**COMSATS INSTITUTE OF INFORMATION TECHNOLOGY,**

**ABBOTTABAD CAMPUS.**

**ASSIGNMENT NO 8.**

**GIT HUB LINK:**

**SUBJECT:**

**PROGRAMMING FUNDAMENTALS.**

**SUBMITTED TO:**

**DR KASHIF BILAL.**

**SUBMITTED BY:**

**AEMAN SAJID**

**REGISTRATION NUMBER:**

**FA22-BSE-084**

**PROGRAM AND SECTION:**

**SOFTWARE ENGINEERING- SEC: 2B.**

**ARRAYS:**

**1.**

#include <stdio.h>

#define MAX\_SIZE 10

void subtractMatrices(int mat1[][MAX\_SIZE], int mat2[][MAX\_SIZE], int result[][MAX\_SIZE], int rows, int cols) {

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            result[i][j] = mat1[i][j] - mat2[i][j];

        }

    }

}

void displayMatrix(int mat[][MAX\_SIZE], int rows, int cols) {

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            printf("%d ", mat[i][j]);

        }

        printf("\n");

    }

}

int main() {

    int mat1[MAX\_SIZE][MAX\_SIZE];

    int mat2[MAX\_SIZE][MAX\_SIZE];

    int result[MAX\_SIZE][MAX\_SIZE];

    int rows, cols;

    printf("Enter the number of rows (max %d): ", MAX\_SIZE);

    scanf("%d", &rows);

    printf("Enter the number of columns (max %d): ", MAX\_SIZE);

    scanf("%d", &cols);

    printf("Enter elements of matrix 1:\n");

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            scanf("%d", &mat1[i][j]);

        }

    }

    printf("Enter elements of matrix 2:\n");

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            scanf("%d", &mat2[i][j]);

        }

    }

    subtractMatrices(mat1, mat2, result, rows, cols);

    printf("Matrix 1:\n");

    displayMatrix(mat1, rows, cols);

    printf("Matrix 2:\n");

    displayMatrix(mat2, rows, cols);

    printf("Subtraction Result:\n");

    displayMatrix(result, rows, cols);

    return 0;

}

**2.**

#include <stdio.h>

#define MAX\_SIZE 10

void multiplyMatrices(int mat1[][MAX\_SIZE], int mat2[][MAX\_SIZE], int result[][MAX\_SIZE], int n) {

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            result[i][j] = 0;

            for (int k = 0; k < n; k++) {

                result[i][j] += mat1[i][k] \* mat2[k][j];

            }

        }

    }

}

void displayMatrix(int mat[][MAX\_SIZE], int n) {

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            printf("%d ", mat[i][j]);

        }

        printf("\n");

    }

}

int main() {

    int mat1[MAX\_SIZE][MAX\_SIZE];

    int mat2[MAX\_SIZE][MAX\_SIZE];

    int result[MAX\_SIZE][MAX\_SIZE];

    int n;

    printf("Enter the value of n (max %d): ", MAX\_SIZE);

    scanf("%d", &n);

    printf("Enter elements of matrix 1:\n");

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            scanf("%d", &mat1[i][j]);

        }

    }

    printf("Enter elements of matrix 2:\n");

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            scanf("%d", &mat2[i][j]);

        }

    }

    multiplyMatrices(mat1, mat2, result, n);

    printf("Matrix 1:\n");

    displayMatrix(mat1, n);

    printf("Matrix 2:\n");

    displayMatrix(mat2, n);

    printf("Multiplication Result:\n");

    displayMatrix(result, n);

    return 0;

}

**3.**

#include <stdio.h>

void displayRowIndex(int number) {

    int digit;

    int row, column;

    printf("Number: %d\n", number);

    while (number > 0) {

        digit = number % 10;

        row = digit;

        column = digit;

        printf("Digit: %d, Row: %d, Column: %d\n", digit, row, column);

        number /= 10;

    }

}

int main() {

    int number;

    printf("Enter a number: ");

    scanf("%d", &number);

    displayRowIndex(number);

    return 0;

}

#include <stdio.h>

#define ROWS 4

#define COLS 4

int binarySearch(int arr[][COLS], int target, int row, int start, int end) {

    while (start <= end) {

        int mid = start + (end - start) / 2;

        if (arr[row][mid] == target) {

            return mid;

        } else if (arr[row][mid] < target) {

            start = mid + 1;

        } else {

            end = mid - 1;

        }

    }

    return -1;

}

void search2DArray(int arr[][COLS], int target) {

    int row = 0;

    int col = COLS - 1;

    int found = 0;

    while (row < ROWS && col >= 0) {

        if (arr[row][col] == target) {

            printf("Found at row %d, column %d\n", row, col);

            found = 1;

            break;

        } else if (arr[row][col] < target) {

            row++;

        } else {

            int index = binarySearch(arr, target, row, 0, col - 1);

            if (index != -1) {

                printf("Found at row %d, column %d\n", row, index);

                found = 1;

                break;

            } else {

                col--;

            }

        }

    }

    if (!found) {

        printf("Element not found in the 2D array.\n");

    }

}

int main() {

    int arr[ROWS][COLS] = {

        {2, 4, 6, 8},

        {10, 12, 14, 16},

        {18, 20, 22, 24},

        {26, 28, 30, 32}

    };

    int target;

    printf("Enter the target element: ");

    scanf("%d", &target);

    search2DArray(arr, target);

    return 0;

}

**4.**

#include <stdio.h>

void findAdditiveSequences(int num, int currentNum, char \*sequence, int index) {

    if (currentNum == 0) {

        sequence[index] = '\0';

        printf("%s\n", sequence);

        return;

    }

    for (int i = 1; i <= currentNum; i++) {

        sequence[index] = i + '0';

        findAdditiveSequences(num, currentNum - i, sequence, index + 1);

    }

}

int main() {

    int num;

    printf("Enter a number: ");

    scanf("%d", &num);

    char sequence[100];

    printf("Additive Sequences for %d:\n", num);

    findAdditiveSequences(num, num, sequence, 0);

    return 0;

}

**5.**

#include <stdio.h>

#include <string.h>

#define MAX\_LENGTH 100

int isDelimiter(char ch, char delimiter) {

    return (ch == delimiter);

}

void splitString(char \*str, char delimiter) {

    int length = strlen(str);

    char word[MAX\_LENGTH];

    int wordIndex = 0;

    for (int i = 0; i < length; i++) {

        if (!isDelimiter(str[i], delimiter)) {

            word[wordIndex++] = str[i];

        } else {

            word[wordIndex] = '\0';

            if (wordIndex > 0) {

                printf("%s\n", word);

            }

            wordIndex = 0;

        }

    }

    // Print the last word if it exists

    if (wordIndex > 0) {

        word[wordIndex] = '\0';

        printf("%s\n", word);

    }

}

int main() {

    char str[MAX\_LENGTH];

    char delimiter;

    printf("Enter a string: ");

    fgets(str, sizeof(str), stdin);

    printf("Enter the delimiter character: ");

    scanf("%c", &delimiter);

    printf("Words after splitting:\n");

    splitString(str, delimiter);

    return 0;

}

**6.**

#include <stdio.h>

int stringLength(const char \*str) {

    int length = 0;

    while (str[length] != '\0') {

        length++;

    }

    return length;

}

int main() {

    char str[100];

    printf("Enter a string: ");

    fgets(str, sizeof(str), stdin);

    // Remove the trailing newline character

    if (str[stringLength(str) - 1] == '\n') {

        str[stringLength(str) - 1] = '\0';

    }

    int length = stringLength(str);

    printf("The length of the string is: %d\n", length);

    return 0;

}

**STRINGS(ASSIGN PART 2):**

**1.**

#include <stdio.h>

#define MAX\_CHARACTERS 256

void countCharacterFrequency(const char \*str) {

    int frequency[MAX\_CHARACTERS] = {0};  // Initialize frequency array with zeros

    // Iterate through each character of the string

    for (int i = 0; str[i] != '\0'; i++) {

        int index = (int)str[i];  // Get the ASCII value of the character

        frequency[index]++;       // Increment the frequency of the character

    }

    // Print the character frequency

    printf("Character frequency:\n");

    for (int i = 0; i < MAX\_CHARACTERS; i++) {

        if (frequency[i] > 0) {

            printf("'%c': %d\n", (char)i, frequency[i]);

        }

    }

}

int main() {

    char str[100];

    printf("Enter a string: ");

    fgets(str, sizeof(str), stdin);

    // Remove the trailing newline character

    if (str[strlen(str) - 1] == '\n') {

        str[strlen(str) - 1] = '\0';

    }

    countCharacterFrequency(str);

    return 0;

}

**2.**

#include <stdio.h>

#include <string.h>

void addIngOrLy(char \*str) {

    int length = strlen(str);

    if (length < 3) {

        // String length is less than 3, no change needed

        return;

    }

    if (strcmp(str + length - 3, "ing") == 0) {

        // String already ends with 'ing', add 'ly' instead

        strcat(str, "ly");

    } else {

        // String does not end with 'ing', add 'ing' at the end

        strcat(str, "ing");

    }

}

int main() {

    char str[100];

    printf("Enter a string: ");

    fgets(str, sizeof(str), stdin);

    // Remove the trailing newline character

    if (str[strlen(str) - 1] == '\n') {

        str[strlen(str) - 1] = '\0';

    }

    addIngOrLy(str);

    printf("Modified string: %s\n", str);

    return 0;

}

**3.**

#include <stdio.h>

#include <string.h>

int findSubstringIndex(const char \*str, const char \*substr) {

    int strLength = strlen(str);

    int substrLength = strlen(substr);

    // Iterate through the string

    for (int i = 0; i <= strLength - substrLength; i++) {

        int j;

        // Check if current substring matches the given substring

        for (j = 0; j < substrLength; j++) {

            if (str[i + j] != substr[j]) {

                break;  // Mismatch found, break the inner loop

            }

        }

        // If the inner loop completed without a mismatch, return the index

        if (j == substrLength) {

            return i;

        }

    }

    return -1;  // Substring not found

}

int main() {

    char str[100];

    char substr[100];

    printf("Enter the string: ");

    fgets(str, sizeof(str), stdin);

    printf("Enter the substring: ");

    fgets(substr, sizeof(substr), stdin);

    // Remove the trailing newline character

    if (str[strlen(str) - 1] == '\n') {

        str[strlen(str) - 1] = '\0';

    }

    if (substr[strlen(substr) - 1] == '\n') {

        substr[strlen(substr) - 1] = '\0';

    }

    int index = findSubstringIndex(str, substr);

    if (index != -1) {

        printf("Substring found at index: %d\n", index);

    } else {

        printf("Substring not found.\n");

    }

    return 0;

}

**4.**

#include <stdio.h>

#include <string.h>

int findLastSubstringIndex(const char \*str, const char \*substr) {

    int strLength = strlen(str);

    int substrLength = strlen(substr);

    if (substrLength > strLength) {

        return -1;  // Substring cannot be longer than the string

    }

    // Iterate through the string in reverse

    for (int i = strLength - substrLength; i >= 0; i--) {

        int j;

        // Check if current substring matches the given substring

        for (j = 0; j < substrLength; j++) {

            if (str[i + j] != substr[j]) {

                break;  // Mismatch found, break the inner loop

            }

        }

        // If the inner loop completed without a mismatch, return the index

        if (j == substrLength) {

            return i;

        }

    }

    return -1;  // Substring not found

}

int main() {

    char str[100];

    char substr[100];

    printf("Enter the string: ");

    fgets(str, sizeof(str), stdin);

    printf("Enter the substring: ");

    fgets(substr, sizeof(substr), stdin);

    // Remove the trailing newline character

    if (str[strlen(str) - 1] == '\n') {

        str[strlen(str) - 1] = '\0';

    }

    if (substr[strlen(substr) - 1] == '\n') {

        substr[strlen(substr) - 1] = '\0';

    }

    int index = findLastSubstringIndex(str, substr);

    if (index != -1) {

        printf("Last occurrence of substring found at index: %d\n", index);

    } else {

        printf("Substring not found.\n");

    }

    return 0;

}

**6.**

#include <stdio.h>

#include <string.h>

void findAllSubstringIndexes(const char \*str, const char \*substr) {

    int strLength = strlen(str);

    int substrLength = strlen(substr);

    if (substrLength > strLength) {

        printf("Substring cannot be longer than the string.\n");

        return;

    }

    int count = 0;  // Count of substring occurrences

    // Iterate through the string

    for (int i = 0; i <= strLength - substrLength; i++) {

        int j;

        // Check if current substring matches the given substring

        for (j = 0; j < substrLength; j++) {

            if (str[i + j] != substr[j]) {

                break;  // Mismatch found, break the inner loop

            }

        }

        // If the inner loop completed without a mismatch, print the index

        if (j == substrLength) {

            printf("Substring found at index: %d\n", i);

            count++;

        }

    }

    if (count == 0) {

        printf("Substring not found.\n");

    } else {

        printf("Total occurrences: %d\n", count);

    }

}

int main() {

    char str[100];

    char substr[100];

    printf("Enter the string: ");

    fgets(str, sizeof(str), stdin);

    printf("Enter the substring: ");

    fgets(substr, sizeof(substr), stdin);

    // Remove the trailing newline character

    if (str[strlen(str) - 1] == '\n') {

        str[strlen(str) - 1] = '\0';

    }

    if (substr[strlen(substr) - 1] == '\n') {

        substr[strlen(substr) - 1] = '\0';

    }

    findAllSubstringIndexes(str, substr);

    return 0;

}

**7.**

#include <stdio.h>

#include <string.h>

#define ALPHABET\_SIZE 26

void caesarEncrypt(char \*str, int shift) {

    int length = strlen(str);

    for (int i = 0; i < length; i++) {

        char ch = str[i];

        // Check if the character is an alphabet

        if (isalpha(ch)) {

            // Determine the case (upper or lower)

            int isUpperCase = isupper(ch);

            // Convert the character to lowercase for ease of calculation

            ch = tolower(ch);

            // Apply the Caesar cipher shift

            ch = (ch - 'a' + shift) % ALPHABET\_SIZE + 'a';

            // Convert back to uppercase if the original character was uppercase

            if (isUpperCase) {

                ch = toupper(ch);

            }

        }

        // Update the character in the string

        str[i] = ch;

    }

}

int main() {

    char str[100];

    int shift;

    printf("Enter a string: ");

    fgets(str, sizeof(str), stdin);

    printf("Enter the shift value: ");

    scanf("%d", &shift);

    // Remove the trailing newline character from the string

    if (str[strlen(str) - 1] == '\n') {

        str[strlen(str) - 1] = '\0';

    }

    caesarEncrypt(str, shift);

    printf("Encrypted string: %s\n", str);

    return 0;

}

**8.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#define ALPHABET\_SIZE 26

void caesarEncrypt(char \*str, int shift) {

    int length = strlen(str);

    for (int i = 0; i < length; i++) {

        char ch = str[i];

        // Check if the character is an alphabet

        if (isalpha(ch)) {

            // Determine the case (upper or lower)

            int isUpperCase = isupper(ch);

            // Convert the character to lowercase for ease of calculation

            ch = tolower(ch);

            // Apply the Caesar cipher shift

            ch = (ch - 'a' + shift) % ALPHABET\_SIZE + 'a';

            // Convert back to uppercase if the original character was uppercase

            if (isUpperCase) {

                ch = toupper(ch);

            }

        }

        // Update the character in the string

        str[i] = ch;

    }

}

int main() {

    char str[100];

    int shift, direction;

    int encryptedShift, encryptedDirection;

    printf("Enter a string: ");

    fgets(str, sizeof(str), stdin);

    // Remove the trailing newline character from the string

    if (str[strlen(str) - 1] == '\n') {

        str[strlen(str) - 1] = '\0';

    }

    // Generate random numbers for shift and direction

    srand(time(NULL));

    shift = rand() % ALPHABET\_SIZE;

    direction = rand() % 2;  // 0 for left shift, 1 for right shift

    // Encrypt the string with the random shift and direction

    caesarEncrypt(str, shift);

    if (direction == 0) {

        encryptedShift = shift;

        encryptedDirection = -1;  // Left shift

    } else {

        encryptedShift = ALPHABET\_SIZE - shift;

        encryptedDirection = 1;   // Right shift

    }

    printf("Encrypted string: %s\n", str);

    printf("Shift: %d\n", encryptedShift);

    printf("Direction: %s\n", encryptedDirection == -1 ? "Left" : "Right");

    return 0;

}

**9.**

#include <stdio.h>

int gcd(int a, int b) {

    if (b == 0) {

        return a;

    }

    return gcd(b, a % b);

}

int lcm(int a, int b) {

    return (a \* b) / gcd(a, b);

}

int calculateLCM(int numbers[], int count) {

    int result = numbers[0];

    for (int i = 1; i < count; i++) {

        result = lcm(result, numbers[i]);

    }

    return result;

}

int main() {

    int numbers[10];

    int count;

    printf("Enter the number of elements (max 10): ");

    scanf("%d", &count);

    if (count < 1 || count > 10) {

        printf("Invalid number of elements. Program exiting.\n");

        return 0;

    }

    printf("Enter the elements: ");

    for (int i = 0; i < count; i++) {

        scanf("%d", &numbers[i]);

    }

    int lcmResult = calculateLCM(numbers, count);

    printf("The least common multiple (LCM) is: %d\n", lcmResult);

    return 0;

}

**10.**

#include <stdio.h>

#include <stdbool.h>

bool isBinaryPalindrome(int num) {

    int reverseNum = 0;

    int temp = num;

    while (temp > 0) {

        reverseNum <<= 1;    // Left shift by 1 to make space for the next bit

        reverseNum |= temp & 1;    // Add the rightmost bit of 'temp' to 'reverseNum'

        temp >>= 1;    // Right shift by 1 to move to the next bit in 'temp'

    }

    return (reverseNum == num);

}

int main() {

    int num;

    printf("Enter a number: ");

    scanf("%d", &num);

    if (isBinaryPalindrome(num)) {

        printf("The binary representation of %d is a palindrome.\n", num);

    } else {

        printf("The binary representation of %d is not a palindrome.\n", num);

    }

    return 0;

}

**11.**

#include <stdio.h>

int main() {

    int monthNumber;

    printf("Enter the month number (1-12): ");

    scanf("%d", &monthNumber);

    switch (monthNumber) {

        case 1:

            printf("January\n");

            break;

        case 2:

            printf("February\n");

            break;

        case 3:

            printf("March\n");

            break;

        case 4:

            printf("April\n");

            break;

        case 5:

            printf("May\n");

            break;

        case 6:

            printf("June\n");

            break;

        case 7:

            printf("July\n");

            break;

        case 8:

            printf("August\n");

            break;

        case 9:

            printf("September\n");

            break;

        case 10:

            printf("October\n");

            break;

        case 11:

            printf("November\n");

            break;

        case 12:

            printf("December\n");

            break;

        default:

            printf("Invalid month number.\n");

            break;

    }

    return 0;

}

**12.**

#include <stdio.h>

int main() {

    int digit;

    printf("Enter a digit (0-9): ");

    scanf("%d", &digit);

    switch (digit) {

        case 0:

            printf("Zero\n");

            break;

        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;

        default:

            printf("Invalid digit.\n");

            break;

    }

    return 0;

}

**13.**

#include <stdio.h>

int performOperation(int operand1, int operand2, char operator) {

    switch (operator) {

        case '+':

            return operand1 + operand2;

        case '-':

            return operand1 - operand2;

        case '\*':

            return operand1 \* operand2;

        case '/':

            return operand1 / operand2;

        default:

            printf("Invalid operator\n");

            return 0;

    }

}

int evaluateExpression(char \*expression) {

    int operandStack[100];

    char operatorStack[100];

    int operandTop = -1;

    int operatorTop = -1;

    int i = 0;

    while (expression[i] != '\0') {

        if (expression[i] == ' ') {

            i++;

            continue;

        }

        if (expression[i] >= '0' && expression[i] <= '9') {

            int operand = 0;

            while (expression[i] >= '0' && expression[i] <= '9') {

                operand = (operand \* 10) + (expression[i] - '0');

                i++;

            }

            operandStack[++operandTop] = operand;

        } else if (expression[i] == '+' || expression[i] == '-' || expression[i] == '\*' || expression[i] == '/') {

            while (operatorTop >= 0 && (operatorStack[operatorTop] == '\*' || operatorStack[operatorTop] == '/')) {

                int operand2 = operandStack[operandTop--];

                int operand1 = operandStack[operandTop--];

                char operator = operatorStack[operatorTop--];

                int result = performOperation(operand1, operand2, operator);

                operandStack[++operandTop] = result;

            }

            operatorStack[++operatorTop] = expression[i];

            i++;

        } else {

            printf("Invalid character in expression\n");

            return 0;

        }

    }

    while (operatorTop >= 0) {

        int operand2 = operandStack[operandTop--];

        int operand1 = operandStack[operandTop--];

        char operator = operatorStack[operatorTop--];

        int result = performOperation(operand1, operand2, operator);

        operandStack[++operandTop] = result;

    }

    return operandStack[operandTop];

}

int main() {

    char expression[100];

    printf("Enter an arithmetic expression: ");

    fgets(expression, sizeof(expression), stdin);

    int result = evaluateExpression(expression);

    printf("Result: %d\n", result);

    return 0;

}

**THANKS!**

**THE END!**