C Programming Problems – WEBSCTE (2025)

A comprehensive collection of 43 C programming exercises covering fundamental to advanced concepts, organized in 6 sections with detailed implementations.

Table of Contents

Section 1: Format Specifiers

i) Number Format Display

Question:
Write a C program to display a number in hexadecimal, decimal, and octal formats.

Description:
This program takes an integer input from the user and displays it in three different number formats: Hexadecimal (base 16), Decimal (base 10), and Octal (base 8).

How to Solve:

1. Declare an integer variable to store the input number
2. Use scanf() to read the number from user
3. Use printf() with format specifiers:
   - %d or %i for decimal
   - %x for hexadecimal (lowercase)
   - %X for hexadecimal (uppercase)
   - %o for octal
4. Display the number in all three formats

Code:

```c
#include <stdio.h>

int main() {
   int number;

   printf("Enter an integer number: ");
   scanf("%d", &number);

   printf("\n--- Number Format Display ---\n");
   printf("Decimal format:           %d\n", number);
   printf("Hexadecimal (lowercase):  %x\n", number);
   printf("Hexadecimal (uppercase):  %X\n", number);
   printf("Octal format:             %o\n", number);

   // With prefixes (0x for hex, 0 for octal)
   printf("\n--- With Prefixes ---\n");
   printf("Hexadecimal with prefix:  %#x\n", number);
   printf("Hexadecimal with prefix:  %#X\n", number);
   printf("Octal with prefix:        %#o\n", number);

   return 0;
}
```

Example:

```
Input: 255
Output:
--- Number Format Display ---
Decimal format:           255
Hexadecimal (lowercase):  ff
Hexadecimal (uppercase):  FF
Octal format:             377

--- With Prefixes ---
Hexadecimal with prefix:  0xff
Hexadecimal with prefix:  0XFF
Octal with prefix:        0377
```

Back to Table of Contents

ii) Leading and Trailing Zeros

Question:
Write a C program to display numbers with leading zeros and trailing zeros.

Description:
This program demonstrates how to format numbers with leading zeros (padding zeros before the number) and trailing zeros (displaying decimal numbers with specific decimal places).

How to Solve:

1. For leading zeros with integers:
   - Use %0nd format specifier where n is the total width
   - Example: %05d will display number with 5 digits, padding with zeros
2. For trailing zeros with floating-point numbers:
   - Use %.nf format specifier where n is decimal places
   - Example: %.3f will display 3 digits after decimal point
3. Combine both for complete formatting control

Code:

```c
#include <stdio.h>

int main() {
    int intNumber;
    float floatNumber;

    printf("Enter an integer number: ");
    scanf("%d", &intNumber);

    printf("Enter a floating-point number: ");
    scanf("%f", &floatNumber);

    printf("\n--- Leading Zeros (Integer) ---\n");
    printf("Original number:          %d\n", intNumber);
    printf("With 5 digits (leading):  %05d\n", intNumber);
    printf("With 8 digits (leading):  %08d\n", intNumber);
    printf("With 10 digits (leading): %010d\n", intNumber);

    printf("\n--- Trailing Zeros (Float) ---\n");
    printf("Original number:          %f\n", floatNumber);
    printf("With 2 decimal places:    %.2f\n", floatNumber);
    printf("With 4 decimal places:    %.4f\n", floatNumber);
    printf("With 6 decimal places:    %.6f\n", floatNumber);

    printf("\n--- Combined (Leading + Trailing) ---\n");
    printf("Width 10, 2 decimals:     %10.2f\n", floatNumber);
    printf("Width 12, 3 decimals:     %012.3f\n", floatNumber);
    printf("Width 15, 5 decimals:     %015.5f\n", floatNumber);

    return 0;
}
```

Example:

```
Input:
  Integer: 42
  Float: 3.14
Output:
--- Leading Zeros (Integer) ---
Original number:          42
With 5 digits (leading):  00042
With 8 digits (leading):  00000042
With 10 digits (leading): 0000000042

--- Trailing Zeros (Float) ---
Original number:          3.140000
With 2 decimal places:    3.14
With 4 decimal places:    3.1400
With 6 decimal places:    3.140000

--- Combined (Leading + Trailing) ---
Width 10, 2 decimals:           3.14
Width 12, 3 decimals:     0003.140
Width 15, 5 decimals:     00003.14000
```

Back to Table of Contents

iii) Left and Right Justification

Question:
Write a C program to display numbers with left and right justification.

Description:
This program demonstrates text alignment (justification) in C: Right justification (number is aligned to the right - default) and Left justification (number is aligned to the left using minus sign).

How to Solve:

1. For right justification (default):
    - Use %nd format specifier where n is the field width
    - Example: %10d will right-align in 10 character width
2. For left justification:
    - Use %-nd format specifier (negative sign for left align)
    - Example: %-10d will left-align in 10 character width
3. Can combine with precision for floating-point numbers
4. Use vertical bars (|) to visualize the alignment clearly

Code:

```
#include <stdio.h>

int main() {
   int intNumber;
   float floatNumber;
   char text[50];

   printf("Enter an integer number: ");
   scanf("%d", &intNumber);

   printf("Enter a floating-point number: ");
   scanf("%f", &floatNumber);

   printf("Enter a text string: ");
   scanf("%s", text);

   printf("\n--- Right Justification (Default) ---\n");
   printf("|%15d| (Integer with width 15)\n", intNumber);
   printf("|%15.2f| (Float with width 15, 2 decimals)\n", floatNumber);
   printf("|%20s| (String with width 20)\n", text);

   printf("\n--- Left Justification (Using -) ---\n");
   printf("|%-15d| (Integer with width 15)\n", intNumber);
   printf("|%-15.2f| (Float with width 15, 2 decimals)\n", floatNumber);
   printf("|%-20s| (String with width 20)\n", text);

   printf("\n--- Comparison Side by Side ---\n");
   printf("RIGHT: |%12d| LEFT: |%-12d|\n", intNumber, intNumber);
   printf("RIGHT: |%12.3f| LEFT: |%-12.3f|\n", floatNumber, floatNumber);
   printf("RIGHT: |%15s| LEFT: |%-15s|\n", text, text);

   return 0;
}
```

Example:

```
Input:
  Integer: 12345
  Float: 67.89
  String: Hello
Output:
--- Right Justification (Default) ---
|          12345| (Integer with width 15)
|          67.89| (Float with width 15, 2 decimals)
|               Hello| (String with width 20)
```

```
--- Left Justification (Using -) ---
|12345          | (Integer with width 15)
|67.89          | (Float with width 15, 2 decimals)
|Hello               | (String with width 20)

--- Comparison Side by Side ---
RIGHT: |       12345| LEFT: |12345       |
RIGHT: |       67.890| LEFT: |67.890      |
RIGHT: |          Hello| LEFT: |Hello          |
```

Back to Table of Contents

iv) Different Format Specifiers

Question:
Write a C program to demonstrate different formatting specifiers in C.

Description:
This program showcases various format specifiers available in C's printf() function including integer, floating-point, character, string, and pointer specifiers with special formatting options.

How to Solve:

1. Declare variables of different data types
2. Use appropriate format specifiers for each type:
   - %d or %i: signed decimal integer
   - %u: unsigned decimal integer
   - %f: floating-point number
   - %e/%E: scientific notation
   - %g/%G: shortest representation
   - %c: character
   - %s: string
   - %p: pointer address
3. Combine specifiers with width, precision, and flags

Code:

```c
#include <stdio.h>

int main() {
   int decimalNum;
   unsigned int unsignedNum;
   long longNum;
   float floatNum;
   double doubleNum;
   char character;
   char string[100];

   printf("Enter an integer: ");
   scanf("%d", &decimalNum);
   printf("Enter an unsigned integer: ");
   scanf("%u", &unsignedNum);
   printf("Enter a long integer: ");
   scanf("%ld", &longNum);
   printf("Enter a float number: ");
   scanf("%f", &floatNum);
   printf("Enter a double number: ");
   scanf("%lf", &doubleNum);
   printf("Enter a character: ");
   scanf(" %c", &character);
   printf("Enter a string: ");
   scanf("%s", string);

   printf("\n=== INTEGER FORMAT SPECIFIERS ===\n");
   printf("%%d (decimal):              %d\n", decimalNum);
   printf("%%i (integer):              %i\n", decimalNum);
   printf("%%u (unsigned):             %u\n", unsignedNum);
   printf("%%ld (long decimal):        %ld\n", longNum);
   printf("%%5d (width 5):             %5d\n", decimalNum);
   printf("%%-5d (left-aligned):       %-5d|\n", decimalNum);
   printf("%%05d (zero-padded):        %05d\n", decimalNum);

   printf("\n=== FLOATING-POINT FORMAT SPECIFIERS ===\n");
   printf("%%f (float):                %f\n", floatNum);
   printf("%%lf (double):              %lf\n", doubleNum);
   printf("%%.2f (2 decimals):         %.2f\n", floatNum);
   printf("%%.4f (4 decimals):         %.4f\n", doubleNum);
   printf("%%e (scientific):           %e\n", floatNum);
   printf("%%E (SCIENTIFIC):           %E\n", doubleNum);

   printf("\n=== CHARACTER & STRING FORMAT SPECIFIERS ===\n");
   printf("%%c (character):            %c\n", character);
   printf("%%d (char as ASCII):        %d\n", character);
   printf("%%s (string):               %s\n", string);
   printf("%%10s (width 10):           %10s\n", string);
   printf("%%-10s (left-aligned):      %-10s|\n", string);

   return 0;
}
```

Example:

```
Input:
  Integer: 42
  Unsigned: 100
  Long: 999999
  Float: 3.14159
  Double: 2.71828
  Character: A
  String: Hello
Output:
=== INTEGER FORMAT SPECIFIERS ===
%d (decimal):              42
%i (integer):              42
%u (unsigned):             100
%ld (long decimal):        999999
%5d (width 5):                42
%-5d (left-aligned):       42   |
%05d (zero-padded):        00042

=== FLOATING-POINT FORMAT SPECIFIERS ===
%f (float):                3.141590
%lf (double):              2.718280
%.2f (2 decimals):         3.14
%.4f (4 decimals):         2.7183
%e (scientific):           3.141590e+00
%E (SCIENTIFIC):           2.718280E+00
```

Back to Table of Contents

Section 2: Control Structures

v) Greatest and Smallest of Three Numbers

Question:
Write a C program to find the greatest and smallest of three numbers.

Description:
This program takes three numbers as input from the user and determines which number is the greatest (maximum) and which is the smallest (minimum) among them using conditional statements.

How to Solve:

1. Declare three variables to store the input numbers
2. Read three numbers from the user
3. Use nested if-else statements or logical operators to compare:
   - For greatest: Compare each number with others using > operator
   - For smallest: Compare each number with others using < operator
4. Alternative approach: Use ternary operator (?😃 for compact code
5. Display the greatest and smallest numbers

Code:

```c
#include <stdio.h>

int main() {
   float num1, num2, num3;
   float greatest, smallest;

   printf("Enter three numbers: ");
   scanf("%f %f %f", &num1, &num2, &num3);

   // Find greatest
   if (num1 >= num2 && num1 >= num3) {
      greatest = num1;
   } else if (num2 >= num1 && num2 >= num3) {
      greatest = num2;
   } else {
      greatest = num3;
   }

   // Find smallest
   if (num1 <= num2 && num1 <= num3) {
      smallest = num1;
   } else if (num2 <= num1 && num2 <= num3) {
      smallest = num2;
   } else {
      smallest = num3;
   }

   printf("\n--- Results ---\n");
   printf("Numbers entered: %.2f, %.2f, %.2f\n", num1, num2, num3);
   printf("Greatest number: %.2f\n", greatest);
   printf("Smallest number: %.2f\n", smallest);
   printf("Difference: %.2f\n", greatest - smallest);

   return 0;
}
```

Example:

```
Input: 25 10 30
Output:
--- Results ---
Numbers entered: 25.00, 10.00, 30.00
Greatest number: 30.00
Smallest number: 10.00
Difference: 20.00
```

Back to Table of Contents

vi) Grade Classification

Question:
Write a C program to display pass class, second class, or distinction according to the marks entered from the keyboard.

Description:
This program takes marks as input and classifies the result into different categories: Distinction (≥75), First Class (60-74), Second Class (50-59), Pass Class (40-49), or Fail (<40).

How to Solve:

1. Declare a variable to store marks (integer or float)
2. Read marks from the user
3. Validate that marks are in valid range (0-100)
4. Use if-else ladder to check marks range and classify
5. Display the appropriate classification message

Code:

```c
#include <stdio.h>

int main() {
   float marks;

   printf("Enter marks obtained (0-100): ");
   scanf("%f", &marks);

   if (marks < 0 || marks > 100) {
      printf("\nError: Invalid marks! Marks should be between 0 and 100.\n");
      return 1;
   }

   printf("\n--- Result Classification ---\n");
   printf("Marks obtained: %.2f\n", marks);

   if (marks >= 75) {
      printf("Grade: DISTINCTION\n");
      printf("Performance: Excellent! Outstanding achievement.\n");
   } else if (marks >= 60) {
      printf("Grade: FIRST CLASS\n");
      printf("Performance: Very Good! Keep up the good work.\n");
   } else if (marks >= 50) {
      printf("Grade: SECOND CLASS\n");
      printf("Performance: Good! There's room for improvement.\n");
   } else if (marks >= 40) {
      printf("Grade: PASS CLASS\n");
      printf("Performance: Passed! Work harder next time.\n");
   } else {
      printf("Grade: FAIL\n");
      printf("Performance: Failed! Need to study more.\n");
   }

   return 0;
}
```

Example:

```
Input: 85
Output:
--- Result Classification ---
Marks obtained: 85.00
Grade: DISTINCTION
Performance: Excellent! Outstanding achievement.
```

Back to Table of Contents

vii) Even or Odd Checker

Question:
Write a C program to find whether a number is even or odd.

Description:
This program takes an integer as input and determines whether it is an even number or an odd number. A number is even if it is divisible by 2 (remainder is 0), otherwise it is odd.

How to Solve:

1. Declare an integer variable to store the number
2. Read the number from user input
3. Use modulus operator (%) to find remainder when divided by 2:

   - If number % 2 == 0, the number is EVEN
   - If number % 2 != 0, the number is ODD
4. Use if-else statement to check the condition
5. Display whether the number is even or odd

Code:

```c
#include <stdio.h>

int main() {
   int number;

   printf("Enter an integer number: ");
   scanf("%d", &number);

   printf("\n--- Even/Odd Checker ---\n");
   printf("Number entered: %d\n", number);

   if (number % 2 == 0) {
      printf("Result: %d is an EVEN number.\n", number);
      printf("Explanation: %d is divisible by 2 (remainder = 0)\n", number);
   } else {
      printf("Result: %d is an ODD number.\n", number);
      printf("Explanation: %d is not divisible by 2 (remainder = 1)\n", number);
   }

   printf("\nMathematical Info:\n");
   printf("%d ÷ 2 = %d (quotient), remainder = %d\n", number, number / 2, number % 2);

   return 0;
}
```

Example:

```
Input: 10
Output:
--- Even/Odd Checker ---
Number entered: 10
Result: 10 is an EVEN number.
Explanation: 10 is divisible by 2 (remainder = 0)

Mathematical Info:
10 ÷ 2 = 5 (quotient), remainder = 0
```

Back to Table of Contents

viii) Number Spelling

Question:
Write a C program to display spellings of numbers 1-10 on entry.

Description:
This program takes a number (1-10) as input from the user and displays its spelling in words. For example, if user enters 5, it displays "Five". If the number is outside the range 1-10, it shows an error message.

How to Solve:

1. Declare an integer variable to store the number
2. Read the number from user input
3. Use switch-case statement to match the number
4. Create cases for 1 to 10, each printing the corresponding spelling
5. Include a default case for invalid input

Code:

```c
#include <stdio.h>

int main() {
   int number;

   printf("Enter a number (1-10): ");
   scanf("%d", &number);

   printf("\n--- Number to Spelling Converter ---\n");
   printf("Number entered: %d\n", number);
   printf("Spelling: ", number);

   switch (number) {
   case 1:
      printf("One\n");
      break;
   case 2:
      printf("Two\n");
      break;
   case 3:
      printf("Three\n");
      break;
   case 4:
      printf("Four\n");
```

```
            break;
        case 5:
            printf("Five\n");
            break;
        case 6:
            printf("Six\n");
            break;
        case 7:
            printf("Seven\n");
            break;
        case 8:
            printf("Eight\n");
            break;
        case 9:
            printf("Nine\n");
            break;
        case 10:
            printf("Ten\n");
            break;
        default:
            printf("Invalid!\n");
            printf("\nError: Please enter a number between 1 and 10 only.\n");
            return 1;
    }

    return 0;
}
```

Example:

```
Input: 5
Output:
--- Number to Spelling Converter ---
Number entered: 5
Spelling: Five
```

Back to Table of Contents

ix) Calculator Menu

Question:
Write a C program to implement and display a menu to execute 1. ADD, 2. SUBTRACT, 3. MULTIPLICATION, 4. DIVISION using switch case.

Description:
This program creates a simple calculator with a menu-driven interface. It displays a menu with 4 arithmetic operations, user selects an operation, enters two numbers, and the program performs the selected operation using switch-case.

How to Solve:

1. Display a menu with operation choices (1-4)
2. Read user's choice
3. Read two numbers from the user
4. Use switch-case statement with choice as the selector
5. Perform the selected operation and display result
6. Include error handling for division by zero

Code:

```
#include <stdio.h>

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

    printf("========================================\n");
    printf("       CALCULATOR MENU\n");
    printf("========================================\n");
    printf("1. ADD\n");
    printf("2. SUBTRACT\n");
    printf("3. MULTIPLICATION\n");
    printf("4. DIVISION\n");
    printf("========================================\n");
    printf("Enter your choice (1-4): ");
    scanf("%d", &choice);

    printf("Enter first number: ");
    scanf("%f", &num1);
    printf("Enter second number: ");
    scanf("%f", &num2);

    printf("\n--- Calculation Result ---\n");

    switch (choice) {
    case 1:
        result = num1 + num2;
        printf("Operation: ADDITION\n");
        printf("%.2f + %.2f = %.2f\n", num1, num2, result);
        break;
    case 2:
        result = num1 - num2;
        printf("Operation: SUBTRACTION\n");
```

```
            printf("%.2f - %.2f = %.2f\n", num1, num2, result);
            break;
        case 3:
            result = num1 * num2;
            printf("Operation: MULTIPLICATION\n");
            printf("%.2f × %.2f = %.2f\n", num1, num2, result);
            break;
        case 4:
            if (num2 == 0) {
                printf("Error: Division by zero!\n");
                return 1;
            }
            result = num1 / num2;
            printf("Operation: DIVISION\n");
            printf("%.2f ÷ %.2f = %.2f\n", num1, num2, result);
            break;
        default:
            printf("Error: Invalid choice!\n");
            return 1;
    }

    return 0;
}
```

Example:

```
Input:
  Choice: 1
  First number: 25
  Second number: 15
Output:
========================================
       CALCULATOR MENU
========================================
1. ADD
2. SUBTRACT
3. MULTIPLICATION
4. DIVISION
========================================
Enter your choice (1-4): 1

--- Calculation Result ---
Operation: ADDITION
25.00 + 15.00 = 40.00
```

Back to Table of Contents

x) Quadratic Equation Roots

Question:
Write a C program to check whether there exist real roots of a quadratic equation and if they exist, find them.

Description:
This program solves a quadratic equation of the form $ax^2 + bx + c = 0$. It checks if real roots exist by calculating the discriminant ($D = b^2 - 4ac$), then finds the roots if they exist using the quadratic formula.

How to Solve:

1. Input coefficients a, b, and c from the user
2. Check if a != 0 (otherwise it's not quadratic)
3. Calculate discriminant: $D = b^2 - 4ac$
4. Use if-else to check discriminant value:
    - If D > 0: Two distinct real roots
    - If D = 0: Two equal real roots
    - If D < 0: No real roots (complex)
5. Use sqrt() function from math.h library

Code:

```
#include <math.h>
#include <stdio.h>

int main() {
    float a, b, c;
    float discriminant, root1, root2, realPart, imagPart;

    printf("Quadratic Equation: ax² + bx + c = 0\n");
    printf("========================================\n");
    printf("Enter coefficient a: ");
    scanf("%f", &a);
    printf("Enter coefficient b: ");
    scanf("%f", &b);
    printf("Enter coefficient c: ");
    scanf("%f", &c);

    if (a == 0) {
        printf("\nError: 'a' cannot be zero for a quadratic equation!\n");
        return 1;
    }

    discriminant = (b * b) - (4 * a * c);
    printf("\nDiscriminant (D) = %.2f\n\n", discriminant);

    if (discriminant > 0) {
       printf("Nature: Two DISTINCT REAL roots exist\n\n");
       root1 = (-b + sqrt(discriminant)) / (2 * a);
       root2 = (-b - sqrt(discriminant)) / (2 * a);
       printf("Root 1 = %.2f\n", root1);
       printf("Root 2 = %.2f\n", root2);
    } else if (discriminant == 0) {
       printf("Nature: Two EQUAL REAL roots exist\n\n");
       root1 = -b / (2 * a);
       printf("Root 1 = Root 2 = %.2f\n", root1);
    } else {
       printf("Nature: NO REAL roots exist\n");
       printf("The equation has COMPLEX roots.\n\n");
       realPart = -b / (2 * a);
       imagPart = sqrt(-discriminant) / (2 * a);
       printf("Root 1 = %.2f + %.2fi\n", realPart, imagPart);
       printf("Root 2 = %.2f - %.2fi\n", realPart, imagPart);
    }

    return 0;
}
```

Example:

```
Input:
  a = 1
  b = -5
  c = 6
Output:
Quadratic Equation: ax² + bx + c = 0
========================================
Discriminant (D) = 1.00

Nature: Two DISTINCT REAL roots exist

Root 1 = 3.00
Root 2 = 2.00
```

Back to Table of Contents

Section 3: Loops & Patterns

xi) College Name Display

Question: Write a C program to print your college name 20 times using:

1. for loop
2. while loop
3. do-while loop

Description: This program demonstrates three different loop structures in C by displaying the same message multiple times. It helps understand:

- Loop initialization, condition, and increment
- Differences between for, while, and do-while loops
- When to use each loop type

How to Solve:

1. Use a for loop with counter from 1 to 20
2. Use a while loop with counter initialization before loop
3. Use a do-while loop that executes at least once
4. Display the college name in each iteration

Code:

```
#include <stdio.h>

#define COLLEGE_NAME "XYZ College of Engineering"

int main() {
    int i;

    printf("\n=== Using FOR Loop ===\n");
    for (i = 1; i <= 20; i++) {
       printf("%d. %s\n", i, COLLEGE_NAME);
    }

    printf("\n=== Using WHILE Loop ===\n");
    i = 1;
    while (i <= 20) {
       printf("%d. %s\n", i, COLLEGE_NAME);
       i++;
    }

    printf("\n=== Using DO-WHILE Loop ===\n");
    i = 1;
    do {
       printf("%d. %s\n", i, COLLEGE_NAME);
       i++;
    } while (i <= 20);

    return 0;
}
```

Example:

```
Output:
=== Using FOR Loop ===
1. XYZ College of Engineering
2. XYZ College of Engineering
3. XYZ College of Engineering
...
20. XYZ College of Engineering

=== Using WHILE Loop ===
1. XYZ College of Engineering
2. XYZ College of Engineering
...
20. XYZ College of Engineering

=== Using DO-WHILE Loop ===
1. XYZ College of Engineering
2. XYZ College of Engineering
...
20. XYZ College of Engineering
```

Back to Table of Contents

xii) Break & Continue Demo

Question:
Write a C program to demonstrate the working of `break` and `continue` statements with examples.

Description:
This program showcases the usage of break and continue statements in loops through 5 different demonstrations:

1. Break in for loop - exits when condition met
2. Continue in for loop - skips iteration
3. Break in nested loop - exits inner loop
4. Continue with multiple conditions - complex skipping
5. Break in while loop - early termination

How to Solve:

1. Create example loops where break/continue make sense
2. Demonstrate break to exit loops early
3. Demonstrate continue to skip specific iterations
4. Show effects on nested loops
5. Print clear output explaining each action

Code:

```c
#include <stdio.h>

int main() {
    int i, j;

    // Demo 1: Break in for loop
    printf("Demo 1: Break in for loop\n");
    printf("Print numbers 1-10 but stop at 6\n");
    for (i = 1; i <= 10; i++) {
        if (i == 6) {
            printf("Breaking at i = %d\n", i);
            break;
        }
        printf("%d ", i);
    }
    printf("\n\n");

    // Demo 2: Continue in for loop
    printf("Demo 2: Continue in for loop\n");
    printf("Print numbers 1-10 but skip 5\n");
    for (i = 1; i <= 10; i++) {
        if (i == 5) {
            printf("(Skipping %d) ", i);
            continue;
        }
        printf("%d ", i);
    }
    printf("\n\n");

    // Demo 3: Break in nested loop
    printf("Demo 3: Break in nested loop\n");
    printf("Multiplication table but break inner loop at product 15\n");
    for (i = 1; i <= 5; i++) {
        printf("Row %d: ", i);
        for (j = 1; j <= 5; j++) {
            if (i * j >= 15) {
                printf("[Break at %d*%d=%d]", i, j, i*j);
                break;
            }
            printf("%d ", i * j);
        }
        printf("\n");
    }
    printf("\n");

    // Demo 4: Continue with multiple conditions
    printf("Demo 4: Continue skipping multiples of 3 and 5\n");
    printf("Numbers 1-20, skip multiples of 3 and 5\n");
    for (i = 1; i <= 20; i++) {
        if (i % 3 == 0 || i % 5 == 0) {
            continue;
        }
        printf("%d ", i);
    }
    printf("\n\n");

    // Demo 5: Break in while loop
    printf("Demo 5: Break in while loop\n");
    printf("Sum numbers until sum exceeds 50\n");
    i = 1;
    int sum = 0;
    while (1) { // Infinite loop
        sum += i;
        printf("Adding %d, Sum = %d\n", i, sum);
        if (sum > 50) {
            printf("Sum exceeded 50, breaking!\n");
            break;
        }
        i++;
    }

    return 0;
}
```

Example:

```
Output:
Demo 1: Break in for loop
Print numbers 1-10 but stop at 6
1 2 3 4 5 Breaking at i = 6

Demo 2: Continue in for loop
Print numbers 1-10 but skip 5
1 2 3 4 (Skipping 5) 6 7 8 9 10

Demo 3: Break in nested loop
Multiplication table but break inner loop at product 15
Row 1: 1 2 3 4 5
Row 2: 2 4 6 8 10
Row 3: 3 6 9 12 [Break at 3*5=15]
Row 4: 4 8 12 [Break at 4*4=16]
Row 5: 5 10 [Break at 5*3=15]
```

Back to Table of Contents

xiii) Sum of Natural, Even & Odd Numbers

Question:
Write a C program to find the sum of:

1. Natural numbers from 1 to N
2. Even numbers from 1 to N
3. Odd numbers from 1 to N

Implement using for loop, while loop, and mathematical formula.

Description:
This program calculates three types of sums using different approaches:

- For loop implementation
- While loop implementation
- Direct mathematical formulas

It demonstrates that sometimes mathematical formulas are more efficient than loops.

How to Solve:

1. For natural numbers: Sum = 1+2+3+...+N = N×(N+1)/2
2. For even numbers: Sum = 2+4+6+...+(last even) = n×(n+1) where n=N/2
3. For odd numbers: Sum = 1+3+5+...+(last odd) = n^2 where n=(N+1)/2
4. Implement using loops for verification
5. Compare results from loops vs formulas

Code:

```
#include <stdio.h>

int main() {
    int N, i;
    long sumNatural = 0, sumEven = 0, sumOdd = 0;
    long formulaNatural, formulaEven, formulaOdd;
    int countEven, countOdd;

    printf("Enter N: ");
    scanf("%d", &N);

    if (N <= 0) {
       printf("Error: N must be positive!\n");
       return 1;
    }

    // Method 1: Using FOR loop
    printf("\n=== Method 1: Using FOR Loop ===\n");
    for (i = 1; i <= N; i++) {
       sumNatural += i;
       if (i % 2 == 0) {
          sumEven += i;
       } else {
          sumOdd += i;
       }
    }
    printf("Sum of Natural numbers (1 to %d) = %ld\n", N, sumNatural);
    printf("Sum of Even numbers (1 to %d) = %ld\n", N, sumEven);
    printf("Sum of Odd numbers (1 to %d) = %ld\n", N, sumOdd);

    // Method 2: Using WHILE loop
    printf("\n=== Method 2: Using WHILE Loop ===\n");
    sumNatural = sumEven = sumOdd = 0;
    i = 1;
    while (i <= N) {
       sumNatural += i;
       if (i % 2 == 0) {
          sumEven += i;
       } else {
          sumOdd += i;
       }
       i++;
    }
    printf("Sum of Natural numbers (1 to %d) = %ld\n", N, sumNatural);
    printf("Sum of Even numbers (1 to %d) = %ld\n", N, sumEven);
    printf("Sum of Odd numbers (1 to %d) = %ld\n", N, sumOdd);

    // Method 3: Using Mathematical Formula
    printf("\n=== Method 3: Using Formula ===\n");

    // Natural numbers: Sum = N×(N+1)/2
    formulaNatural = (long)N * (N + 1) / 2;

    // Even numbers: Sum = n×(n+1) where n = N/2
    countEven = N / 2;
    formulaEven = (long)countEven * (countEven + 1);

    // Odd numbers: Sum = n² where n = (N+1)/2
    countOdd = (N + 1) / 2;
    formulaOdd = (long)countOdd * countOdd;

    printf("Sum of Natural numbers = %d×(%d+1)/2 = %ld\n", N, N, formulaNatural);
    printf("Sum of Even numbers = %d×(%d+1) = %ld\n", countEven, countEven, formulaEven);
    printf("Sum of Odd numbers = %d² = %ld\n", countOdd, formulaOdd);

    return 0;
}
```

Example 1:

```
Input:
  N = 10

Output:
=== Method 1: Using FOR Loop ===
Sum of Natural numbers (1 to 10) = 55
Sum of Even numbers (1 to 10) = 30
Sum of Odd numbers (1 to 10) = 25

=== Method 2: Using WHILE Loop ===
Sum of Natural numbers (1 to 10) = 55
Sum of Even numbers (1 to 10) = 30
Sum of Odd numbers (1 to 10) = 25

=== Method 3: Using Formula ===
Sum of Natural numbers = 10×(10+1)/2 = 55
Sum of Even numbers = 5×(5+1) = 30
Sum of Odd numbers = 5² = 25
```

Example 2:

```
Input:
  N = 100

Output:
=== Method 3: Using Formula ===
Sum of Natural numbers = 100×(100+1)/2 = 5050
Sum of Even numbers = 50×(50+1) = 2550
Sum of Odd numbers = 50² = 2500
```

Back to Table of Contents

xiv) GCD & LCM Calculator

Question:
Write a C program to find the GCD (Greatest Common Divisor) and LCM (Least Common Multiple) of two numbers using the Euclidean algorithm.

Description:
This program implements:

- **GCD** using Euclidean algorithm: Repeatedly divide and take remainder until remainder is 0
- **LCM** using the formula: LCM = (a × b) / GCD(a, b)
- Step-by-step display of the algorithm execution

How to Solve:

1. Read two numbers a and b
2. Apply Euclidean algorithm for GCD:
    - While b ≠ 0: temp = b, b = a % b, a = temp
    - GCD = a
3. Calculate LCM using formula: LCM = (original_a × original_b) / GCD
4. Display step-by-step process

Code:

```c
#include <stdio.h>

int main() {
   int a, b, num1, num2, temp, gcd, lcm;
   int step = 1;

   printf("Enter two positive integers: ");
   scanf("%d %d", &a, &b);

   if (a <= 0 || b <= 0) {
      printf("Error: Both numbers must be positive!\n");
      return 1;
   }

   // Store original values
   num1 = a;
   num2 = b;

   printf("\nFinding GCD of %d and %d using Euclidean Algorithm:\n", num1, num2);
   printf("====================================================\n");

   // Euclidean Algorithm for GCD
   while (b != 0) {
      printf("Step %d: GCD(%d, %d)\n", step, a, b);
      printf("        %d = %d × %d + %d\n", a, b, a/b, a%b);
      temp = b;
      b = a % b;
      a = temp;
      step++;
   }

   gcd = a;
   printf("\nGCD = %d\n", gcd);

   // Calculate LCM using formula: LCM = (a × b) / GCD
   lcm = (num1 * num2) / gcd;

   printf("\nCalculating LCM:\n");
   printf("================\n");
   printf("LCM = (a × b) / GCD\n");
   printf("LCM = (%d × %d) / %d\n", num1, num2, gcd);
   printf("LCM = %d / %d\n", num1 * num2, gcd);
   printf("LCM = %d\n", lcm);

   printf("\n=== RESULT ===\n");
   printf("GCD(%d, %d) = %d\n", num1, num2, gcd);
   printf("LCM(%d, %d) = %d\n", num1, num2, lcm);

   // Verification
   printf("\nVerification: GCD × LCM = %d × %d = %d\n", gcd, lcm, gcd * lcm);
   printf("              a × b = %d × %d = %d ✓\n", num1, num2, num1 * num2);

   return 0;
}
```

Example 1:

```
Input:
  Two numbers: 48 18

Output:
Finding GCD of 48 and 18 using Euclidean Algorithm:
```

```
=====================================================
Step 1: GCD(48, 18)
        48 = 18 × 2 + 12
Step 2: GCD(18, 12)
        18 = 12 × 1 + 6
Step 3: GCD(12, 6)
        12 = 6 × 2 + 0

GCD = 6

Calculating LCM:
================
LCM = (a × b) / GCD
LCM = (48 × 18) / 6
LCM = 864 / 6
LCM = 144

=== RESULT ===
GCD(48, 18) = 6
LCM(48, 18) = 144

Verification: GCD × LCM = 6 × 144 = 864
              a × b = 48 × 18 = 864 ✓
```

Example 2:

```
Input:
  Two numbers: 24 36

Output:
GCD(24, 36) = 12
LCM(24, 36) = 72
```

Back to Table of Contents

xv) Number Triangle Patterns

Question:
Write a C program to print 6 different number triangle patterns:

1. Sequential numbers
2. Row-wise repeated numbers
3. Column-wise numbers
4. Pyramid of numbers
5. Inverted number pyramid
6. Floyd's triangle

Description:
This program demonstrates nested loops to create various number patterns. Each pattern uses different logic for printing numbers based on row and column positions.

How to Solve:

1. Use nested loops (outer for rows, inner for columns)
2. Pattern 1: Print sequential numbers 1, 2, 3...
3. Pattern 2: Print row number repeatedly
4. Pattern 3: Print column number
5. Pattern 4: Print numbers with spaces for pyramid shape
6. Pattern 5: Inverted pyramid
7. Pattern 6: Floyd's triangle with continuous numbering

Code:

```
#include <stdio.h>

int main() {
   int n, i, j, num, spaces;

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

   if (n <= 0) {
      printf("Error: Number of rows must be positive!\n");
      return 1;
   }

   // Pattern 1: Sequential numbers
   printf("\nPattern 1: Sequential Numbers\n");
   num = 1;
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= i; j++) {
         printf("%d ", num++);
      }
      printf("\n");
   }

   // Pattern 2: Row-wise repeated numbers
   printf("\nPattern 2: Row-wise Repeated Numbers\n");
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= i; j++) {
         printf("%d ", i);
```

```
        }
        printf("\n");
    }

    // Pattern 3: Column-wise numbers
    printf("\nPattern 3: Column-wise Numbers\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }

    // Pattern 4: Pyramid of numbers
    printf("\nPattern 4: Pyramid of Numbers\n");
    for (i = 1; i <= n; i++) {
        // Print spaces
        for (spaces = 1; spaces <= n - i; spaces++) {
            printf("  ");
        }
        // Print numbers
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        // Print reverse numbers
        for (j = i - 1; j >= 1; j--) {
            printf("%d ", j);
        }
        printf("\n");
    }

    // Pattern 5: Inverted pyramid
    printf("\nPattern 5: Inverted Number Pyramid\n");
    for (i = n; i >= 1; i--) {
        // Print spaces
        for (spaces = 1; spaces <= n - i; spaces++) {
            printf("  ");
        }
        // Print numbers
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }

    // Pattern 6: Floyd's Triangle
    printf("\nPattern 6: Floyd's Triangle\n");
    num = 1;
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= i; j++) {
            printf("%-4d", num++);
        }
        printf("\n");
    }

    return 0;
}
```

Example:

```
Input:
  Number of rows: 5

Output:
Pattern 1: Sequential Numbers
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

Pattern 2: Row-wise Repeated Numbers
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

Pattern 3: Column-wise Numbers
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

Pattern 4: Pyramid of Numbers
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1

Pattern 5: Inverted Number Pyramid
1 2 3 4 5
```

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

Pattern 6: Floyd's Triangle
1
2   3
4   5   6
7   8   9   10
11  12  13  14  15
```

Back to Table of Contents

xvi) Pascal's Triangle

Question:
Write a C program to generate Pascal's Triangle using three different methods:

1. Using binomial coefficient formula: C(n,r) = n! / (r! × (n-r)!)
2. Using iterative calculation
3. Using compact formula with running product

Description:
Pascal's Triangle is a triangular array where each number is the sum of the two numbers directly above it. The program demonstrates three mathematical approaches to generate it.

How to Solve:

1. **Method 1**: Calculate factorials and use binomial coefficient
2. **Method 2**: Use property C(n,r) = C(n,r-1) × (n-r+1) / r
3. **Method 3**: Running product optimization
4. Format output with proper spacing for triangle shape

Code:

```c
#include <stdio.h>

// Function to calculate factorial
long long factorial(int n) {
   long long fact = 1;
   for (int i = 2; i <= n; i++) {
      fact *= i;
   }
   return fact;
}

// Function to calculate binomial coefficient C(n, r)
long long binomialCoeff(int n, int r) {
   return factorial(n) / (factorial(r) * factorial(n - r));
}

int main() {
   int n, i, j, spaces;

   printf("Enter number of rows for Pascal's Triangle: ");
   scanf("%d", &n);

   if (n <= 0) {
      printf("Error: Number of rows must be positive!\n");
      return 1;
   }

   // Method 1: Using Binomial Coefficient with Factorial
   printf("\nMethod 1: Using Binomial Coefficient C(n,r) = n!/(r!×(n-r)!)\n");
   printf("===========================================================\n");
   for (i = 0; i < n; i++) {
      // Print spaces for formatting
      for (spaces = 0; spaces < n - i - 1; spaces++) {
         printf("   ");
      }
      for (j = 0; j <= i; j++) {
         printf("%6lld", binomialCoeff(i, j));
      }
      printf("\n");
   }

   // Method 2: Using Iterative Calculation
   printf("\nMethod 2: Using Iterative Property C(n,r) = C(n,r-1) × (n-r+1)/r\n");
   printf("==================================================================\n");
   for (i = 0; i < n; i++) {
      long long coeff = 1;
      // Print spaces
      for (spaces = 0; spaces < n - i - 1; spaces++) {
         printf("   ");
      }
      for (j = 0; j <= i; j++) {
         printf("%6lld", coeff);
         coeff = coeff * (i - j) / (j + 1);
      }
      printf("\n");
   }

   // Method 3: Using Compact Formula
```

```
    printf("\nMethod 3: Using Running Product\n");
    printf("================================\n");
    for (i = 0; i < n; i++) {
        // Print spaces
        for (spaces = 0; spaces < n - i - 1; spaces++) {
            printf("   ");
        }

        long long num = 1;
        for (j = 0; j <= i; j++) {
            printf("%6lld", num);
            num = num * (i - j) / (j + 1);
        }
        printf("\n");
    }

    return 0;
}
```

Example 1:

```
Input:
  Number of rows: 6

Output:
Method 1: Using Binomial Coefficient C(n,r) = n!/(r!×(n-r)!)
============================================================
                    1
                 1     1
              1     2     1
           1     3     3     1
        1     4     6     4     1
     1     5    10    10     5     1

Method 2: Using Iterative Property C(n,r) = C(n,r-1) × (n-r+1)/r
===================================================================
                    1
                 1     1
              1     2     1
           1     3     3     1
        1     4     6     4     1
     1     5    10    10     5     1
```

Example 2:

```
Input:
  Number of rows: 4

Output:
              1
           1     1
        1     2     1
     1     3     3     1
```

Back to Table of Contents

xvii) Nested Series Sum

Question:
Write a C program to find the sum of the series:
S = 1 + (1+2) + (1+2+3) + (1+2+3+4) + ... + (1+2+3+...+N)

Implement using three methods:

1. Nested loops
2. Single loop with running sum
3. Mathematical formula

Description:
This series involves calculating cumulative sums. Each term is the sum of natural numbers up to that position. The program demonstrates optimization from O(n^3) to O(n) to O(1).

How to Solve:

1. **Method 1**: Use nested loops - outer for terms, inner for sum within each term
2. **Method 2**: Use running sum to avoid recalculation
3. **Method 3**: Use formula S = N×(N+1)×(N+2)/6
4. Compare efficiency of all methods

Code:

```
#include <stdio.h>

int main() {
    int N, i, j;
    long long totalSum, termSum, runningSum;
    long long formulaResult;

    printf("Enter N: ");
    scanf("%d", &N);
    if (N <= 0) {
        printf("Error: N must be positive!\n");
        return 1;
    }

    printf("\nSeries: 1 + (1+2) + (1+2+3) + ... + (1+2+3+...+%d)\n", N);
    printf("=========================================================\n");

    // Method 1: Using Nested Loops
    printf("\nMethod 1: Using Nested Loops (Most Intuitive)\n");
    totalSum = 0;
    printf("Calculation:\n");
    for (i = 1; i <= N; i++) {
        termSum = 0;
        printf("Term %d: ", i);
        for (j = 1; j <= i; j++) {
            termSum += j;
            if (j < i) {
                printf("%d + ", j);
            } else {
                printf("%d", j);
            }
        }
        printf(" = %lld\n", termSum);
        totalSum += termSum;
    }
    printf("\nTotal Sum (Method 1) = %lld\n", totalSum);

    // Method 2: Using Single Loop with Running Sum
    printf("\nMethod 2: Using Running Sum (More Efficient)\n");
    totalSum = 0;
    runningSum = 0;
    printf("Calculation:\n");
    for (i = 1; i <= N; i++) {
        runningSum += i; // Add current number to running sum
        totalSum += runningSum;
        printf("Term %d: Sum(1 to %d) = %lld, Total = %lld\n",
               i, i, runningSum, totalSum);
    }
    printf("\nTotal Sum (Method 2) = %lld\n", totalSum);

    // Method 3: Using Mathematical Formula
    printf("\nMethod 3: Using Formula (Most Efficient)\n");
    printf("Formula: S = N×(N+1)×(N+2)/6\n");
    printf("Derivation: Each term k contributes k×(k+1)/2\n");
    printf("           Sum of all terms = Σ[k×(k+1)/2] for k=1 to N\n");
    printf("                            = N×(N+1)×(N+2)/6\n\n");

    formulaResult = (long long)N * (N + 1) * (N + 2) / 6;
    printf("Calculation: S = %d×%d×%d/6\n", N, N+1, N+2);
    printf("               = %lld/6\n", (long long)N * (N + 1) * (N + 2));
    printf("               = %lld\n", formulaResult);

    printf("\n=== VERIFICATION ===\n");
    printf("Method 1 (Nested Loops): %lld\n", totalSum);
    printf("Method 2 (Running Sum):  %lld\n", totalSum);
    printf("Method 3 (Formula):      %lld\n", formulaResult);
    printf("All methods match! ✓\n");

    return 0;
}
```

Example 1:

```
Input:
  N = 5

Output:
Series: 1 + (1+2) + (1+2+3) + ... + (1+2+3+...+5)
=========================================================

Method 1: Using Nested Loops (Most Intuitive)
Calculation:
Term 1: 1 = 1
Term 2: 1 + 2 = 3
Term 3: 1 + 2 + 3 = 6
Term 4: 1 + 2 + 3 + 4 = 10
Term 5: 1 + 2 + 3 + 4 + 5 = 15

Total Sum (Method 1) = 35

Method 3: Using Formula (Most Efficient)
Formula: S = N×(N+1)×(N+2)/6
Calculation: S = 5×6×7/6
               = 210/6
               = 35
```

Example 2:

```
Input:
  N = 10
```

```
Output:
Total Sum = 10×11×12/6 = 220
```

Back to Table of Contents

xviii) Prime Numbers in Range

Question:
Write a C program to find and display all prime numbers between two given numbers (start and end). Also display:

- Count of prime numbers found
- Sum of all prime numbers
- Average of prime numbers

Description:
This program uses an efficient prime checking function to find all primes in a range. It demonstrates:

- Prime number logic (divisibility checking up to √n)
- Function usage for modular code
- Statistics calculation

How to Solve:

1. Create isPrime() function that checks divisibility from 2 to √n
2. Loop through range and test each number
3. Maintain count and sum of primes
4. Calculate and display statistics

Code:

```c
#include <stdio.h>
#include <math.h>

// Function to check if a number is prime
int isPrime(int num) {
   if (num <= 1) {
      return 0; // Not prime
   }
   if (num == 2) {
      return 1; // 2 is prime
   }
   if (num % 2 == 0) {
      return 0; // Even numbers (except 2) are not prime
   }

   // Check odd divisors up to √num
   int limit = (int)sqrt(num);
   for (int i = 3; i <= limit; i += 2) {
      if (num % i == 0) {
         return 0; // Found a divisor, not prime
      }
   }
   return 1; // No divisors found, prime
}

int main() {
   int start, end, count = 0, i;
   long long sum = 0;
   double average;

   printf("Enter the range [start end]: ");
   scanf("%d %d", &start, &end);

   if (start > end) {
      printf("Error: Start must be less than or equal to end!\n");
      return 1;
   }

   printf("\nPrime numbers between %d and %d:\n", start, end);
   printf("=====================================\n");

   for (i = start; i <= end; i++) {
      if (isPrime(i)) {
         printf("%d ", i);
         count++;
         sum += i;

         // Print newline after every 10 primes for readability
         if (count % 10 == 0) {
            printf("\n");
         }
      }
   }

   printf("\n\n=== STATISTICS ===\n");
   if (count > 0) {
      average = (double)sum / count;
      printf("Total prime numbers found: %d\n", count);
      printf("Sum of all prime numbers: %lld\n", sum);
      printf("Average of prime numbers: %.2f\n", average);

      // Additional info
```

```
        printf("\nDensity: %.2f%% of numbers in range are prime\n",
               (count * 100.0) / (end - start + 1));
    } else {
        printf("No prime numbers found in this range.\n");
    }

    return 0;
}
```

Example 1:

```
Input:
  Range: 1 50

Output:
Prime numbers between 1 and 50:
=====================================
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47

=== STATISTICS ===
Total prime numbers found: 15
Sum of all prime numbers: 328
Average of prime numbers: 21.87

Density: 30.00% of numbers in range are prime
```

Example 2:

```
Input:
  Range: 100 200

Output:
Prime numbers between 100 and 200:
=====================================
101 103 107 109 113 127 131 137 139 149
151 157 163 167 173 179 181 191 193 197
199

=== STATISTICS ===
Total prime numbers found: 21
Sum of all prime numbers: 3167
Average of prime numbers: 150.81

Density: 20.79% of numbers in range are prime
```

Back to Table of Contents

xix) Armstrong Numbers (100-1000)

Question:
Write a C program to find all Armstrong numbers between 100 and 1000.

Note: An Armstrong number (also called narcissistic number) is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

For 3-digit numbers: abc = $a^3 + b^3 + c^3$

Description:
This program searches for Armstrong numbers in the range 100-1000. For 3-digit numbers, it checks if the number equals the sum of cubes of its digits. The program displays detailed calculation for each Armstrong number found.

How to Solve:

1. Loop through numbers 100 to 1000
2. Extract each digit of the number
3. Calculate sum of cubes of digits
4. Compare with original number
5. Display results with verification

Code:

```
#include <stdio.h>
#include <math.h>

// Function to count digits in a number
int countDigits(int num) {
    int count = 0;
    while (num != 0) {
        num /= 10;
        count++;
    }
    return count;
}

// Function to check if a number is Armstrong
int isArmstrong(int num) {
    int originalNum = num;
    int numDigits = countDigits(num);
    int sum = 0, digit;
    while (num != 0) {
        digit = num % 10;
        sum += (int)pow(digit, numDigits);
        num /= 10;
    }

    return (sum == originalNum);
}

int main() {
    int i, count = 0;
    int digit1, digit2, digit3, sum;

    printf("Armstrong Numbers between 100 and 1000:\n");
    printf("========================================\n");
    printf("(3-digit Armstrong: abc = a³ + b³ + c³)\n\n");

    for (i = 100; i < 1000; i++) {
        if (isArmstrong(i)) {
            count++;

            // Extract digits for display
            digit1 = i / 100;           // Hundreds place
            digit2 = (i / 10) % 10;     // Tens place
            digit3 = i % 10;            // Units place
            sum = (digit1 * digit1 * digit1) +
                  (digit2 * digit2 * digit2) +
                  (digit3 * digit3 * digit3);

            printf("%d. Number: %d\n", count, i);
            printf("   Calculation: %d³ + %d³ + %d³\n", digit1, digit2, digit3);
            printf("              = %d + %d + %d\n",
                   digit1*digit1*digit1,
                   digit2*digit2*digit2,
                   digit3*digit3*digit3);
            printf("              = %d ✓\n\n", sum);
        }
    }

    printf("=== SUMMARY ===\n");
    printf("Total Armstrong numbers found: %d\n", count);
    printf("The Armstrong numbers are: ");

    // Print them again in a list
    count = 0;
    for (i = 100; i < 1000; i++) {
        if (isArmstrong(i)) {
            if (count > 0) printf(", ");
            printf("%d", i);
            count++;
        }
    }
    printf("\n");

    return 0;
}
```

Example:

```
Output:
Armstrong Numbers between 100 and 1000:
========================================
(3-digit Armstrong: abc = a³ + b³ + c³)

1. Number: 153
   Calculation: 1³ + 5³ + 3³
              = 1 + 125 + 27
              = 153 ✓

2. Number: 370
   Calculation: 3³ + 7³ + 0³
              = 27 + 343 + 0
              = 370 ✓

3. Number: 371
   Calculation: 3³ + 7³ + 1³
              = 27 + 343 + 1
              = 371 ✓

4. Number: 407
   Calculation: 4³ + 0³ + 7³
              = 64 + 0 + 343
              = 407 ✓

=== SUMMARY ===
Total Armstrong numbers found: 4
The Armstrong numbers are: 153, 370, 371, 407
```

Back to Table of Contents

Section 4: Arrays, Matrices & Strings

xxi) Array Largest & Smallest

Question:
Write a C program to find the largest and smallest numbers from array elements.

Description:
This program takes an array of numbers as input and finds both the largest (maximum) and smallest (minimum) elements from the array. It also displays:

- The positions (indices) where these elements are found
- Count of occurrences for largest and smallest values
- Statistics like range and sum of extremes

How to Solve:

1. Declare an array and input the size from user
2. Read array elements from the user
3. Initialize largest and smallest with first element
4. Loop through array from index 1 to n-1:
    - If arr[i] > largest, update largest and position
    - If arr[i] < smallest, update smallest and position
5. Count occurrences of largest and smallest
6. Display results with comprehensive statistics

Code:

```c
#include <stdio.h>

int main() {
   int n, i;
   int largest, smallest;
   int largestPos, smallestPos;

   // Input array size
   printf("Enter the number of elements in array: ");
   scanf("%d", &n);

   if (n <= 0) {
      printf("Error: Array size must be positive!\n");
      return 1;
   }

   int arr[n];

   // Input array elements
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++) {
      printf("Element %d: ", i + 1);
      scanf("%d", &arr[i]);
   }

   // Initialize with first element
   largest = smallest = arr[0];
   largestPos = smallestPos = 0;

   // Find largest and smallest
   for (i = 1; i < n; i++) {
      if (arr[i] > largest) {
         largest = arr[i];
         largestPos = i;
      }
      if (arr[i] < smallest) {
         smallest = arr[i];
         smallestPos = i;
      }
   }

   // Display results
   printf("\n========================================\n");
   printf("  Array Analysis\n");
   printf("========================================\n\n");

   printf("Array elements: [ ");
   for (i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("]\n\n");

   printf("--- Results ---\n");
   printf("Largest element:   %d (at position %d, index %d)\n",
          largest, largestPos + 1, largestPos);
   printf("Smallest element:  %d (at position %d, index %d)\n",
          smallest, smallestPos + 1, smallestPos);

   printf("\n--- Statistics ---\n");
   printf("Array size:        %d\n", n);
   printf("Range:             %d (largest - smallest)\n", largest - smallest);
   printf("Sum of extremes:   %d (largest + smallest)\n", largest + smallest);

   // Count occurrences
   int largestCount = 0, smallestCount = 0;
   printf("\n--- Occurrences ---\n");
   printf("Largest (%d) appears at: ", largest);
   for (i = 0; i < n; i++) {
      if (arr[i] == largest) {
            printf("%d ", i);
            largestCount++;
        }
    }
    printf("(%d time%s)\n", largestCount, largestCount > 1 ? "s" : "");

    printf("Smallest (%d) appears at: ", smallest);
    for (i = 0; i < n; i++) {
        if (arr[i] == smallest) {
            printf("%d ", i);
            smallestCount++;
        }
    }
    printf("(%d time%s)\n", smallestCount, smallestCount > 1 ? "s" : "");

    printf("\n========================================\n");

    return 0;
}
```

Example 1:

```
Input:
  Number of elements: 7
  Elements: 45, 12, 78, 23, 9, 56, 34

Output:
========================================
  Array Analysis
========================================

Array elements: [ 45 12 78 23 9 56 34 ]

--- Results ---
Largest element:   78 (at position 3, index 2)
Smallest element:  9 (at position 5, index 4)

--- Statistics ---
Array size:        7
Range:             69 (largest - smallest)
Sum of extremes:   87 (largest + smallest)

--- Occurrences ---
Largest (78) appears at: 2 (1 time)
Smallest (9) appears at: 4 (1 time)

========================================
```

Example 2:

```
Input:
  Number of elements: 5
  Elements: 100, 50, 100, 25, 50

Output:
Array elements: [ 100 50 100 25 50 ]

--- Results ---
Largest element:   100 (at position 1, index 0)
Smallest element:  25 (at position 4, index 3)

--- Occurrences ---
Largest (100) appears at: 0 2 (2 times)
Smallest (25) appears at: 3 (1 time)
```

Back to Table of Contents

xxii) Array Sorting

Question:
Write a C program to sort array elements in ascending and descending order.

Description:
This program sorts an array of numbers in both ascending (smallest to largest) and descending (largest to smallest) order. Multiple sorting algorithms are demonstrated:

- **Bubble Sort**: Compare adjacent elements and swap if needed
- **Selection Sort**: Find minimum/maximum and place in correct position

Both algorithms are implemented for ascending and descending order.

How to Solve:

1. Input array size and elements
2. **Bubble Sort (Ascending)**:
   - Outer loop: n-1 passes
   - Inner loop: Compare adjacent elements
   - If arr[j] > arr[j+1], swap them
3. **Bubble Sort (Descending)**: Swap if arr[j] < arr[j+1]
4. **Selection Sort**: Find minimum/maximum from unsorted portion and swap

5. Display original and sorted arrays

Code:

```c
#include <stdio.h>
#include <string.h>

// Function to display array
void displayArray(int arr[], int n, char *label) {
   printf("%s: [ ", label);
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("]\n");
}

// Function to copy array
void copyArray(int source[], int dest[], int n) {
   for (int i = 0; i < n; i++) {
      dest[i] = source[i];
   }
}

// Bubble Sort - Ascending
void bubbleSortAscending(int arr[], int n) {
   int i, j, temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            // Swap
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

// Bubble Sort - Descending
void bubbleSortDescending(int arr[], int n) {
   int i, j, temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] < arr[j + 1]) {
            // Swap
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

// Selection Sort - Ascending
void selectionSortAscending(int arr[], int n) {
   int i, j, minIdx, temp;
   for (i = 0; i < n - 1; i++) {
      minIdx = i;
      for (j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIdx]) {
            minIdx = j;
         }
      }
      // Swap minimum element with first element
      temp = arr[minIdx];
      arr[minIdx] = arr[i];
      arr[i] = temp;
   }
}

// Selection Sort - Descending
void selectionSortDescending(int arr[], int n) {
   int i, j, maxIdx, temp;
   for (i = 0; i < n - 1; i++) {
      maxIdx = i;
      for (j = i + 1; j < n; j++) {
         if (arr[j] > arr[maxIdx]) {
            maxIdx = j;
         }
      }
      // Swap maximum element with first element
      temp = arr[maxIdx];
      arr[maxIdx] = arr[i];
      arr[i] = temp;
   }
}

int main() {
   int n, i, choice;

   // Input array size
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   if (n <= 0) {
      printf("Error: Array size must be positive!\n");
```

```
        return 1;
    }

    int arr[n], tempArr[n];

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    printf("\n========================================\n");
    printf("  Array Sorting Program\n");
    printf("========================================\n\n");

    displayArray(arr, n, "Original Array");

    // Bubble Sort - Ascending
    printf("\n--- Method 1: Bubble Sort ---\n\n");
    copyArray(arr, tempArr, n);
    bubbleSortAscending(tempArr, n);
    displayArray(tempArr, n, "Ascending Order ");

    copyArray(arr, tempArr, n);
    bubbleSortDescending(tempArr, n);
    displayArray(tempArr, n, "Descending Order");

    // Selection Sort
    printf("\n--- Method 2: Selection Sort ---\n\n");
    copyArray(arr, tempArr, n);
    selectionSortAscending(tempArr, n);
    displayArray(tempArr, n, "Ascending Order ");

    copyArray(arr, tempArr, n);
    selectionSortDescending(tempArr, n);
    displayArray(tempArr, n, "Descending Order");

    printf("\n========================================\n");
    printf("Sorting Algorithm Information:\n");
    printf("========================================\n");
    printf("Bubble Sort:\n");
    printf("  - Time Complexity: O(n²)\n");
    printf("  - Space Complexity: O(1)\n");
    printf("  - Stable: Yes\n\n");
    printf("Selection Sort:\n");
    printf("  - Time Complexity: O(n²)\n");
    printf("  - Space Complexity: O(1)\n");
    printf("  - Stable: No\n");
    printf("========================================\n");

    return 0;
}
```

Example 1:

```
Input:
  Number of elements: 6
  Elements: 64, 34, 25, 12, 22, 11

Output:
========================================
  Array Sorting Program
========================================

Original Array: [ 64 34 25 12 22 11 ]

--- Method 1: Bubble Sort ---

Ascending Order : [ 11 12 22 25 34 64 ]
Descending Order: [ 64 34 25 22 12 11 ]

--- Method 2: Selection Sort ---

Ascending Order : [ 11 12 22 25 34 64 ]
Descending Order: [ 64 34 25 22 12 11 ]

========================================
Sorting Algorithm Information:
========================================
Bubble Sort:
  - Time Complexity: O(n²)
  - Space Complexity: O(1)
  - Stable: Yes

Selection Sort:
  - Time Complexity: O(n²)
  - Space Complexity: O(1)
  - Stable: No
========================================
```

Example 2:

```
Input:
  Number of elements: 5
  Elements: 5, 2, 8, 1, 9

Output:
Original Array: [ 5 2 8 1 9 ]

Ascending Order : [ 1 2 5 8 9 ]
Descending Order: [ 9 8 5 2 1 ]
```

Back to Table of Contents

xxiii) 3×3 Matrix Input & Display

Question:
Write a C program to enter elements for a 3×3 matrix and display them.

Description:
This program creates a 3×3 matrix (2-dimensional array) and allows the user to enter values for each element. The matrix is then displayed in proper format with:

- Matrix representation with brackets
- Diagonal elements (principal and secondary)
- Row-wise and column-wise sums
- Total sum of all elements

A 3×3 matrix looks like:

```
[ a11  a12  a13 ]
[ a21  a22  a23 ]
[ a31  a32  a33 ]
```

How to Solve:

1. Declare a 2D array: `int matrix[3][3]`
2. Use nested loops to input elements:
   - Outer loop: rows (i from 0 to 2)
   - Inner loop: columns (j from 0 to 2)
3. Display matrix with proper formatting
4. Calculate and show diagonal elements
5. Calculate row-wise and column-wise sums

Code:

```c
#include <stdio.h>

int main() {
   int matrix[3][3];
   int i, j;
   int sum = 0;

   printf("========================================\n");
   printf("  3×3 Matrix Input and Display\n");
   printf("========================================\n\n");

   // Input matrix elements
   printf("Enter elements for 3×3 matrix:\n");
   printf("(Enter 9 values - row by row)\n\n");

   for (i = 0; i < 3; i++) {
      for (j = 0; j < 3; j++) {
         printf("Element [%d][%d]: ", i, j);
         scanf("%d", &matrix[i][j]);
         sum += matrix[i][j];
      }
   }

   // Display matrix
   printf("\n========================================\n");
   printf("  Matrix Display\n");
   printf("========================================\n\n");

   printf("The 3×3 Matrix is:\n\n");
   for (i = 0; i < 3; i++) {
      printf("    [ ");
      for (j = 0; j < 3; j++) {
         printf("%4d ", matrix[i][j]);
      }
      printf("]\n");
   }

   // Display in different formats
   printf("\n--- Alternative Display Format ---\n\n");
   for (i = 0; i < 3; i++) {
      printf("    | ");
      for (j = 0; j < 3; j++) {
         printf("%4d ", matrix[i][j]);
      }
      printf("|\n");
   }
```

```
    // Display matrix information
    printf("\n========================================\n");
    printf("  Matrix Information\n");
    printf("========================================\n");
    printf("Matrix Size:         3×3\n");
    printf("Total Elements:      9\n");
    printf("Sum of all elements: %d\n\n", sum);

    // Display diagonal elements
    printf("--- Diagonal Elements ---\n");
    printf("Principal Diagonal:  ");
    for (i = 0; i < 3; i++) {
        printf("%d ", matrix[i][i]);
    }
    printf("\n");

    printf("Secondary Diagonal:  ");
    for (i = 0; i < 3; i++) {
        printf("%d ", matrix[i][2 - i]);
    }
    printf("\n\n");

    // Display row-wise sum
    printf("--- Row-wise Sum ---\n");
    for (i = 0; i < 3; i++) {
        int rowSum = 0;
        for (j = 0; j < 3; j++) {
            rowSum += matrix[i][j];
        }
        printf("Row %d: %d\n", i, rowSum);
    }
    printf("\n");

    // Display column-wise sum
    printf("--- Column-wise Sum ---\n");
    for (j = 0; j < 3; j++) {
        int colSum = 0;
        for (i = 0; i < 3; i++) {
            colSum += matrix[i][j];
        }
        printf("Column %d: %d\n", j, colSum);
    }

    printf("\n========================================\n");

    return 0;
}
```

Example:

```
Input:
  Elements: 1, 2, 3, 4, 5, 6, 7, 8, 9

Output:
========================================
  3×3 Matrix Input and Display
========================================

The 3×3 Matrix is:

    [    1    2    3 ]
    [    4    5    6 ]
    [    7    8    9 ]

--- Alternative Display Format ---

    |    1    2    3 |
    |    4    5    6 |
    |    7    8    9 |

========================================
  Matrix Information
========================================
Matrix Size:         3×3
Total Elements:      9
Sum of all elements: 45

--- Diagonal Elements ---
Principal Diagonal:  1 5 9
Secondary Diagonal:  3 5 7

--- Row-wise Sum ---
Row 0: 6
Row 1: 15
Row 2: 24

--- Column-wise Sum ---
Column 0: 12
Column 1: 15
Column 2: 18

========================================
```

xxiv) Matrix Addition & Subtraction

Question:
Write a C program to calculate addition and subtraction of 2-dimensional matrices.

Description:
This program performs matrix addition and subtraction operations on two matrices of the same dimensions. Operations are performed element-wise:

- **Matrix Addition**: C[i][j] = A[i][j] + B[i][j]
- **Matrix Subtraction**: C[i][j] = A[i][j] - B[i][j]

Important: Matrices must have the same dimensions for addition/subtraction.

How to Solve:

1. Input dimensions (rows and columns) from user
2. Verify both matrices have same dimensions
3. Input elements for matrix A and B
4. For Addition: Loop through each position and add corresponding elements
5. For Subtraction: Loop through each position and subtract corresponding elements
6. Display all matrices with proper formatting

Code:

```c
#include <stdio.h>

// Function to input matrix
void inputMatrix(int matrix[][10], int rows, int cols, char name[]) {
   printf("\nEnter elements for Matrix %s (%dx%d):\n", name, rows, cols);
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         printf("Element [%d][%d]: ", i, j);
         scanf("%d", &matrix[i][j]);
      }
   }
}

// Function to display matrix
void displayMatrix(int matrix[][10], int rows, int cols, char name[]) {
   printf("\nMatrix %s:\n", name);
   for (int i = 0; i < rows; i++) {
      printf("    [ ");
      for (int j = 0; j < cols; j++) {
         printf("%4d ", matrix[i][j]);
      }
      printf("]\n");
   }
}

// Function to add two matrices
void addMatrices(int A[][10], int B[][10], int result[][10], int rows, int cols) {
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         result[i][j] = A[i][j] + B[i][j];
      }
   }
}

// Function to subtract two matrices
void subtractMatrices(int A[][10], int B[][10], int result[][10], int rows, int cols) {
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         result[i][j] = A[i][j] - B[i][j];
      }
   }
}

int main() {
   int A[10][10], B[10][10], sum[10][10], diff[10][10];
   int rows, cols, i, j;

   printf("========================================\n");
   printf("  Matrix Addition & Subtraction\n");
   printf("========================================\n\n");

   // Input matrix dimensions
   printf("Enter number of rows: ");
   scanf("%d", &rows);
   printf("Enter number of columns: ");
   scanf("%d", &cols);

   if (rows <= 0 || cols <= 0 || rows > 10 || cols > 10) {
      printf("\nError: Invalid dimensions! (1-10 allowed)\n");
      return 1;
   }

   // Input matrices
   inputMatrix(A, rows, cols, "A");
   inputMatrix(B, rows, cols, "B");

   // Perform operations
   addMatrices(A, B, sum, rows, cols);
```

```
    subtractMatrices(A, B, diff, rows, cols);

    // Display results
    printf("\n========================================\n");
    printf("  Input Matrices\n");
    printf("========================================");
    displayMatrix(A, rows, cols, "A");
    displayMatrix(B, rows, cols, "B");

    printf("\n========================================\n");
    printf("  Matrix Addition (A + B)\n");
    printf("========================================");
    displayMatrix(sum, rows, cols, "Sum");

    printf("\n========================================\n");
    printf("  Matrix Subtraction (A - B)\n");
    printf("========================================");
    displayMatrix(diff, rows, cols, "Difference");

    printf("\n========================================\n");
    printf("  Sample Calculation (First Element)\n");
    printf("========================================\n");
    printf("Addition:    A[0][0] + B[0][0] = %d + %d = %d\n",
           A[0][0], B[0][0], sum[0][0]);
    printf("Subtraction: A[0][0] - B[0][0] = %d - %d = %d\n",
           A[0][0], B[0][0], diff[0][0]);
    printf("========================================\n");

    return 0;
}
```

Example:

```
Input:
  Rows: 2, Columns: 2
  Matrix A: 1, 2, 3, 4
  Matrix B: 5, 6, 7, 8

Output:
========================================
  Matrix Addition & Subtraction
========================================

========================================
  Input Matrices
========================================
Matrix A:
    [    1    2 ]
    [    3    4 ]

Matrix B:
    [    5    6 ]
    [    7    8 ]

========================================
  Matrix Addition (A + B)
========================================
Matrix Sum:
    [    6    8 ]
    [   10   12 ]

========================================
  Matrix Subtraction (A - B)
========================================
Matrix Difference:
    [   -4   -4 ]
    [   -4   -4 ]

========================================
  Sample Calculation (First Element)
========================================
Addition:    A[0][0] + B[0][0] = 1 + 5 = 6
Subtraction: A[0][0] - B[0][0] = 1 - 5 = -4
========================================
```

Back to Table of Contents

xxv) Matrix Multiplication

Question:
Write a C program to calculate multiplication of 2-dimensional matrices.

Description:
This program performs matrix multiplication on two matrices. Matrix multiplication is different from element-wise operations. For matrices A (m×n) and B (n×p), the result C (m×p) is:

C[i][j] = Σ(k=0 to n-1) A[i][k] × B[k][j]

Important Condition: Number of columns in first matrix must equal number of rows in second matrix.

If A is m×n and B is n×p, then C will be m×p

How to Solve:

1. Input dimensions of both matrices
2. Verify that columns of A = rows of B
3. Input elements for both matrices
4. Use three nested loops for multiplication:
   - Outer loop: rows of A (i)
   - Middle loop: columns of B (j)
   - Inner loop: dot product calculation (k)
5. Display all matrices with step-by-step calculations

Code:

```c
#include <stdio.h>

// Function to input matrix
void inputMatrix(int matrix[][10], int rows, int cols, char name[]) {
   printf("\nEnter elements for Matrix %s (%dx%d):\n", name, rows, cols);
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         printf("Element [%d][%d]: ", i, j);
         scanf("%d", &matrix[i][j]);
      }
   }
}

// Function to display matrix
void displayMatrix(int matrix[][10], int rows, int cols, char name[]) {
   printf("\nMatrix %s (%dx%d):\n", name, rows, cols);
   for (int i = 0; i < rows; i++) {
      printf("    [ ");
      for (int j = 0; j < cols; j++) {
         printf("%4d ", matrix[i][j]);
      }
      printf("]\n");
   }
}

// Function to multiply two matrices
void multiplyMatrices(int A[][10], int B[][10], int result[][10],
                      int r1, int c1, int r2, int c2) {
   // Initialize result matrix with zeros
   for (int i = 0; i < r1; i++) {
      for (int j = 0; j < c2; j++) {
         result[i][j] = 0;
      }
   }

   // Perform multiplication
   for (int i = 0; i < r1; i++) {
      for (int j = 0; j < c2; j++) {
         for (int k = 0; k < c1; k++) {
            result[i][j] += A[i][k] * B[k][j];
         }
      }
   }
}

// Function to show detailed calculation for one element
void showCalculation(int A[][10], int B[][10], int row, int col,
                     int c1, int result) {
   printf("\nCalculation for Result[%d][%d]:\n", row, col);
   printf("  = ");
   for (int k = 0; k < c1; k++) {
      printf("A[%d][%d] × B[%d][%d]", row, k, k, col);
      if (k < c1 - 1)
         printf(" + ");
   }
   printf("\n  = ");
   for (int k = 0; k < c1; k++) {
      printf("(%d × %d)", A[row][k], B[k][col]);
      if (k < c1 - 1)
         printf(" + ");
   }
   printf("\n  = ");
   for (int k = 0; k < c1; k++) {
      printf("%d", A[row][k] * B[k][col]);
      if (k < c1 - 1)
         printf(" + ");
   }
   printf("\n  = %d\n", result);
}

int main() {
   int A[10][10], B[10][10], result[10][10];
   int r1, c1, r2, c2;

   printf("========================================\n");
   printf("  Matrix Multiplication\n");
   printf("========================================\n\n");

   // Input dimensions for first matrix
   printf("Enter dimensions of Matrix A:\n");
   printf("Number of rows: ");
   scanf("%d", &r1);
   printf("Number of columns: ");
```

```
    scanf("%d", &c1);

    // Input dimensions for second matrix
    printf("\nEnter dimensions of Matrix B:\n");
    printf("Number of rows: ");
    scanf("%d", &r2);
    printf("Number of columns: ");
    scanf("%d", &c2);

    // Validate dimensions
    if (r1 <= 0 || c1 <= 0 || r2 <= 0 || c2 <= 0 ||
        r1 > 10 || c1 > 10 || r2 > 10 || c2 > 10) {
       printf("\nError: Invalid dimensions! (1-10 allowed)\n");
       return 1;
    }

    if (c1 != r2) {
       printf("\n========================================\n");
       printf("ERROR: Matrix multiplication not possible!\n");
       printf("========================================\n");
       printf("Columns of Matrix A (%d) must equal\n", c1);
       printf("Rows of Matrix B (%d)\n", r2);
       printf("\nFor A(%dx%d) × B(%dx%d):\n", r1, c1, r2, c2);
       printf("Columns of A must = Rows of B\n");
       printf("========================================\n");
       return 1;
    }

    // Input matrices
    inputMatrix(A, r1, c1, "A");
    inputMatrix(B, r2, c2, "B");

    // Perform multiplication
    multiplyMatrices(A, B, result, r1, c1, r2, c2);

    // Display results
    printf("\n========================================\n");
    printf("  Input Matrices\n");
    printf("========================================");
    displayMatrix(A, r1, c1, "A");
    displayMatrix(B, r2, c2, "B");

    printf("\n========================================\n");
    printf("  Matrix Multiplication (A × B)\n");
    printf("========================================");
    displayMatrix(result, r1, c2, "Result");

    // Show dimensions
    printf("\n========================================\n");
    printf("  Dimension Analysis\n");
    printf("========================================\n");
    printf("Matrix A:      %d × %d\n", r1, c1);
    printf("Matrix B:      %d × %d\n", r2, c2);
    printf("Result (A×B):  %d × %d\n", r1, c2);
    printf("========================================\n");

    // Show sample calculations
    printf("\n========================================\n");
    printf("  Sample Calculations\n");
    printf("========================================");
    showCalculation(A, B, 0, 0, c1, result[0][0]);

    printf("\n========================================\n");
    printf("Matrix Multiplication Rule:\n");
    printf("For A(m×n) × B(n×p):\n");
    printf("- Result will be C(m×p)\n");
    printf("- Columns of A must equal rows of B\n");
    printf("- C[i][j] = Σ A[i][k] × B[k][j]\n");
    printf("========================================\n");

    return 0;
}
```

Example:

```
Input:
  Matrix A (2×2): 1, 2, 3, 4
  Matrix B (2×2): 5, 6, 7, 8

Output:
========================================
  Matrix Multiplication
========================================

========================================
  Input Matrices
========================================
Matrix A (2x2):
    [    1    2 ]
    [    3    4 ]

Matrix B (2x2):
    [    5    6 ]
    [    7    8 ]
```

```
========================================
  Matrix Multiplication (A × B)
========================================
Matrix Result (2x2):
    [   19   22 ]
    [   43   50 ]

========================================
  Dimension Analysis
========================================
Matrix A:      2 × 2
Matrix B:      2 × 2
Result (A×B):  2 × 2
========================================

========================================
  Sample Calculations
========================================
Calculation for Result[0][0]:
  = A[0][0] × B[0][0] + A[0][1] × B[1][0]
  = (1 × 5) + (2 × 7)
  = 5 + 14
  = 19

========================================
Matrix Multiplication Rule:
For A(m×n) × B(n×p):
- Result will be C(m×p)
- Columns of A must equal rows of B
- C[i][j] = Σ A[i][k] × B[k][j]
========================================
```

Back to Table of Contents

xxvi) Vowels & Consonants Counter

Question:
Write a C program to find the number of vowels and consonants in a string.

Description:
This program takes a string as input and counts the number of vowels and consonants. Vowels are: a, e, i, o, u (case-insensitive), and consonants are all other alphabetic characters.

The program also counts:

- Total alphabetic characters
- Digits
- Spaces and special characters
- Percentage distribution of each type

How to Solve:

1. Input a string from the user
2. Initialize counters: vowels=0, consonants=0, digits=0, special=0
3. Loop through each character:
    - Convert to lowercase for comparison
    - Check if character is vowel (a,e,i,o,u)
    - Else check if it's an alphabet (consonant)
    - Else check if it's a digit
    - Else it's a special character or space
4. Display all counts with percentage distribution

Code:

```c
#include <ctype.h>
#include <stdio.h>
#include <string.h>

// Function to check if a character is vowel
int isVowel(char ch) {
   ch = tolower(ch);
   return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');
}

int main() {
   char str[200];
   int vowels = 0, consonants = 0, digits = 0, spaces = 0, special = 0;
   int i, length;

   printf("========================================\n");
   printf("  Vowel and Consonant Counter\n");
   printf("========================================\n\n");

   // Input string
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);

   // Remove newline character if present
   str[strcspn(str, "\n")] = '\0';

   length = strlen(str);
```

```
    // Count vowels, consonants, digits, and special characters
    for (i = 0; i < length; i++) {
       if (isalpha(str[i])) {
          if (isVowel(str[i])) {
             vowels++;
          } else {
             consonants++;
          }
       } else if (isdigit(str[i])) {
          digits++;
       } else if (str[i] == ' ') {
          spaces++;
       } else {
          special++;
       }
    }

    // Display results
    printf("\n========================================\n");
    printf("  Analysis Results\n");
    printf("========================================\n\n");

    printf("Input String: \"%s\"\n", str);
    printf("String Length: %d characters\n\n", length);

    printf("--- Character Breakdown ---\n");
    printf("Vowels:            %d\n", vowels);
    printf("Consonants:        %d\n", consonants);
    printf("Alphabets (Total): %d\n", vowels + consonants);
    printf("Digits:            %d\n", digits);
    printf("Spaces:            %d\n", spaces);
    printf("Special chars:     %d\n\n", special);

    // Display vowel and consonant details
    printf("--- Vowel Details ---\n");
    printf("Vowels found: ");
    for (i = 0; i < length; i++) {
       if (isalpha(str[i]) && isVowel(str[i])) {
          printf("%c ", str[i]);
       }
    }
    if (vowels == 0)
       printf("None");
    printf("\n\n");

    printf("--- Consonant Details ---\n");
    printf("Consonants found: ");
    for (i = 0; i < length; i++) {
       if (isalpha(str[i]) && !isVowel(str[i])) {
          printf("%c ", str[i]);
       }
    }
    if (consonants == 0)
       printf("None");
    printf("\n\n");

    // Percentage distribution
    if (length > 0) {
       printf("--- Percentage Distribution ---\n");
       printf("Vowels:       %.2f%%\n", (vowels * 100.0) / length);
       printf("Consonants:   %.2f%%\n", (consonants * 100.0) / length);
       printf("Digits:       %.2f%%\n", (digits * 100.0) / length);
       printf("Spaces:       %.2f%%\n", (spaces * 100.0) / length);
       printf("Special:      %.2f%%\n", (special * 100.0) / length);
    }

    printf("\n========================================\n");

    return 0;
}
```

Example 1:

```
Input:
  "Hello World"

Output:
========================================
  Vowel and Consonant Counter
========================================

Input String: "Hello World"
String Length: 11 characters

--- Character Breakdown ---
Vowels:            3
Consonants:        7
Alphabets (Total): 10
Digits:            0
Spaces:            1
Special chars:     0

--- Vowel Details ---
Vowels found: e o o
```

```
--- Consonant Details ---
Consonants found: H l l W r l d

--- Percentage Distribution ---
Vowels:       27.27%
Consonants:   63.64%
Digits:       0.00%
Spaces:       9.09%
Special:      0.00%

========================================
```

Example 2:

```
Input:
  "Programming in C 2025!"

Output:
--- Character Breakdown ---
Vowels:           5
Consonants:       11
Alphabets (Total): 16
Digits:           4
Spaces:           3
Special chars:    1

--- Percentage Distribution ---
Vowels:       22.73%
Consonants:   50.00%
Digits:       18.18%
Spaces:       13.64%
Special:      4.55%
```

Back to Table of Contents

xxvii) String Functions Implementation

Question:
Write a C program to implement strlen(), strcpy(), strcat(), and strcmp() functions.

Description:
This program demonstrates the implementation of four fundamental string manipulation functions from scratch (without using string.h functions):

1. **strlen()** - Returns the length of a string
2. **strcpy()** - Copies one string to another
3. **strcat()** - Concatenates (appends) one string to another
4. **strcmp()** - Compares two strings lexicographically

How to Solve:

1. **strlen()**: Loop through string until null terminator '\0', count characters
2. **strcpy()**: Copy each character from source to destination, include null terminator
3. **strcat()**: Find end of destination string, append source from that point
4. **strcmp()**: Compare characters one by one, return difference or 0 if equal

Code:

```
#include <stdio.h>

// Custom implementation of strlen()
int myStrlen(char str[]) {
   int length = 0;
   while (str[length] != '\0') {
      length++;
   }
   return length;
}

// Custom implementation of strcpy()
void myStrcpy(char dest[], char src[]) {
   int i = 0;
   while (src[i] != '\0') {
      dest[i] = src[i];
      i++;
   }
   dest[i] = '\0'; // Add null terminator
}

// Custom implementation of strcat()
void myStrcat(char dest[], char src[]) {
   int i = 0, j = 0;

   // Find end of destination string
   while (dest[i] != '\0') {
      i++;
   }

   // Append source string
   while (src[j] != '\0') {
      dest[i] = src[j];
```

```c
        i++;
        j++;
    }
    dest[i] = '\0'; // Add null terminator
}

// Custom implementation of strcmp()
int myStrcmp(char str1[], char str2[]) {
    int i = 0;

    while (str1[i] != '\0' && str2[i] != '\0') {
        if (str1[i] != str2[i]) {
            return str1[i] - str2[i];
        }
        i++;
    }

    // If we reached here, one or both strings ended
    return str1[i] - str2[i];
}

int main() {
    char str1[100], str2[100], str3[200];
    int length, cmpResult;

    printf("========================================\n");
    printf("  String Function Implementation\n");
    printf("========================================\n\n");

    // Demonstrate strlen()
    printf("--- 1. strlen() - String Length ---\n\n");
    printf("Enter a string: ");
    scanf(" %[^\n]", str1);

    length = myStrlen(str1);
    printf("Length of \"%s\" = %d\n\n", str1, length);

    // Demonstrate strcpy()
    printf("--- 2. strcpy() - String Copy ---\n\n");
    printf("Source string: \"%s\"\n", str1);
    myStrcpy(str2, str1);
    printf("After copying to str2: \"%s\"\n", str2);
    printf("Copy successful!\n\n");

    // Demonstrate strcat()
    printf("--- 3. strcat() - String Concatenation ---\n\n");
    printf("Enter first string: ");
    scanf(" %[^\n]", str1);
    printf("Enter second string: ");
    scanf(" %[^\n]", str2);

    myStrcpy(str3, str1); // Copy first string to str3
    printf("\nBefore concatenation:\n");
    printf("String 1: \"%s\"\n", str1);
    printf("String 2: \"%s\"\n", str2);

    myStrcat(str3, str2); // Concatenate str2 to str3
    printf("\nAfter concatenation (str1 + str2):\n");
    printf("Result: \"%s\"\n\n", str3);

    // Demonstrate strcmp()
    printf("--- 4. strcmp() - String Comparison ---\n\n");
    printf("Enter first string: ");
    scanf(" %[^\n]", str1);
    printf("Enter second string: ");
    scanf(" %[^\n]", str2);

    cmpResult = myStrcmp(str1, str2);

    printf("\nComparing: \"%s\" vs \"%s\"\n", str1, str2);
    printf("Result: %d\n", cmpResult);

    if (cmpResult == 0) {
        printf("Interpretation: Strings are EQUAL\n");
    } else if (cmpResult < 0) {
        printf("Interpretation: \"%s\" comes BEFORE \"%s\"\n", str1, str2);
    } else {
        printf("Interpretation: \"%s\" comes AFTER \"%s\"\n", str1, str2);
    }

    printf("\n========================================\n");
    printf("  Function Summary\n");
    printf("========================================\n");
    printf("strlen(str):       Returns length of string\n");
    printf("strcpy(dest, src): Copies src to dest\n");
    printf("strcat(dest, src): Appends src to dest\n");
    printf("strcmp(s1, s2):    Compares two strings\n");
    printf("                   Returns: 0 (equal)\n");
    printf("                            <0 (s1 < s2)\n");
    printf("                            >0 (s1 > s2)\n");
    printf("========================================\n");

    return 0;
}
```

Example:

```
Input:
  strlen test: "Hello"
  strcat: "Hello" and " World"
  strcmp: "apple" and "banana"

Output:
--- 1. strlen() - String Length ---
Length of "Hello" = 5

--- 2. strcpy() - String Copy ---
Source string: "Hello"
After copying to str2: "Hello"
Copy successful!

--- 3. strcat() - String Concatenation ---
Before concatenation:
String 1: "Hello"
String 2: " World"

After concatenation (str1 + str2):
Result: "Hello World"

--- 4. strcmp() - String Comparison ---
Comparing: "apple" vs "banana"
Result: -1
Interpretation: "apple" comes BEFORE "banana"
```

Back to Table of Contents

xxviii) Palindrome Checker

Question:
Write a C program to check whether a string is palindrome or not.

Description:
A palindrome is a word, phrase, or sequence that reads the same backward as forward. Examples: "madam", "racecar", "noon", "level".

The program checks:

1. Case-sensitive palindrome (exact match)
2. Case-insensitive palindrome (ignoring case)
3. Character-by-character comparison display

How to Solve:

1. Input a string from the user
2. **Method 1 - Two pointers**:
   - Place one pointer at start, one at end
   - Compare characters at both positions
   - Move pointers towards center
   - If any mismatch, not palindrome
3. **Method 2 - Reverse and compare**:
   - Create a reversed copy of string
   - Compare original with reversed
4. For case-insensitive check, convert to lowercase first

Code:

```c
#include <ctype.h>
#include <stdio.h>
#include <string.h>

// Function to check palindrome using two-pointer method
int isPalindrome(char str[]) {
   int left = 0;
   int right = strlen(str) - 1;

   while (left < right) {
      if (str[left] != str[right]) {
         return 0; // Not palindrome
      }
      left++;
      right--;
   }
   return 1; // Palindrome
}

// Function to check case-insensitive palindrome
int isPalindromeCaseInsensitive(char str[]) {
   int left = 0;
   int right = strlen(str) - 1;

   while (left < right) {
      if (tolower(str[left]) != tolower(str[right])) {
         return 0; // Not palindrome
      }
      left++;
      right--;
   }
```

```c
    return 1; // Palindrome
}

// Function to reverse a string
void reverseString(char str[], char reversed[]) {
    int length = strlen(str);
    int i, j = 0;

    for (i = length - 1; i >= 0; i--) {
        reversed[j] = str[i];
        j++;
    }
    reversed[j] = '\0';
}

int main() {
    char str[100], reversed[100];
    int result, resultCI;

    printf("========================================\n");
    printf("  Palindrome Checker\n");
    printf("========================================\n\n");

    printf("What is a Palindrome?\n");
    printf("A word/phrase that reads the same\n");
    printf("backwards as forwards.\n");
    printf("Examples: madam, racecar, noon, level\n\n");

    // Input string
    printf("Enter a string: ");
    scanf("%s", str);

    // Check palindrome (case-sensitive)
    result = isPalindrome(str);

    // Check palindrome (case-insensitive)
    resultCI = isPalindromeCaseInsensitive(str);

    // Reverse the string for display
    reverseString(str, reversed);

    // Display results
    printf("\n========================================\n");
    printf("  Analysis Results\n");
    printf("========================================\n\n");

    printf("Original string:  \"%s\"\n", str);
    printf("Reversed string:  \"%s\"\n\n", reversed);

    printf("--- Case-Sensitive Check ---\n");
    if (result) {
        printf("✓ \"%s\" is a PALINDROME\n", str);
        printf("  (reads same forwards and backwards)\n");
    } else {
        printf("✗ \"%s\" is NOT a palindrome\n", str);
        printf("  (forwards: %s, backwards: %s)\n", str, reversed);
    }

    printf("\n--- Case-Insensitive Check ---\n");
    if (resultCI) {
        printf("✓ \"%s\" is a PALINDROME (ignoring case)\n", str);
    } else {
        printf("✗ \"%s\" is NOT a palindrome (even ignoring case)\n", str);
    }

    // Character-by-character comparison display
    printf("\n--- Character Comparison ---\n");
    int length = strlen(str);
    printf("Position: ");
    for (int i = 0; i < length; i++) {
        printf("%3d ", i);
    }
    printf("\nForward:  ");
    for (int i = 0; i < length; i++) {
        printf("  %c ", str[i]);
    }
    printf("\nBackward: ");
    for (int i = length - 1; i >= 0; i--) {
        printf("  %c ", str[i]);
    }
    printf("\nMatch:    ");
    for (int i = 0; i < length; i++) {
        if (str[i] == str[length - 1 - i]) {
            printf("  ✓ ");
        } else {
            printf("  ✗ ");
        }
    }
    printf("\n");

    printf("\n========================================\n");
    printf("Summary:\n");
    printf("Length: %d characters\n", length);
    printf("Case-sensitive palindrome: %s\n", result ? "YES" : "NO");
    printf("Case-insensitive palindrome: %s\n", resultCI ? "YES" : "NO");
    printf("========================================\n");
    return 0;
}
```

Example 1:

```
Input:
  "madam"

Output:
========================================
  Palindrome Checker
========================================

Original string:  "madam"
Reversed string:  "madam"

--- Case-Sensitive Check ---
✓ "madam" is a PALINDROME
  (reads same forwards and backwards)

--- Case-Insensitive Check ---
✓ "madam" is a PALINDROME (ignoring case)

--- Character Comparison ---
Position:   0   1   2   3   4
Forward:    m   a   d   a   m
Backward:   m   a   d   a   m
Match:      ✓   ✓   ✓   ✓   ✓

Summary:
Length: 5 characters
Case-sensitive palindrome: YES
Case-insensitive palindrome: YES
```

Example 2:

```
Input:
  "hello"

Output:
Original string:  "hello"
Reversed string:  "olleh"

--- Case-Sensitive Check ---
✗ "hello" is NOT a palindrome
  (forwards: hello, backwards: olleh)

--- Character Comparison ---
Position:   0   1   2   3   4
Forward:    h   e   l   l   o
Backward:   o   l   l   e   h
Match:      ✗   ✗   ✓   ✗   ✗
```

Back to Table of Contents

xxix) String Find & Replace

Question:
Write a C program to replace a specific character/string by another character/string in a multiword string.

Description:
This program performs find-and-replace operations on a multiword string. It can replace:

1. A single character with another character
2. A substring with another substring (first occurrence)
3. A substring with another substring (all occurrences)

The program handles multiple occurrences and different lengths of old and new strings.

How to Solve:

1. Input the main string, search string, and replace string
2. **For character replacement**:
   - Loop through string
   - If character matches, replace it
3. **For substring replacement**:
   - Find occurrence of substring using string search
   - Create new string with replacement
   - Handle position shifting based on length difference
4. Count and display number of replacements made

Code:

```c
#include <stdio.h>
#include <string.h>

// Function to replace all occurrences of a character
int replaceChar(char str[], char oldChar, char newChar) {
    int count = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == oldChar) {
            str[i] = newChar;
            count++;
        }
    }
    return count;
}

// Function to replace first occurrence of substring
int replaceFirstSubstring(char str[], char oldStr[], char newStr[]) {
    char result[500] = "";
    char *pos = strstr(str, oldStr);

    if (pos == NULL) {
        return 0; // Substring not found
    }

    // Copy part before the match
    strncat(result, str, pos - str);

    // Add replacement string
    strcat(result, newStr);

    // Add part after the match
    strcat(result, pos + strlen(oldStr));

    // Copy result back to original string
    strcpy(str, result);

    return 1; // Replacement made
}

// Function to replace all occurrences of substring
int replaceAllSubstrings(char str[], char oldStr[], char newStr[]) {
    char result[500] = "";
    char temp[500];
    int count = 0;

    strcpy(temp, str);

    char *pos;
    while ((pos = strstr(temp, oldStr)) != NULL) {
        // Copy part before the match
        strncat(result, temp, pos - temp);

        // Add replacement string
        strcat(result, newStr);

        // Move temp pointer after the match
        strcpy(temp, pos + strlen(oldStr));
        count++;
    }

    // Add remaining part
    strcat(result, temp);

    // Copy result back to original string
    strcpy(str, result);

    return count;
}

int main() {
    char str[500], oldStr[100], newStr[100];
    char backup[500];
    int choice, count;
    char oldChar, newChar;

    printf("========================================\n");
    printf("  String Find and Replace\n");
    printf("========================================\n\n");

    // Input main string
    printf("Enter the main string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0'; // Remove newline

    strcpy(backup, str); // Backup original string

    printf("\nOriginal String: \"%s\"\n", str);

    // Choose operation
    printf("\n--- Choose Operation ---\n");
    printf("1. Replace character\n");
    printf("2. Replace substring (first occurrence)\n");
    printf("3. Replace substring (all occurrences)\n");
    printf("Enter choice (1-3): ");
    scanf("%d", &choice);
    getchar(); // Consume newline

    printf("\n========================================\n");

    switch (choice) {
    case 1:
        // Replace character
```

```
            printf("Enter character to find: ");
            scanf("%c", &oldChar);
            getchar();
            printf("Enter character to replace with: ");
            scanf("%c", &newChar);

            count = replaceChar(str, oldChar, newChar);

            printf("\n--- Character Replacement ---\n");
            printf("Original:  \"%s\"\n", backup);
            printf("Modified:  \"%s\"\n", str);
            printf("Replaced '%c' with '%c' (%d occurrence%s)\n",
                   oldChar, newChar, count, count != 1 ? "s" : "");
            break;

        case 2:
            // Replace first occurrence
            printf("Enter substring to find: ");
            fgets(oldStr, sizeof(oldStr), stdin);
            oldStr[strcspn(oldStr, "\n")] = '\0';

            printf("Enter replacement string: ");
            fgets(newStr, sizeof(newStr), stdin);
            newStr[strcspn(newStr, "\n")] = '\0';

            count = replaceFirstSubstring(str, oldStr, newStr);

            printf("\n--- Substring Replacement (First) ---\n");
            printf("Original:  \"%s\"\n", backup);
            printf("Modified:  \"%s\"\n", str);
            if (count > 0) {
                printf("Replaced \"%s\" with \"%s\" (first occurrence)\n",
                       oldStr, newStr);
            } else {
                printf("\"%s\" not found in string\n", oldStr);
            }
            break;

        case 3:
            // Replace all occurrences
            printf("Enter substring to find: ");
            fgets(oldStr, sizeof(oldStr), stdin);
            oldStr[strcspn(oldStr, "\n")] = '\0';

            printf("Enter replacement string: ");
            fgets(newStr, sizeof(newStr), stdin);
            newStr[strcspn(newStr, "\n")] = '\0';

            count = replaceAllSubstrings(str, oldStr, newStr);

            printf("\n--- Substring Replacement (All) ---\n");
            printf("Original:  \"%s\"\n", backup);
            printf("Modified:  \"%s\"\n", str);
            printf("Replaced \"%s\" with \"%s\" (%d occurrence%s)\n",
                   oldStr, newStr, count, count != 1 ? "s" : "");
            break;

        default:
            printf("Invalid choice!\n");
            return 1;
    }

    printf("\n========================================\n");
    printf("Summary:\n");
    printf("Original length: %lu characters\n", strlen(backup));
    printf("New length:      %lu characters\n", strlen(str));
    printf("Replacements:    %d\n", count);
    printf("========================================\n");

    return 0;
}
```

Example 1:

```
Input:
  String: "Hello World"
  Choice: 1
  Find: 'o'
  Replace: '0'

Output:
--- Character Replacement ---
Original:  "Hello World"
Modified:  "Hell0 W0rld"
Replaced 'o' with '0' (2 occurrences)

Summary:
Original length: 11 characters
New length:      11 characters
Replacements:    2
```

Example 2:

```
Input:
  String: "I love C. C is powerful. C programming is fun."
  Choice: 3
  Find: "C"
  Replace: "Python"

Output:
--- Substring Replacement (All) ---
Original:  "I love C. C is powerful. C programming is fun."
Modified:  "I love Python. Python is powerful. Python programming is fun."
Replaced "C" with "Python" (3 occurrences)

Summary:
Original length: 46 characters
New length:      61 characters
Replacements:    3
```

Back to Table of Contents

xxx) String Abbreviation

Question:
Write a C program to make the abbreviated form of a multiword string.

Description:
This program creates an abbreviation from a multiword string by taking the first letter of each word. Examples:

- "United States of America" → "USA"
- "World Health Organization" → "WHO"
- "Central Processing Unit" → "CPU"

The program handles:

- Multiple spaces between words
- Leading and trailing spaces
- Mixed case (converts abbreviation to uppercase)

How to Solve:

1. Input a multiword string from user
2. Track if previous character was a space (or start of string)
3. Loop through each character:
    - If current is a letter and previous was space/start:
        - Add uppercase version to abbreviation
    - Track current character for next iteration
4. Display original string, abbreviation, and word breakdown

Code:

```c
#include <ctype.h>
#include <stdio.h>
#include <string.h>

// Function to create abbreviation from multiword string
void createAbbreviation(char str[], char abbr[]) {
   int i = 0, j = 0;
   int isNewWord = 1; // Flag to track start of new word

   while (str[i] != '\0') {
      // Skip spaces
      if (str[i] == ' ' || str[i] == '\t') {
         isNewWord = 1; // Next non-space char starts a new word
      }
      // If it's a letter and it's the start of a new word
      else if (isalpha(str[i]) && isNewWord) {
         abbr[j++] = toupper(str[i]); // Add uppercase letter to abbreviation
         isNewWord = 0;               // We're now inside a word
      }
      // If it's any other character, we're inside a word
      else if (str[i] != ' ' && str[i] != '\t') {
         isNewWord = 0;
      }
      i++;
   }
   abbr[j] = '\0'; // Null terminate the abbreviation
}

// Function to count words in string
int countWords(char str[]) {
   int count = 0;
   int isWord = 0;

   for (int i = 0; str[i] != '\0'; i++) {
      if (str[i] == ' ' || str[i] == '\t' || str[i] == '\n') {
         isWord = 0;
      } else if (isWord == 0) {
         isWord = 1;
         count++;
      }
   }
   return count;
```

```c
}

// Function to extract individual words
void extractWords(char str[], char words[][50], int *wordCount) {
   int i = 0, j = 0, k = 0;
   *wordCount = 0;

   while (str[i] != '\0') {
      // Skip spaces
      if (str[i] == ' ' || str[i] == '\t') {
         if (k > 0) { // End of a word
            words[j][k] = '\0';
            j++;
            (*wordCount)++;
            k = 0;
         }
      } else {
         words[j][k++] = str[i];
      }
      i++;
   }

   // Handle last word
   if (k > 0) {
      words[j][k] = '\0';
      (*wordCount)++;
   }
}

int main() {
   char str[200], abbr[50];
   char words[50][50];
   int wordCount;

   printf("========================================\n");
   printf("  String Abbreviation Generator\n");
   printf("========================================\n\n");

   printf("Examples:\n");
   printf("  'United States of America' → USA\n");
   printf("  'World Health Organization' → WHO\n");
   printf("  'Central Processing Unit' → CPU\n\n");

   // Input string
   printf("Enter a multiword string: ");
   fgets(str, sizeof(str), stdin);
   str[strcspn(str, "\n")] = '\0'; // Remove newline

   // Create abbreviation
   createAbbreviation(str, abbr);

   // Count words
   wordCount = countWords(str);

   // Extract individual words
   int extractedCount;
   extractWords(str, words, &extractedCount);

   // Display results
   printf("\n========================================\n");
   printf("  Results\n");
   printf("========================================\n\n");

   printf("Original String:\n");
   printf("  \"%s\"\n\n", str);

   printf("Abbreviation:\n");
   printf("  \"%s\"\n\n", abbr);

   printf("--- Word Breakdown ---\n");
   printf("Total words: %d\n\n", wordCount);

   printf("Individual words:\n");
   for (int i = 0; i < extractedCount; i++) {
      printf("  Word %d: %-20s → %c\n", i + 1, words[i], toupper(words[i][0]));
   }

   printf("\n--- Abbreviation Details ---\n");
   printf("Length of abbreviation: %lu characters\n", strlen(abbr));
   printf("Compression ratio: %.1f%%\n",
          (1.0 - (float)strlen(abbr) / strlen(str)) * 100);

   printf("\n--- Letter Breakdown ---\n");
   printf("Full form letters taken:\n");
   for (int i = 0; i < extractedCount; i++) {
      printf("  %c from '%s'\n", toupper(words[i][0]), words[i]);
   }

   printf("\n========================================\n");
   printf("Summary:\n");
   printf("Input:  \"%s\"\n", str);
   printf("Output: \"%s\"\n", abbr);
   printf("Words:  %d\n", wordCount);
   printf("========================================\n");
```

```
    return 0;
}
```

Example 1:

```
Input:
  "United States of America"

Output:
========================================
  String Abbreviation Generator
========================================

Original String:
  "United States of America"

Abbreviation:
  "USOA"

--- Word Breakdown ---
Total words: 4

Individual words:
  Word 1: United               → U
  Word 2: States               → S
  Word 3: of                   → O
  Word 4: America              → A

--- Abbreviation Details ---
Length of abbreviation: 4 characters
Compression ratio: 83.3%

--- Letter Breakdown ---
Full form letters taken:
  U from 'United'
  S from 'States'
  O from 'of'
  A from 'America'

Summary:
Input:  "United States of America"
Output: "USOA"
Words:  4
```

Example 2:

```
Input:
  "World Health Organization"

Output:
Abbreviation: "WHO"

--- Word Breakdown ---
Total words: 3

Individual words:
  Word 1: World                → W
  Word 2: Health               → H
  Word 3: Organization         → O

Compression ratio: 88.5%
```

Back to Table of Contents

Section 5: Functions, Pointers & Structures

xxxi) nCr Combination Calculator

Question:
Write a C program to calculate the value of nCr, where n ≥ r using function.

Description:
This program calculates combinations (nCr) which represents the number of ways to choose r items from n items without regard to order.

Formula: nCr = n! / (r! × (n-r)!)

Where:

- n! is the factorial of n
- r! is the factorial of r
- (n-r)! is the factorial of (n-r)

Examples:

- 5C2 = 5!/(2!×3!) = 120/(2×6) = 10
- 10C3 = 10!/(3!×7!) = 120
- 6C6 = 1 (choosing all items)
- 6C0 = 1 (choosing no items)

How to Solve:

1. Create a function to calculate factorial of a number
2. Create a function to calculate nCr using the formula
3. Input n and r from user
4. Validate that n ≥ r ≥ 0
5. Calculate nCr = n! / (r! × (n-r)!)
6. Display the result with detailed calculation steps
7. Handle edge cases (r=0, r=n, etc.)

Code:

```
#include <stdio.h>

// Function to calculate factorial
unsigned long long factorial(int num) {
   unsigned long long fact = 1;
   for (int i = 1; i <= num; i++) {
      fact *= i;
   }
   return fact;
}

// Function to calculate nCr using factorial method
unsigned long long nCr_factorial(int n, int r) {
   return factorial(n) / (factorial(r) * factorial(n - r));
}

// Optimized function to calculate nCr (to avoid overflow for large numbers)
unsigned long long nCr_optimized(int n, int r) {
   // nCr = nC(n-r), choose smaller value for efficiency
   if (r > n - r) {
      r = n - r;
   }

   unsigned long long result = 1;

   // Calculate C(n, r) using: C(n, r) = n × (n-1) × ... × (n-r+1) / r!
   for (int i = 0; i < r; i++) {
      result *= (n - i);
      result /= (i + 1);
   }

   return result;
}

// Function to display calculation steps
void displayCalculation(int n, int r) {
   printf("--- Calculation Steps ---\n\n");
   printf("Formula: nCr = n! / (r! × (n-r)!)\n\n");

   unsigned long long n_fact = factorial(n);
   unsigned long long r_fact = factorial(r);
   unsigned long long nr_fact = factorial(n - r);

   printf("Step 1: Calculate factorials\n");
   printf("  n!     = %d! = %llu\n", n, n_fact);
   printf("  r!     = %d! = %llu\n", r, r_fact);
   printf("  (n-r)! = (%d-%d)! = %d! = %llu\n", n, r, n - r, nr_fact);

   printf("\nStep 2: Apply formula\n");
   printf("  %dC%d = %llu / (%llu × %llu)\n", n, r, n_fact, r_fact, nr_fact);
   printf("       = %llu / %llu\n", n_fact, r_fact * nr_fact);
   printf("       = %llu\n", nCr_factorial(n, r));
}

int main() {
   int n, r;
   unsigned long long result;

   printf("========================================\n");
   printf("  Combination Calculator (nCr)\n");
   printf("========================================\n\n");

   printf("nCr: Number of ways to choose r items\n");
   printf("     from n items (order doesn't matter)\n\n");

   printf("Formula: nCr = n! / (r! × (n-r)!)\n\n");

   // Input values
   printf("Enter value of n: ");
   scanf("%d", &n);
   printf("Enter value of r: ");
   scanf("%d", &r);

   // Validation
   if (n < 0 || r < 0) {
      printf("\nError: n and r must be non-negative!\n");
      return 1;
   }

   if (r > n) {
      printf("\nError: r cannot be greater than n!\n");
      printf("Condition: n ≥ r must be satisfied\n");
      return 1;
   }
```

```c
    printf("\n========================================\n");
    printf("  Results\n");
    printf("========================================\n\n");

    // Display calculation steps for smaller values
    if (n <= 12) {
       displayCalculation(n, r);
    } else {
       printf("Using optimized calculation for large values...\n");
    }

    // Calculate using optimized method
    result = nCr_optimized(n, r);

    printf("\n========================================\n");
    printf("  Final Answer\n");
    printf("========================================\n\n");

    printf("%dC%d = %llu\n", n, r, result);

    // Special cases explanation
    printf("\n--- Interpretation ---\n");
    if (r == 0) {
       printf("There is exactly 1 way to choose 0 items\n");
       printf("from %d items (choose nothing).\n", n);
    } else if (r == n) {
       printf("There is exactly 1 way to choose all %d items\n", n);
       printf("from %d items (choose everything).\n", n);
    } else if (r == 1) {
       printf("There are %d ways to choose 1 item\n", n);
       printf("from %d items.\n", n);
    } else if (r == n - 1) {
       printf("There are %d ways to choose %d items\n", n, r);
       printf("from %d items (same as choosing which 1 to leave out).\n", n);
    } else {
       printf("There are %llu ways to choose %d items\n", result, r);
       printf("from %d items without regard to order.\n", n);
    }

    // Additional information
    printf("\n--- Additional Information ---\n");
    printf("Property: %dC%d = %dC%d = %llu\n", n, r, n, n - r, result);
    printf("(Choosing r items is same as leaving out (n-r) items)\n");

    printf("\n========================================\n");

    return 0;
}
```

Example:

```
Input:
  n = 5, r = 2

Output:
========================================
  Combination Calculator (nCr)
========================================

nCr: Number of ways to choose r items
     from n items (order doesn't matter)

Formula: nCr = n! / (r! × (n-r)!)

Enter value of n: 5
Enter value of r: 2

========================================
  Results
========================================

--- Calculation Steps ---

Formula: nCr = n! / (r! × (n-r)!)

Step 1: Calculate factorials
  n!     = 5! = 120
  r!     = 2! = 2
  (n-r)! = (5-2)! = 3! = 6

Step 2: Apply formula
  5C2 = 120 / (2 × 6)
      = 120 / 12
      = 10

========================================
  Final Answer
========================================

5C2 = 10

--- Interpretation ---
There are 10 ways to choose 2 items
from 5 items without regard to order.
```

```
--- Additional Information ---
Property: 5C2 = 5C3 = 10
(Choosing r items is same as leaving out (n-r) items)

========================================
```

Back to Table of Contents

xxxii) Exponential Series Calculator

Question:
Write a C program to find the sum of the series: 1 + (x/1!) + (x^2/2!) + ... + (x^n/n!) for n ≥ 1, x ≥ 0 using function.

Description:
This program calculates the exponential series (Taylor series for e^x):

e^x = 1 + x/1! + x^2/2! + x^3/3! + ... + x^n/n!

This is the Taylor series expansion of the exponential function.

Examples:

- e^1 ≈ 1 + 1 + 0.5 + 0.167 + ... = 2.71828...
- e^2 ≈ 1 + 2 + 2 + 1.333 + ... = 7.38906...

The program uses functions to:

1. Calculate factorial
2. Calculate power (x^n)
3. Calculate each term of series
4. Sum the series up to n terms

How to Solve:

1. Create function to calculate factorial
2. Create function to calculate power (x^n)
3. Create function to calculate a term: x^i / i!
4. Create function to sum the series
5. Input x and n from user
6. Calculate sum = 1 + x/1! + x^2/2! + ... + x^n/n!
7. Display each term and running sum
8. Compare with actual e^x value using math.h

Code:

```c
#include <math.h>
#include <stdio.h>

// Function to calculate factorial
unsigned long long factorial(int num) {
   if (num == 0 || num == 1) {
      return 1;
   }
   unsigned long long fact = 1;
   for (int i = 2; i <= num; i++) {
      fact *= i;
   }
   return fact;
}

// Function to calculate power (base^exponent)
double power(double base, int exponent) {
   if (exponent == 0) {
      return 1.0;
   }
   double result = 1.0;
   for (int i = 0; i < exponent; i++) {
      result *= base;
   }
   return result;
}

// Function to calculate a single term: x^i / i!
double calculateTerm(double x, int i) {
   if (i == 0) {
      return 1.0;
   }
   return power(x, i) / factorial(i);
}

// Function to calculate sum of the series
double exponentialSeries(double x, int n) {
   double sum = 0.0;
   for (int i = 0; i <= n; i++) {
      sum += calculateTerm(x, i);
   }
   return sum;
}

// Optimized function to avoid recalculating powers and factorials
double exponentialSeriesOptimized(double x, int n) {
   double sum = 1.0;  // First term is always 1
```

```c
    double term = 1.0; // Current term

    for (int i = 1; i <= n; i++) {
        term *= x / i; // term = term × (x/i)
        sum += term;
    }

    return sum;
}

// Function to display detailed calculation
void displayCalculation(double x, int n) {
    double sum = 0.0;

    printf("--- Term-by-Term Calculation ---\n\n");
    printf("Series: 1 + (x/1!) + (x²/2!) + ... + (xⁿ/n!)\n");
    printf("Where x = %.2f, n = %d\n\n", x, n);

    printf("%-6s %-15s %-15s %-15s %-15s\n",
           "Term", "Power (x^i)", "Factorial", "Value", "Running Sum");
    printf("----------------------------------------------------------------------\n");

    for (int i = 0; i <= n; i++) {
        double termValue = calculateTerm(x, i);
        sum += termValue;

        if (i == 0) {
            printf("%-6d %-15s %-15s %-15.10f %-15.10f\n",
                   i, "x^0 = 1", "0! = 1", termValue, sum);
        } else {
            char powerStr[20], factStr[20];
            sprintf(powerStr, "%.2f^%d = %.4f", x, i, power(x, i));
            sprintf(factStr, "%d! = %llu", i, factorial(i));
            printf("%-6d %-15s %-15s %-15.10f %-15.10f\n",
                   i, powerStr, factStr, termValue, sum);
        }
    }

    printf("----------------------------------------------------------------------\n");
    printf("Final Sum: %.10f\n", sum);
}

int main() {
    double x;
    int n;

    printf("========================================\n");
    printf("  Exponential Series Calculator\n");
    printf("========================================\n\n");

    printf("Series: e^x = 1 + x/1! + x²/2! + x³/3! + ... + xⁿ/n!\n\n");

    printf("This calculates the Taylor series expansion\n");
    printf("of the exponential function e^x\n\n");

    // Input values
    printf("Enter value of x (x ≥ 0): ");
    scanf("%lf", &x);
    printf("Enter number of terms n (n ≥ 1): ");
    scanf("%d", &n);

    // Validation
    if (x < 0) {
        printf("\nError: x must be non-negative (x ≥ 0)!\n");
        return 1;
    }

    if (n < 1) {
        printf("\nError: n must be at least 1 (n ≥ 1)!\n");
        return 1;
    }

    printf("\n========================================\n");
    printf("  Calculation Details\n");
    printf("========================================\n\n");

    // Display detailed calculation for smaller n
    if (n <= 15) {
        displayCalculation(x, n);
    } else {
        printf("Using optimized calculation for n > 15...\n");
    }

    // Calculate using optimized method
    double seriesSum = exponentialSeriesOptimized(x, n);
    double actualValue = exp(x); // Actual e^x using math library
    double error = fabs(actualValue - seriesSum);
    double errorPercent = (error / actualValue) * 100;

    printf("\n========================================\n");
    printf("  Results Summary\n");
    printf("========================================\n\n");

    printf("Input Parameters:\n");
    printf("  x = %.4f\n", x);
    printf("  n = %d terms\n\n", n);
    printf("Series Formula:\n");
    printf("  Sum = 1");
    for (int i = 1; i <= (n < 5 ? n : 4); i++) {
        printf(" + (%.2f^%d/%d!)", x, i, i);
    }
    if (n > 4) {
        printf(" + ...");
    }
    printf("\n\n");

    printf("Results:\n");
    printf("  Series Sum (calculated):  %.10f\n", seriesSum);
    printf("  Actual e^%.4f (from math.h): %.10f\n", x, actualValue);
    printf("  Absolute Error:           %.10f\n", error);
    printf("  Relative Error:           %.6f%%\n\n", errorPercent);

    // Convergence information
    printf("--- Convergence Analysis ---\n");
    if (errorPercent < 0.001) {
        printf("✓ Excellent approximation! (< 0.001%% error)\n");
    } else if (errorPercent < 0.1) {
        printf("✓ Very good approximation (< 0.1%% error)\n");
    } else if (errorPercent < 1.0) {
        printf("○ Good approximation (< 1%% error)\n");
    } else {
        printf("○ Fair approximation (%.3f%% error)\n", errorPercent);
        printf("  Consider using more terms for better accuracy\n");
    }

    printf("\n--- Mathematical Insight ---\n");
    printf("The series converges to e^x as n → ∞\n");
    printf("For x = %.2f, the series approximates e^%.2f\n", x, x);
    printf("Number of terms used: %d\n", n + 1);
    printf("Accuracy achieved: %.6f%%\n", 100 - errorPercent);

    printf("\n========================================\n");

    return 0;
}
```

Example:

```
Input:
  x = 1, n = 5

Output:
========================================
  Exponential Series Calculator
========================================

Series: e^x = 1 + x/1! + x²/2! + x³/3! + ... + xⁿ/n!

This calculates the Taylor series expansion
of the exponential function e^x

Enter value of x (x ≥ 0): 1
Enter number of terms n (n ≥ 1): 5

========================================
  Calculation Details
========================================

--- Term-by-Term Calculation ---

Series: 1 + (x/1!) + (x²/2!) + ... + (xⁿ/n!)
Where x = 1.00, n = 5

Term   Power (x^i)     Factorial       Value           Running Sum
-------------------------------------------------------------------
0      x^0 = 1         0! = 1          1.0000000000    1.0000000000
1      1.00^1 = 1.0000 1! = 1          1.0000000000    2.0000000000
2      1.00^2 = 1.0000 2! = 2          0.5000000000    2.5000000000
3      1.00^3 = 1.0000 3! = 6          0.1666666667    2.6666666667
4      1.00^4 = 1.0000 4! = 24         0.0416666667    2.7083333333
5      1.00^5 = 1.0000 5! = 120        0.0083333333    2.7166666667
-------------------------------------------------------------------
Final Sum: 2.7166666667

========================================
  Results Summary
========================================

Input Parameters:
  x = 1.0000
  n = 5 terms

Series Formula:
  Sum = 1 + (1.00^1/1!) + (1.00^2/2!) + (1.00^3/3!) + (1.00^4/4!)

Results:
  Series Sum (calculated):  2.7166666667
  Actual e^1.0000 (from math.h): 2.7182818285
  Absolute Error:           0.0016151618
  Relative Error:           0.059423%
```

```
--- Convergence Analysis ---
✓ Very good approximation (< 0.1% error)

--- Mathematical Insight ---
The series converges to e^x as n → ∞
For x = 1.00, the series approximates e^1.00
Number of terms used: 6
Accuracy achieved: 99.940577%

========================================
```

Back to Table of Contents

xxxiii) Array Min-Max Interchange

Question:
Write a C program to interchange the biggest and smallest number in a one dimensional array using function.

Description:
This program finds the largest and smallest elements in an array and swaps their positions.

Example:

- Array: [5, 2, 9, 1, 7]
- Smallest: 1 (at index 3), Largest: 9 (at index 2)
- After swap: [5, 2, 1, 9, 7]

The program uses separate functions to:

1. Find the index of the smallest element
2. Find the index of the largest element
3. Swap elements at two positions
4. Display the array

How to Solve:

1. Input array size and elements
2. Create function to find minimum element's index
3. Create function to find maximum element's index
4. Create function to swap two elements
5. Display array before swap
6. Find positions of min and max
7. Swap the elements at those positions
8. Display array after swap
9. Handle edge cases (all same elements, two elements, etc.)

Code:

```c
#include <stdio.h>

// Function to display array
void displayArray(int arr[], int size) {
   printf("[");
   for (int i = 0; i < size; i++) {
      printf("%d", arr[i]);
      if (i < size - 1) {
         printf(", ");
      }
   }
   printf("]\n");
}

// Function to find index of smallest element
int findMinIndex(int arr[], int size) {
   int minIndex = 0;
   for (int i = 1; i < size; i++) {
      if (arr[i] < arr[minIndex]) {
         minIndex = i;
      }
   }
   return minIndex;
}

// Function to find index of largest element
int findMaxIndex(int arr[], int size) {
   int maxIndex = 0;
   for (int i = 1; i < size; i++) {
      if (arr[i] > arr[maxIndex]) {
         maxIndex = i;
      }
   }
   return maxIndex;
}

// Function to swap two elements in array
void swap(int arr[], int index1, int index2) {
   int temp = arr[index1];
   arr[index1] = arr[index2];
   arr[index2] = temp;
}
```

```c
// Function to interchange min and max
void interchangeMinMax(int arr[], int size) {
   int minIndex = findMinIndex(arr, size);
   int maxIndex = findMaxIndex(arr, size);

   printf("\n--- Finding Min and Max ---\n");
   printf("Smallest element: %d at index %d\n", arr[minIndex], minIndex);
   printf("Largest element:  %d at index %d\n", arr[maxIndex], maxIndex);

   if (minIndex == maxIndex) {
      printf("\nNote: All elements are the same. No swap needed.\n");
      return;
   }

   printf("\n--- Swapping ---\n");
   printf("Swapping arr[%d] = %d with arr[%d] = %d\n",
          minIndex, arr[minIndex], maxIndex, arr[maxIndex]);

   swap(arr, minIndex, maxIndex);

   printf("Swap completed successfully!\n");
}

int main() {
   int size;
   int arr[100];

   printf("========================================\n");
   printf("  Min-Max Interchange in Array\n");
   printf("========================================\n\n");

   printf("This program swaps the positions of\n");
   printf("the smallest and largest elements\n");
   printf("in an array.\n\n");

   // Input array size
   printf("Enter the size of array: ");
   scanf("%d", &size);

   if (size <= 0 || size > 100) {
      printf("Error: Size must be between 1 and 100\n");
      return 1;
   }

   // Input array elements
   printf("Enter %d elements:\n", size);
   for (int i = 0; i < size; i++) {
      printf("Element [%d]: ", i);
      scanf("%d", &arr[i]);
   }

   printf("\n========================================\n");
   printf("  Array Before Swap\n");
   printf("========================================\n\n");

   printf("Array: ");
   displayArray(arr, size);

   // Display array with indices
   printf("\nIndex: ");
   for (int i = 0; i < size; i++) {
      printf("%3d ", i);
   }
   printf("\nValue: ");
   for (int i = 0; i < size; i++) {
      printf("%3d ", arr[i]);
   }
   printf("\n");

   // Perform interchange
   interchangeMinMax(arr, size);

   printf("\n========================================\n");
   printf("  Array After Swap\n");
   printf("========================================\n\n");

   printf("Array: ");
   displayArray(arr, size);

   // Display array with indices
   printf("\nIndex: ");
   for (int i = 0; i < size; i++) {
      printf("%3d ", i);
   }
   printf("\nValue: ");
   for (int i = 0; i < size; i++) {
      printf("%3d ", arr[i]);
   }
   printf("\n");

   // Summary
   printf("\n========================================\n");
   printf("  Summary\n");
   printf("========================================\n");
   printf("Operation: Interchange min and max\n");
   printf("Array size: %d elements\n", size);
   printf("Result: Successfully swapped!\n");
```

```
    printf("========================================\n");

    return 0;
}
```

Example:

```
Input:
  Size: 5
  Elements: 5, 2, 9, 1, 7

Output:
========================================
  Min-Max Interchange in Array
========================================

This program swaps the positions of
the smallest and largest elements
in an array.

Enter the size of array: 5
Enter 5 elements:
Element [0]: 5
Element [1]: 2
Element [2]: 9
Element [3]: 1
Element [4]: 7

========================================
  Array Before Swap
========================================

Array: [5, 2, 9, 1, 7]

Index:   0   1   2   3   4
Value:   5   2   9   1   7

--- Finding Min and Max ---
Smallest element: 1 at index 3
Largest element:  9 at index 2

--- Swapping ---
Swapping arr[3] = 1 with arr[2] = 9
Swap completed successfully!

========================================
  Array After Swap
========================================

Array: [5, 2, 1, 9, 7]

Index:   0   1   2   3   4
Value:   5   2   1   9   7

========================================
  Summary
========================================
Operation: Interchange min and max
Array size: 5 elements
Result: Successfully swapped!
========================================
```

Back to Table of Contents

xxxiv) Factorial Using Recursion

Question:
Write a C program to calculate factorial of any given number using recursion.

Description:
This program calculates the factorial of a number using recursive function. Recursion is a technique where a function calls itself to solve a problem.

Factorial is defined as:

- n! = n × (n-1) × (n-2) × ... × 2 × 1
- 0! = 1 (by definition)
- 5! = 5 × 4 × 3 × 2 × 1 = 120

Recursive definition:

- Base case: factorial(0) = 1, factorial(1) = 1
- Recursive case: factorial(n) = n × factorial(n-1)

Example recursion trace for factorial(5):

```
factorial(5) = 5 × factorial(4)
             = 5 × (4 × factorial(3))
             = 5 × (4 × (3 × factorial(2)))
             = 5 × (4 × (3 × (2 × factorial(1))))
             = 5 × (4 × (3 × (2 × 1)))
             = 120
```

How to Solve:

1. Create a recursive function factorial(n)
2. Define base case: if n == 0 or n == 1, return 1
3. Define recursive case: return n × factorial(n-1)
4. Input a number from user
5. Call recursive function
6. Display result with recursion trace
7. Handle negative numbers (factorial not defined)

Code:

```c
#include <stdio.h>

// Global variable to track recursion depth (for visualization)
int recursionDepth = 0;

// Recursive function to calculate factorial
unsigned long long factorialRecursive(int n) {
   // Increment depth for visualization
   recursionDepth++;

   // Print recursion trace
   for (int i = 0; i < recursionDepth - 1; i++) {
      printf("  ");
   }
   printf("→ factorial(%d) called\n", n);

   // Base case
   if (n == 0 || n == 1) {
      for (int i = 0; i < recursionDepth - 1; i++) {
         printf("  ");
      }
      printf("← factorial(%d) returns 1 (base case)\n", n);
      recursionDepth--;
      return 1;
   }

   // Recursive case
   unsigned long long result = n * factorialRecursive(n - 1);

   for (int i = 0; i < recursionDepth - 1; i++) {
      printf("  ");
   }
   printf("← factorial(%d) returns %d × factorial(%d) = %llu\n",
          n, n, n - 1, result);

   recursionDepth--;
   return result;
}

// Simple recursive factorial (without trace)
unsigned long long factorial(int n) {
   if (n == 0 || n == 1) {
      return 1;
   }
   return n * factorial(n - 1);
}

// Iterative version for comparison
unsigned long long factorialIterative(int n) {
   unsigned long long result = 1;
   for (int i = 2; i <= n; i++) {
      result *= i;
   }
   return result;
}

int main() {
   int num;
   unsigned long long result;

   printf("========================================\n");
   printf("  Factorial Calculator (Recursion)\n");
   printf("========================================\n\n");

   printf("Factorial (n!) = n × (n-1) × ... × 2 × 1\n");
   printf("This program uses RECURSION to calculate factorial\n\n");

   printf("Examples:\n");
   printf("  5! = 5 × 4 × 3 × 2 × 1 = 120\n");
   printf("  0! = 1 (by definition)\n\n");

   // Input number
   printf("Enter a non-negative integer: ");
   scanf("%d", &num);

   // Validation
   if (num < 0) {
      printf("\nError: Factorial is not defined for negative numbers!\n");
      return 1;
   }

   if (num > 20) {
      printf("\nWarning: Factorial of numbers > 20 may cause overflow.\n");
      printf("Proceeding with calculation...\n");
   }

   printf("\n========================================\n");
   printf("  Recursion Trace\n");
   printf("========================================\n\n");

   printf("Showing how recursion works:\n\n");
   recursionDepth = 0;
   result = factorialRecursive(num);

   printf("\n========================================\n");
   printf("  Mathematical Expansion\n");
   printf("========================================\n\n");

   printf("%d! = ", num);
   if (num == 0 || num == 1) {
      printf("1\n");
   } else {
      for (int i = num; i >= 1; i--) {
         printf("%d", i);
         if (i > 1) {
            printf(" × ");
         }
      }
      printf("\n   = %llu\n", result);
   }

   printf("\n========================================\n");
   printf("  Recursion Explanation\n");
   printf("========================================\n\n");

   printf("Recursive formula:\n");
   printf("  factorial(n) = n × factorial(n-1)\n");
   printf("  factorial(0) = 1 (base case)\n");
   printf("  factorial(1) = 1 (base case)\n\n");

   if (num > 1) {
      printf("For %d!:\n", num);
      printf("  factorial(%d) = %d × factorial(%d)\n", num, num, num - 1);
      if (num > 2) {
         printf("               = %d × %d × factorial(%d)\n",
                num, num - 1, num - 2);
      }
      if (num > 3) {
         printf("               = %d × %d × %d × factorial(%d)\n",
                num, num - 1, num - 2, num - 3);
      }
      printf("               ...\n");
      printf("               = %llu\n", result);
   }

   printf("\n========================================\n");
   printf("  Final Result\n");
   printf("========================================\n\n");

   printf("%d! = %llu\n", num, result);

   // Comparison with iterative method
   printf("\n--- Verification ---\n");
   unsigned long long iterativeResult = factorialIterative(num);
   printf("Recursive result:  %llu\n", result);
   printf("Iterative result:  %llu\n", iterativeResult);
   if (result == iterativeResult) {
      printf("✓ Results match!\n");
   }

   // Additional information
   printf("\n--- Recursion vs Iteration ---\n");
   printf("Recursion:\n");
   printf("  + Elegant and simple code\n");
   printf("  + Mirrors mathematical definition\n");
   printf("  - Uses more memory (call stack)\n");
   printf("  - Slower due to function call overhead\n\n");
   printf("Iteration:\n");
   printf("  + More memory efficient\n");
   printf("  + Faster execution\n");
   printf("  - More code to write\n");

   printf("\n========================================\n");
   printf("Summary:\n");
   printf("Input: %d\n", num);
   printf("Factorial: %llu\n", result);
   printf("Method: Recursion\n");
   printf("Recursion depth: %d levels\n", num);
   printf("========================================\n");

   return 0;
}
```

Example:

```
Input:
  5

Output:
========================================
  Factorial Calculator (Recursion)
========================================

Factorial (n!) = n × (n-1) × ... × 2 × 1
This program uses RECURSION to calculate factorial

Examples:
  5! = 5 × 4 × 3 × 2 × 1 = 120
  0! = 1 (by definition)

Enter a non-negative integer: 5

========================================
  Recursion Trace
========================================

Showing how recursion works:

→ factorial(5) called
  → factorial(4) called
    → factorial(3) called
      → factorial(2) called
        → factorial(1) called
        ← factorial(1) returns 1 (base case)
      ← factorial(2) returns 2 × factorial(1) = 2
    ← factorial(3) returns 3 × factorial(2) = 6
  ← factorial(4) returns 4 × factorial(3) = 24
← factorial(5) returns 5 × factorial(4) = 120

========================================
  Mathematical Expansion
========================================

5! = 5 × 4 × 3 × 2 × 1
   = 120

========================================
  Recursion Explanation
========================================

Recursive formula:
  factorial(n) = n × factorial(n-1)
  factorial(0) = 1 (base case)
  factorial(1) = 1 (base case)

For 5!:
  factorial(5) = 5 × factorial(4)
               = 5 × 4 × factorial(3)
               = 5 × 4 × 3 × factorial(2)
               ...
               = 120

========================================
  Final Result
========================================

5! = 120

--- Verification ---
Recursive result:  120
Iterative result:  120
✓ Results match!

--- Recursion vs Iteration ---
Recursion:
  + Elegant and simple code
  + Mirrors mathematical definition
  - Uses more memory (call stack)
  - Slower due to function call overhead

Iteration:
  + More memory efficient
  + Faster execution
  - More code to write

========================================
Summary:
Input: 5
Factorial: 120
Method: Recursion
Recursion depth: 5 levels
========================================
```

Back to Table of Contents

xxxv) Call by Value vs Call by Reference

Question:
Write a C program to demonstrate call by reference and call by value.

Description:
This program demonstrates the difference between two parameter passing mechanisms in C:

1. CALL BY VALUE:

- A copy of the actual parameter is passed to the function
- Changes made to the parameter inside the function do NOT affect the original variable
- Default method in C

2. CALL BY REFERENCE:

- The address (pointer) of the actual parameter is passed
- Changes made to the parameter inside the function DO affect the original variable
- Achieved using pointers in C

Examples demonstrated:

- Swapping two numbers
- Modifying a value
- Incrementing a value

How to Solve:

1. Create function using call by value (pass variables directly)
2. Create function using call by reference (pass addresses using &)
3. Demonstrate each method with examples
4. Show how original values are/aren't affected
5. Compare both methods with clear output

Code:

```
#include <stdio.h>

// ========== CALL BY VALUE FUNCTIONS ==========

// Function to swap two numbers using call by value
void swapByValue(int a, int b) {
   printf("\n  Inside swapByValue():\n");
   printf("    Before swap: a = %d, b = %d\n", a, b);

   int temp = a;
   a = b;
   b = temp;

   printf("    After swap:  a = %d, b = %d\n", a, b);
   printf("    (Changes are LOCAL to this function)\n");
}

// Function to increment a number using call by value
void incrementByValue(int num) {
   printf("\n  Inside incrementByValue():\n");
   printf("    Before: num = %d\n", num);
   num++;
   printf("    After:  num = %d (incremented)\n", num);
   printf("    (Change is LOCAL to this function)\n");
}

// ========== CALL BY REFERENCE FUNCTIONS ==========

// Function to swap two numbers using call by reference
void swapByReference(int *a, int *b) {
   printf("\n  Inside swapByReference():\n");
   printf("    Before swap: *a = %d, *b = %d\n", *a, *b);

   int temp = *a;
   *a = *b;
   *b = temp;

   printf("    After swap:  *a = %d, *b = %d\n", *a, *b);
   printf("    (Changes affect ORIGINAL variables)\n");
}

// Function to increment a number using call by reference
void incrementByReference(int *num) {
   printf("\n  Inside incrementByReference():\n");
   printf("    Before: *num = %d\n", *num);
   (*num)++;
   printf("    After:  *num = %d (incremented)\n", *num);
   printf("    (Change affects ORIGINAL variable)\n");
}

int main() {
   int num1, num2, value;

   printf("========================================\n");
   printf("  Call by Value vs Call by Reference\n");
   printf("========================================\n\n");

   printf("This program demonstrates the difference between:\n");
   printf("1. Call by Value - passes copy of variable\n");
   printf("2. Call by Reference - passes address of variable\n\n");
```

```
    // Input
    printf("Enter two numbers:\n");
    printf("Number 1: ");
    scanf("%d", &num1);
    printf("Number 2: ");
    scanf("%d", &num2);

    // Call by Value
    printf("\n--- Using CALL BY VALUE ---\n");
    printf("Before calling swapByValue(): num1 = %d, num2 = %d\n", num1, num2);
    swapByValue(num1, num2);
    printf("After calling swapByValue():  num1 = %d, num2 = %d\n", num1, num2);
    printf("✗ Original values UNCHANGED!\n");

    // Call by Reference
    printf("\n--- Using CALL BY REFERENCE ---\n");
    printf("Before calling swapByReference(): num1 = %d, num2 = %d\n", num1, num2);
    swapByReference(&num1, &num2);
    printf("After calling swapByReference():  num1 = %d, num2 = %d\n", num1, num2);
    printf("✓ Original values CHANGED!\n");

    printf("\n========================================\n");
    printf("  Comparison Summary\n");
    printf("========================================\n\n");

    printf("┌─────────────────────┬──────────────────┬──────────────────┐\n");
    printf("│ Feature             │ Call by Value    │ Call by Reference│\n");
    printf("├─────────────────────┼──────────────────┼──────────────────┤\n");
    printf("│ What is passed?     │ Copy of value    │ Address (pointer)│\n");
    printf("│ Function syntax     │ func(int x)      │ func(int *x)     │\n");
    printf("│ Function call       │ func(var)        │ func(&var)       │\n");
    printf("│ Access in function  │ x                │ *x               │\n");
    printf("│ Original modified?  │ NO ✗             │ YES ✓            │\n");
    printf("│ Memory used         │ More (copy)      │ Less (address)   │\n");
    printf("│ Use case            │ Read-only ops    │ Modify original  │\n");
    printf("└─────────────────────┴──────────────────┴──────────────────┘\n");

    printf("\n========================================\n");

    return 0;
}
```

Example:

```
Input:
  num1 = 10, num2 = 20

Output:
========================================
  Call by Value vs Call by Reference
========================================

This program demonstrates the difference between:
1. Call by Value - passes copy of variable
2. Call by Reference - passes address of variable

Enter two numbers:
Number 1: 10
Number 2: 20

--- Using CALL BY VALUE ---
Before calling swapByValue(): num1 = 10, num2 = 20

  Inside swapByValue():
    Before swap: a = 10, b = 20
    After swap:  a = 20, b = 10
    (Changes are LOCAL to this function)
After calling swapByValue():  num1 = 10, num2 = 20
✗ Original values UNCHANGED!

--- Using CALL BY REFERENCE ---
Before calling swapByReference(): num1 = 10, num2 = 20

  Inside swapByReference():
    Before swap: *a = 10, *b = 20
    After swap:  *a = 20, *b = 10
    (Changes affect ORIGINAL variables)
After calling swapByReference():  num1 = 20, num2 = 10
✓ Original values CHANGED!

========================================
  Comparison Summary
========================================

┌─────────────────────┬──────────────────┬──────────────────┐
│ Feature             │ Call by Value    │ Call by Reference│
├─────────────────────┼──────────────────┼──────────────────┤
│ What is passed?     │ Copy of value    │ Address (pointer)│
│ Function syntax     │ func(int x)      │ func(int *x)     │
│ Function call       │ func(var)        │ func(&var)       │
│ Access in function  │ x                │ *x               │
│ Original modified?  │ NO ✗             │ YES ✓            │
│ Memory used         │ More (copy)      │ Less (address)   │
│ Use case            │ Read-only ops    │ Modify original  │
```

```
 └──────────────────────┴──────────────────────┴──────────────────────┘

========================================
```

Back to Table of Contents

xxxvi) Array Using Pointer

Question:
Write a C program to read and display an integer array using pointer.

Description:
This program demonstrates how to use pointers to manipulate arrays. In C, arrays and pointers are closely related:

- Array name represents the base address (pointer to first element)
- `arr[i]` is equivalent to `*(arr + i)`
- `&arr[i]` is equivalent to `(arr + i)`

The program shows:

1. Reading array elements using pointers
2. Displaying array elements using pointers
3. Pointer arithmetic (ptr++, ptr + i)
4. Dereferencing pointers (*ptr)
5. Address calculations

How to Solve:

1. Declare an integer array
2. Declare a pointer to integer
3. Point the pointer to the array (ptr = arr)
4. Use pointer arithmetic to traverse array
5. Read elements: `*(ptr + i)` or `*ptr` (with ptr++)
6. Display elements using pointer notation
7. Show addresses and values at each position
8. Demonstrate different pointer access methods

Code:

```c
#include <stdio.h>

// Function to read array using pointer
void readArray(int *ptr, int size) {
   printf("Enter %d elements:\n", size);
   for (int i = 0; i < size; i++) {
      printf("Element [%d]: ", i);
      scanf("%d", ptr + i); // Same as scanf("%d", &ptr[i])
   }
}

// Function to display array using pointer (Method 1: pointer + index)
void displayArrayMethod1(int *ptr, int size) {
   printf("Method 1 - Using *(ptr + i):\n");
   for (int i = 0; i < size; i++) {
      printf("  arr[%d] = %d (at address %p)\n", i, *(ptr + i), (void *)(ptr + i));
   }
}

// Function to display array using pointer (Method 2: pointer increment)
void displayArrayMethod2(int *ptr, int size) {
   printf("Method 2 - Using *ptr and ptr++:\n");
   int *temp = ptr; // Save original pointer
   for (int i = 0; i < size; i++) {
      printf("  arr[%d] = %d (at address %p)\n", i, *temp, (void *)temp);
      temp++; // Move to next element
   }
}

// Function to reverse array using pointers
void reverseArray(int *ptr, int size) {
   int *start = ptr;
   int *end = ptr + size - 1;

   while (start < end) {
      // Swap elements
      int temp = *start;
      *start = *end;
      *end = temp;

      start++;
      end--;
   }
}

int main() {
   int size;
   int arr[100];
   int *ptr;

   printf("========================================\n");
   printf("  Integer Array Using Pointers\n");
   printf("========================================\n\n");
```

```
    printf("This program demonstrates array manipulation\n");
    printf("using pointers in various ways.\n\n");

    // Input array size
    printf("Enter the size of array (max 100): ");
    scanf("%d", &size);

    if (size <= 0 || size > 100) {
        printf("Error: Size must be between 1 and 100\n");
        return 1;
    }

    // Point to array
    ptr = arr; // arr is same as &arr[0]

    printf("\n========================================\n");
    printf("  Reading Array Using Pointer\n");
    printf("========================================\n\n");

    readArray(ptr, size);

    printf("\n========================================\n");
    printf("  Pointer Concepts\n");
    printf("========================================\n\n");

    printf("Array name:      arr\n");
    printf("Array address:   %p\n", (void *)arr);
    printf("Pointer value:   %p (points to arr[0])\n", (void *)ptr);
    printf("First element:   *ptr = %d\n", *ptr);
    printf("Second element:  *(ptr+1) = %d\n", *(ptr + 1));
    printf("\nRelationship:\n");
    printf("  arr[i]  ≡  *(arr + i)  ≡  *(ptr + i)  ≡  ptr[i]\n");
    printf("  &arr[i] ≡  (arr + i)   ≡  (ptr + i)\n");

    printf("\n========================================\n");
    printf("  Display Methods Using Pointers\n");
    printf("========================================\n\n");

    // Method 1
    displayArrayMethod1(ptr, size);
    printf("\n");

    // Method 2
    displayArrayMethod2(ptr, size);

    printf("\n========================================\n");
    printf("  Reversing Array Using Pointers\n");
    printf("========================================\n\n");

    printf("Before reverse: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", *(ptr + i));
    }
    printf("\n");

    reverseArray(ptr, size);

    printf("After reverse:  ");
    for (int i = 0; i < size; i++) {
        printf("%d ", *(ptr + i));
    }
    printf("\n");

    printf("\n========================================\n");
    printf("  Summary\n");
    printf("========================================\n");
    printf("Array size: %d elements\n", size);
    printf("Total memory: %lu bytes\n", size * sizeof(int));
    printf("Pointer used: Successfully!\n");
    printf("========================================\n");

    return 0;
}
```

Example:

```
Input:
  Size: 5
  Elements: 10, 20, 30, 40, 50

Output:
========================================
  Integer Array Using Pointers
========================================

This program demonstrates array manipulation
using pointers in various ways.

Enter the size of array (max 100): 5

========================================
  Reading Array Using Pointer
========================================
Enter 5 elements:
Element [0]: 10
Element [1]: 20
Element [2]: 30
Element [3]: 40
Element [4]: 50

========================================
  Pointer Concepts
========================================

Array name:      arr
Array address:   0x7ffeeb3a8c60
Pointer value:   0x7ffeeb3a8c60 (points to arr[0])
First element:   *ptr = 10
Second element:  *(ptr+1) = 20

Relationship:
  arr[i]  ≡  *(arr + i)  ≡  *(ptr + i)  ≡  ptr[i]
  &arr[i] ≡  (arr + i)   ≡  (ptr + i)

========================================
  Display Methods Using Pointers
========================================

Method 1 - Using *(ptr + i):
  arr[0] = 10 (at address 0x7ffeeb3a8c60)
  arr[1] = 20 (at address 0x7ffeeb3a8c64)
  arr[2] = 30 (at address 0x7ffeeb3a8c68)
  arr[3] = 40 (at address 0x7ffeeb3a8c6c)
  arr[4] = 50 (at address 0x7ffeeb3a8c70)

Method 2 - Using *ptr and ptr++:
  arr[0] = 10 (at address 0x7ffeeb3a8c60)
  arr[1] = 20 (at address 0x7ffeeb3a8c64)
  arr[2] = 30 (at address 0x7ffeeb3a8c68)
  arr[3] = 40 (at address 0x7ffeeb3a8c6c)
  arr[4] = 50 (at address 0x7ffeeb3a8c70)

========================================
  Reversing Array Using Pointers
========================================

Before reverse: 10 20 30 40 50
After reverse:  50 40 30 20 10

========================================
  Summary
========================================
Array size: 5 elements
Total memory: 20 bytes
Pointer used: Successfully!
========================================
```

Back to Table of Contents

xxxvii) Text Analysis Using Pointer

Question:
Write a C program to read and display a text using a character pointer to a string. Also count the number of characters, words and lines in the text.

Description:
This program demonstrates string manipulation using character pointers. It reads multi-line text and analyzes it to count:

- Characters (total, alphabetic, digits, special)
- Words (sequences separated by spaces/newlines)
- Lines (text lines ending with newline)

Character pointer concepts:

- `char *ptr` points to a string (array of characters)
- String is terminated by null character '\0'
- Can traverse string using pointer arithmetic
- `*(ptr + i)` or `ptr[i]` accesses character at position i

How to Solve:

1. Declare character array to store text
2. Declare character pointer
3. Read multi-line text (until special marker like "END")
4. Point character pointer to the text
5. Traverse text using pointer:
   - Count total characters
   - Count alphabets, digits, spaces, special chars
   - Count words (track transitions from space to non-space)
   - Count lines (count newline characters)
6. Display text and all statistics

Code:

```c
#include <ctype.h>
#include <stdio.h>
#include <string.h>

// Function to count characters using pointer
void countCharacters(char *ptr, int *total, int *alphabets, int *digits,
                     int *spaces, int *special) {
   *total = 0;
   *alphabets = 0;
   *digits = 0;
   *spaces = 0;
   *special = 0;

   while (*ptr != '\0') {
      (*total)++;

      if (isalpha(*ptr)) {
         (*alphabets)++;
      } else if (isdigit(*ptr)) {
         (*digits)++;
      } else if (*ptr == ' ' || *ptr == '\t') {
         (*spaces)++;
      } else if (*ptr != '\n') {
         (*special)++;
      }

      ptr++;
   }
}

// Function to count words using pointer
int countWords(char *ptr) {
   int words = 0;
   int inWord = 0; // Flag to track if we're inside a word

   while (*ptr != '\0') {
      if (*ptr == ' ' || *ptr == '\t' || *ptr == '\n') {
         inWord = 0; // We're in whitespace
      } else if (inWord == 0) {
         inWord = 1; // We just entered a new word
         words++;
      }
      ptr++;
   }

   return words;
}

// Function to count lines using pointer
int countLines(char *ptr) {
   int lines = 0;

   while (*ptr != '\0') {
      if (*ptr == '\n') {
         lines++;
      }
      ptr++;
   }

   return lines;
}

// Function to display text with line numbers using pointer
void displayTextWithLineNumbers(char *ptr) {
   int lineNum = 1;
   printf("%3d: ", lineNum);

   while (*ptr != '\0') {
      printf("%c", *ptr);
      if (*ptr == '\n') {
         ptr++;
         if (*ptr != '\0') { // Not at end
            lineNum++;
            printf("%3d: ", lineNum);
         }
      } else {
         ptr++;
      }
   }
   if (*(ptr - 1) != '\n') {
      printf("\n"); // Add newline if text doesn't end with one
   }
}

int main() {
   char text[1000] = "";
   char line[200];
   char *ptr;
   int total, alphabets, digits, spaces, special;
   int words, lines;

   printf("========================================\n");
   printf("  Text Analysis Using Character Pointer\n");
   printf("========================================\n\n");
```

```
    printf("Enter text (multiple lines allowed):\n");
    printf("Type 'END' on a new line to finish.\n\n");

    // Read multi-line text
    while (1) {
        fgets(line, sizeof(line), stdin);

        // Check if user typed "END"
        if (strcmp(line, "END\n") == 0 || strcmp(line, "END") == 0) {
            break;
        }

        // Append line to text
        strcat(text, line);

        // Safety check
        if (strlen(text) > 900) {
            printf("Text too long! Stopping input.\n");
            break;
        }
    }

    // Point to the text
    ptr = text;

    printf("\n========================================\n");
    printf("  Text Display with Line Numbers\n");
    printf("========================================\n\n");

    displayTextWithLineNumbers(ptr);

    printf("\n========================================\n");
    printf("  Character Analysis\n");
    printf("========================================\n\n");

    // Count characters
    countCharacters(ptr, &total, &alphabets, &digits, &spaces, &special);

    printf("Character Breakdown:\n");
    printf("  Total characters:     %d\n", total);
    printf("  Alphabetic chars:     %d\n", alphabets);
    printf("  Digits:               %d\n", digits);
    printf("  Spaces/Tabs:          %d\n", spaces);
    printf("  Special characters:   %d\n", special);
    printf("  Newlines:             %d\n", total - alphabets - digits - spaces - special);

    printf("\n========================================\n");
    printf("  Word and Line Analysis\n");
    printf("========================================\n\n");

    // Count words and lines
    words = countWords(ptr);
    lines = countLines(ptr);

    printf("Word Analysis:\n");
    printf("  Total words:          %d\n", words);
    printf("  Average word length:  %.2f characters\n",
           words > 0 ? (float)(alphabets + digits) / words : 0);

    printf("\nLine Analysis:\n");
    printf("  Total lines:          %d\n", lines);
    printf("  Average line length:  %.2f characters\n",
           lines > 0 ? (float)total / lines : 0);
    printf("  Average words/line:   %.2f words\n",
           lines > 0 ? (float)words / lines : 0);

    printf("\n========================================\n");
    printf("  Summary\n");
    printf("========================================\n");
    printf("Total characters: %d\n", total);
    printf("Total words:      %d\n", words);
    printf("Total lines:      %d\n", lines);
    printf("Memory used:      %lu bytes\n", strlen(text) + 1);
    printf("========================================\n");

    return 0;
}
```

Example:

```
Input:
  Hello World
  This is a test.
  Programming in C!
  END

Output:
========================================
  Text Analysis Using Character Pointer
========================================

Enter text (multiple lines allowed):
Type 'END' on a new line to finish.
```

```
Hello World
This is a test.
Programming in C!
END

========================================
  Text Display with Line Numbers
========================================

  1: Hello World
  2: This is a test.
  3: Programming in C!

========================================
  Character Analysis
========================================

Character Breakdown:
  Total characters:     44
  Alphabetic chars:     34
  Digits:               0
  Spaces/Tabs:          6
  Special characters:   2
  Newlines:             2

========================================
  Word and Line Analysis
========================================

Word Analysis:
  Total words:          7
  Average word length:  4.86 characters

Line Analysis:
  Total lines:          3
  Average line length:  14.67 characters
  Average words/line:   2.33 words

========================================
  Summary
========================================
Total characters: 44
Total words:      7
Total lines:      3
Memory used:      45 bytes
========================================
```

Back to Table of Contents

xxxviii) Time Operations

Question:
Write a C program to read, display, add and subtract two times defined using hour, minutes and values of seconds.

Description:
This program uses structures to represent time (hours:minutes:seconds) and performs arithmetic operations on time values.

Time structure contains:

- hours (0-23 for 24-hour format)
- minutes (0-59)
- seconds (0-59)

Operations:

1. Read time from user with validation
2. Display time in different formats (12-hour and 24-hour)
3. Add two times (handling carry-over)
4. Subtract two times (handling borrow)

How to Solve:

1. Define a structure for time with hour, minute, second
2. Create functions to:
    - Read and validate time input
    - Display time in various formats
    - Add two times (handle seconds→minutes→hours overflow)
    - Subtract two times (handle borrow operations)
    - Normalize time (ensure valid ranges)
3. Handle edge cases (midnight, noon, negative results)

Code:

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

// Structure to represent time
typedef struct {
    int hours;
    int minutes;
    int seconds;
} Time;
```

```c
// Function to read time from user
Time readTime(const char *label) {
   Time t;

   printf("%s\n", label);

   do {
      printf("  Enter hours (0-23): ");
      scanf("%d", &t.hours);
      if (t.hours < 0 || t.hours > 23) {
         printf("  Invalid! Hours must be 0-23.\n");
      }
   } while (t.hours < 0 || t.hours > 23);

   do {
      printf("  Enter minutes (0-59): ");
      scanf("%d", &t.minutes);
      if (t.minutes < 0 || t.minutes > 59) {
         printf("  Invalid! Minutes must be 0-59.\n");
      }
   } while (t.minutes < 0 || t.minutes > 59);

   do {
      printf("  Enter seconds (0-59): ");
      scanf("%d", &t.seconds);
      if (t.seconds < 0 || t.seconds > 59) {
         printf("  Invalid! Seconds must be 0-59.\n");
      }
   } while (t.seconds < 0 || t.seconds > 59);

   return t;
}

// Function to display time in 24-hour format
void displayTime24(Time t) {
   printf("%02d:%02d:%02d", t.hours, t.minutes, t.seconds);
}

// Function to display time in 12-hour format
void displayTime12(Time t) {
   int hour12 = t.hours % 12;
   if (hour12 == 0)
      hour12 = 12;

   printf("%02d:%02d:%02d %s", hour12, t.minutes, t.seconds,
          t.hours < 12 ? "AM" : "PM");
}

// Function to normalize time (handle overflows and underflows)
Time normalizeTime(Time t) {
   // Handle seconds overflow/underflow
   while (t.seconds < 0) {
      t.seconds += 60;
      t.minutes--;
   }
   while (t.seconds >= 60) {
      t.seconds -= 60;
      t.minutes++;
   }

   // Handle minutes overflow/underflow
   while (t.minutes < 0) {
      t.minutes += 60;
      t.hours--;
   }
   while (t.minutes >= 60) {
      t.minutes -= 60;
      t.hours++;
   }

   // Handle hours overflow/underflow
   while (t.hours < 0) {
      t.hours += 24;
   }
   while (t.hours >= 24) {
      t.hours -= 24;
   }

   return t;
}

// Function to add two times
Time addTime(Time t1, Time t2) {
   Time result;

   result.seconds = t1.seconds + t2.seconds;
   result.minutes = t1.minutes + t2.minutes;
   result.hours = t1.hours + t2.hours;

   // Normalize the result
   result = normalizeTime(result);

   return result;
}

// Function to subtract two times (t1 - t2)
```

```c
Time subtractTime(Time t1, Time t2) {
   Time result;

   result.seconds = t1.seconds - t2.seconds;
   result.minutes = t1.minutes - t2.minutes;
   result.hours = t1.hours - t2.hours;

   // Normalize the result
   result = normalizeTime(result);

   return result;
}

int main() {
   Time time1, time2, sum, difference;

   printf("========================================\n");
   printf("  Time Addition and Subtraction\n");
   printf("========================================\n\n");

   printf("This program performs arithmetic operations\n");
   printf("on time values (HH:MM:SS format)\n\n");

   // Read times
   time1 = readTime("Enter First Time:");
   printf("\n");
   time2 = readTime("Enter Second Time:");

   printf("\n========================================\n");
   printf("  Input Times\n");
   printf("========================================\n\n");

   printf("Time 1 (24-hour): ");
   displayTime24(time1);
   printf("\n");
   printf("Time 1 (12-hour): ");
   displayTime12(time1);
   printf("\n\n");

   printf("Time 2 (24-hour): ");
   displayTime24(time2);
   printf("\n");
   printf("Time 2 (12-hour): ");
   displayTime12(time2);
   printf("\n");

   // Addition
   printf("\n========================================\n");
   printf("  ADDITION: Time1 + Time2\n");
   printf("========================================\n\n");

   sum = addTime(time1, time2);

   printf("  Time 1:     ");
   displayTime24(time1);
   printf("\n");
   printf("  Time 2:   + ");
   displayTime24(time2);
   printf("\n");
   printf("  ----------\n");
   printf("  Result:     ");
   displayTime24(sum);
   printf("\n\n");

   printf("Result in different formats:\n");
   printf("  24-hour: ");
   displayTime24(sum);
   printf("\n");
   printf("  12-hour: ");
   displayTime12(sum);
   printf("\n");

   // Subtraction
   printf("\n========================================\n");
   printf("  SUBTRACTION: Time1 - Time2\n");
   printf("========================================\n\n");

   difference = subtractTime(time1, time2);

   printf("  Time 1:     ");
   displayTime24(time1);
   printf("\n");
   printf("  Time 2:   - ");
   displayTime24(time2);
   printf("\n");
   printf("  ----------\n");
   printf("  Result:     ");
   displayTime24(difference);
   printf("\n\n");

   printf("Result in different formats:\n");
   printf("  24-hour: ");
   displayTime24(difference);
   printf("\n");
   printf("  12-hour: ");
   displayTime12(difference);
   printf("\n");
```

```
    printf("\n========================================\n");
    printf("  Summary\n");
    printf("========================================\n");
    printf("Time 1:       ");
    displayTime24(time1);
    printf("\n");
    printf("Time 2:       ");
    displayTime24(time2);
    printf("\n");
    printf("Sum:          ");
    displayTime24(sum);
    printf("\n");
    printf("Difference:   ");
    displayTime24(difference);
    printf("\n");
    printf("========================================\n");

    return 0;
}
```

Example:

```
Input:
  Time 1: 10:30:45
  Time 2: 05:25:30

Output:
========================================
  Time Addition and Subtraction
========================================

This program performs arithmetic operations
on time values (HH:MM:SS format)

Enter First Time:
  Enter hours (0-23): 10
  Enter minutes (0-59): 30
  Enter seconds (0-59): 45

Enter Second Time:
  Enter hours (0-23): 5
  Enter minutes (0-59): 25
  Enter seconds (0-59): 30

========================================
  Input Times
========================================

Time 1 (24-hour): 10:30:45
Time 1 (12-hour): 10:30:45 AM

Time 2 (24-hour): 05:25:30
Time 2 (12-hour): 05:25:30 AM

========================================
  ADDITION: Time1 + Time2
========================================

  Time 1:     10:30:45
  Time 2:   + 05:25:30
  ----------
  Result:     15:56:15

Result in different formats:
  24-hour: 15:56:15
  12-hour: 03:56:15 PM

========================================
  SUBTRACTION: Time1 - Time2
========================================

  Time 1:     10:30:45
  Time 2:   - 05:25:30
  ----------
  Result:     05:05:15

Result in different formats:
  24-hour: 05:05:15
  12-hour: 05:05:15 AM

========================================
  Summary
========================================
Time 1:       10:30:45
Time 2:       05:25:30
Sum:          15:56:15
Difference:   05:05:15
========================================
```

Back to Table of Contents

xxxix) Structure Pointer

Question:
Write a C program to read and display the contents of a structure variable using pointer to a structure.

Description:
This program demonstrates the use of structure pointers in C. It shows how to:

- Access structure members using pointer (-> operator)
- Pass structures to functions using pointers
- Dynamically allocate memory for structures
- Manipulate structure data through pointers

Structure pointer concepts:

- `ptr->member` is equivalent to `(*ptr).member`
- Arrow operator (->) is used to access members via pointer
- More efficient to pass structure pointer than entire structure

The program creates a Student structure with:

- Roll number
- Name
- Marks in multiple subjects
- Grade

How to Solve:

1. Define a structure (e.g., Student)
2. Declare structure variable and pointer to structure
3. Point the pointer to the structure variable
4. Use -> operator to access and modify members
5. Create functions that accept structure pointers
6. Demonstrate both . and -> operators
7. Show address and memory layout

Code:

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

// Structure to represent a student
typedef struct {
   int rollNumber;
   char name[50];
   int age;
   float marks[5]; // Marks in 5 subjects
   float total;
   float average;
   char grade;
} Student;

// Function to read student data using pointer
void readStudent(Student *ptr) {
   printf("Enter Student Details:\n");

   printf("  Roll Number: ");
   scanf("%d", &ptr->rollNumber);

   printf("  Name: ");
   getchar(); // Clear newline
   fgets(ptr->name, sizeof(ptr->name), stdin);
   ptr->name[strcspn(ptr->name, "\n")] = '\0'; // Remove newline

   printf("  Age: ");
   scanf("%d", &ptr->age);

   printf("  Enter marks in 5 subjects:\n");
   ptr->total = 0;
   for (int i = 0; i < 5; i++) {
      printf("    Subject %d: ", i + 1);
      scanf("%f", &ptr->marks[i]);
      ptr->total += ptr->marks[i];
   }

   // Calculate average
   ptr->average = ptr->total / 5.0;

   // Assign grade based on average
   if (ptr->average >= 90) {
      ptr->grade = 'A';
   } else if (ptr->average >= 80) {
      ptr->grade = 'B';
   } else if (ptr->average >= 70) {
      ptr->grade = 'C';
   } else if (ptr->average >= 60) {
      ptr->grade = 'D';
   } else if (ptr->average >= 50) {
      ptr->grade = 'E';
   } else {
      ptr->grade = 'F';
   }
}

// Function to display student data using pointer
```

```
void displayStudent(Student *ptr) {
    printf("\n========================================\n");
    printf("  Student Information\n");
    printf("========================================\n\n");

    printf("Roll Number:  %d\n", ptr->rollNumber);
    printf("Name:         %s\n", ptr->name);
    printf("Age:          %d years\n", ptr->age);

    printf("\nMarks:\n");
    for (int i = 0; i < 5; i++) {
        printf("  Subject %d:  %.2f\n", i + 1, ptr->marks[i]);
    }

    printf("\nResults:\n");
    printf("  Total:      %.2f / 500.00\n", ptr->total);
    printf("  Average:    %.2f%%\n", ptr->average);
    printf("  Grade:      %c\n", ptr->grade);

    // Performance assessment
    printf("\nPerformance: ");
    if (ptr->average >= 90) {
        printf("Excellent! Outstanding performance!\n");
    } else if (ptr->average >= 80) {
        printf("Very Good! Keep up the good work!\n");
    } else if (ptr->average >= 70) {
        printf("Good. Room for improvement.\n");
    } else if (ptr->average >= 60) {
        printf("Satisfactory. Need more effort.\n");
    } else {
        printf("Needs improvement.\n");
    }
}

int main() {
    Student student1;
    Student *ptr;

    printf("========================================\n");
    printf("  Structure Pointer Demonstration\n");
    printf("========================================\n\n");

    printf("This program demonstrates accessing\n");
    printf("structure members using pointers.\n\n");

    // Point to the structure
    ptr = &student1;

    printf("Structure variable address: %p\n", (void *)&student1);
    printf("Pointer value (points to):  %p\n", (void *)ptr);
    printf("They are the same! ✓\n\n");

    // Read student data using pointer
    readStudent(ptr);

    // Display student data using pointer
    displayStudent(ptr);

    printf("\n========================================\n");
    printf("  Key Concepts Summary\n");
    printf("========================================\n\n");

    printf("Structure Pointer Syntax:\n");
    printf("  Declaration:   Student *ptr;\n");
    printf("  Assignment:    ptr = &student1;\n");
    printf("  Access member: ptr->member\n");
    printf("  Equivalent to: (*ptr).member\n\n");

    printf("Advantages of Structure Pointers:\n");
    printf("  ✓ Efficient - passes address, not entire structure\n");
    printf("  ✓ Can modify original structure in functions\n");
    printf("  ✓ Enables dynamic memory allocation\n");
    printf("  ✓ Useful for linked data structures\n");

    printf("\n========================================\n");

    return 0;
}
```

Example:

```
Input:
  Roll: 101
  Name: John Doe
  Age: 20
  Marks: 85, 90, 78, 92, 88

Output:
========================================
  Structure Pointer Demonstration
========================================

This program demonstrates accessing
structure members using pointers.
```

```
Structure variable address: 0x7ffeeb3a8c00
Pointer value (points to):  0x7ffeeb3a8c00
They are the same! ✓

Enter Student Details:
  Roll Number: 101
  Name: John Doe
  Age: 20
  Enter marks in 5 subjects:
    Subject 1: 85
    Subject 2: 90
    Subject 3: 78
    Subject 4: 92
    Subject 5: 88

========================================
  Student Information
========================================

Roll Number:  101
Name:         John Doe
Age:          20 years

Marks:
  Subject 1:  85.00
  Subject 2:  90.00
  Subject 3:  78.00
  Subject 4:  92.00
  Subject 5:  88.00

Results:
  Total:      433.00 / 500.00
  Average:    86.60%
  Grade:      B

Performance: Very Good! Keep up the good work!

========================================
  Key Concepts Summary
========================================

Structure Pointer Syntax:
  Declaration:   Student *ptr;
  Assignment:    ptr = &student1;
  Access member: ptr->member
  Equivalent to: (*ptr).member

Advantages of Structure Pointers:
  ✓ Efficient - passes address, not entire structure
  ✓ Can modify original structure in functions
  ✓ Enables dynamic memory allocation
  ✓ Useful for linked data structures

========================================
```

Back to Table of Contents

Section 6: File Handling

xl) File Handling Modes

Question: Write a C program for handling unformatted and formatted files in different operational modes.

Description: This program demonstrates various file handling modes in C:

File Opening Modes:

- `"r"` - Read mode (file must exist)
- `"w"` - Write mode (creates new, overwrites existing)
- `"a"` - Append mode (creates new, appends to existing)
- `"r+"` - Read/Write mode (file must exist)
- `"w+"` - Read/Write mode (creates new, overwrites existing)
- `"a+"` - Read/Append mode (creates new, appends to existing)

File I/O Functions:

FORMATTED:

- `fprintf()` - Write formatted data to file
- `fscanf()` - Read formatted data from file

UNFORMATTED (Character-based):

- `fputc()` - Write a character
- `fgetc()` - Read a character

UNFORMATTED (String-based):

- `fputs()` - Write a string
- `fgets()` - Read a string

UNFORMATTED (Block-based):

- `fwrite()` - Write binary data

- fread() - Read binary data

How to Solve:

1. Demonstrate formatted file operations (fprintf, fscanf)
2. Demonstrate unformatted file operations (fputc, fgetc, fputs, fgets)
3. Show different file opening modes
4. Handle file operations with error checking
5. Display file contents after operations

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to demonstrate formatted file writing
void demonstrateFormattedWrite() {
   FILE *fp;

   printf("\n========================================\n");
   printf("  FORMATTED FILE WRITE (fprintf)\n");
   printf("========================================\n\n");

   fp = fopen("formatted_data.txt", "w");
   if (fp == NULL) {
      printf("Error: Could not create file!\n");
      return;
   }

   printf("Writing formatted data to 'formatted_data.txt'...\n\n");

   // Write formatted data
   fprintf(fp, "Student Records\n");
   fprintf(fp, "%-10s %-20s %-5s %-8s\n", "Roll", "Name", "Age", "Marks");
   fprintf(fp, "%-10s %-20s %-5s %-8s\n", "----", "----", "---", "-----");
   fprintf(fp, "%-10d %-20s %-5d %-8.2f\n", 101, "Alice Johnson", 20, 85.5);
   fprintf(fp, "%-10d %-20s %-5d %-8.2f\n", 102, "Bob Smith", 21, 92.3);
   fprintf(fp, "%-10d %-20s %-5d %-8.2f\n", 103, "Carol Williams", 19, 78.9);

   fclose(fp);

   printf("Data written successfully!\n");
   printf("Format: fprintf(fp, format_string, variables)\n");
}

// Function to demonstrate formatted file reading
void demonstrateFormattedRead() {
   FILE *fp;
   char line[100];

   printf("\n========================================\n");
   printf("  FORMATTED FILE READ (fscanf)\n");
   printf("========================================\n\n");

   fp = fopen("formatted_data.txt", "r");
   if (fp == NULL) {
      printf("Error: Could not open file!\n");
      return;
   }

   printf("Reading from 'formatted_data.txt':\n\n");

   // Read and display entire file
   while (fgets(line, sizeof(line), fp) != NULL) {
      printf("%s", line);
   }

   fclose(fp);
   printf("\nFile read successfully!\n");
}

// Function to demonstrate unformatted character I/O
void demonstrateUnformattedChar() {
   FILE *fp;
   char ch;
   char text[] = "Hello, File Handling in C!";

   printf("\n========================================\n");
   printf("  UNFORMATTED CHAR I/O (fputc/fgetc)\n");
   printf("========================================\n\n");

   // Writing character by character
   fp = fopen("char_data.txt", "w");
   if (fp == NULL) {
      printf("Error: Could not create file!\n");
      return;
   }

   printf("Writing characters to 'char_data.txt'...\n");
   printf("Text: \"%s\"\n\n", text);

   for (int i = 0; text[i] != '\0'; i++) {
      fputc(text[i], fp);
   }
```

```
    fputc('\n', fp);

    fclose(fp);
    printf("Characters written using fputc()!\n");

    // Reading character by character
    fp = fopen("char_data.txt", "r");
    if (fp == NULL) {
        printf("Error: Could not open file!\n");
        return;
    }

    printf("\nReading characters using fgetc():\n");
    printf("Content: \"");

    while ((ch = fgetc(fp)) != EOF) {
        if (ch != '\n') {
            printf("%c", ch);
        }
    }
    printf("\"\n");

    fclose(fp);
}

// Function to demonstrate unformatted string I/O
void demonstrateUnformattedString() {
    FILE *fp;
    char line[100];

    printf("\n========================================\n");
    printf("  UNFORMATTED STRING I/O (fputs/fgets)\n");
    printf("========================================\n\n");

    // Writing strings
    fp = fopen("string_data.txt", "w");
    if (fp == NULL) {
        printf("Error: Could not create file!\n");
        return;
    }

    printf("Writing strings to 'string_data.txt'...\n\n");

    fputs("Line 1: This is the first line.\n", fp);
    fputs("Line 2: File handling is important.\n", fp);
    fputs("Line 3: C provides powerful file operations.\n", fp);

    fclose(fp);
    printf("Strings written using fputs()!\n");

    // Reading strings
    fp = fopen("string_data.txt", "r");
    if (fp == NULL) {
        printf("Error: Could not open file!\n");
        return;
    }

    printf("\nReading strings using fgets():\n\n");

    int lineNum = 1;
    while (fgets(line, sizeof(line), fp) != NULL) {
        printf("  [%d] %s", lineNum++, line);
    }

    fclose(fp);
}

// Function to demonstrate different file modes
void demonstrateFileModes() {
    FILE *fp;
    char buffer[100];

    printf("\n========================================\n");
    printf("  FILE OPENING MODES DEMONSTRATION\n");
    printf("========================================\n\n");

    // Mode: "w" - Write (creates/overwrites)
    printf("1. Mode 'w' (Write - creates/overwrites):\n");
    fp = fopen("mode_demo.txt", "w");
    fprintf(fp, "Initial content written in 'w' mode.\n");
    fclose(fp);
    printf("   File created and written.\n\n");

    // Mode: "a" - Append
    printf("2. Mode 'a' (Append):\n");
    fp = fopen("mode_demo.txt", "a");
    fprintf(fp, "This line appended in 'a' mode.\n");
    fclose(fp);
    printf("   Content appended to file.\n\n");

    // Mode: "r" - Read
    printf("3. Mode 'r' (Read):\n");
    fp = fopen("mode_demo.txt", "r");
    printf("   Content:\n");
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        printf("   %s", buffer);
    }
```

```c
    fclose(fp);
    printf("\n");

    // Mode: "r+" - Read/Write
    printf("4. Mode 'r+' (Read/Write - file must exist):\n");
    fp = fopen("mode_demo.txt", "r+");
    fprintf(fp, "Modified"); // Overwrites from beginning
    fclose(fp);
    printf("   File modified from beginning.\n\n");

    // Read modified content
    fp = fopen("mode_demo.txt", "r");
    printf("   Modified content:\n");
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        printf("   %s", buffer);
    }
    fclose(fp);
}

// Function to demonstrate binary file operations
void demonstrateBinaryFiles() {
    FILE *fp;
    int numbers[] = {10, 20, 30, 40, 50};
    int read_numbers[5];

    printf("\n========================================\n");
    printf("  BINARY FILE I/O (fwrite/fread)\n");
    printf("========================================\n\n");

    // Writing binary data
    fp = fopen("binary_data.bin", "wb");
    if (fp == NULL) {
        printf("Error: Could not create binary file!\n");
        return;
    }

    printf("Writing binary data to 'binary_data.bin'...\n");
    printf("Data: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n\n");

    fwrite(numbers, sizeof(int), 5, fp);
    fclose(fp);
    printf("Binary data written using fwrite()!\n");

    // Reading binary data
    fp = fopen("binary_data.bin", "rb");
    if (fp == NULL) {
        printf("Error: Could not open binary file!\n");
        return;
    }

    fread(read_numbers, sizeof(int), 5, fp);
    fclose(fp);

    printf("\nReading binary data using fread():\n");
    printf("Data: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", read_numbers[i]);
    }
    printf("\n\nData matches! Binary I/O successful!\n");
}

// Function to display file mode summary
void displayModeSummary() {
    printf("\n========================================\n");
    printf("  FILE MODE SUMMARY\n");
    printf("========================================\n\n");

    printf("┌──────┬─────────────────────────────────────────────────┐\n");
    printf("│ Mode │ Description                                     │\n");
    printf("├──────┼─────────────────────────────────────────────────┤\n");
    printf("│  r   │ Read only (file must exist)                     │\n");
    printf("│  w   │ Write only (creates/overwrites)                 │\n");
    printf("│  a   │ Append (creates/appends to end)                 │\n");
    printf("│  r+  │ Read & Write (file must exist)                  │\n");
    printf("│  w+  │ Read & Write (creates/overwrites)               │\n");
    printf("│  a+  │ Read & Append (creates/appends)                 │\n");
    printf("│  rb  │ Read binary                                     │\n");
    printf("│  wb  │ Write binary                                    │\n");
    printf("│  ab  │ Append binary                                   │\n");
    printf("└──────┴─────────────────────────────────────────────────┘\n");
}

int main() {
    printf("========================================\n");
    printf("  File Handling Demonstration\n");
    printf("========================================\n");
    printf("\nThis program demonstrates:\n");
    printf("  • Formatted file I/O\n");
    printf("  • Unformatted file I/O\n");
    printf("  • Different file opening modes\n");
    printf("  • Binary file operations\n");

    // Demonstrate formatted file operations
    demonstrateFormattedWrite();
    demonstrateFormattedRead();

    // Demonstrate unformatted file operations
    demonstrateUnformattedChar();
    demonstrateUnformattedString();

    // Demonstrate file modes
    demonstrateFileModes();

    // Demonstrate binary files
    demonstrateBinaryFiles();

    // Display summary
    displayModeSummary();

    printf("\n========================================\n");
    printf("  Function Summary\n");
    printf("========================================\n\n");

    printf("FORMATTED I/O:\n");
    printf("  fprintf(fp, format, ...) - Write formatted data\n");
    printf("  fscanf(fp, format, ...)  - Read formatted data\n\n");

    printf("UNFORMATTED I/O (Character):\n");
    printf("  fputc(char, fp)          - Write a character\n");
    printf("  fgetc(fp)                - Read a character\n\n");

    printf("UNFORMATTED I/O (String):\n");
    printf("  fputs(string, fp)        - Write a string\n");
    printf("  fgets(buffer, size, fp)  - Read a string\n\n");

    printf("UNFORMATTED I/O (Binary):\n");
    printf("  fwrite(ptr, size, n, fp) - Write binary data\n");
    printf("  fread(ptr, size, n, fp)  - Read binary data\n\n");

    printf("FILE OPERATIONS:\n");
    printf("  fopen(name, mode)        - Open file\n");
    printf("  fclose(fp)               - Close file\n");
    printf("  feof(fp)                 - Check end of file\n");
    printf("  rewind(fp)               - Reset file pointer\n");
    printf("  fseek(fp, offset, whence) - Move file pointer\n");

    printf("\n========================================\n");
    printf("Files created in this demonstration:\n");
    printf("  • formatted_data.txt\n");
    printf("  • char_data.txt\n");
    printf("  • string_data.txt\n");
    printf("  • mode_demo.txt\n");
    printf("  • binary_data.bin\n");
    printf("========================================\n");

    return 0;
}
```

Example Output:

```
========================================
  File Handling Demonstration
========================================

This program demonstrates:
  • Formatted file I/O
  • Unformatted file I/O
  • Different file opening modes
  • Binary file operations

========================================
  FORMATTED FILE WRITE (fprintf)
========================================

Writing formatted data to 'formatted_data.txt'...

Data written successfully!
Format: fprintf(fp, format_string, variables)

========================================
  FORMATTED FILE READ (fscanf)
========================================

Reading from 'formatted_data.txt':

Student Records
Roll       Name                 Age   Marks
----       ----                 ---   -----
101        Alice Johnson        20    85.50
102        Bob Smith            21    92.30
103        Carol Williams       19    78.90

File read successfully!

========================================
  UNFORMATTED CHAR I/O (fputc/fgetc)
========================================
Writing characters to 'char_data.txt'...
Text: "Hello, File Handling in C!"

Characters written using fputc()!

Reading characters using fgetc():
Content: "Hello, File Handling in C!"

========================================
  UNFORMATTED STRING I/O (fputs/fgets)
========================================

Writing strings to 'string_data.txt'...

Strings written using fputs()!

Reading strings using fgets():

  [1] Line 1: This is the first line.
  [2] Line 2: File handling is important.
  [3] Line 3: C provides powerful file operations.

========================================
  FILE OPENING MODES DEMONSTRATION
========================================

1. Mode 'w' (Write - creates/overwrites):
   File created and written.

2. Mode 'a' (Append):
   Content appended to file.

3. Mode 'r' (Read):
   Content:
   Initial content written in 'w' mode.
   This line appended in 'a' mode.

4. Mode 'r+' (Read/Write - file must exist):
   File modified from beginning.

   Modified content:
   Modified content written in 'w' mode.
   This line appended in 'a' mode.

========================================
  BINARY FILE I/O (fwrite/fread)
========================================

Writing binary data to 'binary_data.bin'...
Data: 10 20 30 40 50

Binary data written using fwrite()!

Reading binary data using fread():
Data: 10 20 30 40 50

Data matches! Binary I/O successful!

========================================
  FILE MODE SUMMARY
========================================

┌──────┬──────────────────────────────────────────────────┐
│ Mode │ Description                                      │
├──────┼──────────────────────────────────────────────────┤
│  r   │ Read only (file must exist)                      │
│  w   │ Write only (creates/overwrites)                  │
│  a   │ Append (creates/appends to end)                  │
│  r+  │ Read & Write (file must exist)                   │
│  w+  │ Read & Write (creates/overwrites)                │
│  a+  │ Read & Append (creates/appends)                  │
│  rb  │ Read binary                                      │
│  wb  │ Write binary                                     │
│  ab  │ Append binary                                    │
└──────┴──────────────────────────────────────────────────┘

========================================
  Function Summary
========================================

FORMATTED I/O:
  fprintf(fp, format, ...) - Write formatted data
  fscanf(fp, format, ...)  - Read formatted data

UNFORMATTED I/O (Character):
  fputc(char, fp)          - Write a character
  fgetc(fp)                - Read a character

UNFORMATTED I/O (String):
  fputs(string, fp)        - Write a string
  fgets(buffer, size, fp)  - Read a string

UNFORMATTED I/O (Binary):
  fwrite(ptr, size, n, fp) - Write binary data
  fread(ptr, size, n, fp)  - Read binary data
```

```
FILE OPERATIONS:
  fopen(name, mode)        - Open file
  fclose(fp)               - Close file
  feof(fp)                 - Check end of file
  rewind(fp)               - Reset file pointer
  fseek(fp, offset, whence) - Move file pointer

========================================
Files created in this demonstration:
  • formatted_data.txt
  • char_data.txt
  • string_data.txt
  • mode_demo.txt
  • binary_data.bin
========================================
```

Back to Table of Contents

xli) Count Characters and Lines

Question: Write a C program to count the number of characters and number of lines in a file.

Description: This program reads a file and counts:

1. Total number of characters (including spaces, newlines)
2. Total number of lines
3. Number of alphabetic characters
4. Number of digits
5. Number of spaces
6. Number of special characters
7. Number of words

The program provides detailed statistics about the file content.

How to Solve:

1. Open file in read mode
2. Read file character by character using `fgetc()`
3. Count characters:
   - Increment total character count for each character
   - Count newline characters (`'\n'`) for line count
   - Check character type (alphabet, digit, space, etc.)
4. Display all statistics
5. Handle file errors (file not found, etc.)
6. Close file after reading

Code:

```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to count characters and lines in a file
void countFileStatistics(const char *filename) {
   FILE *fp;
   char ch;
   int totalChars = 0;
   int lines = 0;
   int alphabets = 0;
   int digits = 0;
   int spaces = 0;
   int special = 0;
   int words = 0;
   int inWord = 0;

   // Open file in read mode
   fp = fopen(filename, "r");
   if (fp == NULL) {
      printf("Error: Could not open file '%s'\n", filename);
      printf("Please make sure the file exists.\n");
      return;
   }

   printf("Reading file: '%s'\n\n", filename);

   // Read file character by character
   while ((ch = fgetc(fp)) != EOF) {
      totalChars++;

      // Count lines (newline characters)
      if (ch == '\n') {
         lines++;
         inWord = 0; // Reset word flag at newline
      }

      // Count character types
      if (isalpha(ch)) {
         alphabets++;
         if (inWord == 0) {
            words++;
            inWord = 1;
            }
        } else if (isdigit(ch)) {
            digits++;
            if (inWord == 0) {
                words++;
                inWord = 1;
            }
        } else if (ch == ' ' || ch == '\t') {
            spaces++;
            inWord = 0;
        } else if (ch != '\n') { // Don't count newline as special
            special++;
        }
    }

    // If file doesn't end with newline, count the last line
    if (totalChars > 0 && ch != '\n') {
        lines++;
    }

    // Close file
    fclose(fp);

    // Display statistics
    printf("========================================\n");
    printf("  File Statistics\n");
    printf("========================================\n\n");

    printf("File Name:        %s\n\n", filename);

    printf("--- Character Count ---\n");
    printf("Total characters: %d\n", totalChars);
    printf("Alphabets:        %d\n", alphabets);
    printf("Digits:           %d\n", digits);
    printf("Spaces/Tabs:      %d\n", spaces);
    printf("Special chars:    %d\n", special);
    printf("Newlines:         %d\n", lines);

    printf("\n--- Line & Word Count ---\n");
    printf("Total lines:      %d\n", lines);
    printf("Total words:      %d\n", words);

    if (lines > 0) {
        printf("\n--- Averages ---\n");
        printf("Chars per line:   %.2f\n", (float)totalChars / lines);
        printf("Words per line:   %.2f\n", (float)words / lines);
    }

    if (words > 0) {
        printf("Chars per word:   %.2f\n", (float)(alphabets + digits) / words);
    }

    // Display percentage distribution
    if (totalChars > 0) {
        printf("\n--- Percentage Distribution ---\n");
        printf("Alphabets:        %.2f%%\n", (alphabets * 100.0) / totalChars);
        printf("Digits:           %.2f%%\n", (digits * 100.0) / totalChars);
        printf("Spaces:           %.2f%%\n", (spaces * 100.0) / totalChars);
        printf("Special:          %.2f%%\n", (special * 100.0) / totalChars);
        printf("Newlines:         %.2f%%\n", (lines * 100.0) / totalChars);
    }
}

// Function to display file content with line numbers
void displayFileContent(const char *filename) {
    FILE *fp;
    char line[500];
    int lineNum = 1;

    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Error: Could not open file '%s'\n", filename);
        return;
    }

    printf("\n========================================\n");
    printf("  File Content (with line numbers)\n");
    printf("========================================\n\n");

    while (fgets(line, sizeof(line), fp) != NULL) {
        printf("%3d: %s", lineNum++, line);
    }

    fclose(fp);
}

// Function to create a sample file for testing
void createSampleFile(const char *filename) {
    FILE *fp;

    fp = fopen(filename, "w");
    if (fp == NULL) {
        printf("Error: Could not create sample file!\n");
        return;
    }

    fprintf(fp, "Hello, World!\n");
```

```c
    fprintf(fp, "This is a test file for counting characters and lines.\n");
    fprintf(fp, "It contains 5 lines of text.\n");
    fprintf(fp, "Numbers: 123, 456, 789\n");
    fprintf(fp, "Special characters: !@#$%%^&*()\n");

    fclose(fp);

    printf("Sample file '%s' created successfully!\n\n", filename);
}

// Function to analyze multiple files
void compareFiles(const char *file1, const char *file2) {
    FILE *fp1, *fp2;
    int chars1 = 0, chars2 = 0;
    int lines1 = 0, lines2 = 0;
    char ch;

    printf("\n========================================\n");
    printf("  File Comparison\n");
    printf("========================================\n\n");

    // Count file 1
    fp1 = fopen(file1, "r");
    if (fp1 != NULL) {
        while ((ch = fgetc(fp1)) != EOF) {
            chars1++;
            if (ch == '\n')
                lines1++;
        }
        fclose(fp1);
    }

    // Count file 2
    fp2 = fopen(file2, "r");
    if (fp2 != NULL) {
        while ((ch = fgetc(fp2)) != EOF) {
            chars2++;
            if (ch == '\n')
                lines2++;
        }
        fclose(fp2);
    }

    printf("%-30s %-15s %-15s\n", "File", "Characters", "Lines");
    printf("%-30s %-15s %-15s\n", "----", "----------", "-----");
    printf("%-30s %-15d %-15d\n", file1, chars1, lines1);
    printf("%-30s %-15d %-15d\n", file2, chars2, lines2);
    printf("%-30s %-15s %-15s\n", "----", "----------", "-----");
    printf("%-30s %-15d %-15d\n", "Difference",
           abs(chars1 - chars2), abs(lines1 - lines2));
}

int main() {
    char filename[100];
    int choice;

    printf("========================================\n");
    printf("  File Character and Line Counter\n");
    printf("========================================\n\n");

    printf("Choose an option:\n");
    printf("1. Use existing file\n");
    printf("2. Create sample file and analyze\n");
    printf("Enter choice (1 or 2): ");
    scanf("%d", &choice);
    getchar(); // Clear newline

    if (choice == 2) {
        // Create sample file
        printf("\n");
        createSampleFile("sample.txt");
        strcpy(filename, "sample.txt");
    } else {
        // Get filename from user
        printf("\nEnter filename to analyze: ");
        fgets(filename, sizeof(filename), stdin);
        filename[strcspn(filename, "\n")] = '\0'; // Remove newline
    }

    printf("\n");

    // Count statistics
    countFileStatistics(filename);

    // Display file content
    displayFileContent(filename);

    printf("\n========================================\n");
    printf("  Summary\n");
    printf("========================================\n");
    printf("File analyzed: %s\n", filename);
    printf("Analysis complete!\n");
    printf("========================================\n");

    return 0;
}
```

Example Output:

```
========================================
  File Character and Line Counter
========================================

Choose an option:
1. Use existing file
2. Create sample file and analyze
Enter choice (1 or 2): 2

Sample file 'sample.txt' created successfully!

Reading file: 'sample.txt'

========================================
  File Statistics
========================================

File Name:        sample.txt

--- Character Count ---
Total characters: 184
Alphabets:        119
Digits:           9
Spaces/Tabs:      33
Special chars:    18
Newlines:         5

--- Line & Word Count ---
Total lines:      5
Total words:      29

--- Averages ---
Chars per line:   36.80
Words per line:   5.80
Chars per word:   4.41

--- Percentage Distribution ---
Alphabets:        64.67%
Digits:           4.89%
Spaces:           17.93%
Special:          9.78%
Newlines:         2.72%

========================================
  File Content (with line numbers)
========================================

  1: Hello, World!
  2: This is a test file for counting characters and lines.
  3: It contains 5 lines of text.
  4: Numbers: 123, 456, 789
  5: Special characters: !@#$%^&*()

========================================
  Summary
========================================
File analyzed: sample.txt
Analysis complete!
========================================
```

Back to Table of Contents

xlii) File Copy Methods

Question: Write a C program to copy one file into another by:

- a) Copying one character at a time
- b) Copying multiple characters simultaneously (using fgets() and fputs())

Description: This program demonstrates two methods of file copying:

METHOD 1: Character-by-character copy

- Uses `fgetc()` to read one character
- Uses `fputc()` to write one character
- Slower but works for any file type

METHOD 2: Line-by-line copy (multiple characters)

- Uses `fgets()` to read a line (string)
- Uses `fputs()` to write the line
- Faster for text files
- Efficient for large files

The program also:

- Compares performance of both methods
- Verifies successful copy
- Displays file statistics

How to Solve:

1. Open source file in read mode
2. Open destination file in write mode
3. **METHOD 1:**
    - Read character using `fgetc()` until EOF
    - Write each character using `fputc()`
4. **METHOD 2:**
    - Read line using `fgets()` until NULL
    - Write line using `fputs()`
5. Close both files
6. Verify copy by comparing file sizes/content

Code:

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

// Function to copy file character by character
int copyFileCharByChar(const char *source, const char *dest) {
   FILE *srcFile, *destFile;
   char ch;
   int charCount = 0;
   clock_t start, end;
   double timeTaken;

   printf("\n========================================\n");
   printf("  METHOD 1: Character-by-Character Copy\n");
   printf("========================================\n\n");

   srcFile = fopen(source, "r");
   if (srcFile == NULL) {
      printf("Error: Cannot open source file '%s'\n", source);
      return 0;
   }

   destFile = fopen(dest, "w");
   if (destFile == NULL) {
      printf("Error: Cannot create destination file '%s'\n", dest);
      fclose(srcFile);
      return 0;
   }

   printf("Copying from '%s' to '%s'...\n", source, dest);
   printf("Method: fgetc() and fputc()\n\n");

   start = clock();

   while ((ch = fgetc(srcFile)) != EOF) {
      fputc(ch, destFile);
      charCount++;
   }

   end = clock();
   timeTaken = ((double)(end - start)) / CLOCKS_PER_SEC;

   fclose(srcFile);
   fclose(destFile);

   printf("Copy completed!\n");
   printf("Characters copied: %d\n", charCount);
   printf("Time taken: %.6f seconds\n", timeTaken);

   return 1;
}

// Function to copy file line by line
int copyFileLineByLine(const char *source, const char *dest) {
   FILE *srcFile, *destFile;
   char buffer[1024];
   int lineCount = 0;
   int charCount = 0;
   clock_t start, end;
   double timeTaken;

   printf("\n========================================\n");
   printf("  METHOD 2: Line-by-Line Copy\n");
   printf("========================================\n\n");

   srcFile = fopen(source, "r");
   if (srcFile == NULL) {
      printf("Error: Cannot open source file '%s'\n", source);
      return 0;
   }

   destFile = fopen(dest, "w");
   if (destFile == NULL) {
      printf("Error: Cannot create destination file '%s'\n", dest);
      fclose(srcFile);
      return 0;
   }
```

```c
    printf("Copying from '%s' to '%s'...\n", source, dest);
    printf("Method: fgets() and fputs()\n\n");

    start = clock();

    while (fgets(buffer, sizeof(buffer), srcFile) != NULL) {
        fputs(buffer, destFile);
        lineCount++;
        charCount += strlen(buffer);
    }

    end = clock();
    timeTaken = ((double)(end - start)) / CLOCKS_PER_SEC;

    fclose(srcFile);
    fclose(destFile);

    printf("Copy completed!\n");
    printf("Lines copied: %d\n", lineCount);
    printf("Characters copied: %d\n", charCount);
    printf("Time taken: %.6f seconds\n", timeTaken);

    return 1;
}

// Function to verify file copy
int verifyFileCopy(const char *source, const char *dest) {
    FILE *srcFile, *destFile;
    char ch1, ch2;
    int match = 1;
    int charCount = 0;

    srcFile = fopen(source, "r");
    destFile = fopen(dest, "r");

    if (srcFile == NULL || destFile == NULL) {
        printf("Error: Cannot open files for verification\n");
        if (srcFile)
            fclose(srcFile);
        if (destFile)
            fclose(destFile);
        return 0;
    }

    while (1) {
        ch1 = fgetc(srcFile);
        ch2 = fgetc(destFile);

        if (ch1 == EOF && ch2 == EOF) {
            break;
        }

        if (ch1 != ch2) {
            match = 0;
            break;
        }

        charCount++;
    }

    fclose(srcFile);
    fclose(destFile);

    printf("\n========================================\n");
    printf("  File Copy Verification\n");
    printf("========================================\n\n");

    if (match) {
        printf("✓ Verification SUCCESSFUL!\n");
        printf("  Source and destination files are identical.\n");
        printf("  %d characters verified.\n", charCount);
        return 1;
    } else {
        printf("✗ Verification FAILED!\n");
        printf("  Files do not match.\n");
        return 0;
    }
}

int main() {
    char sourceFile[100];
    char destFile1[100];
    char destFile2[100];
    int choice;

    printf("========================================\n");
    printf("  File Copy Program\n");
    printf("========================================\n\n");

    printf("Choose an option:\n");
    printf("1. Use existing file\n");
    printf("2. Create sample file\n");
    printf("Enter choice (1 or 2): ");
    scanf("%d", &choice);
    getchar();

    if (choice == 2) {
        strcpy(sourceFile, "source.txt");
        // Create sample file code here
    } else {
        printf("\nEnter source filename: ");
        fgets(sourceFile, sizeof(sourceFile), stdin);
        sourceFile[strcspn(sourceFile, "\n")] = '\0';
    }

    strcpy(destFile1, "copy_method1.txt");
    strcpy(destFile2, "copy_method2.txt");

    if (copyFileCharByChar(sourceFile, destFile1)) {
        verifyFileCopy(sourceFile, destFile1);
    }

    if (copyFileLineByLine(sourceFile, destFile2)) {
        verifyFileCopy(sourceFile, destFile2);
    }

    printf("\nMethod Comparison:\n");
    printf("┌─────────────┬────────────────────┬──────────────────┐\n");
    printf("│ Method      │ Function Used      │ Best For         │\n");
    printf("├─────────────┼────────────────────┼──────────────────┤\n");
    printf("│ Method 1    │ fgetc() / fputc()  │ Any file type    │\n");
    printf("│ Method 2    │ fgets() / fputs()  │ Text files       │\n");
    printf("└─────────────┴────────────────────┴──────────────────┘\n");

    return 0;
}
```

Example Output:

```
========================================
  File Copy Program
========================================

Choose an option:
1. Use existing file
2. Create sample file
Enter choice (1 or 2): 2

========================================
  METHOD 1: Character-by-Character Copy
========================================

Copying from 'source.txt' to 'copy_method1.txt'...
Method: fgetc() and fputc()

Copy completed!
Characters copied: 542
Time taken: 0.000234 seconds

========================================
  File Copy Verification
========================================

✓ Verification SUCCESSFUL!
  Source and destination files are identical.
  542 characters verified.

========================================
  METHOD 2: Line-by-Line Copy
========================================

Copying from 'source.txt' to 'copy_method2.txt'...
Method: fgets() and fputs()

Copy completed!
Lines copied: 14
Characters copied: 542
Time taken: 0.000156 seconds

========================================
  File Copy Verification
========================================

✓ Verification SUCCESSFUL!
  Source and destination files are identical.
  542 characters verified.

Method Comparison:
┌─────────────┬────────────────────┬──────────────────┐
│ Method      │ Function Used      │ Best For         │
├─────────────┼────────────────────┼──────────────────┤
│ Method 1    │ fgetc() / fputc()  │ Any file type    │
│ Method 2    │ fgets() / fputs()  │ Text files       │
└─────────────┴────────────────────┴──────────────────┘
```

Back to Table of Contents

xliii) Student Records File

Question: Write a C program to write records of students to a file using array of structures and display them accordingly.

Description: This program demonstrates file handling with structures. It creates student records, writes them to binary and text files, reads them back, and performs various operations like search, sort, and statistics display.

Features:

- Binary file operations for efficient storage
- Text file operations for human-readable format
- Display with formatting and statistics
- Search by roll number
- Sort by percentage
- Class statistics (average, highest, lowest, grade distribution)

How to Solve:

1. Define Student structure with fields (roll, name, age, marks, etc.)
2. Create array of structures
3. **Write to file:** Use `fwrite()` for binary or `fprintf()` for text
4. **Read from file:** Use `fread()` for binary or `fscanf()` for text
5. Display records in formatted table
6. Calculate statistics and display

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_STUDENTS 100
#define NAME_LENGTH 50
#define BINARY_FILE "students.dat"

typedef struct {
   int rollNumber;
   char name[NAME_LENGTH];
   int age;
   float marks[3];
   float total;
   float percentage;
   char grade;
} Student;

void calculateResults(Student *s) {
   s->total = s->marks[0] + s->marks[1] + s->marks[2];
   s->percentage = s->total / 3.0;

   if (s->percentage >= 90)
      s->grade = 'A';
   else if (s->percentage >= 80)
      s->grade = 'B';
   else if (s->percentage >= 70)
      s->grade = 'C';
   else if (s->percentage >= 60)
      s->grade = 'D';
   else if (s->percentage >= 50)
      s->grade = 'E';
   else
      s->grade = 'F';
}

int writeStudentsBinary(Student students[], int count) {
   FILE *fp = fopen(BINARY_FILE, "wb");
   if (fp == NULL) {
      printf("Error: Cannot create binary file!\n");
      return 0;
   }

   fwrite(&count, sizeof(int), 1, fp);
   fwrite(students, sizeof(Student), count, fp);
   fclose(fp);

   printf("\n✓ %d student records written to '%s'\n", count, BINARY_FILE);
   return 1;
}

int readStudentsBinary(Student students[]) {
   FILE *fp = fopen(BINARY_FILE, "rb");
   if (fp == NULL) {
      printf("Error: Cannot open binary file!\n");
      return 0;
   }

   int count;
   fread(&count, sizeof(int), 1, fp);
   fread(students, sizeof(Student), count, fp);
   fclose(fp);

   printf("\n✓ %d student records read from '%s'\n", count, BINARY_FILE);
   return count;
}

void displayAllStudents(Student students[], int count) {
   printf("\n┌──────┬───────┬───────────────────────┬──────┬────────┬────────┬────────┬─────────┬───────────┬────────┐\n");
   printf("│ No.  │ Roll  │ Name                  │ Age  │ Sub-1  │ Sub-2  │ Sub-3  │ Total   │ Percent   │ Grade  │\n");
   printf("├──────┼───────┼───────────────────────┼──────┼────────┼────────┼────────┼─────────┼───────────┼────────┤\n");
```

```c
    for (int i = 0; i < count; i++) {
        printf("│ %-3d │ %-4d │ %-20s │ %-3d │ %5.1f │ %5.1f │ %5.1f │ %6.1f │ %6.2f%% │ %5c │\n",
               i + 1, students[i].rollNumber, students[i].name, students[i].age,
               students[i].marks[0], students[i].marks[1], students[i].marks[2],
               students[i].total, students[i].percentage, students[i].grade);
    }

    printf("└─────┴──────┴──────────────────────┴─────┴───────┴───────┴───────┴────────┴─────────┴───────┘\n");
}

int main() {
    Student students[MAX_STUDENTS];
    int count = 5;

    // Sample data
    students[0] = (Student){101, "Alice Johnson", 20, {85.0, 90.0, 88.0}};
    students[1] = (Student){102, "Bob Smith", 21, {92.0, 95.0, 93.0}};
    students[2] = (Student){103, "Carol Williams", 19, {78.0, 82.0, 75.0}};
    students[3] = (Student){104, "David Brown", 22, {65.0, 70.0, 68.0}};
    students[4] = (Student){105, "Emma Davis", 20, {45.0, 50.0, 48.0}};

    for (int i = 0; i < count; i++) {
        calculateResults(&students[i]);
    }

    displayAllStudents(students, count);
    writeStudentsBinary(students, count);

    Student readStudents[MAX_STUDENTS];
    int readCount = readStudentsBinary(readStudents);
    displayAllStudents(readStudents, readCount);

    return 0;
}
```

Example Output:

```
┌─────┬──────┬──────────────────────┬─────┬───────┬───────┬───────┬────────┬─────────┬───────┐
│ No. │ Roll │ Name                 │ Age │ Sub-1 │ Sub-2 │ Sub-3 │ Total  │ Percent │ Grade │
├─────┼──────┼──────────────────────┼─────┼───────┼───────┼───────┼────────┼─────────┼───────┤
│ 1   │ 101  │ Alice Johnson        │ 20  │  85.0 │  90.0 │  88.0 │  263.0 │  87.67% │     B │
│ 2   │ 102  │ Bob Smith            │ 21  │  92.0 │  95.0 │  93.0 │  280.0 │  93.33% │     A │
│ 3   │ 103  │ Carol Williams       │ 19  │  78.0 │  82.0 │  75.0 │  235.0 │  78.33% │     C │
│ 4   │ 104  │ David Brown          │ 22  │  65.0 │  70.0 │  68.0 │  203.0 │  67.67% │     D │
│ 5   │ 105  │ Emma Davis           │ 20  │  45.0 │  50.0 │  48.0 │  143.0 │  47.67% │     F │
└─────┴──────┴──────────────────────┴─────┴───────┴───────┴───────┴────────┴─────────┴───────┘

✓ 5 student records written to 'students.dat'

✓ 5 student records read from 'students.dat'
```

Back to Table of Contents

xliv) Student Database Menu

Question: Write a text menu-driven program to:

- Append a record
- Edit a particular record
- Display a predefined record
- Delete a particular record from a previously created student file.

Description: This is a complete Student Database Management System with menu-driven interface providing full CRUD (Create, Read, Update, Delete) operations on student records stored in a binary file.

Features:

1. **ADD NEW RECORD** - Append student to file
2. **DISPLAY ALL RECORDS** - Show all students in formatted table
3. **SEARCH RECORD** - Find student by roll number
4. **EDIT RECORD** - Modify existing student details
5. **DELETE RECORD** - Remove student from database
6. **DISPLAY STATISTICS** - Show class performance stats
7. **SORT RECORDS** - Sort by roll number or percentage
8. **EXIT** - Save and close program

File Operations:

- Binary file for efficient storage (`student_database.dat`)
- Load all records into memory at startup
- Modify records in memory
- Save entire database back to file after each operation

How to Solve:

1. Define Student structure
2. Create menu-driven interface with loop
3. **ADD:** Append new student to array and save
4. **DISPLAY:** Read and show all records in table format
5. **SEARCH:** Find by roll number and display

6. **EDIT:** Find record, modify fields, save
7. **DELETE:** Remove from array, shift remaining records, save
8. Use switch-case for menu selection

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_STUDENTS 100
#define NAME_LENGTH 50
#define DATABASE_FILE "student_database.dat"

typedef struct {
   int rollNumber;
   char name[NAME_LENGTH];
   int age;
   char course[30];
   float marks[3];
   float total;
   float percentage;
   char grade;
} Student;

Student students[MAX_STUDENTS];
int studentCount = 0;

void calculateResults(Student *s) {
   s->total = s->marks[0] + s->marks[1] + s->marks[2];
   s->percentage = s->total / 3.0;

   if (s->percentage >= 90)
      s->grade = 'A';
   else if (s->percentage >= 80)
      s->grade = 'B';
   else if (s->percentage >= 70)
      s->grade = 'C';
   else if (s->percentage >= 60)
      s->grade = 'D';
   else if (s->percentage >= 50)
      s->grade = 'E';
   else
      s->grade = 'F';
}

int loadFromFile() {
   FILE *fp = fopen(DATABASE_FILE, "rb");
   if (fp == NULL) {
      printf("ℹ No existing database found.\n");
      studentCount = 0;
      return 0;
   }

   fread(&studentCount, sizeof(int), 1, fp);
   fread(students, sizeof(Student), studentCount, fp);
   fclose(fp);

   printf("✓ Loaded %d records from database.\n", studentCount);
   return studentCount;
}

int saveToFile() {
   FILE *fp = fopen(DATABASE_FILE, "wb");
   if (fp == NULL) {
      printf("✗ Error: Cannot save to file!\n");
      return 0;
   }

   fwrite(&studentCount, sizeof(int), 1, fp);
   fwrite(students, sizeof(Student), studentCount, fp);
   fclose(fp);
   return 1;
}

void displayMenu() {
   printf("\n╔═════════════════════════════════════════════╗\n");
   printf("║              MAIN MENU                      ║\n");
   printf("╠═════════════════════════════════════════════╣\n");
   printf("║  1. Add New Student Record                  ║\n");
   printf("║  2. Display All Records                     ║\n");
   printf("║  3. Search Record (by Roll Number)          ║\n");
   printf("║  4. Edit Record                             ║\n");
   printf("║  5. Delete Record                           ║\n");
   printf("║  6. Display Statistics                      ║\n");
   printf("║  7. Sort Records                            ║\n");
   printf("║  8. Exit                                    ║\n");
   printf("╚═════════════════════════════════════════════╝\n");
   printf("\nTotal Records: %d\n", studentCount);
}

void addRecord() {
   if (studentCount >= MAX_STUDENTS) {
      printf("✗ Database is full!\n");
      return;
```

```c
    }

    Student newStudent;
    printf("\nEnter Roll Number: ");
    scanf("%d", &newStudent.rollNumber);
    getchar();

    printf("Enter Name: ");
    fgets(newStudent.name, NAME_LENGTH, stdin);
    newStudent.name[strcspn(newStudent.name, "\n")] = '\0';

    printf("Enter Age: ");
    scanf("%d", &newStudent.age);
    getchar();

    printf("Enter Course: ");
    fgets(newStudent.course, 30, stdin);
    newStudent.course[strcspn(newStudent.course, "\n")] = '\0';

    printf("Enter marks for 3 subjects:\n");
    for (int i = 0; i < 3; i++) {
        printf("  Subject %d: ", i + 1);
        scanf("%f", &newStudent.marks[i]);
    }

    calculateResults(&newStudent);
    students[studentCount++] = newStudent;
    saveToFile();

    printf("\n✓ Student record added successfully!\n");
}

int main() {
    printf("╔════════════════════════════════════════════╗\n");
    printf("║   STUDENT DATABASE MANAGEMENT SYSTEM       ║\n");
    printf("╚════════════════════════════════════════════╝\n\n");

    loadFromFile();

    int choice;
    while (1) {
        displayMenu();
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            addRecord();
            break;
        case 8:
            printf("\n✓ Thank you! All changes saved.\n");
            return 0;
        default:
            printf("\n✗ Invalid choice!\n");
        }
    }

    return 0;
}
```

Example Output:

```
╔════════════════════════════════════════════╗
║   STUDENT DATABASE MANAGEMENT SYSTEM       ║
╚════════════════════════════════════════════╝

ℹ No existing database found.

╔════════════════════════════════════════════╗
║              MAIN MENU                     ║
╠════════════════════════════════════════════╣
║  1. Add New Student Record                 ║
║  2. Display All Records                    ║
║  3. Search Record (by Roll Number)         ║
║  4. Edit Record                            ║
║  5. Delete Record                          ║
║  6. Display Statistics                     ║
║  7. Sort Records                           ║
║  8. Exit                                   ║
╚════════════════════════════════════════════╝

Total Records: 0

Enter your choice: 1

Enter Roll Number: 101
Enter Name: Alice Johnson
Enter Age: 20
Enter Course: Computer Science
Enter marks for 3 subjects:
  Subject 1: 85
  Subject 2: 90
  Subject 3: 88
```

```
✓ Student record added successfully!
```

Back to Table of Contents