LAB MANUAL

Compiler Design Laboratory

(CSE606)

By

Pearl Agnihotri (22000405)

Third Year, Semester 6

Course Incharge: Prof. Vaibhavi Patel

NAVRACHANA UNIVERSITY

a UGC recognized University

Department of Computer Science and Engineering

School of Engineering and Technology

Navrachana University, Vadodara

Spring Semester

Jan – May, 2025

TABLE OF CONTENT

Experiment - 1

Aim:

a) Write a program to recognize strings starts with ‘a’ over {a, b}.
b) Write a program to recognize strings end with ‘a’.
c) Write a program to recognize strings end with ‘ab’. Take the input from text file.
d) Write a program to recognize strings contains ‘ab’. Take the input from text file.

Code:

a)

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

#define MAX_LENGTH 100

// Function to recognize strings that start with 'a'
void recognizeString(const char *string) {
    if (string[0] == 'a') {
        printf("The string '%s' starts with 'a'.\n", string);
    } else {
        printf("The string '%s' does not start with 'a'.\n", string);
    }
}

int main() {
    char string[MAX_LENGTH];

    printf("Enter words to check if they start with 'a' (type 'exit' to
stop):\n");

    while (1) {
        // Input a string from the user
        printf("Enter a word: ");
        fgets(string, MAX_LENGTH, stdin);

        // Remove trailing newline if present
        size_t len = strlen(string);
        if (len > 0 && string[len - 1] == '\n') {
            string[len - 1] = '\0';
        }

        // Check if the user wants to exit
        if (strcmp(string, "exit") == 0) {
            break;
        }

        // Recognize the string
        recognizeString(string);
    }

    printf("Program terminated.\n");
    return 0;
}
```

b)

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

#define MAX_LENGTH 100

// Function to recognize strings that end with 'a'
void recognizeString(const char *string) {
    size_t len = strlen(string);
    if (len > 0 && string[len - 1] == 'a') {
        printf("The string '%s' ends with 'a'.\n", string);
    } else {
        printf("The string '%s' does not end with 'a'.\n", string);
    }
}

int main() {
    char string[MAX_LENGTH];

    printf("Enter words to check if they end with 'a' (type 'exit' to
stop):\n");

    while (1) {
        // Input a string from the user
        printf("Enter a word: ");
        fgets(string, MAX_LENGTH, stdin);

        // Remove trailing newline if present
        size_t len = strlen(string);
        if (len > 0 && string[len - 1] == '\n') {
            string[len - 1] = '\0';
        }

        // Check if the user wants to exit
        if (strcmp(string, "exit") == 0) {
            break;
        }

        // Recognize the string
        recognizeString(string);
    }

    printf("Program terminated.\n");
    return 0;
}
```

c)

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

#define MAX_LINE_LENGTH 100

void checkStringsEndingWithAb(const char *filePath) {
    FILE *file = fopen(filePath, "r");
    if (file == NULL) {
        printf("Error: Could not open file '%s'\n", filePath);
        return;
    }

    char line[MAX_LINE_LENGTH];
```

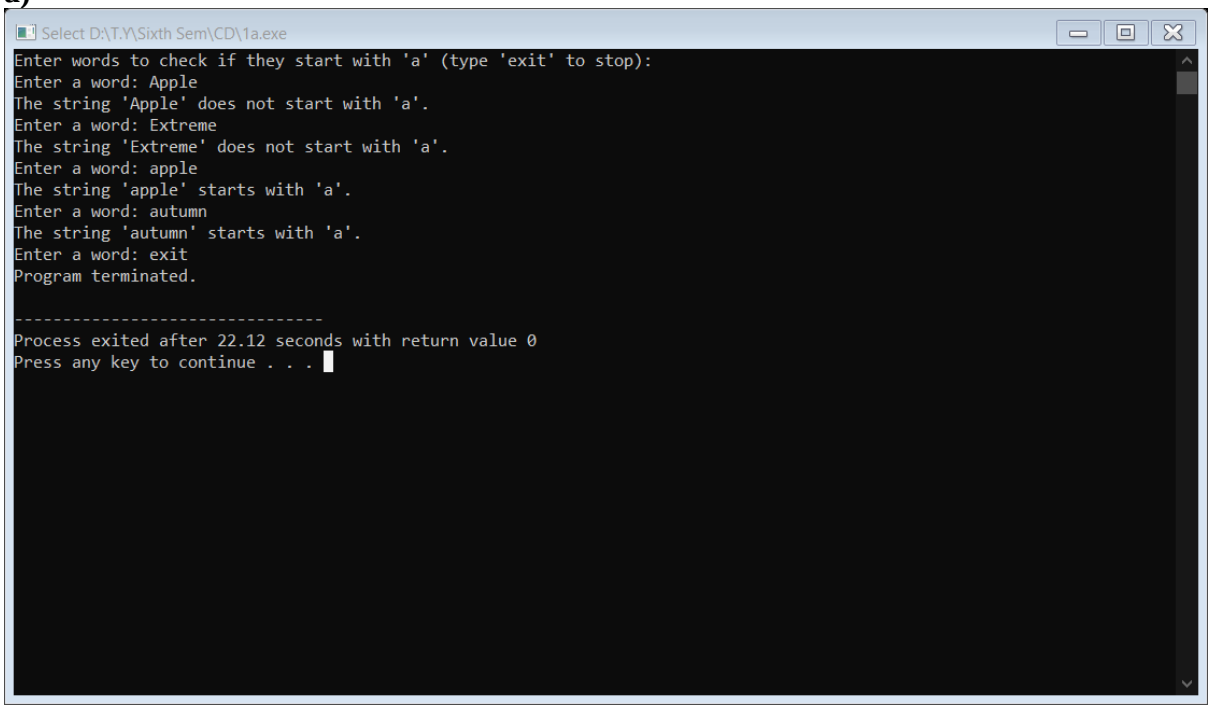
```
    printf("Strings that end with 'ab':\n");
    while (fgets(line, sizeof(line), file) != NULL) {
        // Remove trailing newline character if present
        size_t len = strlen(line);
        if (len > 0 && line[len - 1] == '\n') {
            line[len - 1] = '\0';
        }

        // Check if the string ends with "ab"
        len = strlen(line);
        if (len >= 2 && line[len - 2] == 'a' && line[len - 1] == 'b') {
            printf("%s\n", line);
        }
    }

    fclose(file);
}

int main() {
    char filePath[100];
    printf("Enter the path to the text file: ");
    scanf("%s", filePath);

    checkStringsEndingWithAb(filePath);

    return 0;
}
```

d)

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

#define MAX_LINE_LENGTH 1000

// Function to check if a string contains 'ab'
int contains_ab(const char *str) {
    return strstr(str, "ab") != NULL;  // Returns true if 'ab' is found
}

int main() {
    FILE *file;
    char line[MAX_LINE_LENGTH];

    // Open the text file for reading
    file = fopen("TrainingforC.txt", "r"); // Replace "input.txt" with your
file name
    if (file == NULL) {
        printf("File not found.\n");
        return 1;
    }

    // Read the file line by line
    while (fgets(line, sizeof(line), file)) {
        if (contains_ab(line)) {
            printf("Line containing 'ab': %s", line);
        }
    }

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

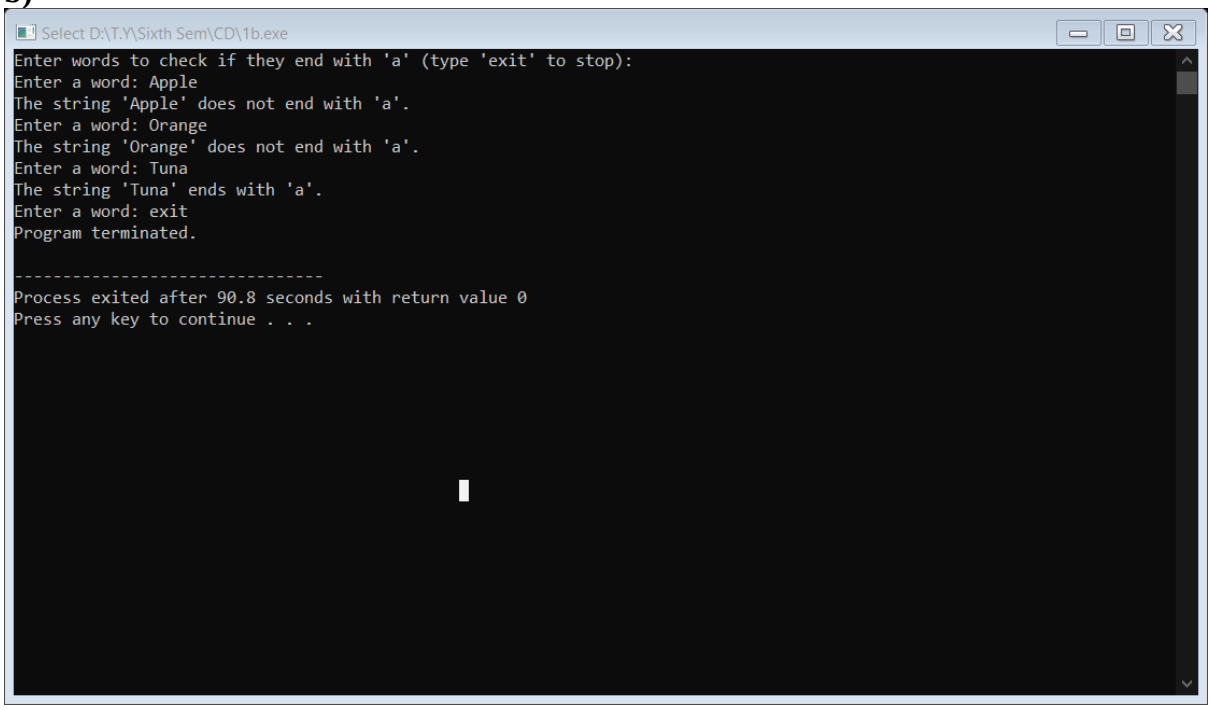
Output:

a)


Select D:\T.Y\Sixth Sem\CD\1a.exe
Enter words to check if they start with 'a' (type 'exit' to stop):
Enter a word: Apple
The string 'Apple' does not start with 'a'.
Enter a word: Extreme
The string 'Extreme' does not start with 'a'.
Enter a word: apple
The string 'apple' starts with 'a'.
Enter a word: autumn
The string 'autumn' starts with 'a'.
Enter a word: exit
Program terminated.

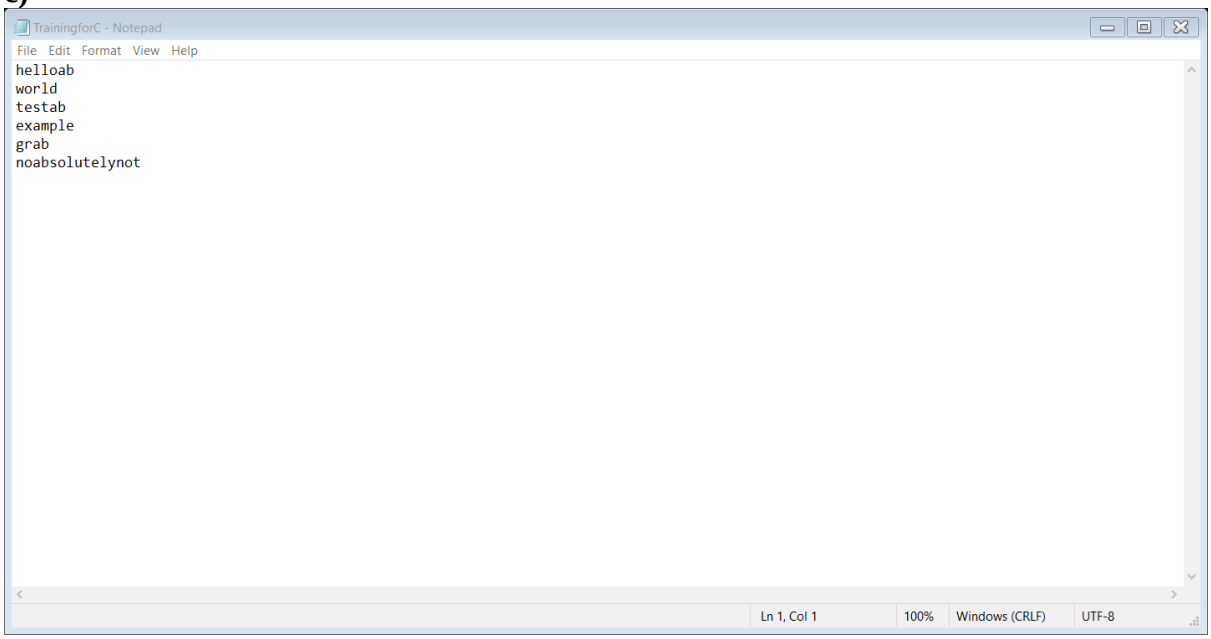
Process exited after 22.12 seconds with return value 0
Press any key to continue . . .

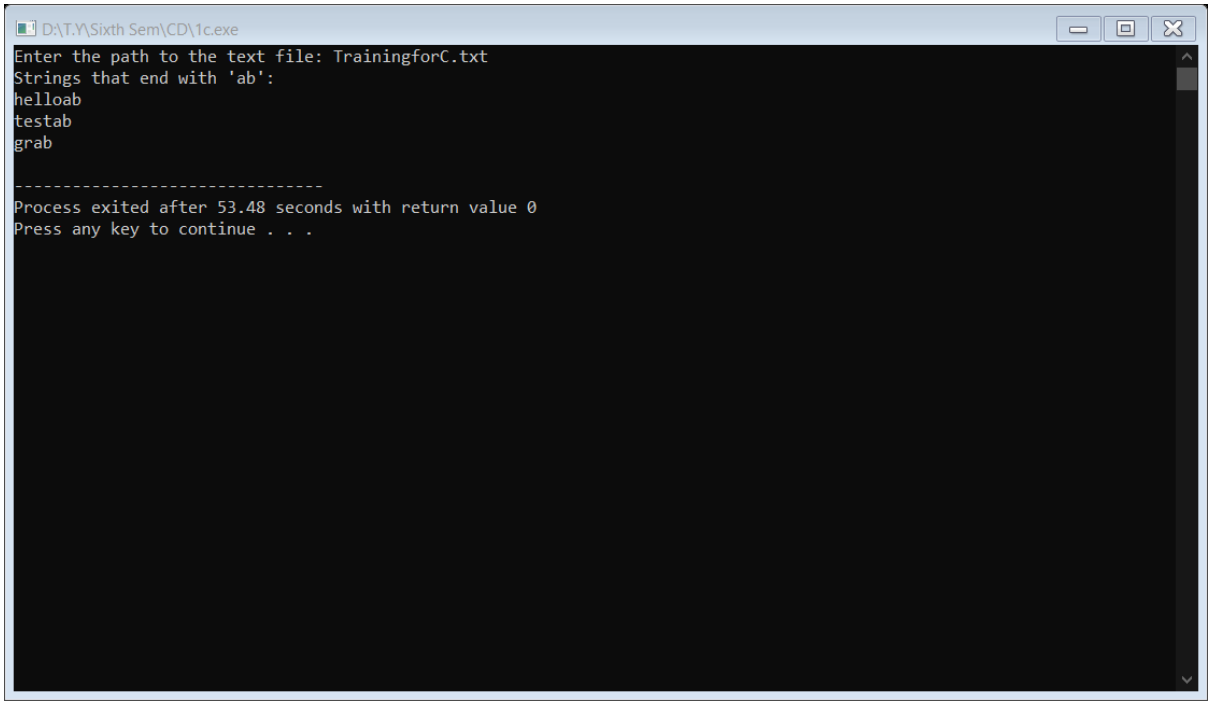
b)


Select D:\T.Y\Sixth Sem\CD\1b.exe
Enter words to check if they end with 'a' (type 'exit' to stop):
Enter a word: Apple
The string 'Apple' does not end with 'a'.
Enter a word: Orange
The string 'Orange' does not end with 'a'.
Enter a word: Tuna
The string 'Tuna' ends with 'a'.
Enter a word: exit
Program terminated.

Process exited after 90.8 seconds with return value 0
Press any key to continue . . .

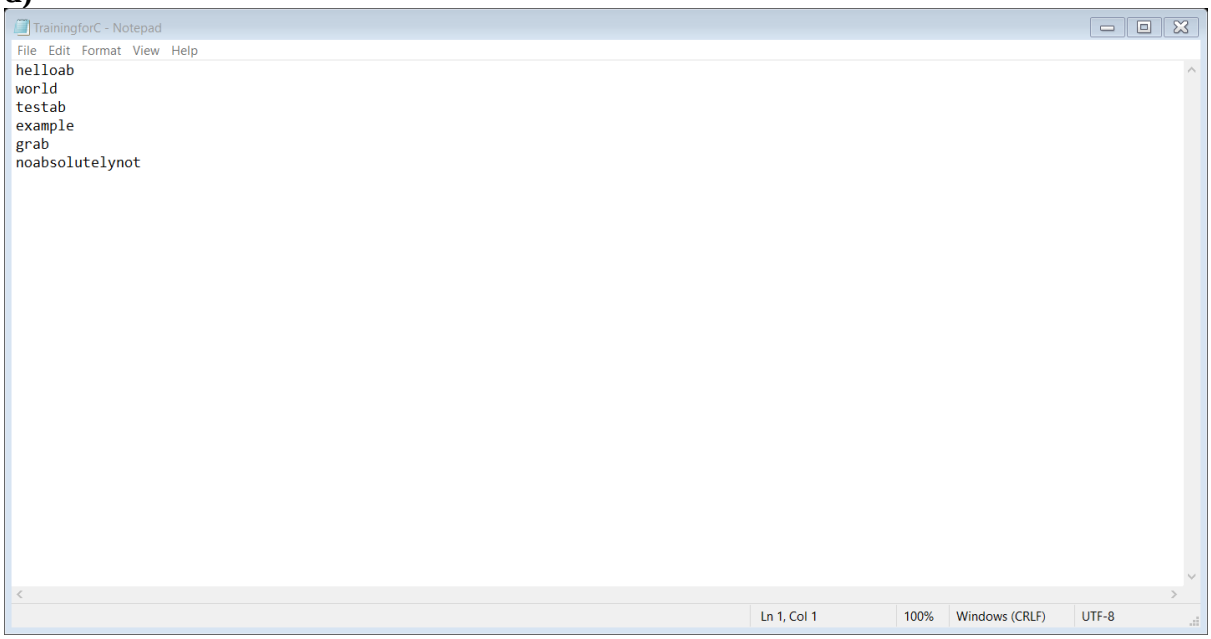
c)

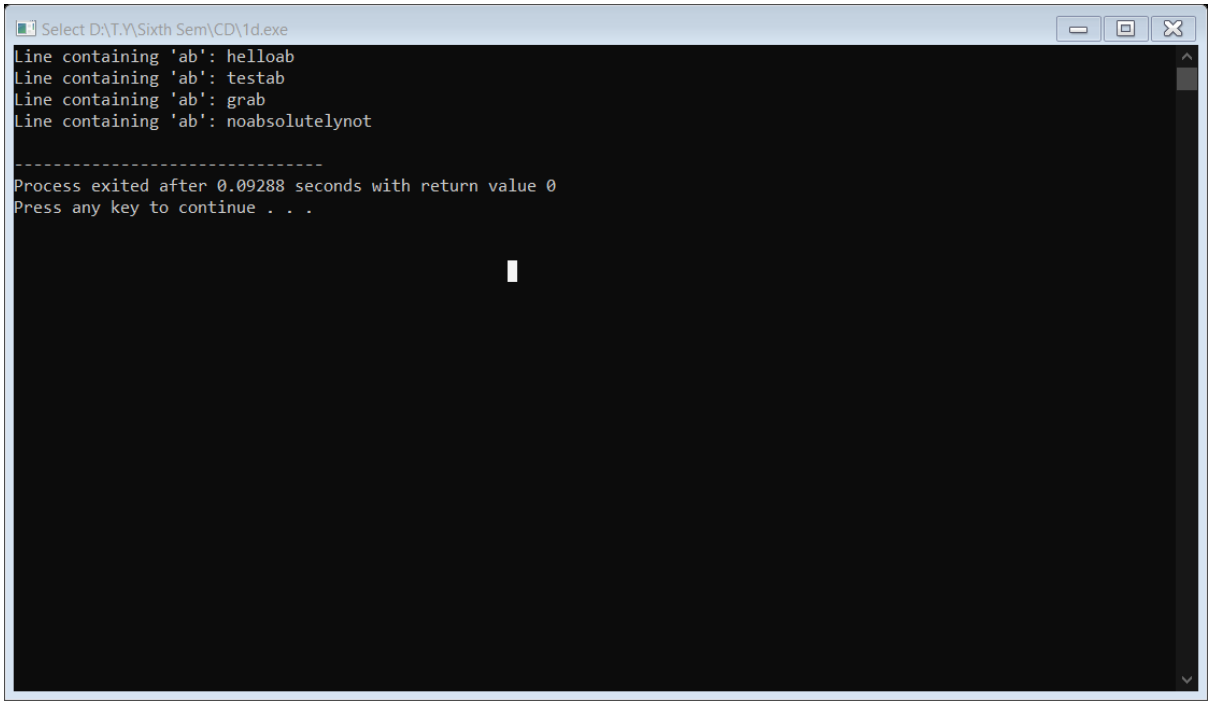

TrainingforC - Notepad
File Edit Format View Help
helloab
world
testab
example
grab
noabsolutelynot
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8


D:\T.Y\Sixth Sem\CD\1c.exe
Enter the path to the text file: TrainingforC.txt
Strings that end with 'ab':
helloab
testab
grab

Process exited after 53.48 seconds with return value 0
Press any key to continue . . .

d)


TrainingforC - Notepad
File Edit Format View Help
helloab
world
testab
example
grab
noabsolutelynot
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8


Select D:\T.Y\Sixth Sem\CD\1d.exe
Line containing 'ab': helloab
Line containing 'ab': testab
Line containing 'ab': grab
Line containing 'ab': noabsolutelynot

Process exited after 0.09288 seconds with return value 0
Press any key to continue . . .

Experiment - 2

Aim:

a) Write a program to recognize the valid identifiers and keywords.
b) Write a program to recognize the valid operators.
c) Write a program to recognize the valid number.
d) Write a program to recognize the valid comments.
e) Program to implement Lexical Analyzer.

Code:

a)

```
#include <stdio.h>

#include <ctype.h>

#include <string.h>


// Keywords list

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"

};

const int num_keywords = sizeof(keywords) / sizeof(keywords[0]);


// Check if string is a keyword

int is_keyword(const char *str) {

    int i;  // Declare i outside the loop (C89 Fix)

    for (i = 0; i < num_keywords; i++) {

        if (strcmp(str, keywords[i]) == 0)

            return 1; // It is a keyword

    }

    return 0;

}


// Check if string is a valid identifier
```

```
void check_identifier(const char *str) {
    int state = 0;
    int i;  // Declare i outside the loop (C89 Fix)

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

        switch (state) {
            case 0: // Start state
                if (isalpha(c) || c == '_')
                    state = 1; // Valid identifier
                else {
                    state = -1;
                    break;
                }
            case 1: // Valid identifier state
                if (!(isalnum(c) || c == '_'))
                    state = -1;
                break;
        }
        if (state == -1) break; // Exit loop if invalid
    }

    // Final state check
    switch (state) {
        case 1:
            printf("\"%s\" is %s\n", str, is_keyword(str) ? "a Keyword" :
"a Valid Identifier");
            break;
        default:
            printf("\"%s\" is an Invalid Identifier\n", str);
    }
}

// Main function
```

```
int main() {
    char input[50];

    printf("Enter a string (Type 'exit' to stop):\n");

    while (1) {
        printf("\nInput: ");
        scanf("%49s", input);

        if (strcmp(input, "exit") == 0) {
            printf("Exiting program.\n");
            break;
        }

        check_identifier(input);
    }

    return 0;
}
```

b)

```
//to recognize the valid operators
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

int main() {
    char input[50];
    const char *validOperators[] = {
        "+", "-", "*", "/", "%",  // Arithmetic
        "=", "+=", "-=", "*=", "/=", "%=", // Assignment
        "==", "!=", ">", "<", ">=", "<=", // Relational
```

```
    "&&", "||", "!", // Logical
    "&", "|", "^", "~", "<<", ">>", // Bitwise
    "++", "--", // Increment/Decrement
    ",", ".", "->", // Structure/Union member access
    "(", ")", "[", "]", "{", "}", // Parentheses, brackets, braces
    "?", ":", // Ternary operator
    "sizeof", // Unary operator
    "->", "." // Pointer-to-member operators (less common)
};

int numOperators = sizeof(validOperators) / sizeof(validOperators[0]);

printf("Enter a potential C operator (or 'exit' to quit): ");

while (1) {
    scanf("%49s", input);

    if (strcmp(input, "exit") == 0) {
        break;
    }

    bool found = false;
    int i = 0; // Initialize loop counter
    while (i < numOperators) { // While loop
        switch (strcmp(input, validOperators[i])) { // Switch statement
            case 0: // Match found
                found = true;
                i = numOperators; // A way to break the while loop
```

```
                    break;

                default: // No match, go to next operator

                    i++;

                    break;

            }

        }



        if (found) {

            printf("\"%s\" is a valid C operator.\n", input);

        } else {

            printf("\"%s\" is NOT a valid C operator.\n", input);

        }



        printf("Enter another operator (or 'exit' to quit): ");

    }



    printf("Exiting.\n");

    return 0;

}
```

c)

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

void check_valid_number(char *input) {
    int state = 0, i = 0;
    char lexeme[100];

    while (input[i] != '\0') {
        char c = input[i];

        switch (state) {
            case 0:
                if (isdigit(c)) {
                    state = 1;  // Transition to integer state
```

```
        } else if (c == '.') {
            state = 2;  // Starts with a dot, expecting digits
        } else {
            printf("Invalid number: %s\n", input);
            return;
        }
        break;

    case 1:  // Integer state
        if (isdigit(c)) {
            state = 1;
        } else if (c == '.') {
            state = 3;  // Transition to decimal part
        } else if (c == 'E' || c == 'e') {
            state = 5;  // Transition to exponent part
        } else {
            printf("%s is a valid number\n", input);
            return;
        }
        break;

    case 2:  // Starts with a dot
        if (isdigit(c)) {
            state = 3;
        } else {
            printf("Invalid number: %s\n", input);
            return;
        }
        break;

    case 3:  // Decimal part
        if (isdigit(c)) {
            state = 3;
        } else if (c == 'E' || c == 'e') {
            state = 5;
        } else {
            printf("%s is a valid number\n", input);
            return;
        }
        break;

    case 5:  // Exponent part
        if (c == '+' || c == '-') {
            state = 6;
        } else if (isdigit(c)) {
            state = 7;
        } else {
            printf("Invalid number: %s\n", input);
            return;
        }
        break;

    case 6:  // Sign after exponent
        if (isdigit(c)) {
            state = 7;
        } else {
            printf("Invalid number: %s\n", input);
            return;
        }
        break;
```

```
            case 7:  // Digits after exponent
                if (isdigit(c)) {
                    state = 7;
                } else {
                    printf("%s is a valid number\n", input);
                    return;
                }
                break;
        }
        i++;
    }

    // If loop exits normally, check if we ended in a valid state
    if (state == 1 || state == 3 || state == 7) {
        printf("%s is a valid number\n", input);
    } else {
        printf("Invalid number: %s\n", input);
    }
}

int main() {
    char input[100];

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

    check_valid_number(input);

    return 0;
}
```

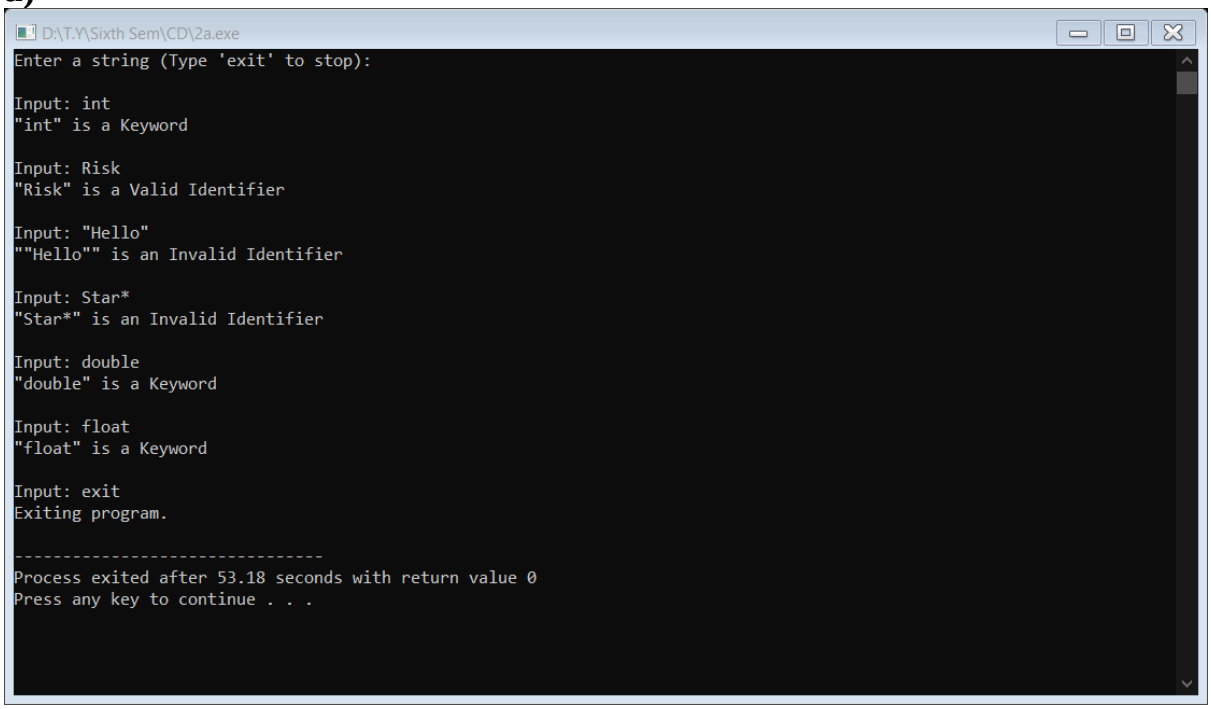
d)

```
#include <stdio.h>

#include <string.h>


void check_comment(const char *str) {

    int len = strlen(str);


    // Single-line comment check

    if (len >= 2 && str[0] == '/' && str[1] == '/') {

        printf("Valid Single-line Comment\n");

        return;

    }


    // Multi-line comment check

    if (len >= 4 && str[0] == '/' && str[1] == '*' && str[len - 2] == '*'
&& str[len - 1] == '/') {

        printf("Valid Multi-line Comment\n");
```

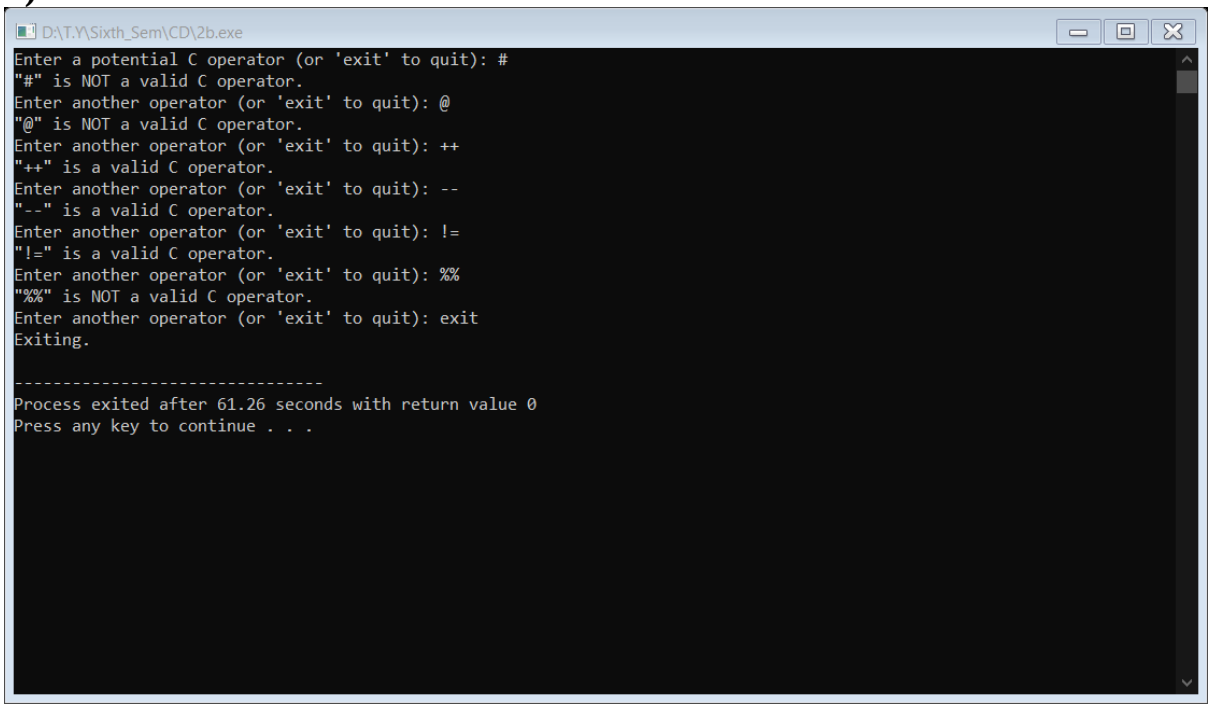
```
        return;
    }

    printf("Not a Valid Comment\n");
}

int main() {
    char input[100];

    printf("Enter a comment to check (Type 'exit' to stop):\n");

    while (1) {
        printf("\nInput: ");
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = 0; // Remove newline character

        if (strcmp(input, "exit") == 0) {
            printf("Exiting program.\n");
            break;
        }

        check_comment(input);
    }

    return 0;
}
```

Output:

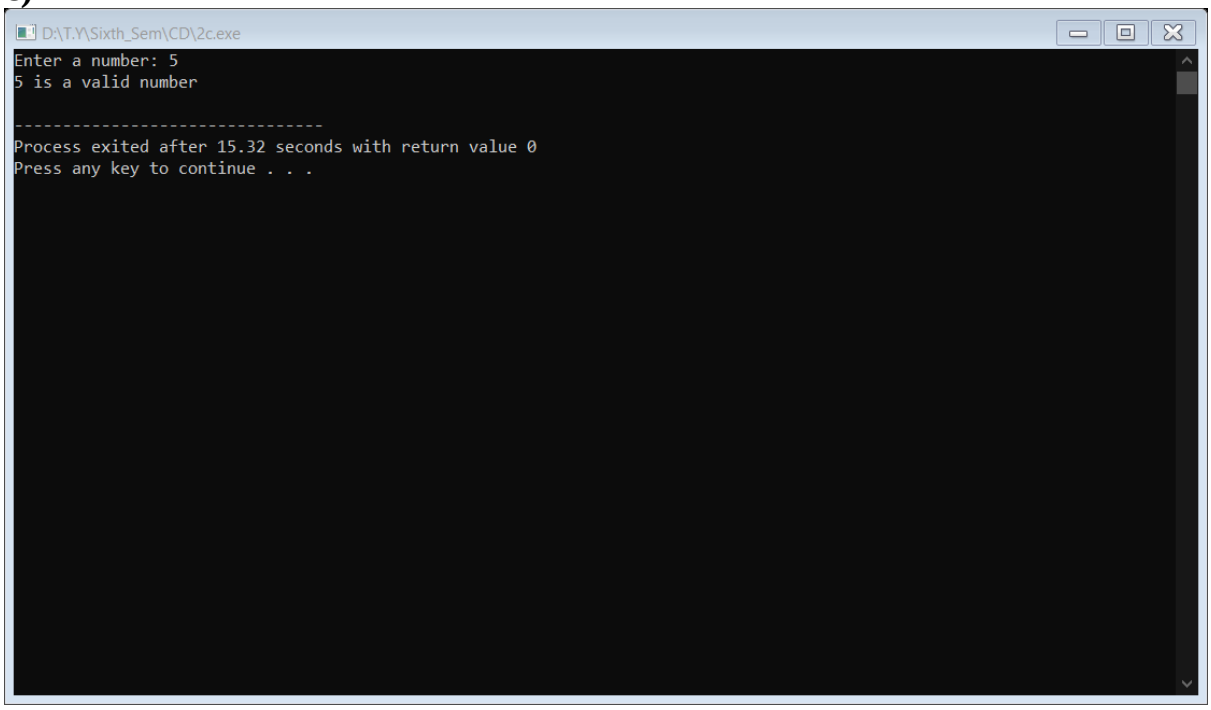
a)


D:\T.Y\Sixth Sem\CD\2a.exe
Enter a string (Type 'exit' to stop):
Input: int
"int" is a Keyword
Input: Risk
"Risk" is a Valid Identifier
Input: "Hello"
""Hello"" is an Invalid Identifier
Input: Star*
"Star*" is an Invalid Identifier
Input: double
"double" is a Keyword
Input: float
"float" is a Keyword
Input: exit
Exiting program.
Process exited after 53.18 seconds with return value 0
Press any key to continue . . .

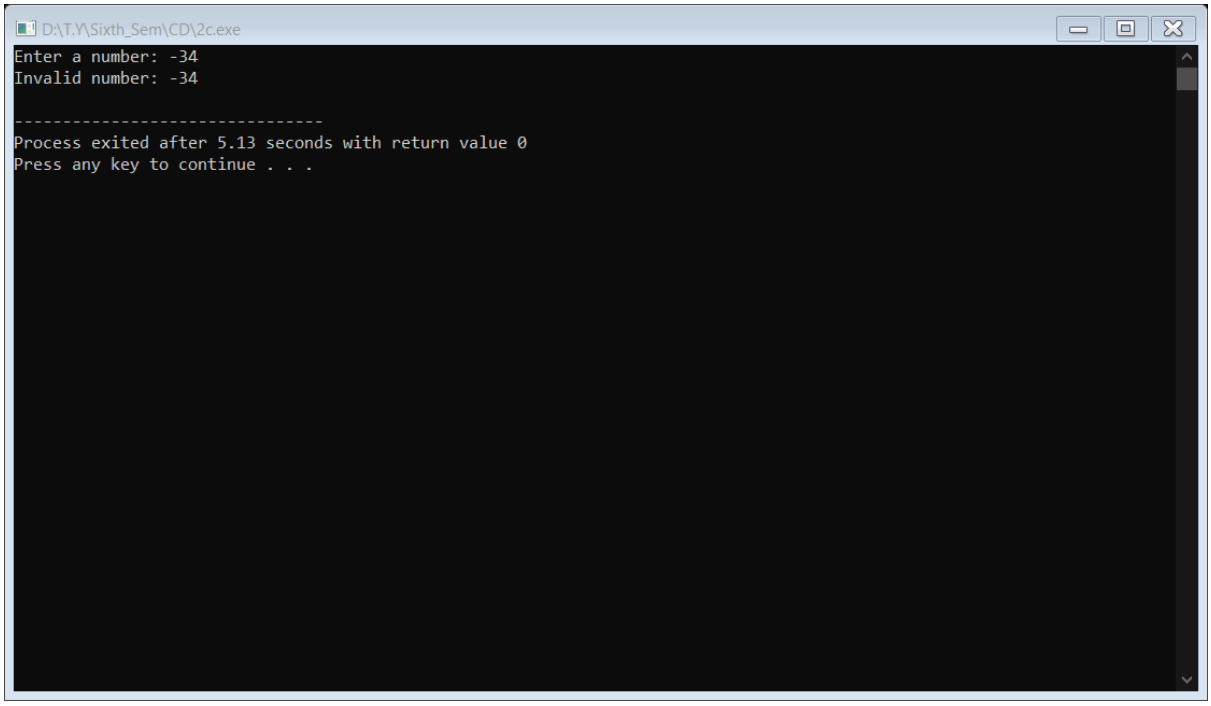
b)


D:\T.Y\Sixth_Sem\CD\2b.exe
Enter a potential C operator (or 'exit' to quit): #
"#" is NOT a valid C operator.
Enter another operator (or 'exit' to quit): @
"@" is NOT a valid C operator.
Enter another operator (or 'exit' to quit): ++
"++" is a valid C operator.
Enter another operator (or 'exit' to quit): --
"--" is a valid C operator.
Enter another operator (or 'exit' to quit): !=
"!=" is a valid C operator.
Enter another operator (or 'exit' to quit): %%
"%%" is NOT a valid C operator.
Enter another operator (or 'exit' to quit): exit
Exiting.
Process exited after 61.26 seconds with return value 0
Press any key to continue . . .

c)

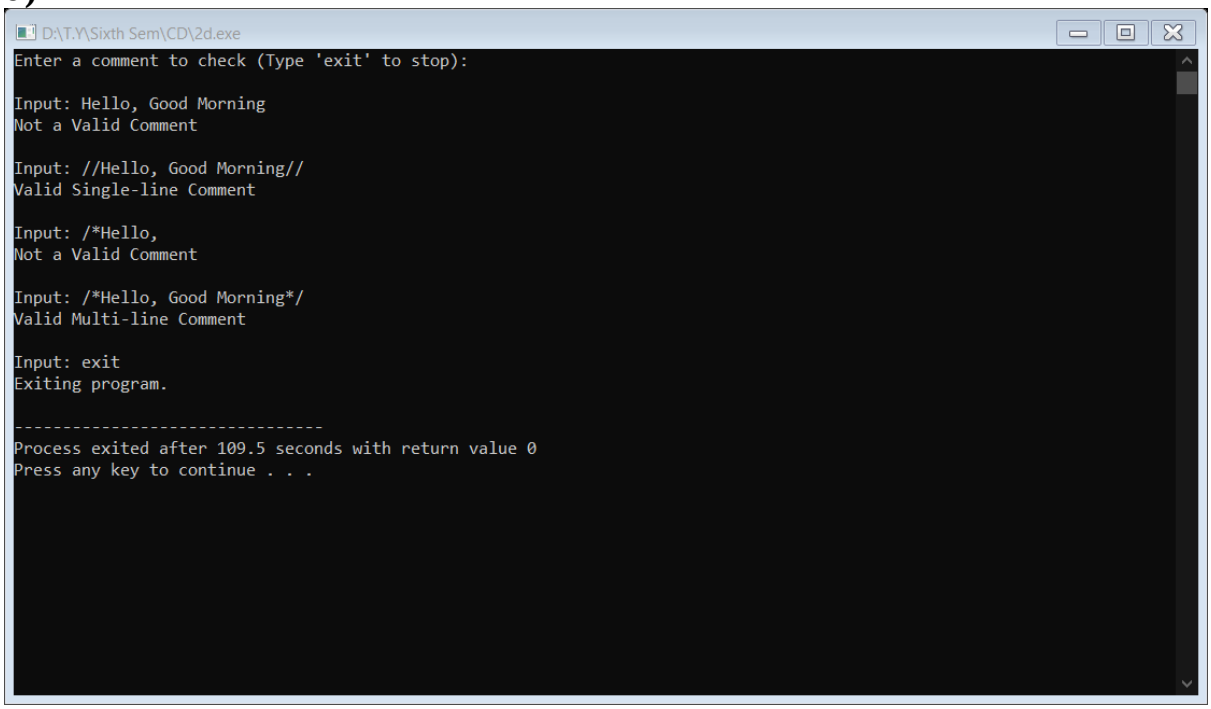

D:\T.Y\Sixth_Sem\CD\2c.exe
Enter a number: 5
5 is a valid number

Process exited after 15.32 seconds with return value 0
Press any key to continue . . .


D:\T.Y\Sixth_Sem\CD\2c.exe
Enter a number: -34
Invalid number: -34

Process exited after 5.13 seconds with return value 0
Press any key to continue . . .

d)


D:\T.Y\Sixth Sem\CD\2d.exe
Enter a comment to check (Type 'exit' to stop):
Input: Hello, Good Morning
Not a Valid Comment
Input: //Hello, Good Morning//
Valid Single-line Comment
Input: /*Hello,
Not a Valid Comment
Input: /*Hello, Good Morning*/
Valid Multi-line Comment
Input: exit
Exiting program.

Process exited after 109.5 seconds with return value 0
Press any key to continue . . .

Experiment - 3

Aim: To Study about Lexical Analyzer Generator (LEX) and Flex (Fast Lexical Analyzer).

Introduction:

A Lexical Analyzer converts an input stream (source code) into a sequence of tokens, which are then used by the parser in a compiler. Lex and Flex are tools designed for this purpose.

1. Lexical Analyzer Generator (LEX)

LEX is a tool used to generate lexical analyzers. It takes a set of **regular expressions** (token patterns) as input and produces a C program that can identify these tokens.

Working of LEX:

1. **Specification File:**
   A LEX program consists of three sections:
   - **Definition Section:** Declare header files and global variables.
   - **Rules Section:** Define token patterns using regular expressions.
   - **C Code Section:** Additional helper functions (optional).
2. **Compilation Process:**
   - The **LEX file (.l)** is compiled using lex to generate lex.yy.c.
   - The lex.yy.c file is compiled with a C compiler (gcc lex.yy.c -o output).
   - The executable processes input and tokenizes it.

Example LEX Program:

```
%{
#include <stdio.h>
%}
```

```
%%
[0-9]+    { printf("Number: %s\n", yytext); }
[a-zA-Z]+ { printf("Identifier: %s\n", yytext); }
.         { printf("Special Symbol: %s\n", yytext); }
%%
int main() {
   yylex();
   return 0;
}
int yywrap() { return 1; }
```

Commands to Run:

```
lex filename.l
gcc lex.yy.c -o output
./output < input.txt
```

2. Fast Lexical Analyzer (FLEX)

Flex is an improved and faster version of **Lex**. It provides better performance and extended functionality.

Key Features of FLEX:

- Works similarly to **Lex**, but faster.
- Generates a more optimized lex.yy.c.
- Supports additional options like debugging and performance tuning.

Example FLEX Program:

(Same structure as LEX)

```
%{
#include <stdio.h>
%}

%%
[0-9]+    { printf("Number: %s\n", yytext); }
[a-zA-Z]+ { printf("Identifier: %s\n", yytext); }
.         { printf("Special Symbol: %s\n", yytext); }
%%

int main() {
   yylex();
   return 0;
}
int yywrap() { return 1; }
```

Commands to Run:

```
flex filename.l
gcc lex.yy.c -o output
./output < input.txt
```

Comparison: LEX vs FLEX

Feature	LEX	FLEX
Speed	Slower	Faster
Compatibility	Traditional UNIX tool	GNU version, supports more platforms
Debugging	Limited	More debugging options
Performance	Basic optimization	Highly optimized DFA

Conclusion:

- Lex and Flex automate the creation of lexical analyzers.
- Flex is an enhanced version of Lex and is more commonly used today.
- These tools simplify token generation in compiler design.

Experiment - 4

Aim: Implement following programs using Lex.

a) Write a Lex program to take input from text file and count no of characters, no. of lines & no. of words.
b) Write a Lex program to take input from text file and count number of vowels and consonants.
c) Write a Lex program to print out all numbers from the given file.
d) Write a Lex program which adds line numbers to the given file and display the same into different file.
e) Write a Lex program to printout all markup tags and HTML comments in file.

Code:

a) L File Code

```
%{
#include<stdio.h>

int l = 0, w = 0, c = 0;
int in_word = 0; // Flag to track words
%}
%%
\n      { l++; c++; in_word = 0; }  // Newline increases line and
character count, resets word flag
[ \t]+  { c += yyleng; in_word = 0; }  // Spaces & tabs count as
characters, reset word flag
.       { c++; if (!in_word) { w++; in_word = 1; } }  // Count
characters, track words
%%

void main() {
    yyin = fopen("input.txt", "r");
    if (!yyin) {
        printf("Error opening file\n");
        return;
    }
    yylex();
    fclose(yyin);

    printf("This file contains:\n");
    printf("%d characters\n", c);
    printf("%d words\n", w);
    printf("%d lines\n", l);
}

int yywrap() { return 1; }
```

b) L File Code

```
%{
#include<stdio.h>

int vowels = 0, consonants = 0;
%}
%%
[aAeEiIoOuU]   { vowels++; }       // Count vowels
[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z]  { consonants++; }  // Count
consonants
.             { }  // Ignore other characters
\n            { }  // Ignore newlines
```

```
%%
void main() {
    yyin = fopen("input.txt", "r");
    if (!yyin) {
        printf("Error opening file\n");
        return;
    }
    yylex();
    fclose(yyin);

    printf("This file contains:\n");
    printf("%d vowels\n", vowels);
    printf("%d consonants\n", consonants);
}

int yywrap() { return 1; }
```

c) L File Code:

```
%{
#include <stdio.h>

%}
%%
[0-9]+  { printf("%s\n", yytext); }  // Match numbers (integers)
.       { }  // Ignore other characters
%%
void main() {
    yyin = fopen("input.txt", "r");
    if (!yyin) {
        printf("Error opening file\n");
        return;
    }
    printf("Numbers found in the file:\n");
    yylex();
    fclose(yyin);
}

int yywrap() { return 1; }
```

d) L File Code:

```
%{
#include <stdio.h>

int line_number = 1;
FILE *output;
%}
%%
^.*   { fprintf(output, "%d: %s\n", line_number++, yytext); }  // Add
line numbers
%%
void main() {
    yyin = fopen("input.txt", "r");  // Open input file
    if (!yyin) {
        printf("Error opening input file\n");
        return;
    }

    output = fopen("output.txt", "w");  // Open output file
    if (!output) {
        printf("Error opening output file\n");
```

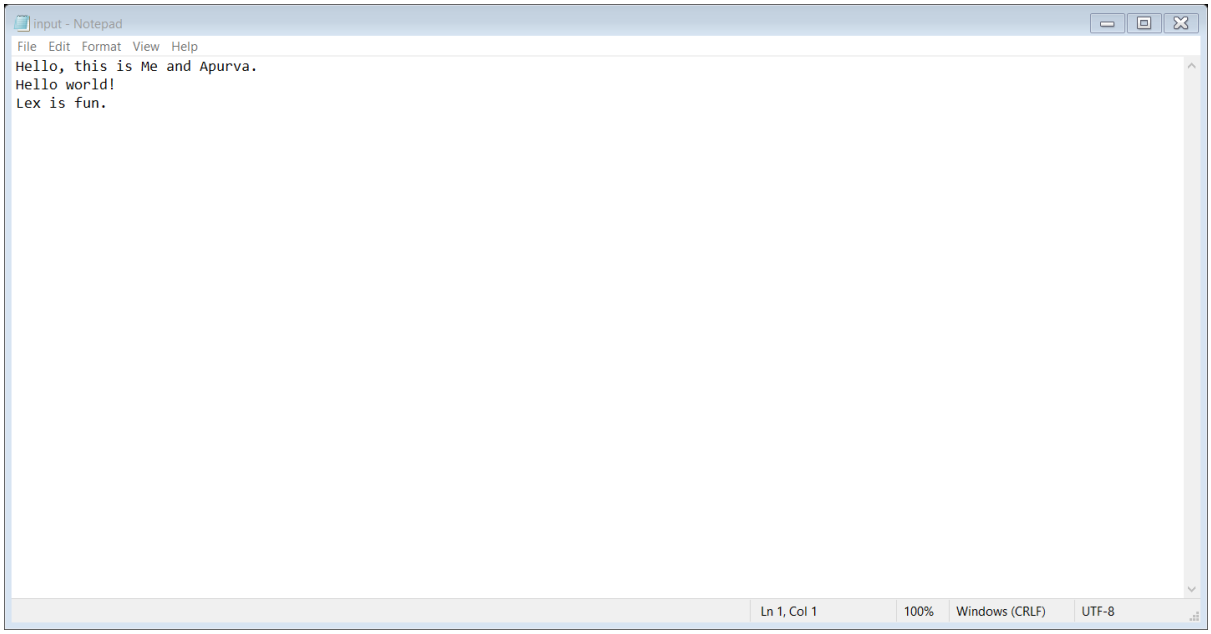
```
        return;
    }

    yylex();  // Process file

    fclose(yyin);
    fclose(output);

    printf("Processed file saved as 'output.txt'\n");
}

int yywrap() { return 1; }
```

e) L File Code:

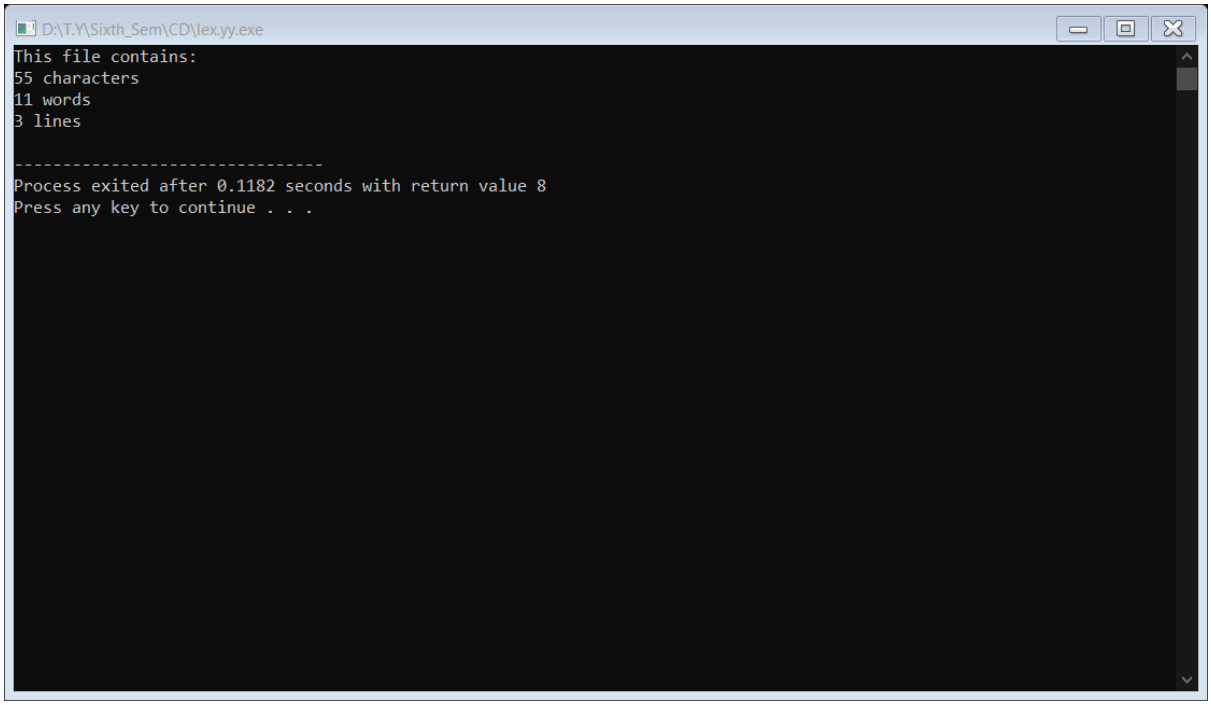
```
%{
#include <stdio.h>
%}
%%
"<!--".*"-->"   { printf("Comment: %s\n", yytext); }  // Match HTML
comments
"<"[^>]+">"     { printf("Tag: %s\n", yytext); }      // Match HTML
tags
[ \t\n]+        { }  // Ignore spaces and newlines
.               { }  // Ignore other content
%%
void main() {
    yyin = fopen("input.html", "r");  // Open input file
    if (!yyin) {
        printf("Error opening file\n");
        return;
    }

    printf("Extracted Markup Tags and Comments:\n");
    yylex();  // Process file

    fclose(yyin);
}

int yywrap() { return 1; }
```

Output:

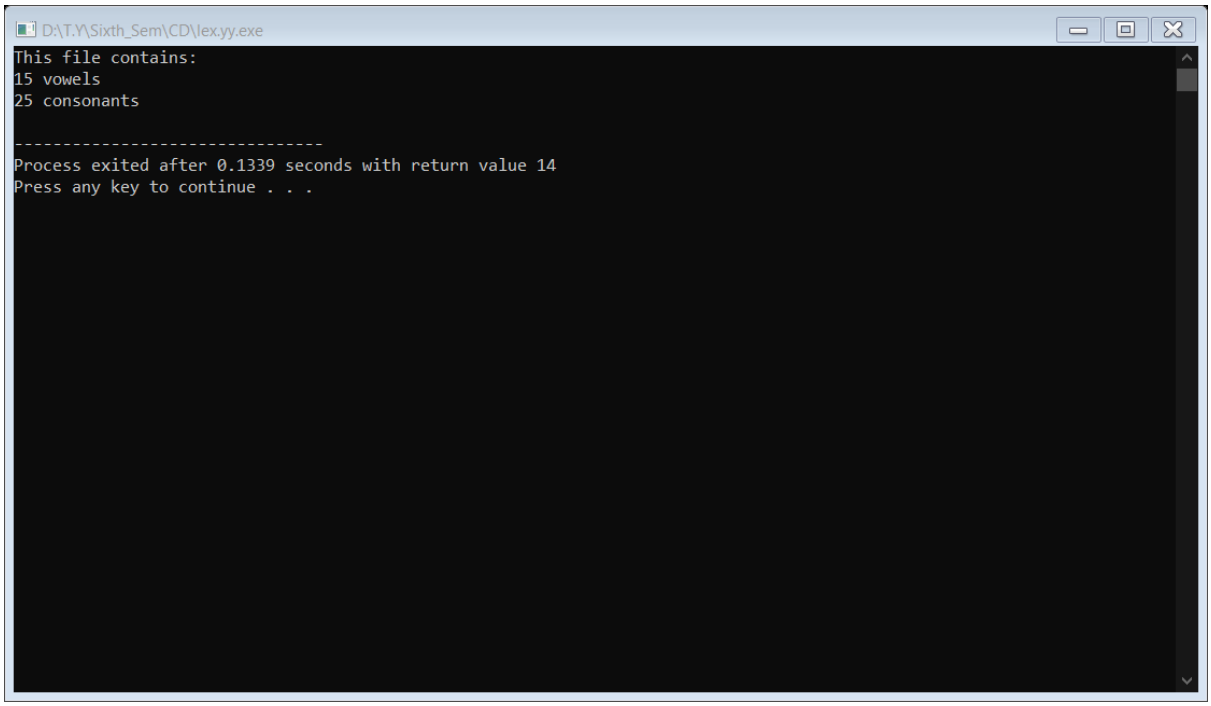
Text File:



a)

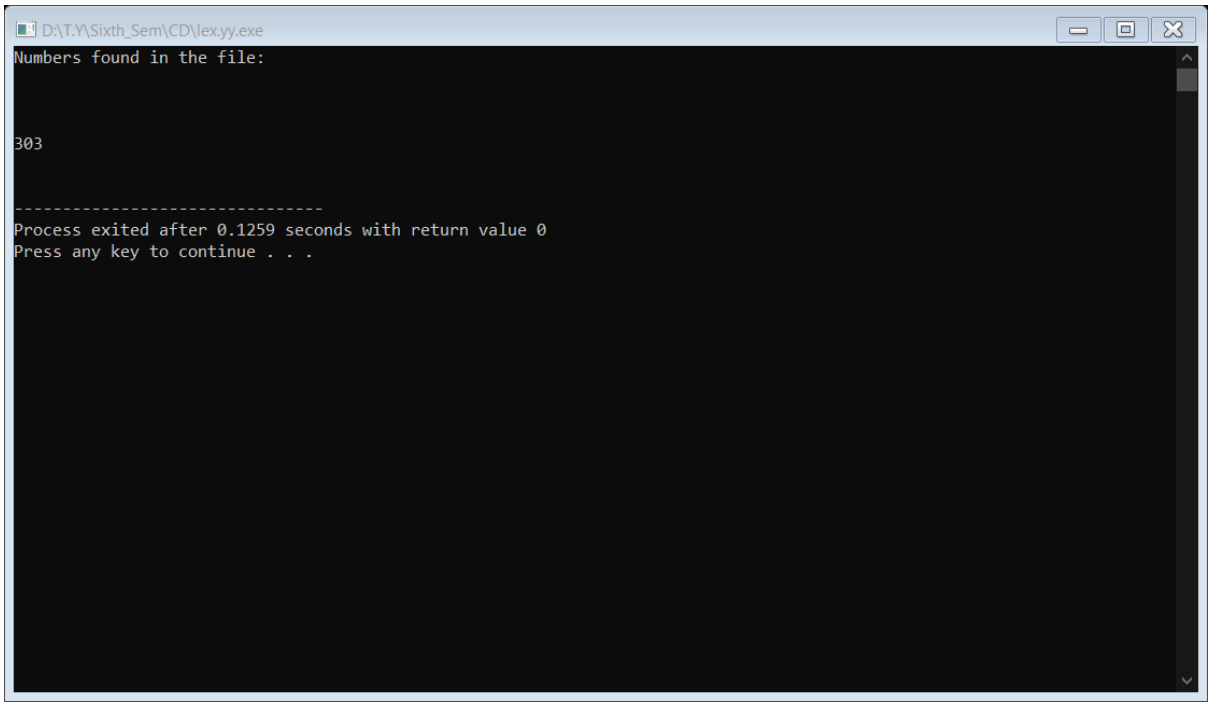


b)


D:\T.Y\Sixth_Sem\CD\lex.yy.exe
This file contains:
15 vowels
25 consonants

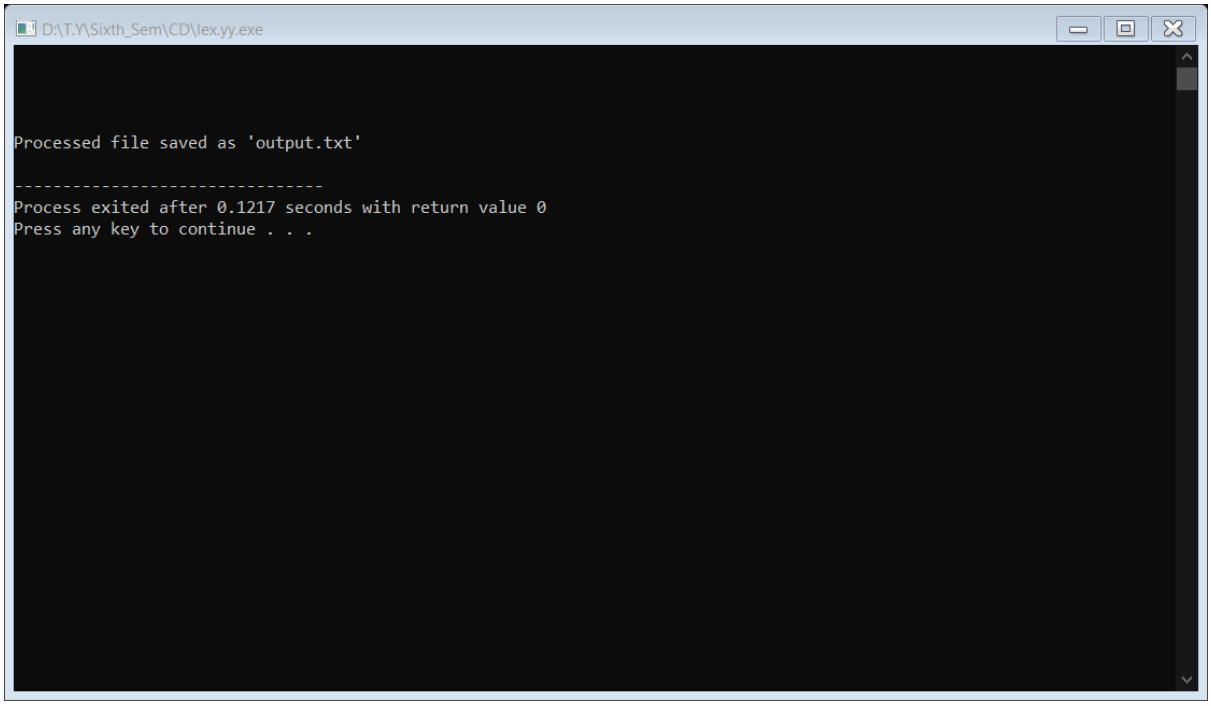
Process exited after 0.1339 seconds with return value 14
Press any key to continue . . .

c)

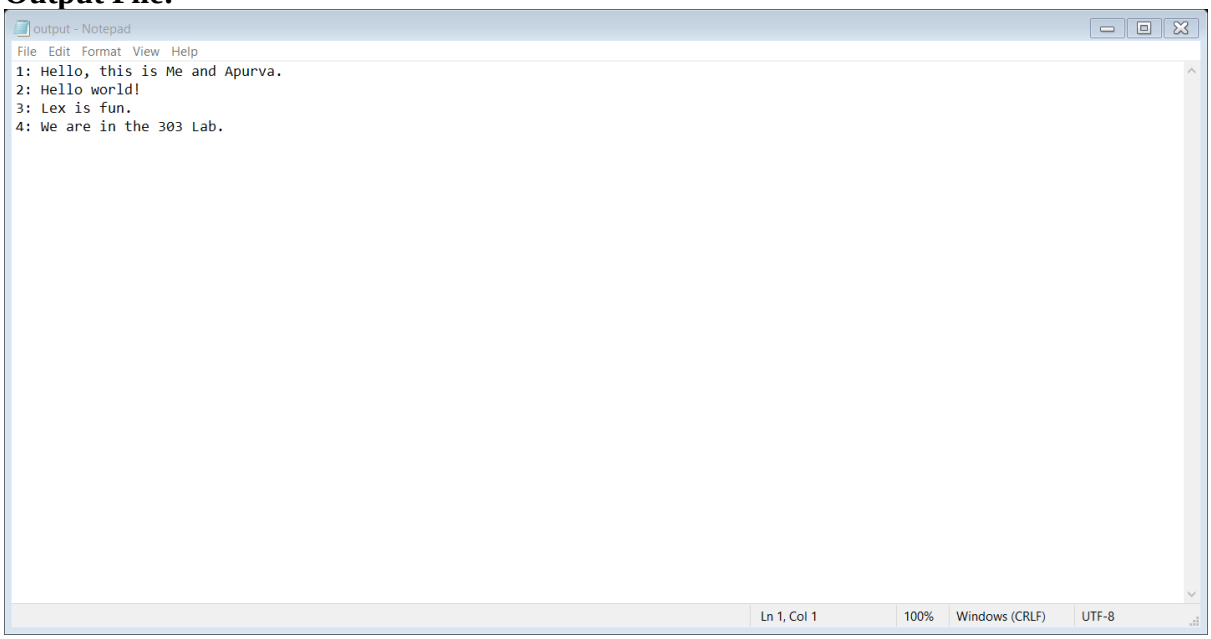

D:\T.Y\Sixth_Sem\CD\lex.yy.exe
Numbers found in the file:
303

Process exited after 0.1259 seconds with return value 0
Press any key to continue . . .

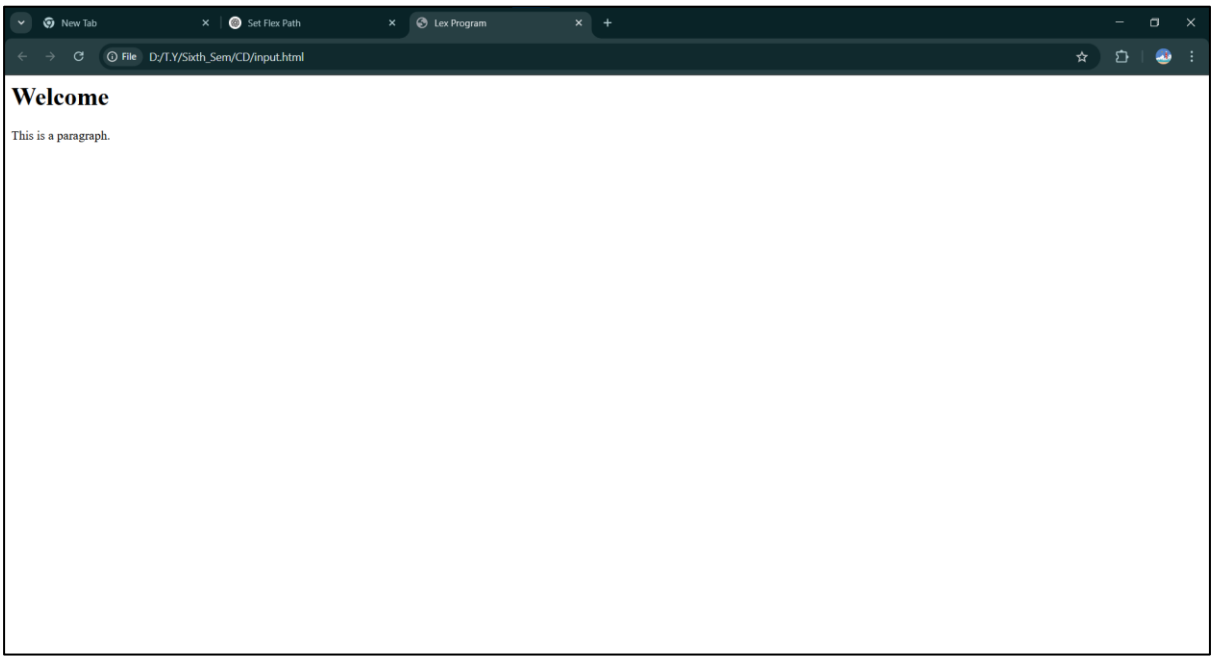
d)

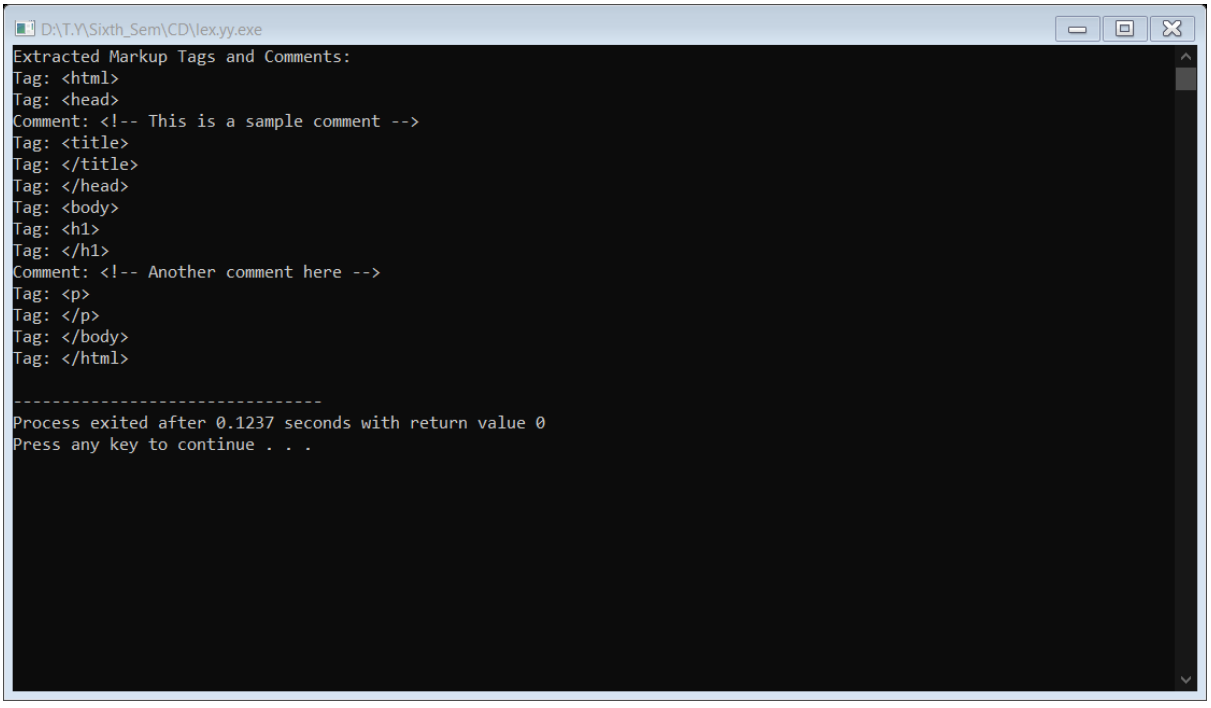


Output File:



e)


New Tab
Set Flex Path
Lex Program
File
D:/T.Y/Sixth_Sem/CD/input.html
Welcome
This is a paragraph.


D:\T.Y\Sixth_Sem\CD\lex.yy.exe
Extracted Markup Tags and Comments:
Tag: <html>
Tag: <head>
Comment: <!-- This is a sample comment -->
Tag: <title>
Tag: </title>
Tag: </head>
Tag: <body>
Tag: <h1>
Tag: </h1>
Comment: <!-- Another comment here -->
Tag:
Tag:
Tag: </body>
Tag: </html>

Process exited after 0.1237 seconds with return value 0
Press any key to continue . . .

Experiment - 5

Aim:

a) Write a Lex program to count the number of C comment lines from a given C program. Also eliminate them and copy that program into separate file.
b) Write a Lex program to recognize keywords, identifiers, operators, numbers, special symbols, literals from a given C program.

Code:

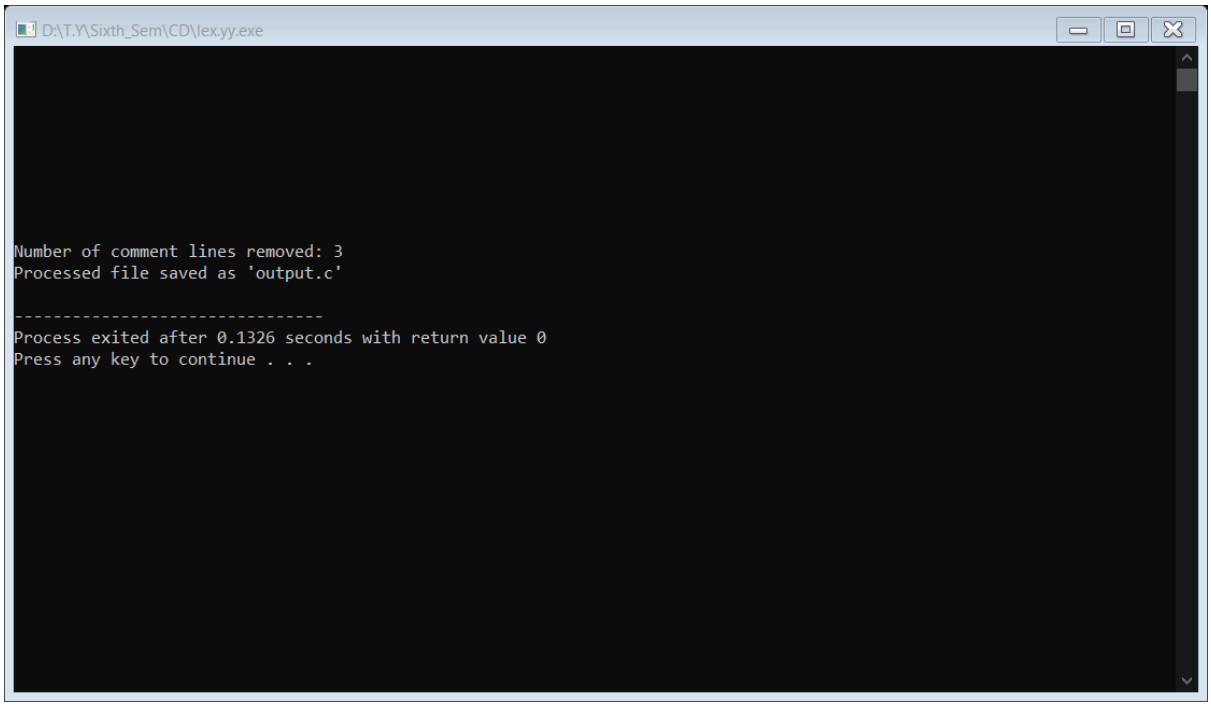
a) L File Code

```
%{
#include <stdio.h>

int comment_count = 0;
FILE *output;
%}
%%
"//".* { comment_count++; }  // Count and ignore single-line comments
"/\*"([^*]|\*+[^*/])*\*+"/" { comment_count++; }  // Count and ignore
multi-line comments
[^\n] { fprintf(output, "%s", yytext); }  // Write non-comment
content to output file
\n    { fprintf(output, "\n"); }  // Preserve newlines
%%
void main() {
    yyin = fopen("5a.c", "r");  // Open input C file
    if (!yyin) {
        printf("Error opening input file\n");
        return;
    }

    output = fopen("output.c", "w");  // Open output file (without
comments)
    if (!output) {
        printf("Error creating output file\n");
        return;
    }

    yylex();  // Process the input file

    fclose(yyin);
    fclose(output);

    printf("Number of comment lines removed: %d\n", comment_count);
    printf("Processed file saved as 'output.c'\n");
}

int yywrap() { return 1; }
```

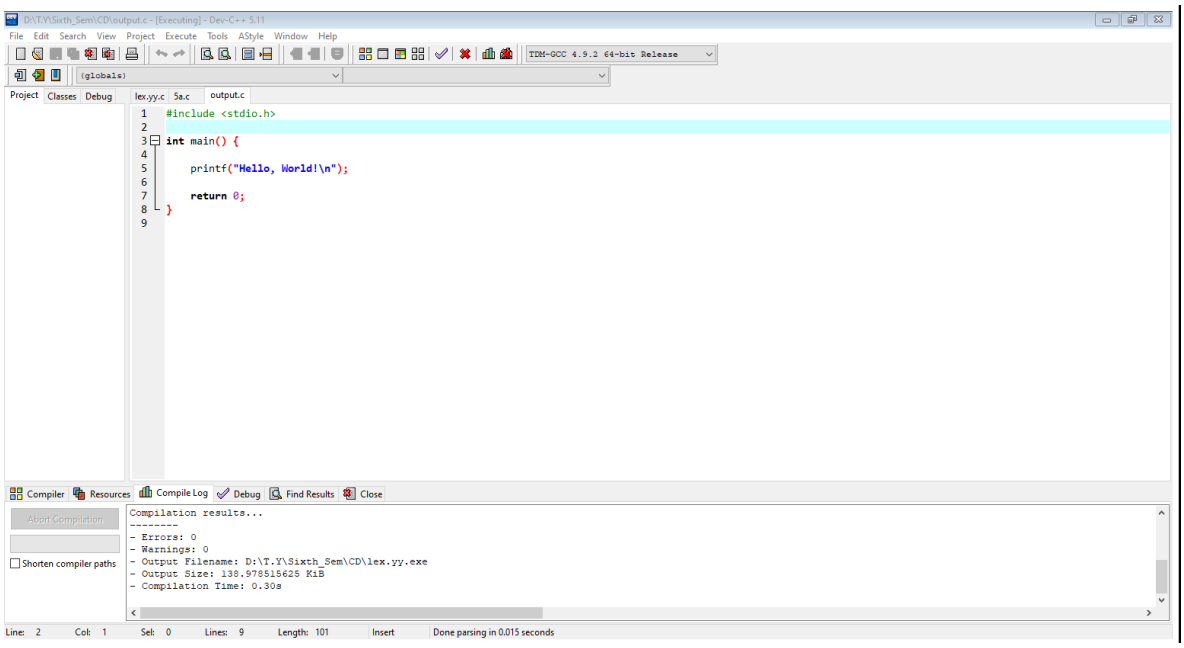
b)

Output:

a)


D:\T.Y\Sixth_Sem\CD\lex.yy.exe
Number of comment lines removed: 3
Processed file saved as 'output.c'

Process exited after 0.1326 seconds with return value 0
Press any key to continue . . .


#include <stdio.h>
int main() {
printf("Hello, World!\n");
return 0;
}
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: D:\T.Y\Sixth_Sem\CD\lex.yy.exe
- Output Size: 138.978515625 KiB
- Compilation Time: 0.30s

b)

Experiment - 6

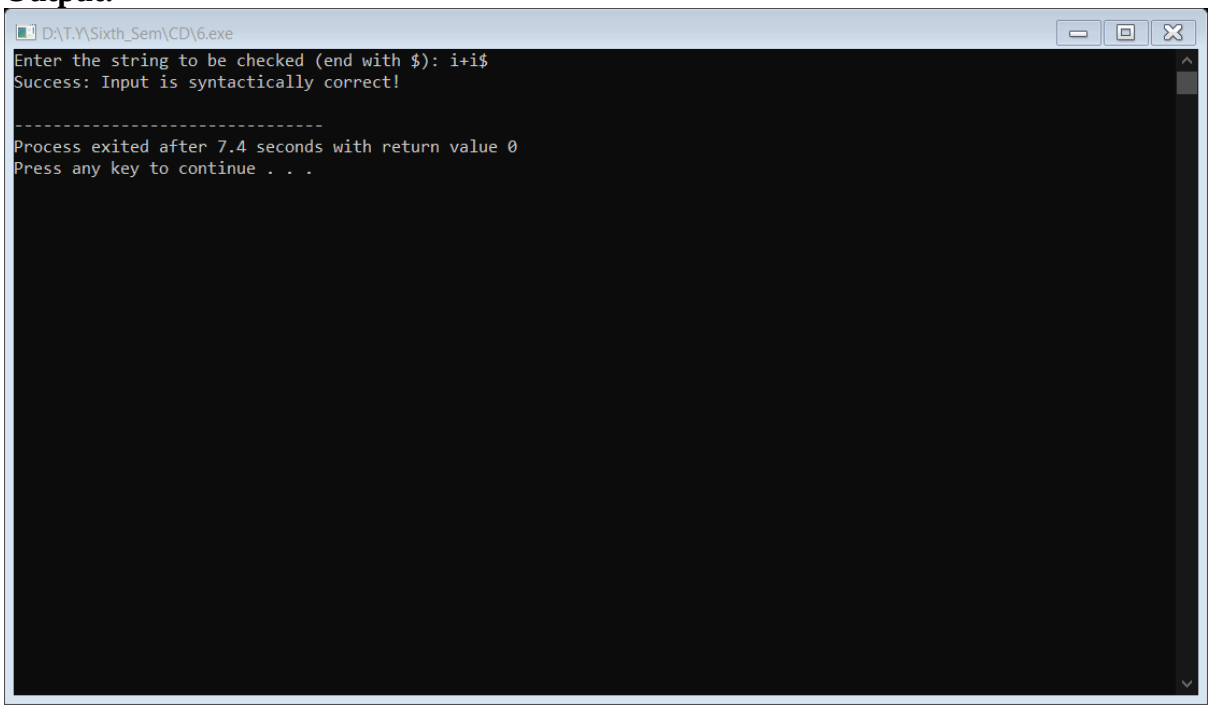
Aim: Program to implement Recursive Descent Parsing in C.

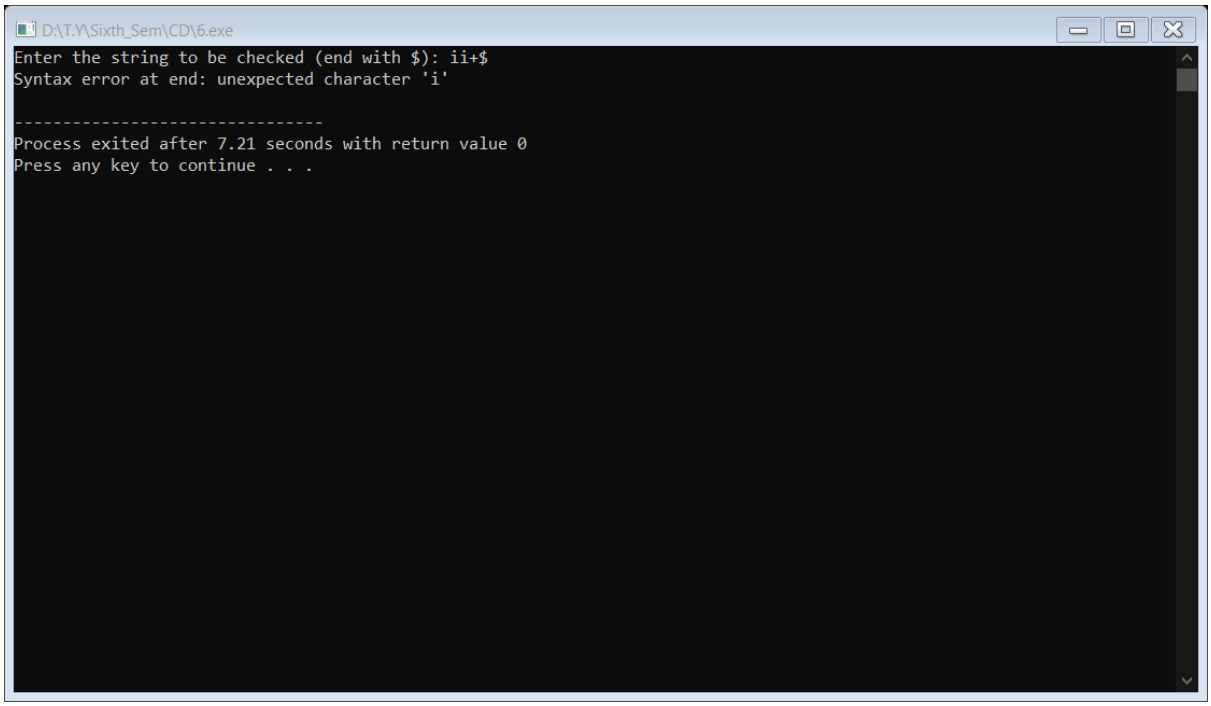
Code:

```
#include<stdio.h>     // For input/output functions like printf, scanf
#include<stdlib.h>    // For exit() function

/*
Grammar:
E  -> iE_
E_ -> +iE_ | -iE_ | e (empty)
*/

char s[20];           // Input string (e.g., "i+i-i$")
int i = 1;            // Index to track position in string (starts at 1
since s[0] goes to 'l')
char l;               // Current character being processed

int match(char t)     // Function to match current character with expected
token 't'
{
    if (l == t) {     // If match successful
        l = s[i];     // Move to next character in string
        i++;          // Increment index
    } else {          // If match fails
        printf("Syntax error");  // Print error
        exit(1);                 // Exit program
    }
}

int E_()              // Function for E_ ? +iE_ | -iE_ | e
{
    if (l == '+') {       // If current character is '+'
        match('+');       // Match '+'
        match('i');       // Match 'i' after '+'
        E_();             // Recursively process the rest
    } else if (l == '-') { // If current character is '-'
        match('-');        // Match '-'
        match('i');        // Match 'i' after '-'
        E_();              // Recursively process the rest
    } else {
        return 1;          // If neither '+' nor '-', return success
(epsilon case)
    }
}

int E()               // Function for E ? iE_
{
    if (l == 'i') {   // If current character is 'i'
        match('i');   // Match 'i'
        E_();         // Then check for E_
    }
}

int main()
{
    printf("\n Enter the set of characters to be checked :"); // Prompt for
input
    scanf("%s", &s);         // Read input string into array 's'
    l = s[0];                // Set the first character of input to 'l'
```

```
    E();                        // Start parsing from rule E
    if (l == '$') {             // If end marker '$' is reached after
successful parsing
        printf("Success \n"); // Input is valid
    } else {
        printf("syntax error"); // Input is invalid if not ending with '$'
    }
    return 0;                   // Exit successfully
}
    return 0;
}
```

Output:





Experiment - 7

Aim:

a) To Study about Yet Another Compiler-Compiler(YACC).
b) Create Yacc and Lex specification files to recognizes arithmetic expressions involving +, -, * and / .
c) Create Yacc and Lex specification files are used to generate a calculator which accepts integer type arguments.
d) Create Yacc and Lex specification files are used to convert infix expression to postfix expression.

Code:

a) Installation process
b) L File:

```
%{
#include "yacc.tab.h"   // Include token definitions from yacc output
%}

%%
[0-9]+      { yylval = atoi(yytext); return NUMBER; }  // Return
numbers as NUMBER
[+\-*/]     { return yytext[0]; }                      // Return
operators directly
[\n]        { return 0; }                              // End input
on newline
[ \t]       ;                                          // Skip
whitespace
.           { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() {
    return 1;
}
```

Y File:

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

int yylex();           // Declare yylex() from lex.l
int yyerror(const char *s);  // Declare yyerror()
%}

%token NUMBER

%%
expr: expr '+' term   { printf("Add\n"); }
    | expr '-' term   { printf("Subtract\n"); }
    | term
    ;

term: term '*' factor { printf("Multiply\n"); }
    | term '/' factor { printf("Divide\n"); }
    | factor
    ;

factor: NUMBER
      ;
%%
```

```
int main() {
    printf("Enter an arithmetic expression:\n");
    yyparse();
    return 0;
}

int yyerror(const char *s) {
    printf("Syntax Error: %s\n", s);
    return 0;
}
```

c) L File:

```
%{
#include <stdlib.h>
void yyerror(char *);
#include "yacc.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext); return NUM;}
[-+*\n] {return *yytext;}
[ \t] { }
. yyerror("invalid character");
%%
int yywrap() {
 return 0;
}
```

Y File:

```
%{
 #include <stdio.h>
 int yylex(void);
 void yyerror(char *);
%}
%token NUM
%%
S: E '\n' { printf("%d\n", $1); return(0); }
E: E '+' T   { $$ = $1 + $3; }
 | E '-' T  { $$ = $1 - $3; }
 | T        { $$ = $1; }
T : T '*' F { $$ = $1 * $3; }
 | F        { $$ = $1; }
F:NUM   { $$ = $1; }
%%
void yyerror(char *s) {
 fprintf(stderr, "%s\n", s);
}
int main() {
 yyparse();
 return 0;
}
```

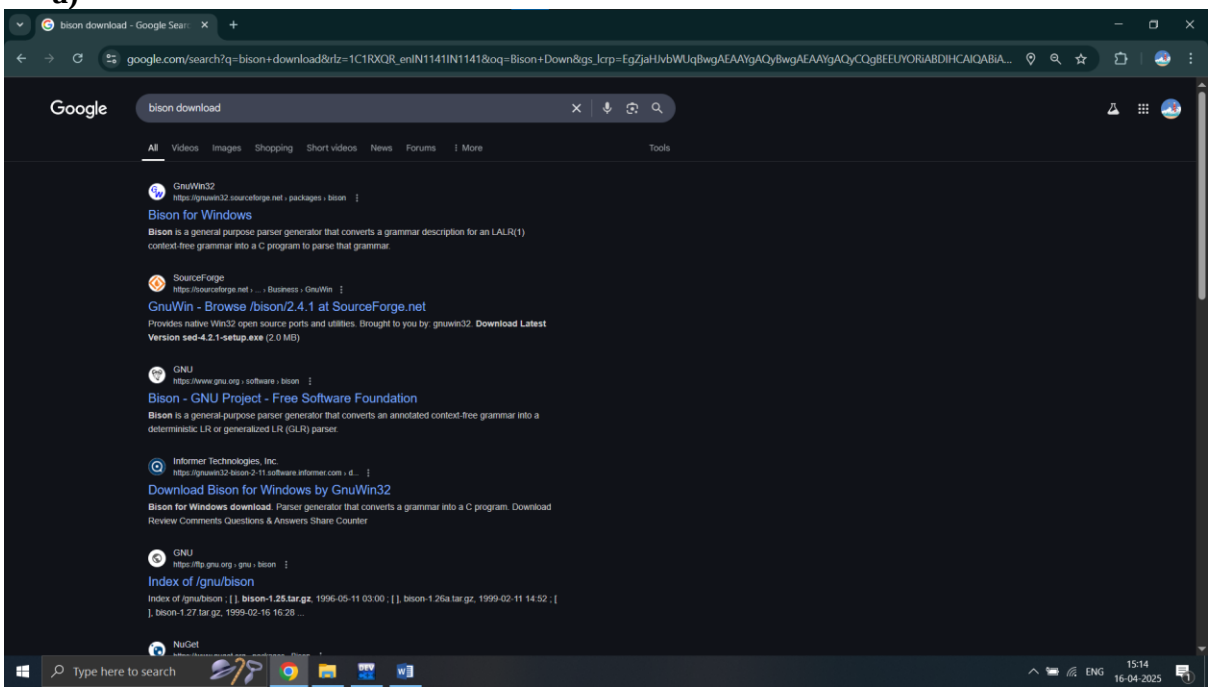
d) L File:

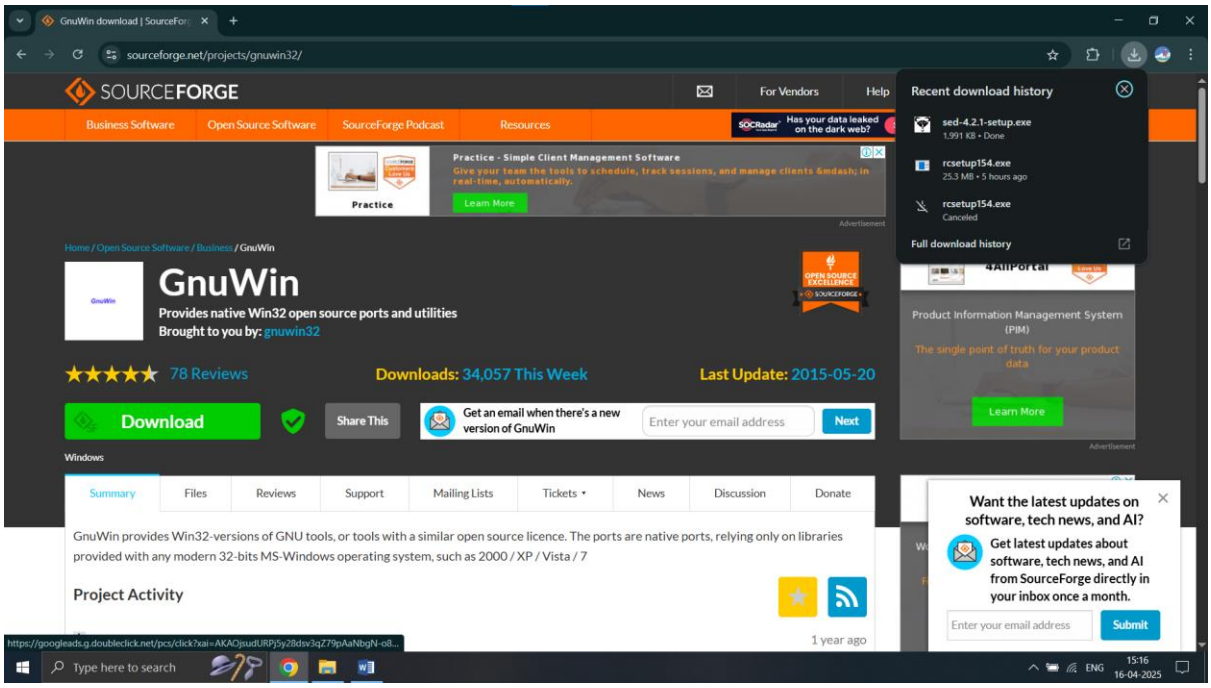
```
%{
#include<stdio.h>
#include "first.tab.h"
void yyerror(char *);
%}

%%
[0-9]+ { yylval.num = atoi(yytext); return INTEGER; }
```

```
[A-Za-z_][A-Za-z0-9_]* { yylval.str = yytext; return ID; }
[-+;\n*] { return *yytext; }
[ \t] ;
. yyerror("invalid character");
%%

int yywrap(){
    return 1;
}

Y File:
%{
    #include<stdio.h>
    int yylex(void);
    void yyerror(char *);
%}
%union{
    char *str;
    int num;
}
%token <num> INTEGER
%token <str> ID
%%
S: E '\n' {printf("\n");}
E: E '+' T {printf("+ "); }
 | E '-' T {printf("- "); }
 | T { }
T : T '*' F {printf("* "); }
 | F { }
F:INTEGER { printf("%d ", $1);}
 | ID { printf("%s ", $1);}
%%
void yyerror(char *s){
    printf("%s\n", s);
}
int main(){yyparse();return 0;}
```

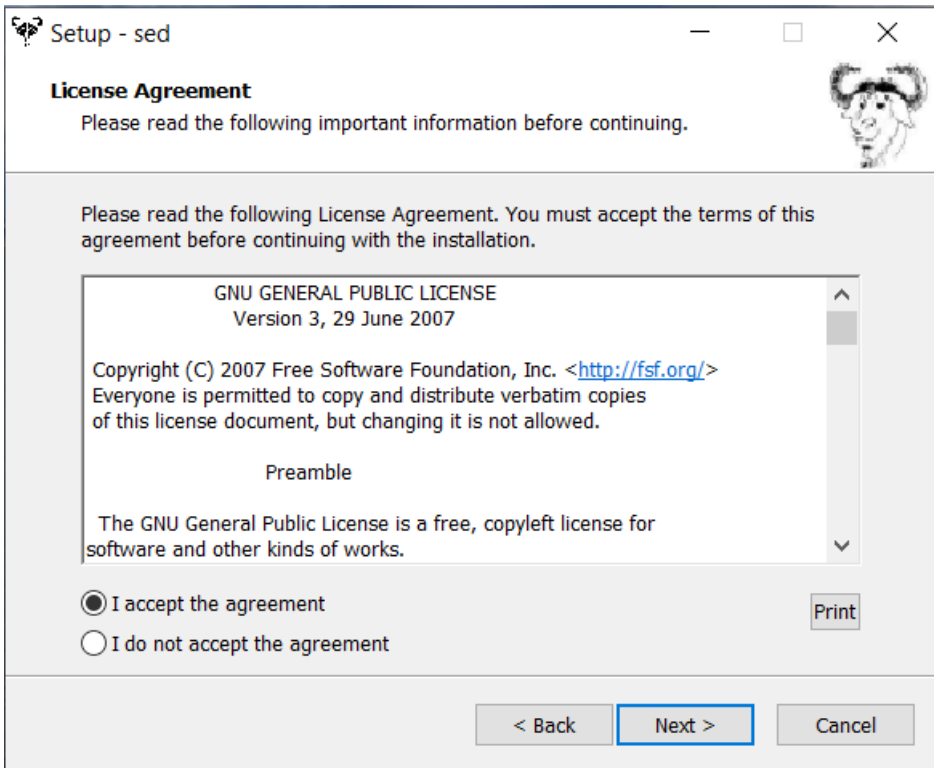
Output:

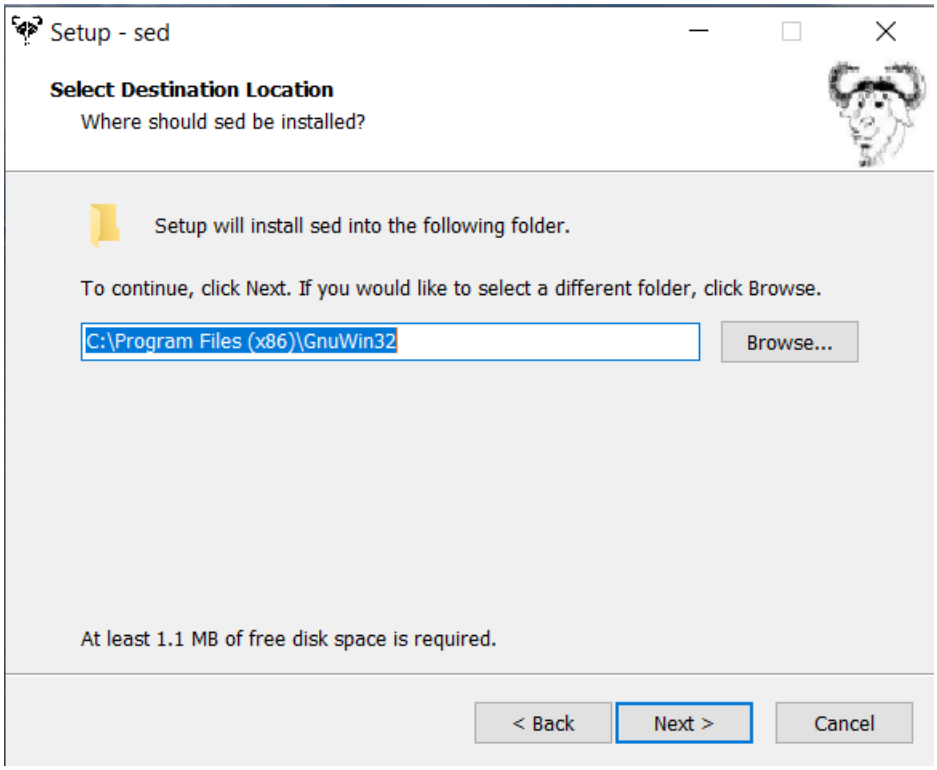
a)

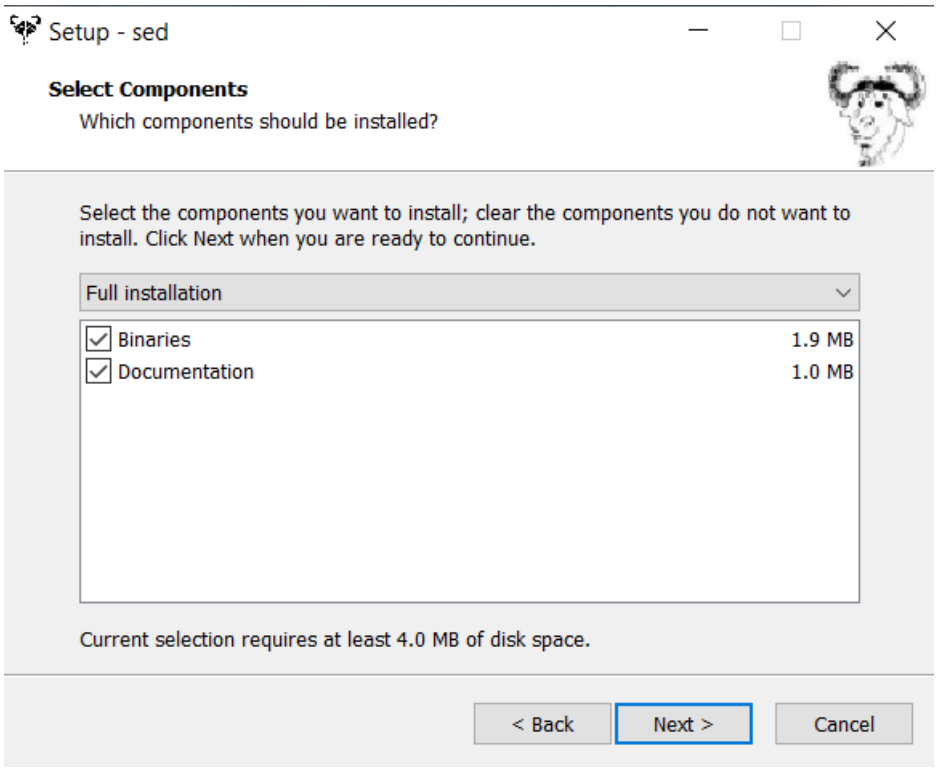

bison download - Google Search
google.com/search?q=bison+download&rlz=1C1RXQR_enIN1141IN1141&oq=Bison+Down&gs_lcrp=EgZjaHJvbWUqBwgAEAAYgACyBwgAEAAYgACyCQgBEEUYORiABDIHCAIQABiA...
Google
bison download
All Videos Images Shopping Short videos News Forums More Tools
GnuWin32
https://gnuwin32.sourceforge.net › packages › bison
Bison for Windows
Bison is a general purpose parser generator that converts a grammar description for an LALR(1) context-free grammar into a C program to parse that grammar.
SourceForge
https://sourceforge.net › ... › Business › GnuWin
GnuWin - Browse /bison/2.4.1 at SourceForge.net
Provides native Win32 open source ports and utilities. Brought to you by: gnuwin32. Download Latest Version sed-4.2.1-setup.exe (2.0 MB)
GNU
https://www.gnu.org › software › bison
Bison - GNU Project - Free Software Foundation
Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser.
Informer Technologies, Inc.
https://gnuwin32-bison-2-11.software.informer.com › d...
Download Bison for Windows by GnuWin32
Bison for Windows download. Parser generator that converts a grammar into a C program. Download Review Comments Questions & Answers Share Counter
GNU
https://ftp.gnu.org › gnu › bison
Index of /gnu/bison
Index of /gnu/bison . [ ], bison-1.25.tar.gz, 1996-05-11 03:00 , [ ], bison-1.26a.tar.gz, 1999-02-11 14:52 , [ ], bison-1.27.tar.gz, 1999-02-16 16:28 ...
NuGet
Type here to search
ENG
15:14
16-04-2025

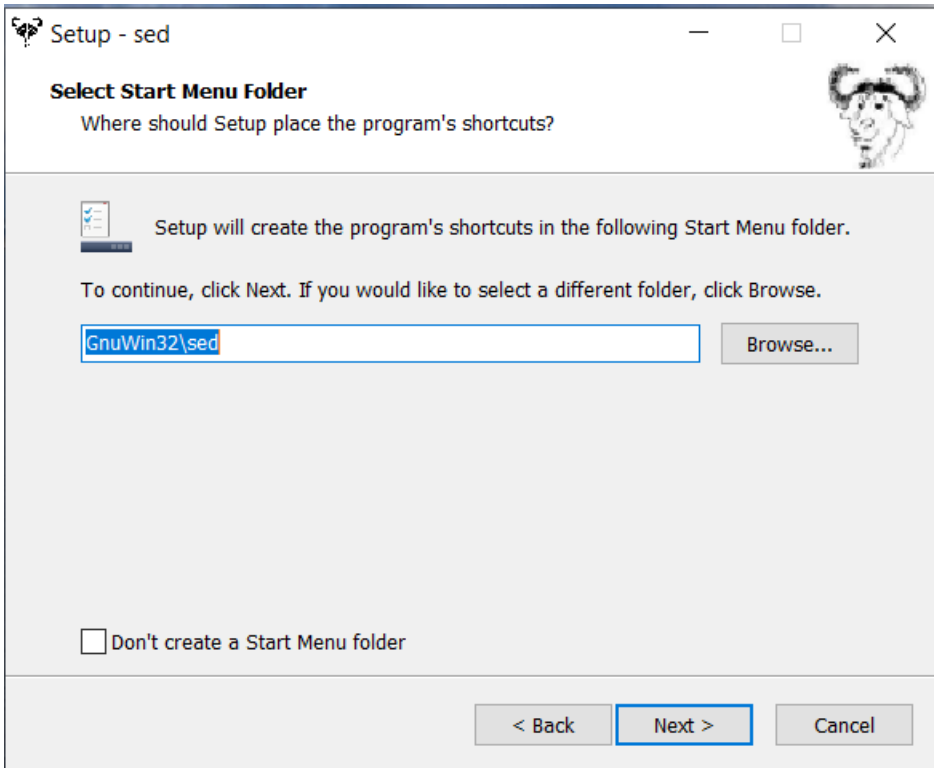

GnuWin download | SourceForge.net
sourceforge.net/projects/gnuwin32/
SOURCEFORGE
For Vendors
Help
Business Software
Open Source Software
SourceForge Podcast
Resources
Has your data leaked on the dark web?
Recent download history
sed-4.2.1-setup.exe
1.991 KB • Done
rcsetup154.exe
25.3 MB • 5 hours ago
rcsetup154.exe
Canceled
Full download history
Practice - Simple Client Management Software
Give your team the tools to schedule, track sessions, and manage clients — in real-time, automatically.
Learn More
Practice
Advertisement
Home / Open Source Software / Business / GnuWin
GnuWin
Provides native Win32 open source ports and utilities
Brought to you by: gnuwin32
78 Reviews
Downloads: 34,057 This Week
Last Update: 2015-05-20
Download
Share This
Get an email when there's a new version of GnuWin
Enter your email address
Next
Windows
Summary
Files
Reviews
Support
Mailing Lists
Tickets
News
Discussion
Donate
GnuWin provides Win32-versions of GNU tools, or tools with a similar open source licence. The ports are native ports, relying only on libraries provided with any modern 32-bits MS-Windows operating system, such as 2000 / XP / Vista / 7
Project Activity
1 year ago
Product Information Management System (PIM)
The single point of truth for your product data
Learn More
Want the latest updates on software, tech news, and AI?
Get latest updates about software, tech news, and AI from SourceForge directly in your inbox once a month.
Enter your email address
Submit
Type here to search
ENG
15:16
16-04-2025

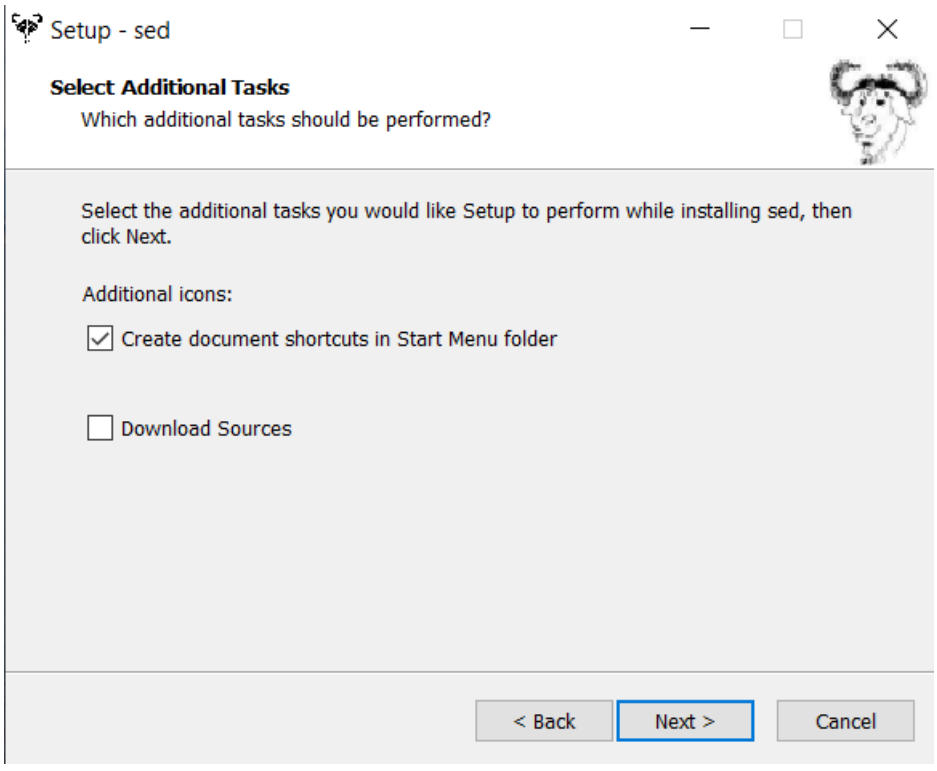

Setup - sed
Welcome to the sed Setup Wizard
This will install sed-4.2.1 (stream editor; Complete package, except sources) on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel

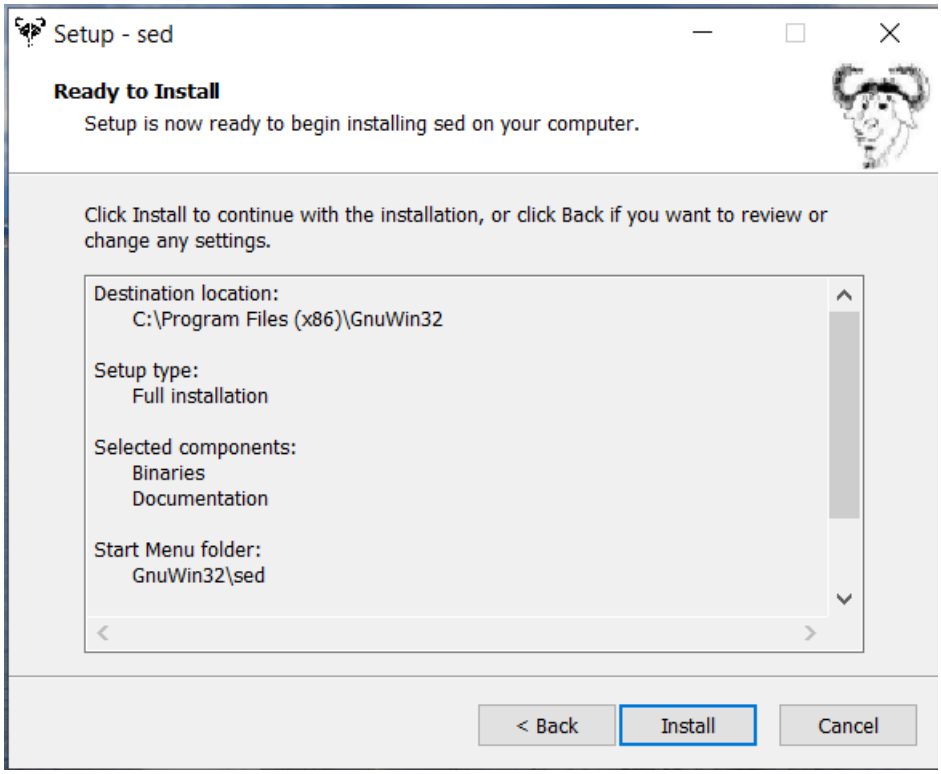

Setup - sed
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
The GNU General Public License is a free, copyleft license for
software and other kinds of works.
I accept the agreement
I do not accept the agreement
Print
< Back
Next >
Cancel

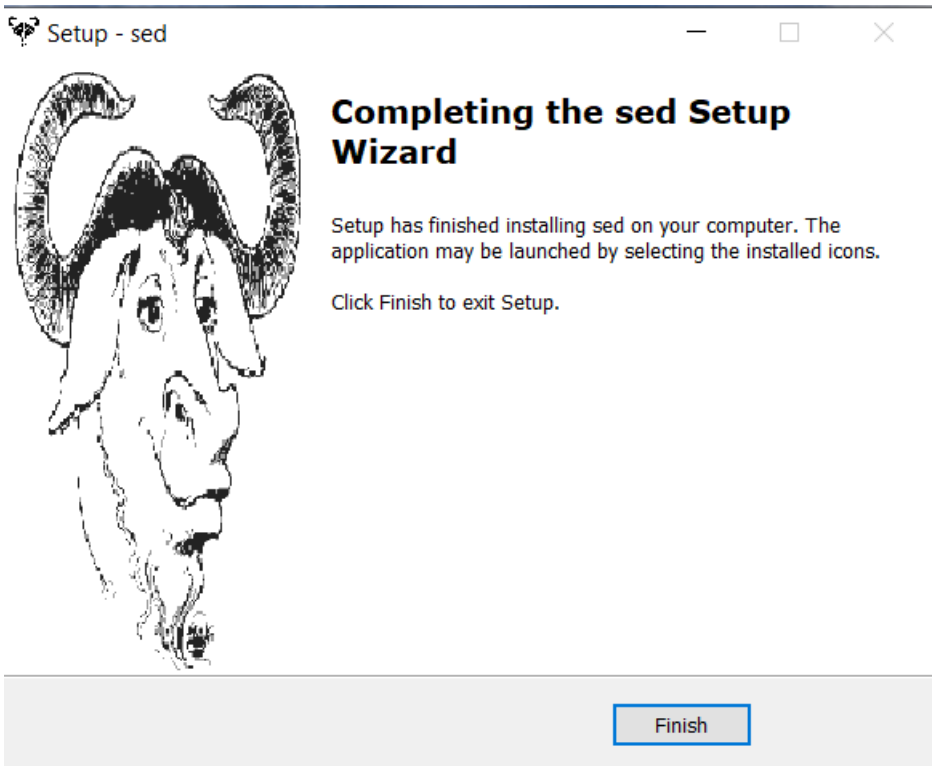

Setup - sed
Select Destination Location
Where should sed be installed?
Setup will install sed into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files (x86)\GnuWin32
Browse...
At least 1.1 MB of free disk space is required.
< Back
Next >
Cancel

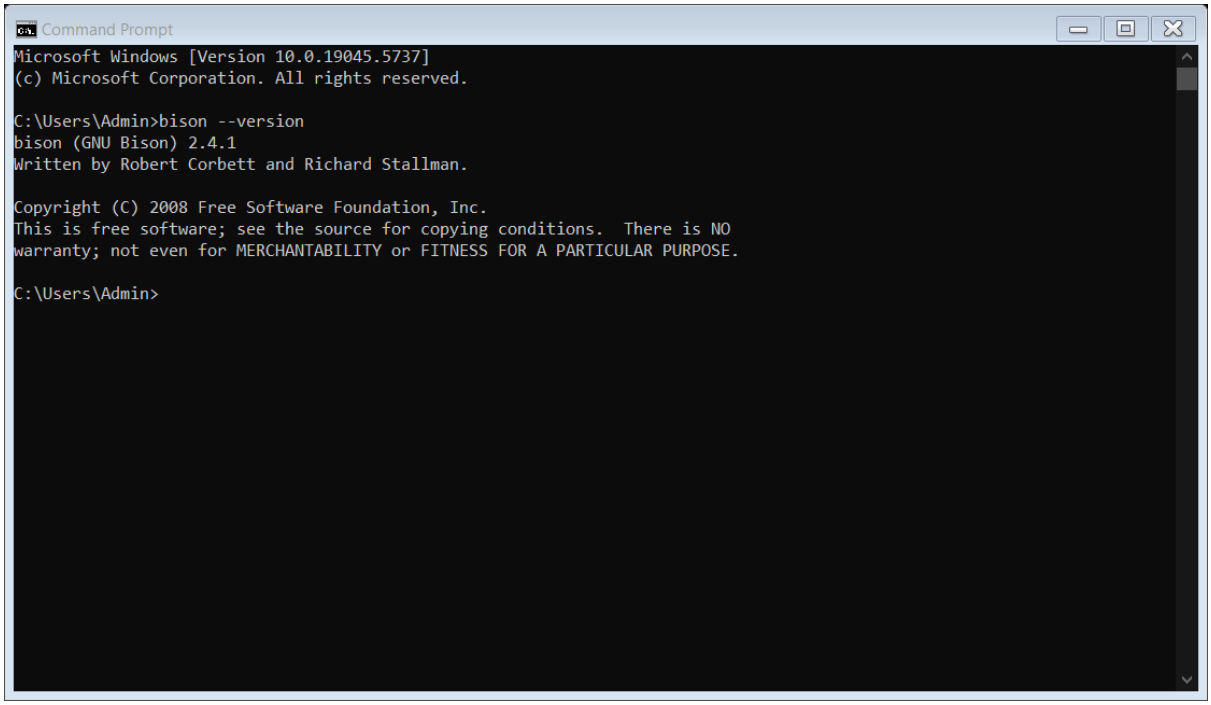

Setup - sed
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Full installation
Binaries 1.9 MB
Documentation 1.0 MB
Current selection requires at least 4.0 MB of disk space.
< Back
Next >
Cancel


Setup - sed
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
GnuWin32\sed
Browse...
Don't create a Start Menu folder
< Back
Next >
Cancel


Setup - sed
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing sed, then click Next.
Additional icons:
Create document shortcuts in Start Menu folder
Download Sources
< Back
Next >
Cancel

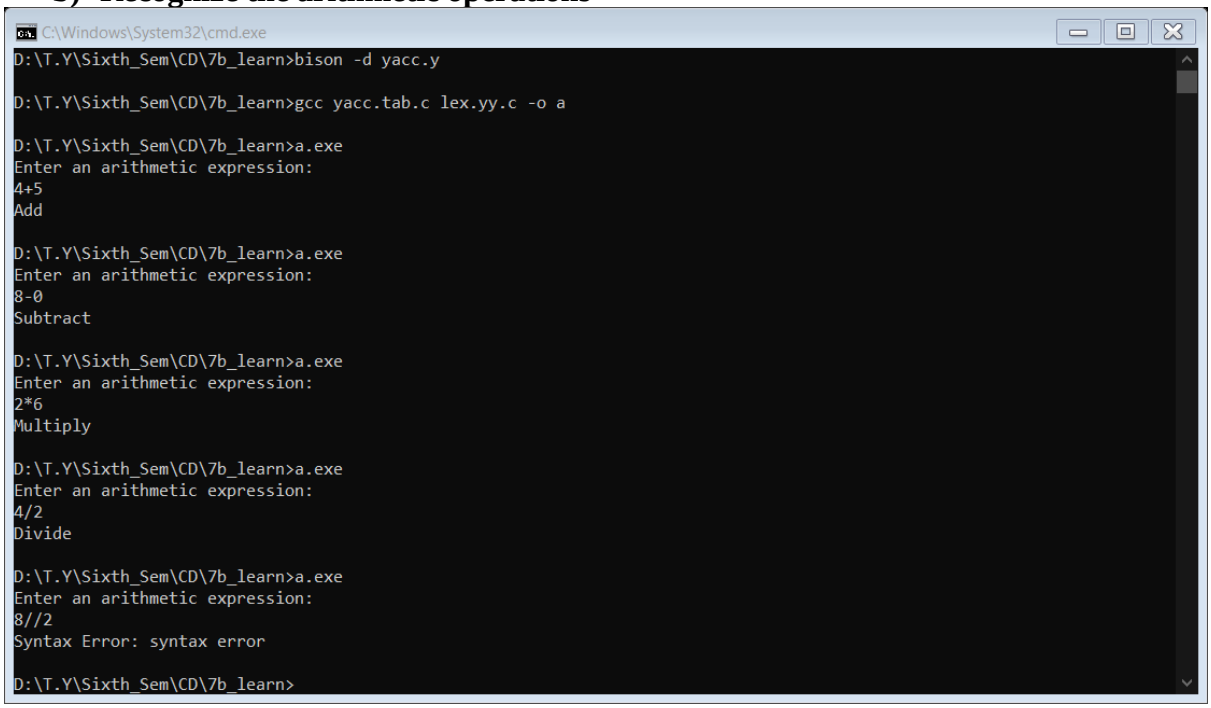

Setup - sed
Ready to Install
Setup is now ready to begin installing sed on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Destination location:
C:\Program Files (x86)\GnuWin32
Setup type:
Full installation
Selected components:
Binaries
Documentation
Start Menu folder:
GnuWin32\sed
< Back
Install
Cancel


Setup - sed
Completing the sed Setup Wizard
Setup has finished installing sed on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
Finish

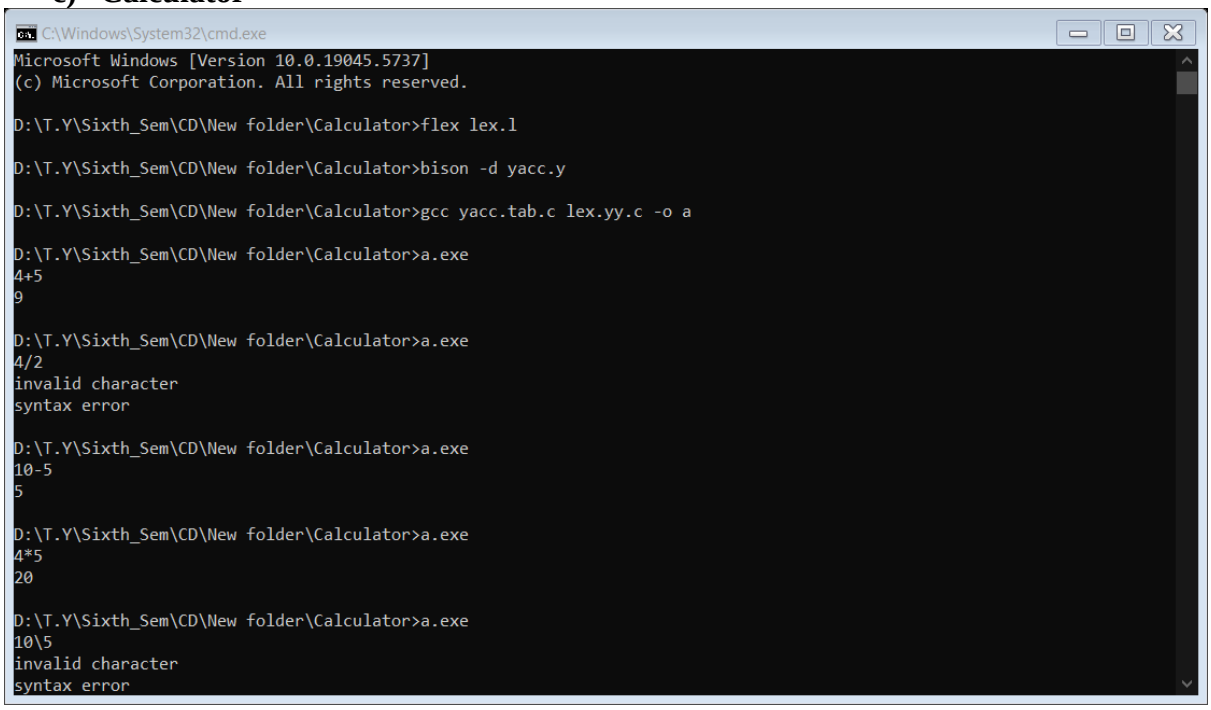


To run, following are the commands:
bison –d yacc.y
flex filename.l
gcc lex.yy.c yacc.tab.c
After this an .exe file will be generated

b) Recognize the arithmetic operations



c) Calculator



d) Infix to postfix