1.

Title: Program to Recognize and Count Vowels and Consonants in a Given String using Lexical Analysis

Procedure

Lexical analysis is the process of scanning an input string and dividing it into meaningful components called tokens.
In this program, we perform lexical analysis on the given string to identify whether each character is a vowel, consonant, or other symbol.

Steps followed:

1. Start the program.
2. Take an input string from the user.
3. Initialize two counters: vowelCount and consonantCount.
4. Traverse each character of the string:
    - If the character is a vowel (a, e, i, o, u — both lowercase and uppercase), increment vowelCount.
    - Else if the character is an alphabet and not a vowel, increment consonantCount.
    - Ignore spaces, digits, and special characters.
5. Display the total number of vowels and consonants.
6. Stop the program.

Program

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

int main() {
   char str[200];
   int vowelCount = 0, consonantCount = 0;
   int i;

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


   for (i = 0; str[i] != '\0'; i++) {
      char ch = tolower(str[i]);

      if (ch >= 'a' && ch <= 'z') {
               if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
            vowelCount++;
         else
            consonantCount++;
      }
   }
```

```
    printf("\n===== Result =====\n");
    printf("Input String: %s", str);
    printf("Total Vowels     : %d\n", vowelCount);
    printf("Total Consonants : %d\n", consonantCount);

    return 0;
}
```

Output

Input:

```
Enter a string: Lexical Analysis is Fun!
```

Output:

```
===== Result =====
Input String: Lexical Analysis
Total Vowels     : 6
Total Consonants : 9
```

2.

Title: Program to Recognize and Count Uppercase and Lowercase Letters in a Given String

Procedure

This program reads an input string from the user and analyzes it to count the number of uppercase and lowercase letters.

Steps followed:

1. Start the program.
2. Take an input string from the user.
3. Initialize two counters: uppercaseCount and lowercaseCount.
4. Traverse each character of the string:
    - If the character lies between 'A' and 'Z', increment uppercaseCount.
    - Else if the character lies between 'a' and 'z', increment lowercaseCount.
    - Ignore digits, spaces, and special characters.
5. Display the total number of uppercase and lowercase letters.
6. Stop the program.

Program

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

int main() {
   char str[200];
   int uppercaseCount = 0, lowercaseCount = 0;
   int i;

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

   // Traverse the string
   for (i = 0; str[i] != '\0'; i++) {
      if (str[i] >= 'A' && str[i] <= 'Z') {
         uppercaseCount++;
      }
      else if (str[i] >= 'a' && str[i] <= 'z') {
         lowercaseCount++;
      }
   }

   // Output the result
   printf("\n===== Result =====\n");
   printf("Input String       : %s", str);
   printf("Total Uppercase    : %d\n", uppercaseCount);
```

```
    printf("Total Lowercase    : %d\n", lowercaseCount);

    return 0;
}
```

Output

Input:

```
Enter a string: Lexical Analysis Is Fun!
```

Output:

```
===== Result =====
Input String       : Lexical Analysis Is Fun!
Total Uppercase    : 4
Total Lowercase    : 16
```

3.

Title: Program to Recognize and Count Tokens from an Input File

Procedure

A token is the smallest meaningful unit in a program or sentence.
Tokens can be keywords, identifiers, operators, numbers, strings, or special characters.

In this program, we will:

- Read an input file.
- Extract tokens by separating words based on spaces, tabs, newlines, and punctuation.
- Count the total number of tokens.

Steps Followed:

1. Start the program.
2. Open the input file in read mode.
3. Check if the file exists; if not, display an error message.
4. Read the file line by line.
5. Use a delimiter-based splitting method to separate tokens (spaces, tabs, and punctuation).
6. Count the total number of tokens.
7. Display all tokens and the total token count.
8. Close the file and end the program.

Program

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

int main() {
   FILE *fp;
   char filename[100], buffer[1000];
   char *token;
   int tokenCount = 0;

   printf("Enter the file name: ");
   scanf("%s", filename);

   fp = fopen(filename, "r");
   if (fp == NULL) {
      printf("Error: Cannot open file %s\n", filename);
      return 1;
   }
```

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

    while (fgets(buffer, sizeof(buffer), fp)) {
        token = strtok(buffer, " ,.;:\n\t!?(){}[]");
        while (token != NULL) {
            printf("%s\n", token);
            tokenCount++;
            token = strtok(NULL, " ,.;:\n\t!?(){}[]");
        }
    }

    printf("\nTotal Tokens: %d\n", tokenCount);

    fclose(fp);
    return 0;
}
```

Output

Sample Input File (input.txt)

```
Hello World!
Lexical analysis is fun.
Let's count tokens from this file.
```

Program Execution

Input:

```
Enter the file name: input.txt
```

Output:

```
===== Tokens Found =====
Hello
World
Lexical
analysis
is
fun
Let's
count
tokens
from
this
file

Total Tokens: 11
```

4.

Title: Program to Recognize a String Ending with "aa"

Procedure

This program checks whether a given input string ends with the substring "aa".

Steps Followed:

1. Start the program.
2. Take an input string from the user.
3. Find the length of the string.
4. Check the last two characters:
    - If both are 'a', then the string ends with "aa".
    - Otherwise, the string does not end with "aa".
5. Display the result.
6. Stop the program.

Program

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

int main() {
  char str[100];
  int len;

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

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

  // Find length
  len = strlen(str);

  // Check if string ends with "aa"
  if (len >= 2 && str[len - 1] == 'a' && str[len - 2] == 'a') {
    printf("\nThe string \"%s\" ends with \"aa\".\n", str);
  } else {
    printf("\nThe string \"%s\" does NOT end with \"aa\".\n", str);
  }

  return 0;
}
```

Output

Sample Input & Output

Case 1: When the string ends with "aa"
Input:

```
Enter a string: BUBTaa
```

Output:

```
The string "BUBTaa" ends with "aa".
```

Case 2: When the string does NOT end with "aa"
Input:

```
Enter a string: Lexical
```

Output:

```
The string "Lexical" does NOT end with "aa".
```

5.

Title: Program to Recognize a String Starting and Ending with "bb"

Procedure

This program checks whether a given input string starts and ends with the substring "bb".

Steps Followed:

1. Start the program.
2. Take an input string from the user.
3. Find the length of the string.
4. Check the first two characters and the last two characters:
   - If both the start and end have 'b' and 'b', the string is valid.
   - Otherwise, it is not valid.
5. Display the result.
6. Stop the program.

Program

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

int main() {
    char str[100];
    int len;

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

    str[strcspn(str, "\n")] = '\0';

    len = strlen(str);

    if (len >= 4 && str[0] == 'b' && str[1] == 'b' && str[len - 2] == 'b' && str[len - 1] == 'b') {
        printf("\nThe string \"%s\" starts and ends with \"bb\".\n", str);
    } else {
        printf("\nThe string \"%s\" does NOT start and end with \"bb\".\n", str);
    }

    return 0;
}
```

Output

Sample Input & Output

Case 1: When the string starts and ends with "bb"
Input:

```
Enter a string: bbcollegebb
```

Output:

```
The string "bbcollegebb" starts and ends with "bb".
```

Case 2: When the string does NOT start and end with "bb"
Input:

```
Enter a string: lexicalbb
```

Output:

```
The string "lexicalbb" does NOT start and end with "bb".
```

6.

Title: Program to Recognize a Number Having '1' in the 3rd Position and '9' in the 7th Position

Procedure

This program checks whether a given number contains '1' in the 3rd position (from the left) and '9' in the 7th position.

Steps Followed:

1. Start the program.
2. Take a number as input from the user (as a string for easy position checking).
3. Find the length of the number.
4. Check if the length is at least 7 digits.
5. Verify:
    - If the 3rd digit (index 2) is '1'
    - And the 7th digit (index 6) is '9'
6. If both conditions are true → print valid number.
7. Otherwise → print invalid number.
8. Stop the program.

Program

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

int main() {
  char number[100];
  int len;

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

  len = strlen(number);

  if (len >= 7) {
    if (number[2] == '1' && number[6] == '9') {
      printf("\nThe number %s is VALID.\n", number);
      printf("It has '1' at 3rd position and '9' at 7th position.\n");
    } else {
      printf("\nThe number %s is INVALID.\n", number);
      printf("It does not meet the required position criteria.\n");
    }
  } else {
    printf("\nThe number must have at least 7 digits.\n");
  }
```

```
    return 0;
}
```

Output

Sample Input & Output

Case 1: When the number is valid ☐
Input:

```
Enter a number: 451123987
```

Output:

```
The number 451123987 is VALID.
It has '1' at 3rd position and '9' at 7th position.
```

Case 2: When the number is invalid ☐
Input:

```
Enter a number: 783457123
```

Output:

```
The number 783457123 is INVALID.
It does not meet the required position criteria.
```

Case 3: When the number has less than 7 digits ☐
Input:

```
Enter a number: 45123
```

Output:

```
The number must have at least 7 digits.
```

7.

Title: Program to Count the Number of Characters, Words, Spaces, and End-of-Lines in a Given Input File

Procedure

This program reads an input file and counts:

- Characters → All alphabets, digits, and symbols except newline
- Words → Groups of characters separated by spaces or newlines
- Spaces → All blank spaces in the file
- End-of-Lines → Number of newline characters (\n)

Steps Followed:

1. Start the program.
2. Open the input file in read mode.
3. Check if the file exists; if not, display an error message.
4. Initialize counters: charCount, wordCount, spaceCount, and lineCount.
5. Read the file character by character:
    - If the character is a space → increment spaceCount.
    - If the character is a newline → increment lineCount and wordCount if the previous character wasn't a space.
    - If the character is not space or newline → increment charCount.
    - Count words when a space, newline, or EOF is reached.
6. Display the final counts.
7. Close the file and stop the program.

Program

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

int main() {
  FILE *fp;
  char filename[100];
  char ch;
  int charCount = 0, wordCount = 0, spaceCount = 0, lineCount = 0;
  int inWord = 0;

  printf("Enter the file name: ");
  scanf("%s", filename);

  fp = fopen(filename, "r");
  if (fp == NULL) {
    printf("Error: Cannot open file %s\n", filename);
    return 1;
  }

  while ((ch = fgetc(fp)) != EOF) {
    if (ch != '\n')
        charCount++;

    if (ch == ' ')
        spaceCount++;

    if (ch == '\n')
        lineCount++;

    if (isspace(ch)) {
        inWord = 0;
    } else if (inWord == 0) {
        inWord = 1;
        wordCount++;
    }
  }

  fclose(fp);

  printf("\n===== File Statistics =====\n");
  printf("Total Characters : %d\n", charCount);
  printf("Total Words      : %d\n", wordCount);
  printf("Total Spaces     : %d\n", spaceCount);
  printf("Total Lines      : %d\n", lineCount);

  return 0;
}
```

4. Output

Sample Input File (input.txt)

Input:

```
Lexical analysis is fun.
We are learning file handling in C.
Counting characters, words, and lines!
```

Program Execution

```
Enter the file name: input.txt
```

Output:

```
===== File Statistics =====
Total Characters : 86
Total Words      : 13

Total Spaces     : 12

Total Lines      : 3
```

8.

Title: Program to Count the Number of Comment Lines in a Given C Program

Procedure

This program reads a C program file and counts the total number of comment lines.
In C, there are two types of comments:

1. Single-line comments → Start with //
2. Multi-line comments → Start with /* and end with */

Steps Followed:

1. Start the program.
2. Open the C source code file in read mode.
3. Initialize a counter commentCount to 0.
4. Read the file character by character.
5. Check for:
    - // → Count as a single-line comment and read until a newline.
    - /* ... */ → Count all lines inside as multi-line comments.
6. Display the total number of comment lines.
7. Close the file and end the program.

Program

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

int main() {
  FILE *fp;
  char filename[100];
  char ch;
  int commentCount = 0;

  printf("Enter the C program file name: ");
  scanf("%s", filename);

  fp = fopen(filename, "r");
  if (fp == NULL) {
    printf("Error: Cannot open file %s\n", filename);
    return 1;
  }

  while ((ch = fgetc(fp)) != EOF) {
    if (ch == '/') {
      char next = fgetc(fp);
```

```
            if (next == '/') {
                commentCount++;
                while ((ch = fgetc(fp)) != '\n' && ch != EOF);
            }
            else if (next == '*') {
                commentCount++;
                while ((ch = fgetc(fp)) != EOF) {
                    if (ch == '\n') commentCount++; // count each line inside comment
                    if (ch == '*') {
                        char end = fgetc(fp);
                        if (end == '/')
                            break;
                    }
                }
            }
        }
    }

    fclose(fp);

    printf("\n===== Result =====\n");
    printf("Total Comment Lines: %d\n", commentCount);

    return 0;
}
```

Output

Sample Input File (sample.c)

```
#include <stdio.h>

// This is a single-line comment

int main() {
    /* This is a
       multi-line comment */
    printf("Hello World\n");  // Prints a
message
    return 0;
}
```

Program Execution

Input:

```
Enter the C program file name: sample.c
```

Output:

```
===== Result =====
Total Comment Lines: 4
```

9.

Title: Program to Recognize Whether a Given Sentence is Simple or Compound

Procedure

In English grammar, a simple sentence contains only one independent clause, while a compound sentence contains two or more independent clauses, usually connected by conjunctions such as:

Conjunctions: and, but, or, so, yet, nor, for.

Approach:

- Take a sentence as input.
- Search for coordinating conjunctions within the sentence.
- If any conjunction is found → classify as compound.
- Otherwise → classify as simple.

Steps Followed:

1. Start the program.
2. Take an input sentence from the user.
3. Create a list of coordinating conjunctions.
4. Scan the sentence for any matching conjunction.
5. If a match is found → print compound sentence.
6. Otherwise → print simple sentence.
7. Stop the program.

Program

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

int main() {
   char sentence[500];
   char *conjunctions[] = {"and", "but", "or", "so", "yet", "nor", "for"};
   int i, isCompound = 0;

   printf("Enter a sentence: ");
   fgets(sentence, sizeof(sentence), stdin);

   sentence[strcspn(sentence, "\n")] = '\0';

   for (i = 0; sentence[i]; i++) {
      sentence[i] = tolower(sentence[i]);
   }
```

```
    for (i = 0; i < 7; i++) {
        if (strstr(sentence, conjunctions[i]) != NULL) {
            isCompound = 1;
            break;
        }
    }

    printf("\n===== Result =====\n");
    printf("Input Sentence: %s\n", sentence);
    if (isCompound)
        printf("Sentence Type: Compound Sentence\n");
    else
        printf("Sentence Type: Simple Sentence\n");

    return 0;
}
```

Output

Case 1: Simple Sentence

Input:

```
Enter a sentence: I love programming.
```

Output:

```
===== Result =====
Input Sentence: i love programming.
Sentence Type: Simple Sentence
```

Case 2: Compound Sentence

Input:

```
Enter a sentence: I love programming and I enjoy solving problems.
```

Output:

```
===== Result =====
Input Sentence: i love programming and i enjoy solving problems.
Sentence Type: Compound Sentence
```

10.

Title: Program to Recognize and Count the Number of Identifiers in a Given Input File

Procedure

1. Objective
   To write a program that reads a source code file and identifies all the valid identifiers, then counts their total number.
2. Concept
   - Identifiers in C/C++ are names used for variables, functions, arrays, classes, etc.
   - Rules for valid identifiers:
     - Must start with a letter (A-Z, a-z) or underscore (_).
     - Can contain letters, digits (0-9), and underscores.
     - Cannot be a reserved keyword.
     - Cannot start with a digit.
   - Example of valid identifiers:
     sum, _value1, counter_2025
     Example of invalid identifiers:
     1num, float, @value
3. Approach
   - Open the input file in read mode.
   - Read the file word by word.
   - For each word:
     - Check if it's a keyword → skip.
     - Check if it follows identifier rules.
     - If valid → count it.
   - Finally, display all identifiers and their total count.
4. Tools Used
   - Language: C
   - Compiler: GCC / MinGW / Turbo C
   - Input: Text file containing C/C++ code.

Program

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

#define MAX 50

// List of C keywords
const char *keywords[] = {
   "auto","break","case","char","const","continue","default","do","double","else",
   "enum","extern","float","for","goto","if","int","long","register","return",
   "short","signed","sizeof","static","struct","switch","typedef","union",
   "unsigned","void","volatile","while"
};
```

```
int isKeyword(const char *word) {
    for (int i = 0; i < 32; i++) {
        if (strcmp(word, keywords[i]) == 0)
            return 1;
    }
    return 0;
}

int isIdentifier(const char *word) {
    if (!(isalpha(word[0]) || word[0] == '_'))
        return 0; // Must start with letter or underscore

    for (int i = 1; word[i] != '\0'; i++) {
        if (!(isalnum(word[i]) || word[i] == '_'))
            return 0;
    }
    return !isKeyword(word);
}

int main() {
    FILE *fp;
    char word[MAX];
    int count = 0;

    // Open input file
    fp = fopen("input.c", "r");
    if (fp == NULL) {
        printf("Error: Cannot open input file!\n");
        return 1;
    }

    printf("Identifiers found:\n");

    // Read words from the file
    while (fscanf(fp, "%s", word) != EOF) {
        if (isIdentifier(word)) {
            printf("%s\n", word);
            count++;
        }
    }

    printf("\nTotal Number of Identifiers = %d\n", count);

    fclose(fp);
    return 0;
}
```

Output

Input File (input.c)

```
#include <stdio.h>
int main() {
   int num1 = 10, num2 = 20, sum;
   sum = num1 + num2;
   printf("Sum = %d\n", sum);
   return 0;
}
```

Program Execution

```
Identifiers found:
main
num1
num2
sum
printf

Total Number of Identifiers = 5
```