Spring 2025, MIS 102 – COMPUTER PROGRAMMING

Quiz 2

姓名:_____ 學號:_____ 系級:_____

1. [56 pts]

1. Which of the following is the typical range of a signed **`char`** in C (assuming an 8-bit, two's complement system)?

   a. 0 to 255

   b. -128 to 127

   c. -127 to 127

   d. -256 to 255

   ANS：B

2. In a sentinel-controlled repetition using a **`while`** loop, the sentinel value is?

   a. A value that causes the loop to execute two extra iterations

   b. A value used to to signal when the loop should stop

   c. Always equal to zero

   d. Mandatory for **`for`** loops

   ANS：B

3. Which statement best describes the **`break`** keyword within a switch-case structure?

   a. It breaks out of the entire program

   b. It is only used in loops

c. It prevents further case blocks from executing

d. It continues to the next case label

ANS : C

4. Which of the following correctly describes a sentinel-controlled loop?

a. It always uses **`i++`** as an increment expression

b. It terminates when a special "flag" value is encountered

c. It requires multiple initialization statements in the condition

d. It must be a **`for`** loop

ANS : B

5. In C, what does the **`sizeof`** operator return?

a. The number of bits in the data type

b. The number of possible values the data type can hold

c. The size (in bytes) of a variable or data type

d. The execution time of a function

ANS : C

6. In a **`switch`** statement, if a **`case`** block does not end with a **`break`**, what generally happens?

a. The program ends abruptly

b. The switch condition is evaluated again

c. Execution falls through to the next **`case`** block

d. Nothing; C automatically inserts a **`break`**

ANS : C

7. What happens when the **`continue`** statement is executed inside a **`for`** loop?

a. The loop is immediately exited

b. The loop counter resets to its initial value

c. The program jumps to the **`break`** statement

d. The current iteration is skipped, and the loop proceeds with the next iteration

ANS : D

8. Which header file in C provides the **`bool`** type and the keywords **`true`** and **`false`**?

a. #include <stdio.h>

b. #include <stdbool.h>

c. #include <math.h>

d. #include <stdlib.h>

ANS : B

9. Which statement best describes the risk when **`break`** is used in nested loops?

a. It terminates all active loops simultaneously

b. It resumes execution at the start of the outermost loop

c. It only exits the innermost loop in which it appears

d. It has no effect unless paired with an explicit **`goto`**

ANS : C

Explanation: In C, `break` exits the *innermost* loop or switch block only. Exiting multiple nested loops typically requires additional logic (e.g., a flag variable or function return).

10. Suppose you have a **`switch`** statement with multiple **`case`** labels sharing a block of code before a single **`break`**. Which statement is most accurate regarding this use of fall-through?

a. It is illegal in C and results in a compiler error

b. It is safe but requires that the **`break`** statement be in the default label

c. It is a valid language feature, but must be used deliberately to avoid unexpected behavior

d. It automatically executes only the first **`case`** label

ANS : C

Explanation: Fall-through is allowed in C, but it can be confusing if not documented or intended. Multiple cases can indeed lead to the same execution path until a `break` is encountered.

11. The future value calculation using " $FV = P \times (1 + r)^n$ ", can be implemented in C with **`pow(1 + annualRate, year).`** Which header file must be included to use **`pow()`**?
    a. `#include <stdbool.h>`
    b. `#include <stdlib.h>`
    c. `#include <math.h>`
    d. `#include <time.h>`

    ANS : C

12. Why is relying on operator precedence alone (without parentheses) considered a risky practice in complex boolean expressions?
    a. Parentheses always slow down the compiler's parse time
    b. Operator precedence is different for each C compiler
    c. Complex expressions can become hard to read or maintain, leading to subtle logical errors
    d. All C operators have the exact same precedence level, making parentheses mandatory

    ANS : C

    Explanation: Although C's operator precedence is standardized, heavily relying on it in complex expressions can lead to code that is difficult to understand and prone to mistakes. Parentheses make the intended logic explicit and more maintainable.

13. In a program that uses both **`unsigned`** and signed integer types, which of the following scenarios can lead to logic errors if not handled properly?
    a. Mixing different loops (**`for`** vs. **`while`**)
    b. Comparisons where an **`unsigned`** variable is expected to be negative, but negative values are never valid for **`unsigned`**
    c. Using **`switch`** statements without **`default`**

d. Declaring multiple **`break`** statements inside a single loop

ANS: B

Explanation: An `unsigned` integer can't represent negative numbers. If you compare an unsigned variable against a signed negative value, implicit conversions or unexpected comparisons can occur, resulting in logic errors.

14. What is one key reason to replace a **`goto`** statement with structured loops or functions?

a. **`goto`** cannot jump forward in the code, only backward

b. **`goto`** changes the value of local variables unexpectedly

c. Structured loops and functions improve readability and maintenance by controlling flow explicitly without arbitrary jumps

d. Modern C compilers do not support **`goto`** anymore

ANS: C

Explanation: Although `goto` remains part of C, it tends to make code less structured. Explicit loops (`for`, `while`, etc.) and function calls make control flow clearer, leading to code that is easier to follow and maintain.

2. [44 pts]

Q1 [C] (22 pts): **Menu-Driven Operations**: Write a menu-driven C program that continuously performs different mathematical operations based on the user's choice until the user selects an exit option. Your program must:

1. present a menu with three options:
   - **Option 1 (Summation)**: Prompt for two integers a and b and compute the sum of all integers from a to b (inclusive).
   - **Option 2 (Product)**: Prompt for two integers x and y and compute the product of all integers from x to y (inclusive).
   - **Option 3 (Base-Exponent Power)**: Prompt the user for an integer base B and an integer exponent E. Compute B^E using the C standard library
   - **Option 4 (Exit)**: Break out of the loop and end the program.
2. Use a loop that keeps presenting the menu until the user selects Exit.

3. Handle negative or invalid inputs gracefully (e.g., if the user enters an invalid menu choice, display an error message, then re-display the menu).

Here is an example of the program output:

```
Menu:
1. Compute Summation
2. Compute Product
3. Compute Power (Baseˆ Exponent)
4. Exit
Enter your choice: 1

Enter starting integer (a): 2
Enter ending integer (b): 5
Summation of numbers from 2 to 5 = 14

Menu:
1. Compute Summation
2. Compute Product
3. Compute Power (Baseˆ Exponent)
4. Exit
Enter your choice: 3

Enter base (B): 2
Enter exponent (E): 5
2ˆ5 = 32

Menu:
1. Compute Summation
2. Compute Product
3. Compute Power (Baseˆ Exponent)
4. Exit
Enter your choice: 4

Exiting program...
```

ANS:

```
#include <stdio.h>
#include <math.h>   // Needed for pow()

int main() {
   int choice = 0;

   do {
      // Display Menu
      printf("\nMenu:\n");
      printf("1. Compute Summation\n");
      printf("2. Compute Product\n");
      printf("3. Compute Power (Baseˆ Exponent)\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");

      if (scanf("%d", &choice) != 1) {
         // Handle invalid (non-integer) input
```

```
    printf("Invalid input. Exiting.\n");
    return 1;
}

switch (choice) {
    case 1: {
        // Summation
        int a, b;
        printf("Enter starting integer (a): ");
        scanf("%d", &a);
        printf("Enter ending integer (b): ");
        scanf("%d", &b);

        int sum = 0;
        int start = (a < b) ? a : b;
        int end   = (a < b) ? b : a;

        for (int i = start; i <= end; i++) {
            sum += i;
        }

        printf("Summation of numbers from %d to %d = %d\n", a, b, sum);
        break;
    }
    case 2: {
        // Product
        int x, y;
        printf("Enter starting integer (x): ");
        scanf("%d", &x);
        printf("Enter ending integer (y): ");
        scanf("%d", &y);

        long long product = 1;  // 64-bit to reduce risk of overflow
        int start = (x < y) ? x : y;
        int end   = (x < y) ? y : x;

        // If the range includes 0, product is 0
        if (start <= 0 && end >= 0) {
            product = 0;
        } else {
            for (int i = start; i <= end; i++) {
                product *= i;
            }
        }

        printf("Product of numbers from %d to %d = %lld\n", x, y, product);
        break;
    }
    case 3: {
        // BaseˆExponent Power
        // If you want to handle negative exponents,
        // consider storing the result in a double variable.
        int base, exponent;
        printf("Enter base (B): ");
        scanf("%d", &base);
        printf("Enter exponent (E): ");
```

```
            scanf("%d", &exponent);

            // For integer exponents, result can be stored as double
            // to handle negative exponents as well.
            double result = pow((double)base, (double)exponent);

            // If exponent >= 0, you can also store result in a long long, but negative
exponents
            // require floating-point representation.
            printf("%dˆ%d = %.2f\n", base, exponent, result);
            break;
        }
        case 4:
            // Exit
            printf("Exiting program...\n");
            break;

        default:
            printf("Invalid choice. Please try again.\n");
            break;
    }

  } while (choice != 4);

  return 0;
}
```

Q2 [Python] (22 pts): **Text Analysis with Sentinel-Controlled Input**: Write a Python script that reads multiple lines of text from the user, analyzes each line, and then prints an aggregated summary once the user is finished. Your program must:

1. Continuously prompt the user to enter a line of text.
2. End (break out of the input loop) as soon as the user enters an **empty line** (i.e., presses Enter without typing anything). Treat this empty line as the sentinel value.
3. For each non-empty line of text:
    - Count the total number of characters in that line (excluding any trailing newline).
    - Count how many of those characters are vowels (`a, e, i, o, u` in either lowercase or uppercase)

Here is an example of the program output:

```
Enter a line of text (press Enter on an empty line to stop):
Hello, World!
Enter a line of text (press Enter on an empty line to stop):
Programming in Python is fun.
Enter a line of text (press Enter on an empty line to stop):
        # user presses Enter immediately here, no text typed
```

```
--- Summary Report ---
Line 1:
  Total characters: 13
  Total vowels: 3
Line 2:
  Total characters: 29
  Total vowels: 8

Highest number of vowels in a single line: 8
Thank you for using the Text Analysis Tool!
```

ANS:

```
def main():
    lines = []

    while True:
        user_input = input("Enter a line of text (press Enter on an empty line to stop): ")
        # Sentinel condition: empty line -> break
        if user_input == "":
            break
        lines.append(user_input)

    print("\n--- Summary Report ---")
    highest_vowels = 0

    for i, line in enumerate(lines, start=1):
        char_count = len(line)
        vowel_count = 0
        for ch in line:
            if ch.lower() in ['a', 'e', 'i', 'o', 'u']:
                vowel_count += 1

        if vowel_count > highest_vowels:
            highest_vowels = vowel_count

        print(f"Line {i}:")
        print(f"  Total characters: {char_count}")
        print(f"  Total vowels: {vowel_count}")

    print(f"\nHighest number of vowels in a single line: {highest_vowels}")
    print("Thank you for using the Text Analysis Tool!")

if __name__ == "__main__":
    main()
```