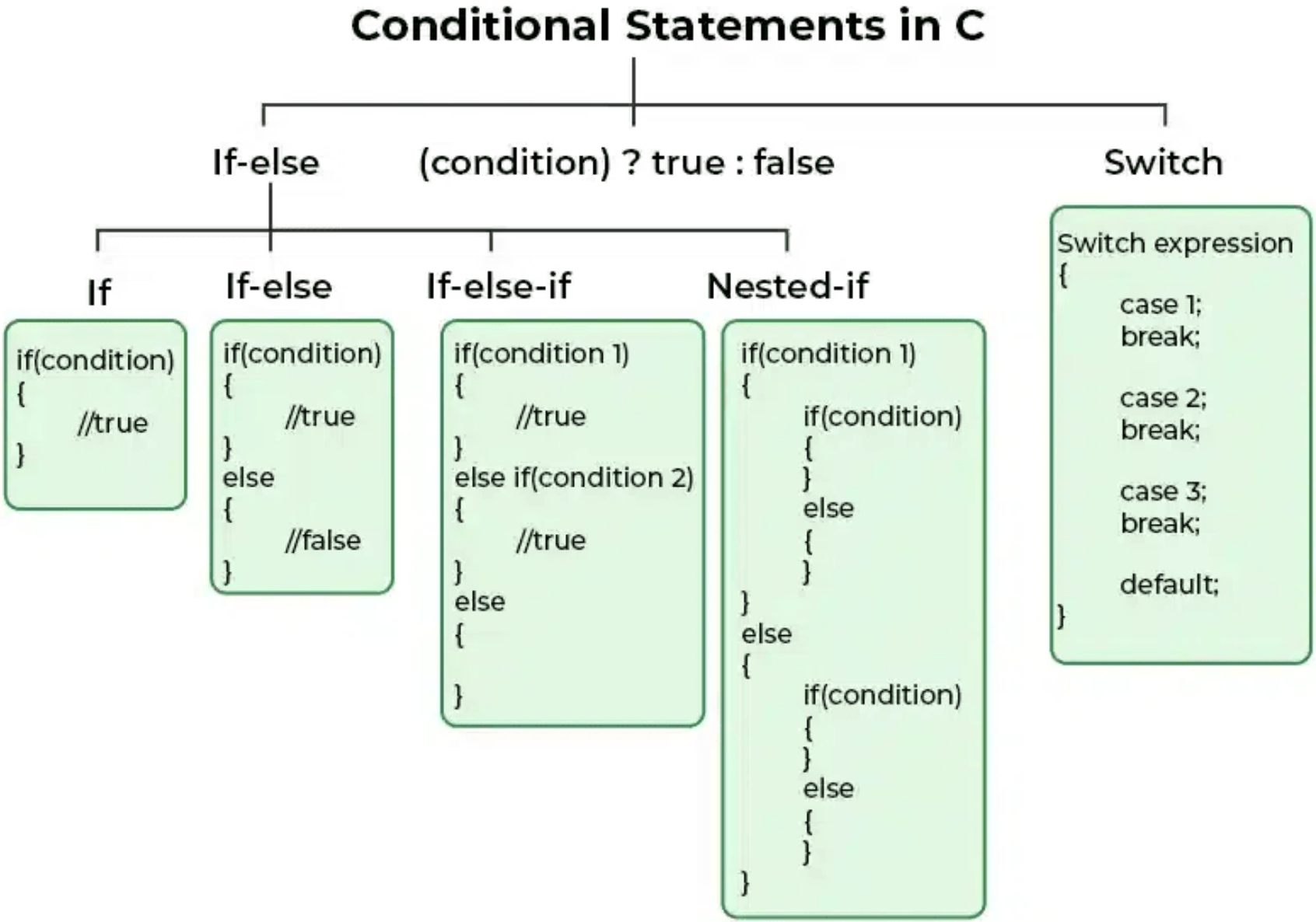
Outline

Introduction to Control Statements

Control statements in C are crucial for decision-making in programs. They allow the code to take different paths based on conditions.

- Types of Control Statements: **if**, **if-else**, **switch**.
- Decision statements allow program logic to branch conditionally.
- They guide the program flow.

Control Statements



The 'If' Statement - Basics

Syntax

```
if (condition) {
    // code to execute if condition is true
}
```

Example

Check if a number is positive:

```
if (number > 0) {
    printf("Positive");
}
```

Example of 'If' Statement

Code

```
int num = 4;
if (num % 2 == 0) {
    printf("Even");
} else {
    printf("Odd");
}
```

Output

"Even" for num = 4

Understanding 'If-Else' Statements

Syntax

```
if (condition) {
/* code if true */
}
else {
/* code if false */
}
```

- The 'if-else' statement allows more complex decision paths.
- Used when two paths of execution are required.

'If-Else' Example

Problem

Determine the grade based on a score.

Code

```
int score = 85;
if (score >= 90) {
    printf("Grade A");
} else if (score >= 80) {
    printf("Grade B");
} else if (score >= 70) {
    printf("Grade C");
} else {
    printf("Grade F");
}
```

The 'Else-If' Ladder Explained

Syntax

```
if (condition1) {
/* code1 */
}
else if (condition2) {
/* code2 */
}
...
else {
/* default code */
}
```

- An 'else-if' ladder is used for multiple conditions.
- It checks multiple conditions in sequence.

'Else-If' Complex Example

Problem

Categorize age groups based on years.

Code

```
int age = 25;
if (age < 13) {
    printf("Child");
} else if (age < 20) {
    printf("Teen");
} else if (age < 60) {
    printf("Adult");
} else {
    printf("Senior");
}
```

Introduction to 'Switch' Statements

Syntax

```
switch (expression) {
    case value1:
        // code block
        break;
    case value2:
        // code block
        break;
    default:
        // default code block
}
```

- Switch statements offer a way to select one of many code blocks to be executed.
- Useful for cases with multiple specific values to check against.

'Switch' Statement Syntax

Syntax

```
switch (expression) {
    case value1:
        // code block
        break;
    case value2:
        // code block
        break;
    default:
        // default code block
}
```

Example Output

The output depends on the expression value and can execute any of the case blocks or the default block if no case matches.

'Switch' Statement Complex Example

Problem

Execute different operations based on user choice.

Code

```
char operation = '+';
int a = 10, b = 5;
switch (operation) {
    case '+': printf("%d", a + b); break;
    case '-': printf("%d", a - b); break;
    case '*': printf("%d", a * b); break;
    case '/': printf("%d", a / b); break;
    default: printf("Invalid operation");
}
```

'Switch' Case Example

Code

```
int day = 3;
switch (day) {
    case 1: printf("Monday"); break;
    case 2: printf("Tuesday"); break;
    case 3: printf("Wednesday"); break;
    // Additional cases
}
```

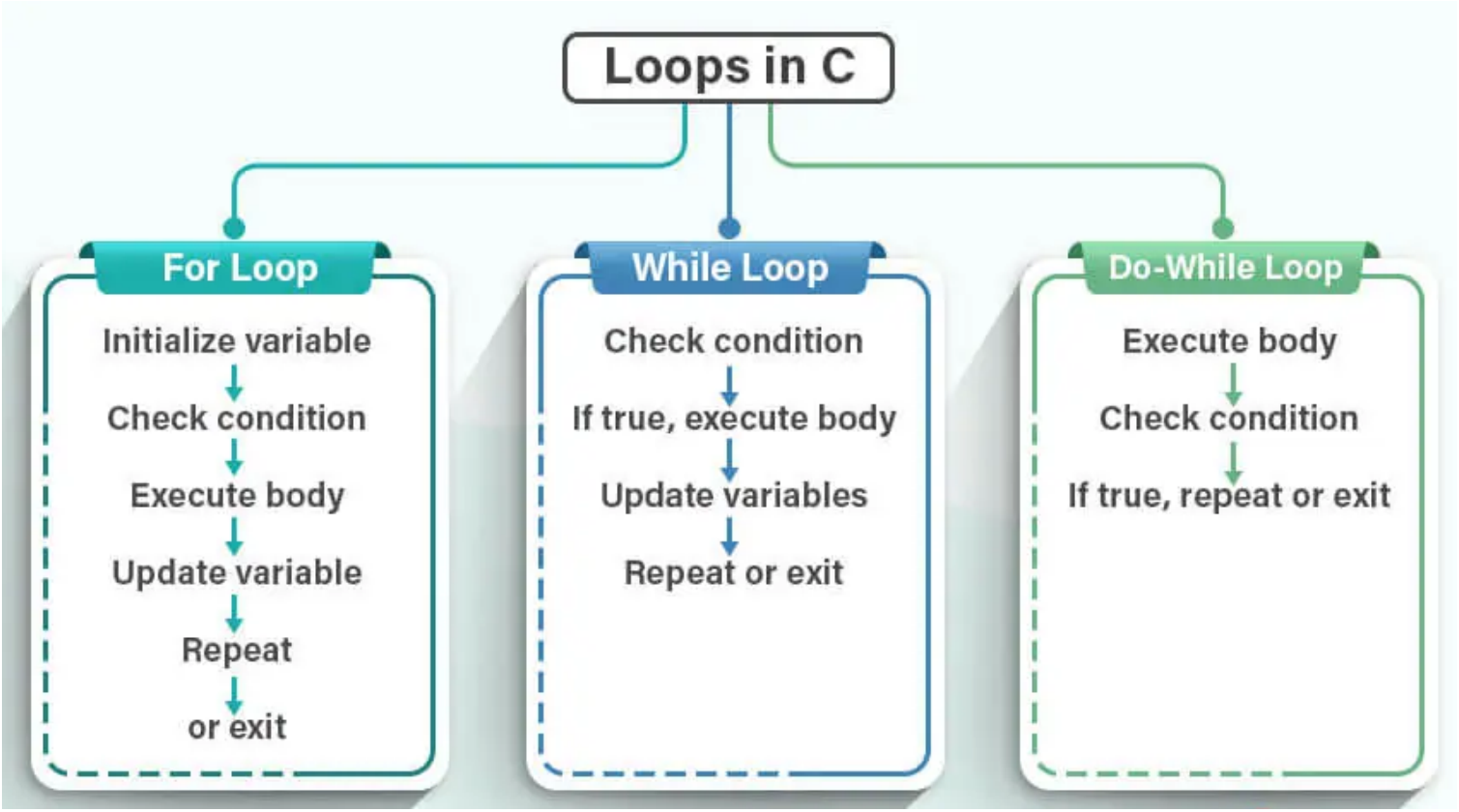
Output

```
"Wednesday"
```

Loops in C: An Overview

Loops in C are used to execute a block of code repeatedly.

- Types of loops: **while**, **do-while**, **for**.
- Loops are ideal for tasks that require repetition.



'While' Loop Explained

Syntax

```
while (condition) {
    // code to be executed
}
```

Example

Print numbers from 1 to 5.

```
int i = 1;
while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

'While' Loop Example Output

Code

```
int i = 1;
while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

Output

```
1 2 3 4 5
```

'Do-While' Loop Basics

Syntax

```
do {
/* code */
}
while (condition);
```

- A 'do-while' loop ensures that the code block is executed at least once.

'Do-While' Loop Example

Problem

Print numbers from 1 to 5, using a 'do-while' loop.

Code

```
int i = 1;
do {
    printf("%d ", i);
    i++;
} while (i <= 5);
```

Understanding 'For' Loops

Syntax

```
for (initialization; condition; increment) {
/* code */
}
```

- The 'for' loop is used for a specific number of iterations.

'For' Loop Syntax and Example

Syntax

```
for (initialization; condition; update) {
    // code to be executed
}
```

Example

Print numbers from 1 to 5.

```
for (int i = 1; i <= 5; i++) {
    printf("%d ", i);
}
```

Nested Loops Concept

Nested loops are loops inside another loop, commonly used for multi-dimensional data processing.

- Inner loop runs completely for each iteration of the outer loop.

Nested Loops Example

Problem

Print a 3x3 matrix of numbers.

Code

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        printf("%d ", i*j);
    }
    printf("\n");
}
```

'Break' Statement in Loops

The 'break' statement is used to exit a loop prematurely.

- Immediately stops loop execution and moves to the next statement after the loop.

Using 'Break' - Example

Problem

Exit the loop when a specific condition is met.

Code

```
for (int i = 1; i <= 10; i++) {
    if (i == 6) break;
    printf("%d ", i);
}
```

'Continue' Statement Explained

The 'continue' statement skips the current iteration of a loop and moves to the next.

- Often used to skip certain conditions within a loop.

'Continue' in Action

Problem

Skip even numbers in a loop.

Code

```
for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) continue;
    printf("%d ", i);
}
```

The 'Goto' Statement

The 'goto' statement directs the flow to a labeled part of the program.

- Can be used for exiting multiple nested loops.
- Generally discouraged due to readability and maintainability issues.

'Goto' Use Case

Problem

Using 'goto' for error handling in nested loops.

Code

```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (/* error condition */) {
            goto error;
        }
        // process data
    }
}
error:
    // error handling code
```

Impact of 'Goto' on Flow

Understanding the impact of 'goto' on program flow.

- 'goto' can make the code hard to read and maintain.
- Best practices suggest limiting its use to specific cases like error handling.

Quiz: Control Structures

Time for a quick quiz to test your understanding of control structures.

1. What is the difference between 'break' and 'continue' in a loop?
2. Provide a scenario where a 'goto' might be useful.

Quiz: Control Structures

Time for a quick quiz to test your understanding of control structures.

1. What is the difference between 'break' and 'continue' in a loop?
2. Provide a scenario where a 'goto' might be useful.

Answers

1. 'break' exits the loop completely, while 'continue' skips to the next iteration.
2. 'goto' can be useful in nested loops for error handling, jumping out of multiple loop levels.

Code Challenge: If-Else

Challenge

Write a program using 'if-else' to categorize a character as a vowel or consonant.

Code Challenge: If-Else

Challenge

Write a program using 'if-else' to categorize a character as a vowel or consonant.

Solution

```
char ch = 'a';
if (ch == 'a' || ch == 'e' || ch == 'i' ||
    ch == 'o' || ch == 'u') {
    printf("%c is a vowel", ch);
} else {
    printf("%c is a consonant", ch);
}
```

Advanced 'If' Statements

Problem

Determine the most significant digit of a number.

Advanced 'If' Statements

Problem

Determine the most significant digit of a number.

Solution

```
int num = 982;
int significant = num;
while (significant >= 10) {
    significant /= 10;
}
printf("Most significant digit: %d", significant);
```

Complex 'Switch' Cases

Problem

Execute different math operations based on user input.

Complex 'Switch' Cases

Problem

Execute different math operations based on user input.

Solution

```
char op = '+';
int a = 10, b = 5, result;
switch (op) {
    case '+': result = a + b; break;
    case '-': result = a - b; break;
    case '*': result = a * b; break;
    case '/': result = a / b; break;
    default: printf("Invalid operation");
}
printf("Result: %d", result);
```

Loops with Multiple Conditions

Problem

Print numbers divisible by either 2 or 3 up to 20.

Loops with Multiple Conditions

Problem

Print numbers divisible by either 2 or 3 up to 20.

Solution

```
for (int i = 1; i <= 20; i++) {
    if (i % 2 == 0 || i % 3 == 0) {
        printf("%d ", i);
    }
}
```

Infinite Loops and Prevention

Understanding and preventing infinite loops in your code.

- Common in 'while' and 'for' loops with incorrect conditions or updates.
- Always ensure that the loop's exit condition will be met.

Debugging Loop Errors

Problem

Identify and fix the error in the following loop.

Debugging Loop Errors

Problem

Identify and fix the error in the following loop.

Code with Error

```
for (int i = 0; i != 10; i += 2) {
    printf("%d ", i);
}
```

Debugging Loop Errors

Problem

Identify and fix the error in the following loop.

Code with Error

```
for (int i = 0; i != 10; i += 2) {
    printf("%d ", i);
}
```

Fixed Code

```
for (int i = 0; i < 10; i += 2) {
    printf("%d ", i);
}
```

Quiz: Loop Constructs

Challenge your understanding of loop constructs with these questions.

1. Explain the difference between a 'while' loop and a 'do-while' loop.
2. Give an example of a scenario where a nested loop is necessary.

Quiz: Loop Constructs

Challenge your understanding of loop constructs with these questions.

1. Explain the difference between a 'while' loop and a 'do-while' loop.
2. Give an example of a scenario where a nested loop is necessary.

Answers

1. A 'while' loop checks the condition before executing, while a 'do-while' loop executes at least once before checking.
2. Nested loops are used for multi-dimensional array processing, like matrix operations.

Quiz Solutions and Discussion with Real-Life Scenarios

Case Studies and Scenarios

- While vs Do-While Loops:
  - Case Study: Sensor Data Processing - Using 'while' for continuous monitoring and 'do-while' for initial setup procedures.
- Applications of Nested Loops:
  - Real-Life Scenario: E-commerce Website - Displaying product categories (outer loop) and products (inner loop).
  - Application in Gaming: Developing a grid-based puzzle game using nested loops for grid management.

Effective Use of 'Break'

Problem

Use 'break' to exit a loop when a prime number is found.

Effective Use of 'Break'

Problem

Use 'break' to exit a loop when a prime number is found.

Solution

```
int n = 50, flag = 0;
for (int i = 2; i <= n; i++) {
    flag = 1;
    for (int j = 2; j <= i / 2; j++) {
        if (i % j == 0) {
            flag = 0;
            break;
        }
    }
    if (flag == 1) {
        printf("First prime number: %d", i);
        break;
    }
}
```

When to Use 'Continue'

Problem

Skip numbers that are not prime in a loop.

When to Use 'Continue'

Problem

Skip numbers that are not prime in a loop.

Solution

```
for (int num = 2; num <= 20; num++) {
    int isPrime = 1;
    for (int i = 2; i <= num / 2; i++) {
        if (num % i == 0) {
            isPrime = 0;
            break;
        }
    }
    if (!isPrime) continue;
    printf("%d ", num);
}
```

'Goto': Good or Bad?

A critical look at the 'goto' statement.

- Can simplify error handling in complex code.
- However, it can lead to "spaghetti code" that is hard to read and maintain.
- Use sparingly and with clear intent.

Code Challenge: Break and Continue

Challenge

Write a loop that prints numbers from 1 to 10 but skips any number that is a multiple of 3, and stops the loop if the number is 7.

Code Challenge: Break and Continue

Challenge

Write a loop that prints numbers from 1 to 10 but skips any number that is a multiple of 3, and stops the loop if the number is 7.

Solution

```
for (int i = 1; i <= 10; i++) {
    if (i % 3 == 0) continue;
    if (i == 7) break;
    printf("%d ", i);
}
```

Code Challenge: Goto Usage

Challenge

Implement a nested loop structure where an error in the inner loop leads to a jump to an error-handling section using 'goto'.

Code Challenge: Goto Usage

Challenge

Implement a nested loop structure where an error in the inner loop leads to a jump to an error-handling section using 'goto'.

Solution

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (/* error condition */) goto handleError;
        // process data
    }
}
handleError:
    // error handling code
```

Understanding Conditional Logic

Exploring more complex conditional logic in C programming.

- Compound conditions using logical operators (&&, ||).
- Application in decision-making algorithms.

Boolean Expressions in Control

Advanced Example

Develop a program that checks if a number is both positive and even.

Boolean Expressions in Control

Advanced Example

Develop a program that checks if a number is both positive and even.

Solution

```
int num = 8;
if (num > 0 && num % 2 == 0) {
    printf("Number is positive and even");
} else {
    printf("Condition not met");
}
```

Control Flow Best Practices

Best practices for writing clean and efficient control flow in C.

- Keep conditions simple and readable.
- Avoid deeply nested structures when possible.
- Use 'switch' statements for clarity in multiple-choice scenarios.

Avoiding Common Mistakes

Highlighting common pitfalls in control flow and how to avoid them.

- Infinite loops due to incorrect conditions.
- Misuse of 'break' and 'continue' in complex loops.
- Overuse or incorrect use of 'goto'.

Advanced Control Structures

Exploring more complex control structures for sophisticated program logic.

- Nested if-else statements for multi-level decision making.
- Combining loops with conditional statements for advanced data processing.

Combining Loops and Conditionals

Problem

Create a program that finds the first 5 prime numbers in a range.

Combining Loops and Conditionals

Problem

Create a program that finds the first 5 prime numbers in a range.

Solution

```
int count = 0;
for (int i = 2; count < 5; i++) {
    int isPrime = 1;
    for (int j = 2; j <= i / 2; j++) {
        if (i % j == 0) {
            isPrime = 0;
            break;
        }
    }
    if (isPrime) {
        printf("%d ", i);
        count++;
    }
}
```

Case Study: Simple Game Development

Scenario

Developing a simple number guessing game using loops and conditionals.

Game Logic

- The computer picks a random number between 1 and 100.
- The player guesses the number; the computer provides hints ('higher' or 'lower').
- The game continues until the player guesses correctly or exits.

Code Overview

Provide an overview of the game's code, highlighting the use of a 'while' loop for the guessing process and 'if-else' statements for hinting.

Number Guessing Game - Code and Output

Code Snippet

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int number, guess, attempts = 0;
    srand(time(0));
    number = rand() % 100 + 1;//Random number from 1 to 100
    printf("Guess the number (1 to 100):\n");
    do {
        scanf("%d", &guess);
        attempts++;
        if (guess > number) {
            printf("Lower!\n");
        } else if (guess < number) {
            printf("Higher!\n");
        }
    } while (guess != number);
    printf("Correct! Attempts: %d", attempts);
    return 0;
}
```

Gameplay Output

```
Guess:  50 → Lower!
Guess:  25 → Higher!
Guess:  37 → Correct!
Attempts:  3
```

Real-Time Case Challenge 1

Challenge Description

Develop a program to simulate a traffic light controller using control statements. The traffic light should change from green to yellow, then to red, and back to green, with appropriate time delays.

Solution to Case Challenge 1

Traffic Light Controller Code

```
#include <stdio.h>
#include <unistd.h>  // For sleep function

int main() {
    char light = 'G';  // Start with Green
    while (1) {
        switch (light) {
            case 'G': printf("Green\n"); sleep(5); light = 'Y'; break;
            case 'Y': printf("Yellow\n"); sleep(2); light = 'R'; break;
            case 'R': printf("Red\n"); sleep(7); light = 'G'; break;
        }
    }
    return 0;
}
```

Explanation

This program simulates a traffic light using a switch-case inside a loop. The ‘sleep‘ function provides the delay.

Real-Time Case Challenge 2

Challenge Description

Create an algorithm for a simple checkout system in a store. The system should calculate the total cost, apply discounts if applicable, and display the final amount.

Solution to Case Challenge 2

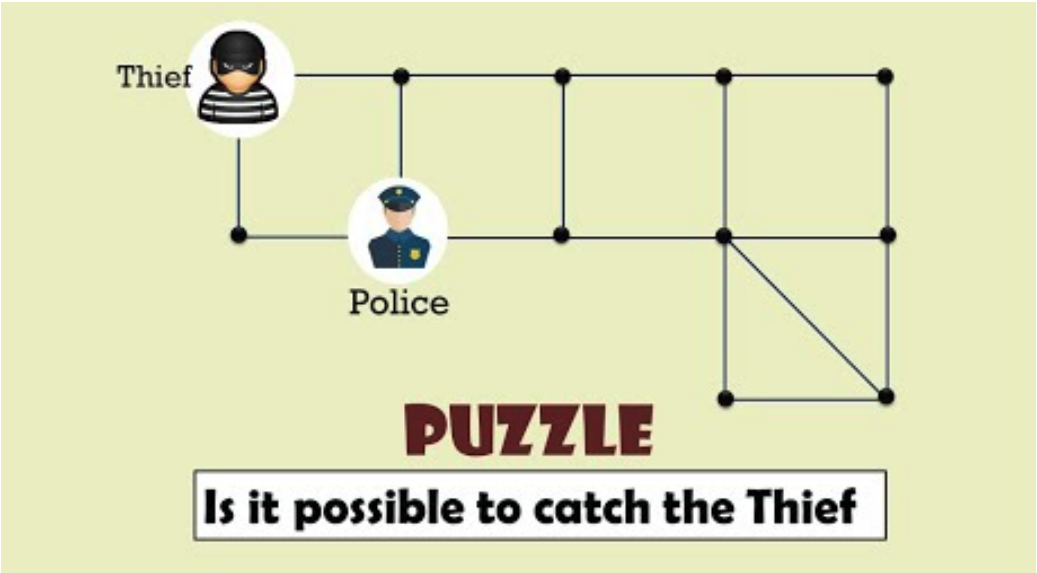
Checkout System Code

```
#include <stdio.h>
int main() {
    double price, total = 0;
    int count, discount = 0;
    printf("Enter number of items: ");
    scanf("%d", &count);
    for (int i = 0; i < count; i++) {
        printf("Enter price of item %d: ", i + 1);
        scanf("%lf", &price);
        total += price;
    }
    if (total > 100) discount = 10;  // 10% discount
    printf("Total: %.2f, Discount: %d%%, Final: %.2f\n",
           total, discount, total * (1 - discount / 100.0));
    return 0;
}
```

Police-Thief Puzzle Description

Puzzle Scenario

A police officer and a thief are on a grid. The police officer must catch the thief, who tries to evade capture. Both can move a one step in each turn.



Task

Design an algorithm to simulate this scenario. Consider factors like movement distance per turn and strategy to either capture or evade.

Solution to the Police-Thief Puzzle

Algorithm Approach

A strategy-based algorithm where both the police and the thief make decisions each turn, considering factors like distance and potential movement.

Pseudo-Code

```
Initialize positions of Police (P) and Thief (T)
while (distance between P and T is not zero) {
    Update position of P based on strategy
    Update position of T based on evasion
    Check for capture condition
}
```

Key Concepts

Use loops for continuous checking, conditional statements for decision-making, and movement strategies.

Police-Thief Grid Puzzle Simulation

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int policePos, thiefPos;
    srand(time(NULL));
    policePos = (rand() % 2 == 0) ? 1 : 4;
    thiefPos = (policePos == 1) ? 4 : 5;
    printf("Police: %d, Thief: %d\n", policePos, thiefPos);
    while (policePos != thiefPos) {
        policePos = (policePos % 5) + 1;
        thiefPos = (thiefPos % 5) + 1;
        printf("Police: %d, Thief: %d\n", policePos, thiefPos);
        if (policePos == thiefPos) {
            printf("Thief caught at %d\n", thiefPos);
            break;
        }
    }
    return 0;
}
```

Crazy Gangster Puzzle

Puzzle Scenario

- Let's play a game of Russian roulette. You are tied to your chair and can't get up. Here's a gun.
- Crazy gangster with a 6-shot revolver
- Initially all six chambers are empty.
- He puts two bullets into adjacent chambers(in front of you), spins it and closes the cylinder.
- He now puts the gun to your head & pulls the trigger...

Crazy Gangster Puzzle

Puzzle Scenario

- Let's play a game of Russian roulette. You are tied to your chair and can't get up. Here's a gun.
- Crazy gangster with a 6-shot revolver
- Initially all six chambers are empty.
- He puts two bullets into adjacent chambers(in front of you), spins it and closes the cylinder.
- He now puts the gun to your head & pulls the trigger...
- Luckily you survive.
- He is going to pull the trigger again...

Crazy Gangster Puzzle

Puzzle Scenario

- Let's play a game of Russian roulette. You are tied to your chair and can't get up. Here's a gun.
- Crazy gangster with a 6-shot revolver
- Initially all six chambers are empty.
- He puts two bullets into adjacent chambers(in front of you), spins it and closes the cylinder.
- He now puts the gun to your head & pulls the trigger...
- Luckily you survive.
- He is going to pull the trigger again...
- But before he proceeds he asks you for your preference whether:
- 1 - He should re-spin the cylinder then pull the trigger
- 2 - He should pull the trigger right away (no spin)

Crazy Gangster Puzzle - Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define CHAMBERS 6
int main() {
  srand(time(0));
  int chambers[CHAMBERS];
  for(int i=0; i<CHAMBERS; i++) {
    chambers[i] = 0;
  }

  int bullet1 = rand() % CHAMBERS;
  chambers[bullet1] = 1;
  int bullet2 = (bullet1 + 1) % CHAMBERS;
  chambers[bullet2] = 1;
  printf("Bullets loaded in chambers
         %d and %d\n", bullet1, bullet2);
```

```
  for(int i=0; i<100; i++) {
    int swap = rand() % CHAMBERS;
    int temp = chambers[swap];
    chambers[swap] = chambers[0];
    chambers[0] = temp;
  }

  int chamberFired = rand() % CHAMBERS;
  if(chambers[chamberFired] == 1) {
    printf("Shot 1: You were hit and died!
    return 0;
  }

  printf("Shot 1: You survived!\n");

  // Option to respin

  // Second trigger pull
  if(chambers[chamberFired] == 1) {
    printf("Shot 2: Bullet fired. Dead!");
  } else {
    printf("You survived!");
  }
```

Quiz Question 1

Question

A vending machine dispenses drinks only when the correct combination of buttons is pressed. Implement this scenario in C using control structures.

Quiz Question 1

Question

A vending machine dispenses drinks only when the correct combination of buttons is pressed. Implement this scenario in C using control structures.

Solution to Quiz Question 1

Vending Machine Program

```
#include <stdio.h>
int main() {
    char button1, button2;
    printf("Enter button combination: ");
    scanf(" %c %c", &button1, &button2);

    if (button1 == 'A' && button2 == '1') {
        printf("Dispensing Cola\n");
    } else if (button1 == 'B' && button2 == '2') {
        printf("Dispensing Sprite\n");
    } else {
        printf("Invalid combination\n");
    }
    return 0;
}
```

Code Debugging Challenge

Challenge

Identify and correct the error in this C program intended to calculate the sum of digits of a number.

Code Snippet

```
#include <stdio.h>
int main() {
    int n = 1234, sum = 0;
    while (n > 0) {
        sum = sum + n % 10;
        n = n / 10;
    }
    printf("Sum of digits: %d", sum);
    return 0;
}
```

Solution to Code Debugging Challenge

Corrected Code

The provided code is actually correct. It successfully calculates the sum of digits of the number 1234.

Output

Sum of digits: 10

Guess the Output Question 2

Question

What will be the output of the following C code snippet?

```
#include <stdio.h>
int main() {
    int x = 5;
    printf("%d %d %d", x++, x, ++x);
    return 0;
}
```

Loop Control Challenge

Question

Predict the output of the following C code snippet involving a loop and a control statement:

```
#include <stdio.h>
int main() {
    for (int i = 0; i < 5; i++) {
        if (i == 2) continue;
        if (i == 4) break;
        printf("%d ", i);
    }
    return 0;
}
```

Solution to Loop Control Challenge

Explanation

The loop iterates from 0 to 4. The 'continue' statement skips the printing when 'i' is 2. The 'break' statement stops the loop when 'i' reaches 4. Therefore, the numbers printed are 0, 1, and 3.

Output

0 1 3

Real-Time Scenario Quiz

Scenario-Based Question

Write a C program to simulate an automated door system that opens when a sensor detects a person and closes after a certain delay.

Solution to Real-Time Scenario Quiz

Pseudocode Solution

```
#include <stdio.h>
#include <unistd.h> // For sleep function

int main() {
    int sensorInput;
    printf("Enter sensor input (1 for person detected, 0 other
    scanf("%d", &sensorInput);

    if (sensorInput == 1) {
        printf("Door opening...\n");
        sleep(5); // Represents delay
        printf("Door closing...\n");
    } else {
        printf("No person detected. Door remains closed.\n");
    }
    return 0;
```

Embedded C Program for 8051 - LED Blinking

Program: LED Blinking on 8051

Control an LED connected to a port on the 8051 microcontroller.

Code Snippet

```
#include <reg51.h>

void delay() {
    // Simple delay function
    int i, j;
    for(i=0; i<1000; i++) {
        for(j=0; j<100; j++) {}
    }
}

void main() {
    while(1) {
        P1 = 0xFF; // Turn ON all LEDs
        delay();
        P1 = 0x00; // Turn OFF all LEDs
        delay();
    }
}
```

Explanation of LED Blinking Program

Program Operation

The program continuously turns all LEDs on and off with a delay in between. It uses port 1 (P1) of the 8051 to control the LEDs.

Expected Output

The connected LEDs will blink on and off indefinitely.

Embedded C Program for 8051 - Button Controlled LED

Program: Button Controlled LED on 8051

Using a button to control an LED connected to the 8051 microcontroller.

Code Snippet

```
#include <reg51.h>

void main() {
    P1 = 0xFF; // Configure Port 1 as input port
    while(1) {
        if (P1 == 0xFE) { // Check if button pressed
            P2 = 0xFF; // Turn ON LED on Port 2
        } else {
            P2 = 0x00; // Turn OFF LED
        }
    }
}
```

Explanation of Button Controlled LED Program

Program Operation

The program reads input from a button connected to Port 1. When the button is pressed, it turns on an LED connected to Port 2.

Expected Output

The LED on Port 2 toggles its state based on the button press.

Advanced Embedded C Program for 8051

Program: Temperature Controlled Fan Speed

Using nested if-else and loop structures to control a fan's speed based on temperature readings.

Program Operation

The program reads temperature from a sensor connected to Port 1. Depending on the temperature reading, it controls the fan speed connected to Port 2 using nested if-else statements.

Expected Output

The fan speed varies (off, medium, high) based on the temperature reading.

Code Snippet of Temperature Controlled Fan Speed Program

Code Snippet

```
#include <reg51.h>
#define MAX_TEMP 30
#define MID_TEMP 20
void delay() {
    int i, j;
    for(i=0; i<1000; i++) {
        for(j=0; j<100; j++) {}
    }
}
void main() {
    int temp;
    while(1) {
        temp = P1; // Assume temperature reading from sensor
        if (temp > MAX_TEMP) {
            P2 = 0x03; // High speed
        } else if (temp > MID_TEMP) {
            P2 = 0x01; // Medium speed
        } else {
            P2 = 0x00; // Fan off
        }
        delay();
    }
}
```