PATHAN FIZA FAIYAZKHAN

22000986(A2)

SCHOOL OF ENGINEERING & TECHNOLOGY

BACHELOR OF TECHNOLOGY

COMPILER DESIGN

6TH SEMESTER

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Laboratory Manual

TABLE OF CONTENT

Practical-1

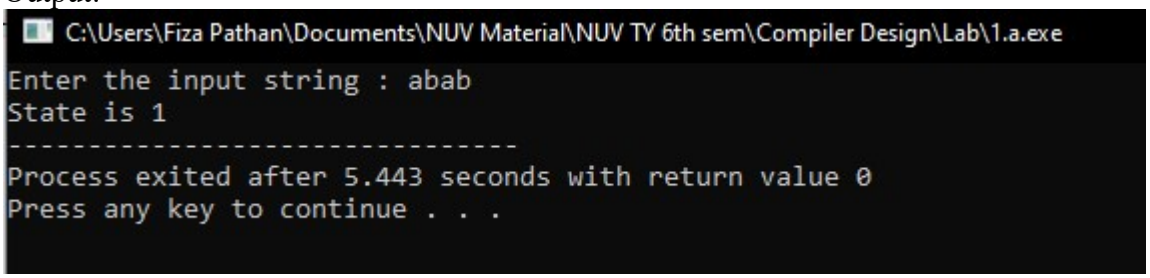
a) Write a program to recognize strings starts with 'a' over {a, b}.

Input:

```
#include<stdio.h>
int main ()
{
    char input[10];
    int state=0,i=0;
    printf("Enter the input string : ");
    scanf("%s",input);
    while(input[i]!='\0'){
            switch (state){
                    case 0:
                            if (input[i]=='a'){
                                    state = 1;
                                    i++;
                            }
                            else if (input[i]=='b'){
                                    state = 2;
                                    i++;
                            }
                            else {
                                    state = 3;
                                    i++;
                            }
                            break;
                    case 1:
                            if (input[i]=='a'||input[i]=='b'){
                                    state = 1;
                                    i++;
                            }
                            else {
                                    state = 3;
                                    i++;
                            }
                            break;
                    case 2:
                            if (input[i]=='a'||input[i]=='b'){
                                    state = 2;
                                    i++;
                            }
                            else {
                                    state = 3;
                                    i++;
```

```
                        }
                        break;
                case 3:
                        state=3;
                        i++;
                        break;
            }
    }
    printf("State is %d",state);
    return 0;
}
```

Output:

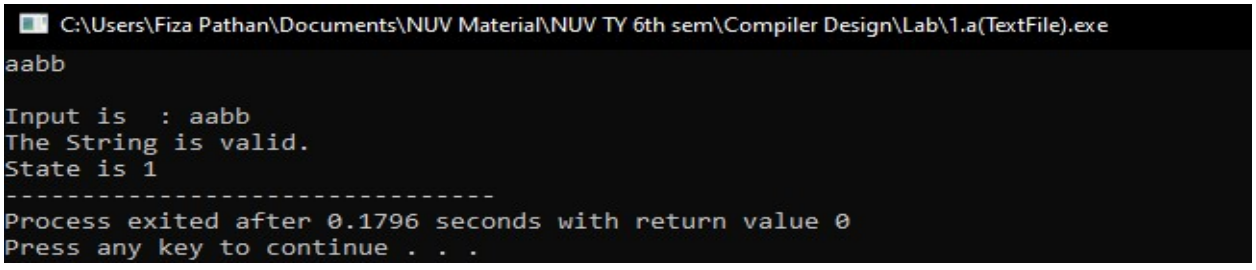


Input (With File):

```
#include<stdio.h>
 #include <stdlib.h>
int main ()
{
    char input[100];
    int state=0,i=0;
    //write a program for string starts with 'a'.
    FILE* ptr;
    ptr = fopen("Hello.txt", "r");
    if (ptr == NULL)
    {
    printf("Error While opening file");
    exit(1);
    }
    if(fgets(input, 80, ptr) != NULL)
    {
    puts(input);
    }
    fclose(ptr);
    printf("\nInput is  : %s ",input);
//    printf("Enter : ");
```

```
//  scanf("%s",input);
    while(input[i]!='\0'){
        switch (state){
            case 0:
                if (input[i]=='a'){
                    state = 1;
                    i++;
                }
                else {
                    state = 2;
                    i++;
                }
                break;
            case 1:
                state = 1;
                i++;
                break;
            case 2:
                state = 2;
                i++;
                break;
        }
    }
    if(state == 1){
        printf("\nThe String is valid.",input);
    }
    else if(state == 2){
        printf("\nThe String is invalid.",input);
    }
    printf("\nState is %d",state);
    return 0;
}
```

Output:



b) Write a program to recognize strings end with 'a'.

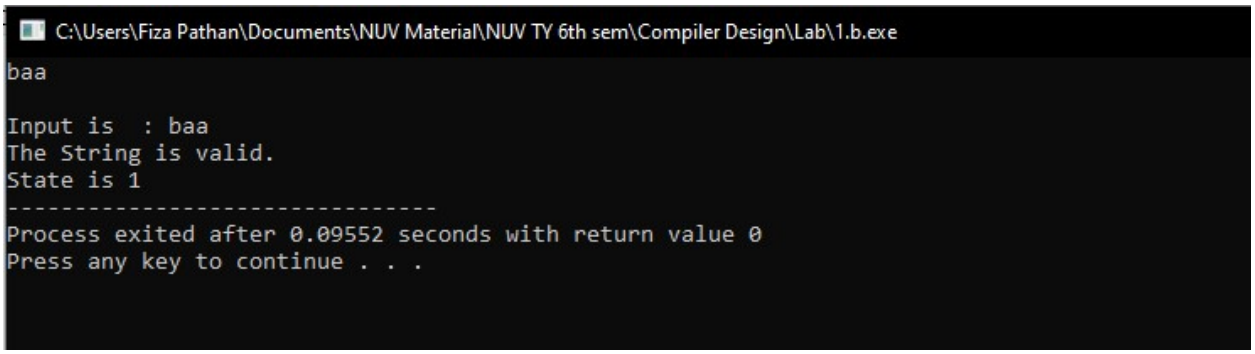
Input:

```
#include<stdio.h>
 #include <stdlib.h>
int main ()
{
                    char input[100];
                    int state=0,i=0;
                    //write a program for string ends with 'a'.
                    FILE* ptr;
                    ptr = fopen("Hello.txt", "r");
                    if (ptr == NULL)
                    {
                    printf("Error While opening file");
                    exit(1);
                    }
                    if(fgets(input, 80, ptr) != NULL)
                    {
                    puts(input);
                    }
                    fclose(ptr);
                    printf("\nInput is  : %s ",input);
//                  printf("Enter : ");
//                  scanf("%s",input);
                    while(input[i]!='\0'){
                     switch (state){
                              case 0:
```

```
                if (input[i]=='a'){
                        state = 1;
                        i++;
                }
                else {
                        state = 0;
                        i++;
                }
                break;
        case 1:
                if (input[i]=='a'){
                state = 1;
                i++;
                }
                else{
                        state = 0;
                        i++;
                }
                break;
        case 2:
                state = 2;
                i++;
                break;
 }
}
if(state == 1){
```

```
        printf("\nThe String is valid.",input);
        }
        else if(state == 0){
        printf("\nThe String is invalid.",input);
        }
        printf("\nState is %d",state);
        return 0;
}
```

Output:



c) Write a program to recognize strings end with 'ab'. Take the input from text file.

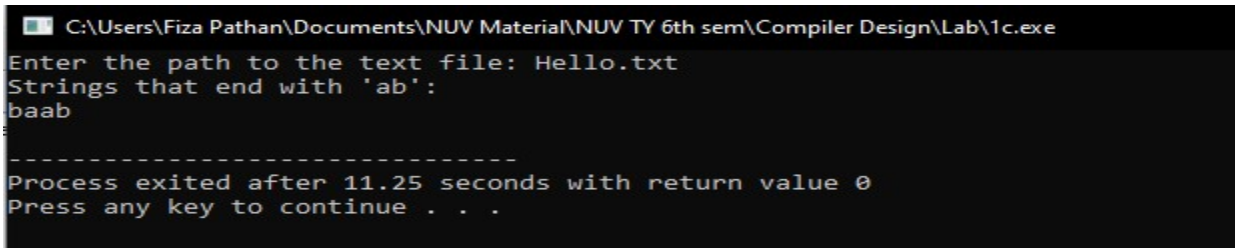
Input:

```
#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';
        }
        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;
}
```

Output:



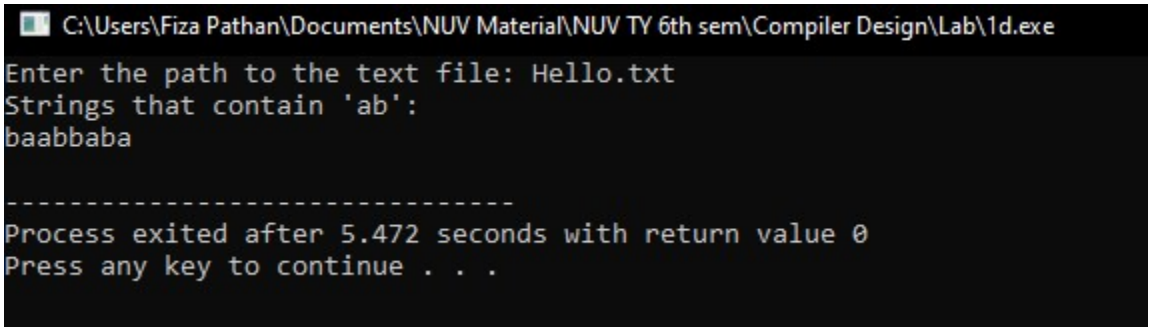
d) Write a program to recognize strings contains 'ab'. Take the input from text file.

Input:

```
#include <stdio.h>

#include <string.h>

#define MAX_LINE_LENGTH 100 // Maximum length for each line

// Function to check strings containing "ab"

void checkStringsContainingAb(const char *filePath) {

    FILE *file = fopen(filePath, "r"); // Open the file in read mode

    if (file == NULL) {

        printf("Error: Could not open file '%s'\n", filePath);

        return; // Exit the function if the file cannot be opened

    }

    char line[MAX_LINE_LENGTH]; // Buffer to hold each line

    printf("Strings that contain 'ab':\n");

    // Read each line from the file

    while (fgets(line, sizeof(line), file) != NULL) {
```

```
        // Remove the 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 contains "ab"
        if (strstr(line, "ab") != NULL) {
            printf("%s\n", line);
        }
    }
    fclose(file); // Close the file
}
int main() {
    char filePath[100];
    // Prompt the user for the input file path
    printf("Enter the path to the text file: ");
    scanf("%s", filePath);
    // Call the function to check strings
    checkStringsContainingAb(filePath);
   return 0;
}
```

Output:


C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\1d.exe
Enter the path to the text file: Hello.txt
Strings that contain 'ab':
baabbaba

Process exited after 5.472 seconds with return value 0
Press any key to continue . . .

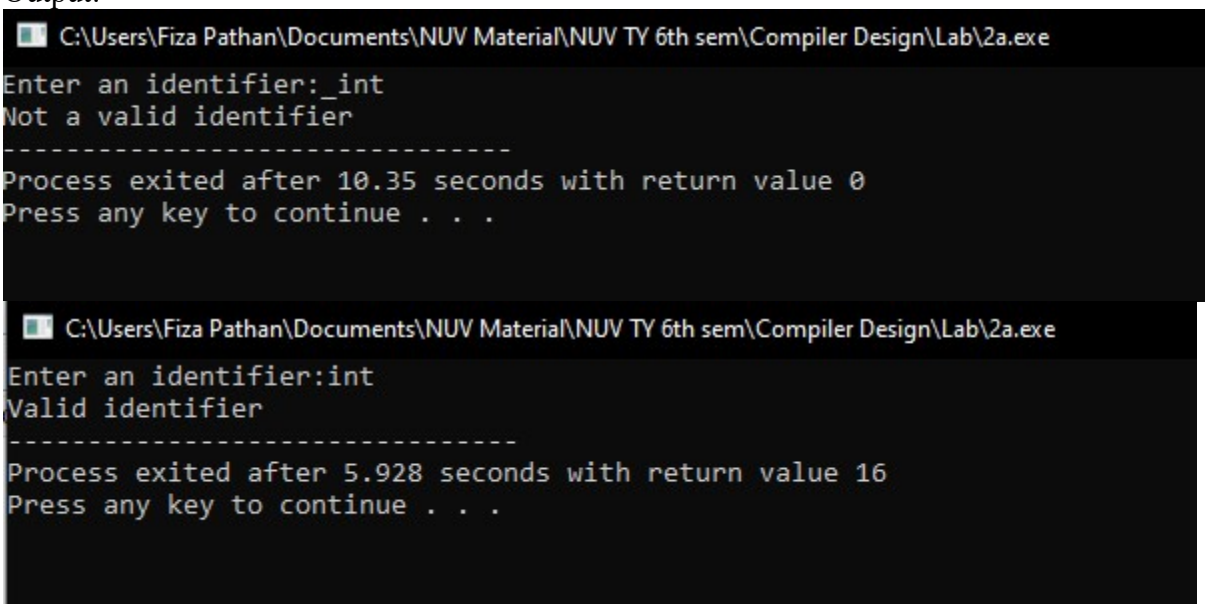
Practical-2

a) Write a program to recognize the valid identifiers.

Input:

```
#include <stdio.h>
#include <ctype.h>
void main()
{
        char a[10];
        int flag, i=1;
        printf("Enter an identifier:");
        scanf("%s",&a);
        if(isalpha(a[0])){

                flag = 1; // If the first character is an alphabet, set flag = 1 (indicating a
valid start).
        }
        else
                printf("Not a valid identifier");
        while (a[i] != '\0') {
      if (!isalnum(a[i]) && a[i] != '_') {
        flag = 0;
        break;
      }
      i++;
   }
        if(flag == 1){
                printf("Valid identifier");
        }
}
```

Output:



b) Write a program to recognize the valid operators.

Input:

```
#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;
```

```
}
```

Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\2b.exe
Enter a potential C 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 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 118.4 seconds with return value 0
Press any key to continue . . .
```

c) Write a program to recognize the valid Unsigned number.

Input:

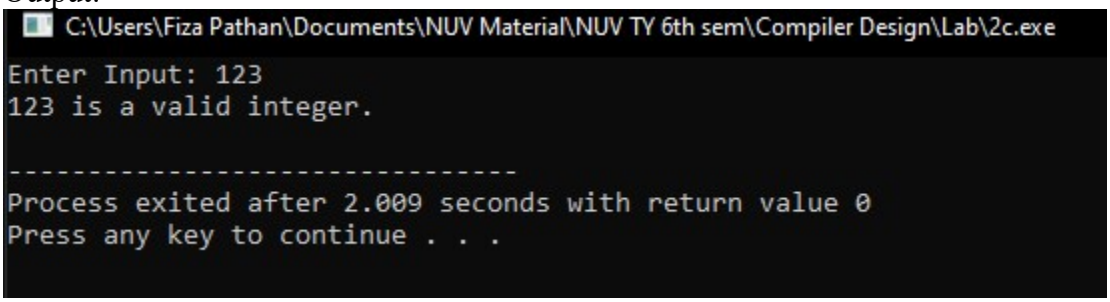
```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
int main() {
   char buffer[100]; // Buffer to store each number
   int i = 0, f = 0, state = 0;
   char lexeme[100];
   printf("Enter Input: ");
   scanf("%s",&buffer);
      while (buffer[f] != '\0') {
         switch (state) {
            case 0:
               if (isdigit(buffer[f])) {
```

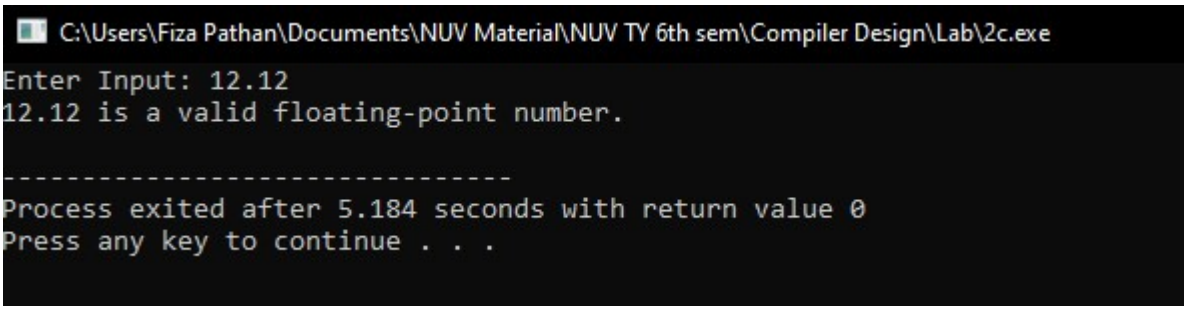
```
            state = 1;
            lexeme[i++] = buffer[f];
        } else {
            state = 99;
        }
        break;
    case 1:
        if (isdigit(buffer[f])) {
            state = 1;
            lexeme[i++] = buffer[f];
        } else if (buffer[f] == '.') {
            state = 2;
            lexeme[i++] = buffer[f];
        } else if (buffer[f] == 'e' || buffer[f] == 'E') {
            state = 4;
            lexeme[i++] = buffer[f];
        } else {
            lexeme[i] = '\0';
            printf("%s is a valid integer.\n", lexeme);
        }
        break;
    case 2:
        if (isdigit(buffer[f])) {
            state = 3;
            lexeme[i++] = buffer[f];
        } else {
            lexeme[i] = '\0';
            printf("%s is an invalid number.\n", lexeme);
        }
        break;
```

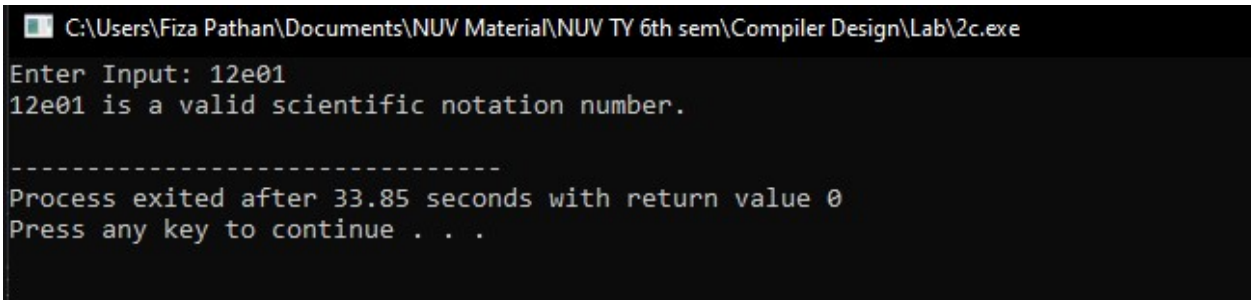
```
case 3:
   if (isdigit(buffer[f])) {
      state = 3;
      lexeme[i++] = buffer[f];
   } else if (buffer[f] == 'e' || buffer[f] == 'E') {
      state = 4;
      lexeme[i++] = buffer[f];
   } else {
      lexeme[i] = '\0';
      printf("%s is a valid floating-point number.\n", lexeme);
   }
   break;
case 4:
   if (isdigit(buffer[f])) {
      state = 6;
      lexeme[i++] = buffer[f];
   } else if (buffer[f] == '+' || buffer[f] == '-') {
      state = 5;
      lexeme[i++] = buffer[f];
   } else {
      lexeme[i] = '\0';
      printf("%s is an invalid number.\n", lexeme);
   }
   break;
case 5:
   if (isdigit(buffer[f])) {
      state = 6;
      lexeme[i++] = buffer[f];
   } else {
      lexeme[i] = '\0';
```

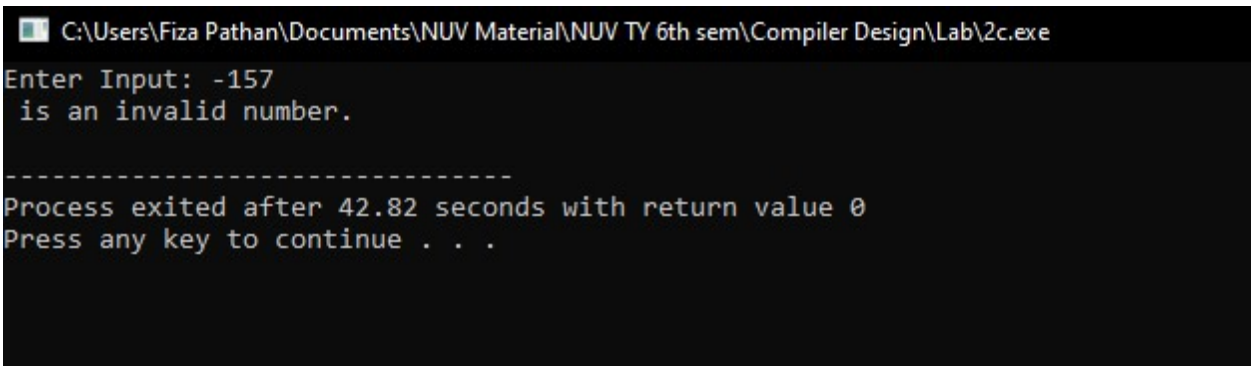
```
                printf("%s is an invalid number.\n", lexeme);
            }
            break;
        case 6:
            if (isdigit(buffer[f])) {
                state = 6;
                lexeme[i++] = buffer[f];
            } else {
                lexeme[i] = '\0';
                printf("%s is a valid scientific notation number.\n", lexeme);
            }
            break;
    }
    f++;
}
// Print classification based on final state
lexeme[i] = '\0';
if (state == 1) {
    printf("%s is a valid integer.\n", lexeme);
} else if (state == 3) {
    printf("%s is a valid floating-point number.\n", lexeme);
} else if (state == 6) {
    printf("%s is a valid scientific notation number.\n", lexeme);
} else {
    printf("%s is an invalid number.\n", lexeme);
}
return 0;
}
```

Output:









d) Write a program to recognize the valid comments. Take the input from text file.

Input:

//accept only comments single line and multiline both.

#include<stdio.h>

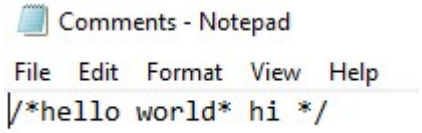
```
int main(){
                        char input[100];
                        int state =0, i=0;
                        FILE *file;
                        file = fopen("Comments.txt","r");
                        if(file==NULL){
                         printf("Error: Couldn't open the file.\n");
                         return 1;
                        }
                        if(fgets(input,sizeof(input),file)==NULL){
                         printf("Error: Couldn't read the file or file is empty.");
                         fclose(file);
                         return 1;
                        }
                        fclose(file);
                        for (i = 0; input[i] != '\0'; i++) {
    if (input[i] == '\n') {
        input[i] = '\0';
        break;
    }
  }
  i = 0;
                        while(input[i]!='\0'){
                         switch(state){
                                 case 0:
                                         if(input[i]=='/')state = 1;
                                         else state =3;
```

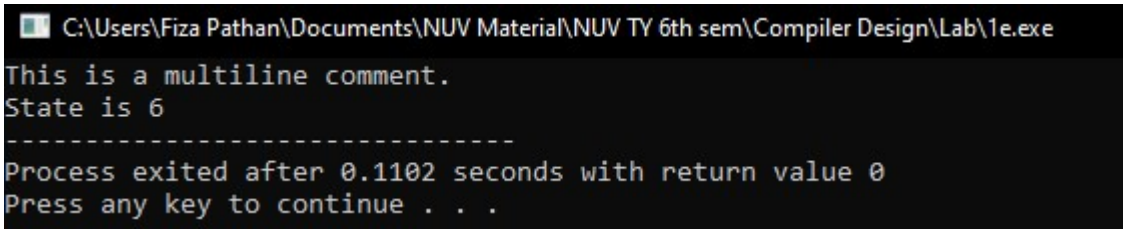
```
                break;
        case 1:
                if(input[i]=='/') state=2;
                else if(input[i]=='*') state =4;
                else state=3;
                break;
        case 2:
                state = 2;
                break;
        case 3:
                state =3;
                break;
        case 4:
                if(input[i]='*')state=5;
                else state=4;
                break;
        case 5:
                if(input[i]=='/') state =6;
                else state = 4;
                break;
        case 6:
                state = 3;
                break;
 }
 i++;
}
if(state==0){
```

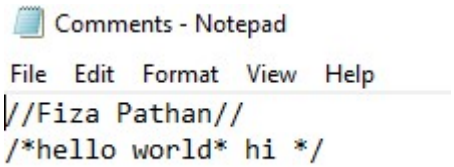
```
 printf("This is not a comment.");
 printf("\nState is %d",state);
}
else if(state==1){
 printf("This is not a comment.");
 printf("\nState is %d",state);
}
else if(state==2){
 printf("This is a single line comment.");
 printf("\nState is %d",state);
}
else if(state==3){
 printf("This is not a comment.");
 printf("\nState is %d",state);
}
else if(state==4){
 printf("This is not a comment.");
 printf("\nState is %d",state);
}
else if(state==5){
 printf("This is not a comment.");
 printf("\nState is %d",state);
}
else if(state==6){
 printf("This is a multiline comment.");
 printf("\nState is %d",state);
}
```

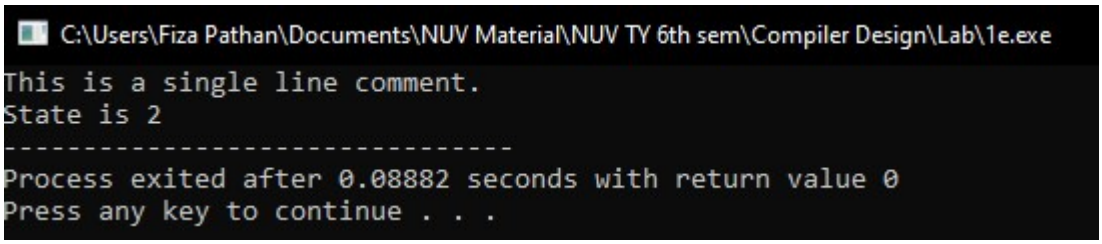
```
        return 0;
}
```

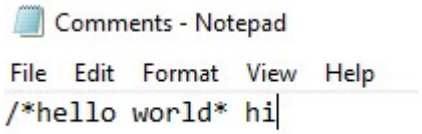
Output:

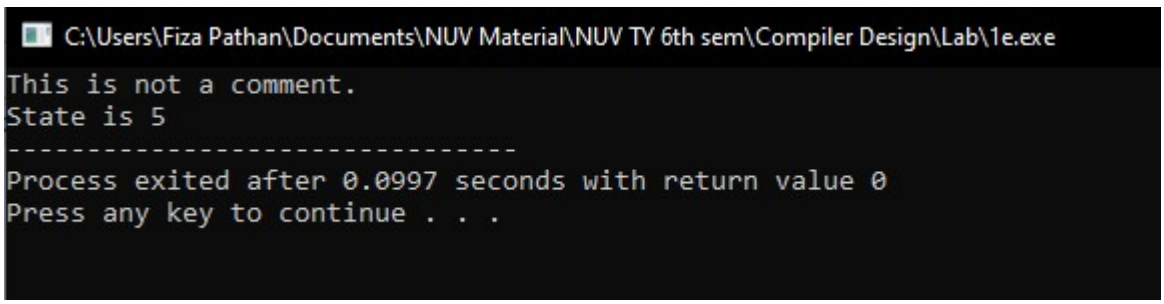












e) Program to implement Lexical Analyzer.

Input:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define BUFFER_SIZE 1000
void check_keyword_or_identifier(char *lexeme);
void recognize_number(char *lexeme);
void recognize_operator(char c);
void recognize_comment(char *buffer, int *index);
int main() {
   FILE *f1;
   char *buffer;
   char lexeme[50];
   char c;
   int i = 0, f = 0, state = 0;
   f1 = fopen("Lexical_Analyzer.txt", "r");
   if (f1 == NULL) {
      printf("Error: Could not open Lexical_Analyzer.txt\n");
   return 0;
   }
   fseek(f1, 0, SEEK_END);
   long file_size = ftell(f1);
   rewind(f1);
   buffer = (char *)malloc(file_size + 1);
   fread(buffer, 1, file_size, f1);
   buffer[file_size] = '\0';
   fclose(f1);
   while (buffer[f] != '\0') {
      c = buffer[f];
```

```
switch (state) {
    case 0:
        if (isalpha(c) || c == '_') {
            state = 1;
            lexeme[i++] = c;
        }
        else if (isdigit(c)) {
            state = 2;
            lexeme[i++] = c;
        }
        else if (c == '/' && (buffer[f + 1] == '/' || buffer[f + 1] == '*')) {
            recognize_comment(buffer, &f);
            state = 0;
        }
        else if (strchr("+-*/%=<>!", c)) {
            recognize_operator(c);
            state = 0;
        }
        else if (strchr(";,{}()", c)) {
            printf("%c is a symbol\n", c);
            state = 0;
        }
        else if (isspace(c)) {
            state = 0;
        }
        break;
    case 1:
        if (isalnum(c) || c == '_') {
            lexeme[i++] = c;
```

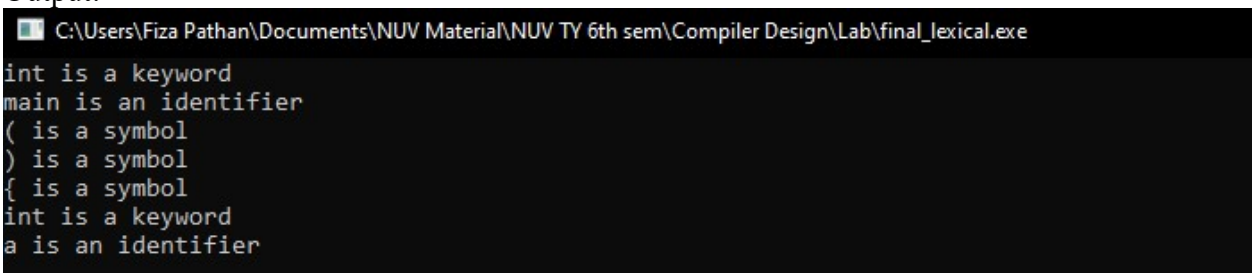
```
    } else {
        lexeme[i] = '\0';
        check_keyword_or_identifier(lexeme);
        i = 0;
        state = 0;
        f--;
    }
    break;
case 2:
    if (isdigit(c)) {
        lexeme[i++] = c;
    } else if (c == '.') {
        state = 3;
        lexeme[i++] = c;
    } else if (c == 'E' || c == 'e') {
        state = 4;
        lexeme[i++] = c;
    } else {
        lexeme[i] = '\0';
        recognize_number(lexeme);
        i = 0;
        state = 0;
        f--;
    }
    break;
case 3:
    if (isdigit(c)) {
        lexeme[i++] = c;
    } else {
        lexeme[i] = '\0';
```

```
            recognize_number(lexeme);
            i = 0;
            state = 0;
            f--;
        }
        break;
    case 4:
        if (isdigit(c) || c == '+' || c == '-') {
            state = 5;
            lexeme[i++] = c;
        } else {
            lexeme[i] = '\0';
            recognize_number(lexeme);
            i = 0;
            state = 0;
            f--;
        }
        break;
    case 5:
        if (isdigit(c)) {
            lexeme[i++] = c;
        } else {
            lexeme[i] = '\0';
            recognize_number(lexeme);
            i = 0;
            state = 0;
            f--;
        }
        break;
}
```

```
        f++;
    }
    free(buffer);
}
void check_keyword_or_identifier(char *lexeme) {
    int i =0;
          char *keywords[] = {
        "auto", "break", "case", "char", "const", "continue", "default", "do",
        "double", "else", "enum", "extern", "float", "for", "goto", "if",
        "inline", "int", "long", "register", "restrict", "return", "short", "signed",
        "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned",
        "void", "volatile", "while"
    };
    for (i = 0; i < 32; i++) {
        if (strcmp(lexeme, keywords[i]) == 0) {
            printf("%s is a keyword\n", lexeme);
            return;
        }
    }
    printf("%s is an identifier\n", lexeme);
}
void recognize_number(char *lexeme) {
    printf("%s is a valid number\n", lexeme);
}
void recognize_operator(char c) {
    char operators[][3] = {"+", "-", "*", "/", "%", "=", "==", "!=", "<", ">", "<=", ">="};
    char next = getchar();
    char op[3] = {c, next, '\0'};
          int i = 0;
    for (i = 0; i < 12; i++) {
```

```
        if (strcmp(op, operators[i]) == 0) {
            printf("%s is an operator\n", op);
            return;
        }
    }
    printf("%c is an operator\n", c);
    ungetc(next, stdin);
}
void recognize_comment(char *buffer, int *index) {
    if (buffer[*index] == '/' && buffer[*index + 1] == '/') {
        printf("// is a single-line comment\n");
        while (buffer[*index] != '\n' && buffer[*index] != '\0') (*index)++;
    }
    else if (buffer[*index] == '/' && buffer[*index + 1] == '*') {
        printf("/* is the start of a multi-line comment\n");
        (*index) += 2;
        while (!(buffer[*index] == '*' && buffer[*index + 1] == '/') && buffer[*index] !=
'\0') (*index)++;
        if (buffer[*index] == '*' && buffer[*index + 1] == '/') {
            printf("*/ is the end of a multi-line comment\n");
            (*index) += 2;
        }
    }
}
```

Output:



Practical-3

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

Lex (short for Lexical Analyzer) is a tool used to generate scanners (tokenizers). It was originally developed at AT&T Bell Labs. Lex helps in breaking down an input stream into meaningful tokens, which are then used by a parser.

Key Features of Lex:

- Used for pattern matching in text.
- Works with regular expressions to identify tokens.
- Generates a C program (lex.yy.c) that performs the scanning.
- Typically used with YACC (Yet Another Compiler Compiler) for parsing.

Flex (Fast Lexical Analyzer) is an improved and faster version of Lex. It generates more efficient and portable C code than Lex and is widely used in modern applications.

Key Features of Flex:

- Faster and more efficient than Lex.
- Compatible with Lex (so you can use Lex code in Flex).
- Generates a C file (lex.yy.c) containing the scanner logic.
- Works with Bison (GNU's YACC equivalent) for building compilers.

How Lex/Flex Works?

Write a Lex/Flex program (file.l) containing patterns and associated actions.

1. Generate the scanner code by running: flex file.l
2. This creates a C source file: lex.yy.c.
3. Compile the generated C code with a C compiler like GCC: gcc lex.yy.c
4. Run the scanner on input text or directly run c file by file.exe: ./scanner < input.txt

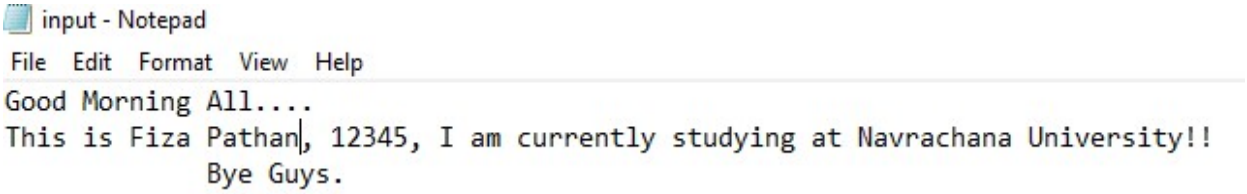
Applications of Lex & Flex

- Compilers & Interpreters (Tokenizing programming languages)
- Text Processing (Extracting words, numbers, patterns)
- Log Analysis (Processing system logs)
- Natural Language Processing (NLP) (Tokenizing sentences)
- Data Validation (Checking input formats)

Sample Program:

```
%{
   #include<stdio.h>
   int letters=0;
%}
%%
[a-zA-Z]  {letters++;}
\n  ;
.  ;
%%
void main(){
   yyin=fopen("input.txt","r");
   yylex();
   printf("This file is containing %d letters",letters);
}
int yywrap(){  return(1);}
```

input.txt File:



Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_Sample>flex sample.l

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_Sample>gcc lex.yy.c

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_Sample>a.exe
This file is containing 83 letters
```

Practical-4

Implement following programs using Lex.

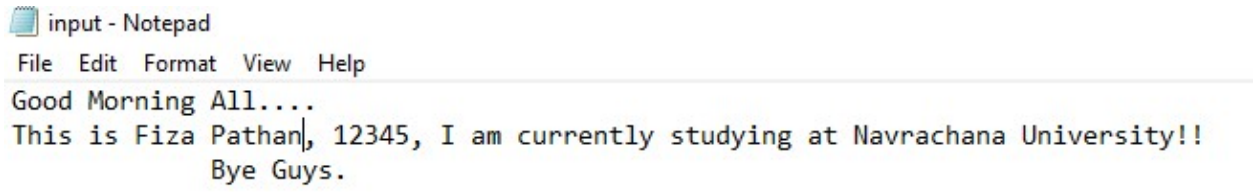
a) **Write a Lex program to take input from text file and count no of characters, no. of lines & no. of words.**

Input:

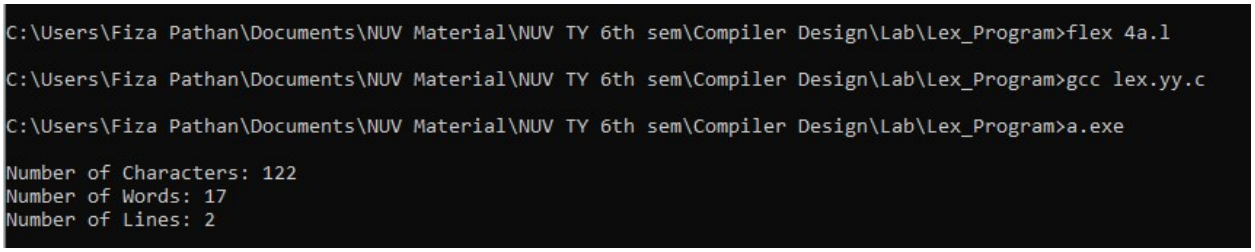
```
%{
#include <stdio.h>
int char_count = 0, word_count = 0, line_count = 0;
%}
%%
\n          { line_count++; char_count++; }
[^\n\t ]+   { word_count++; char_count += yyleng; }
.           { char_count++; }
%%
int main() {
   FILE *file = fopen("input.txt", "r");  // Open the file
   if (!file) {
      printf("Error: Could not open file 'input.txt'\n");
      return 1;
   }
   yyin = file;  // Set Lex input to the file
   yylex();      // Process the file
   printf("\nNumber of Characters: %d", char_count);
   printf("\nNumber of Words: %d", word_count);
   printf("\nNumber of Lines: %d\n", line_count);
   fclose(file);  // Close the file
   return 0;
}
int yywrap() {
   return 1;
```

```
}
```

Input.txt file:



Output:



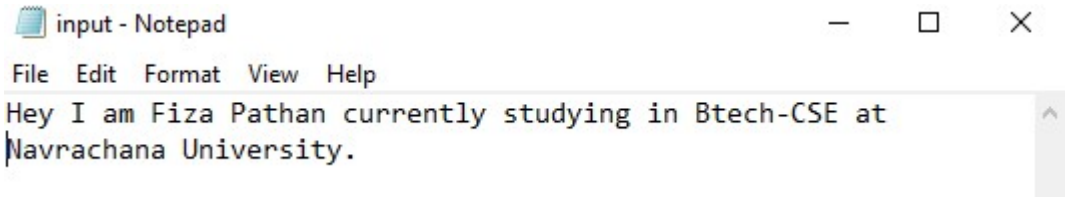
b) Write a Lex program to take input from text file and count number of vowels and consonants.

Input:

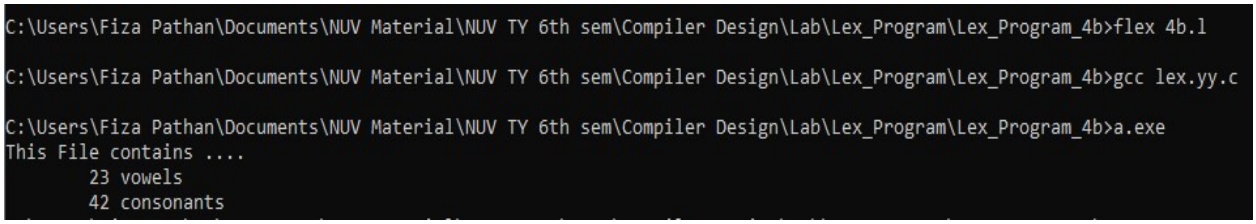
```
%{
#include <stdio.h>
int consonants=0, vowels=0;
%}
%%
[aeiouAEIOU]  { vowels++; }
[a-zA-Z]  {consonants++;}
\n      ;
.       ;
%%
int main() {
   FILE *file = fopen("input.txt", "r");  // Open the file
   if (!file) {
      printf("Error: Could not open file 'input.txt'\n");
```

```
        return 1;
    }
    yyin = file;  // Set Lex input to the file
    yylex();      // Process the file
    printf("This File contains ....");
    printf("\n\t%d vowels",vowels);
    printf("\n\t%d consonants", consonants);
    fclose(file);  // Close the file
    return 0;
}
int yywrap() {
    return 1;
}
```

Input.txt file:



Output:



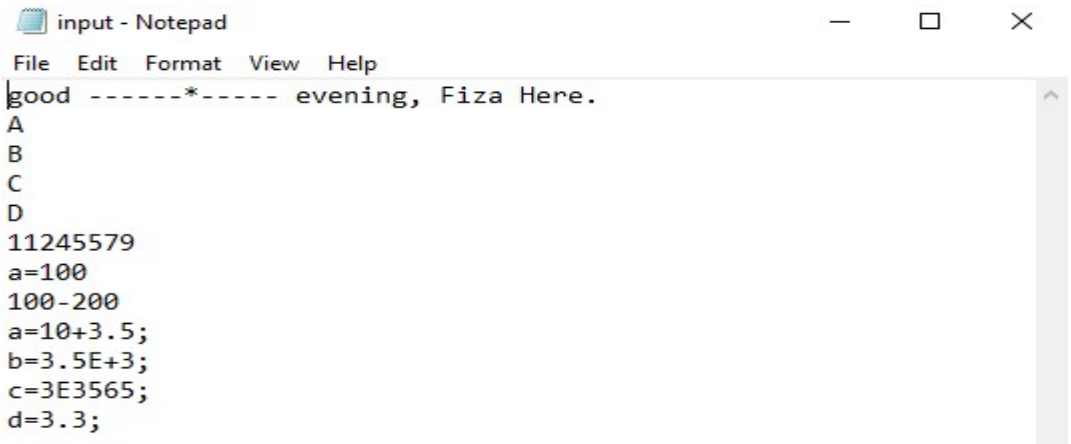
c) Write a Lex program to print out all numbers from the given file.

Input:

```
%{
    #include <stdio.h>
    int numbers = 0;
%}
```

```
%%
[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?    { printf("\n%s is a valid number.", yytext); numbers++;
}
\n                              ;
.                               ;  /* Ignore other characters */
%%
int main()
{
   yyin = fopen("input.txt", "r");
   if (!yyin) {
      perror("Error opening file");
      return 1;
   }
   yylex();  // Process input file
   fclose(yyin); // Close file
   return 0;
}
int yywrap()
{
   return 1;
}
```

Input.txt file:



Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_4c>flex 4c.l

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_4c>gcc lex.yy.c

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_4c>a.exe

11245579 is a valid number.
100 is a valid number.
100 is a valid number.
200 is a valid number.
10 is a valid number.
3.5 is a valid number.
3.5E+3 is a valid number.
3E3565 is a valid number.
3.3 is a valid number.
```

d) Write a Lex program which adds line numbers to the given file and display the same into different file.

Input:

```
%{
   int line_number = 1;
%}
%%
.*  {fprintf(yyout,"%d : %s ",line_number,yytext);line_number++;}
%%
int main()
{
   yyin = fopen("input.txt","r");
   yyout=fopen("op.txt","w");
   yylex();
   printf("done");
   return 0;
}
int yywrap()
{
   return(1);
}
```

Input.txt file:

input - Notepad

File Edit Format View Help

```
good ------*----- evening, Fiza Here.
A
B
C
D
11245579
a=100
100-200
a=10+3.5;
b=3.5E+3;
c=3E3565;
d=3.3;
```

op.txt file:

op - Notepad

File Edit Format View Help

```
1 : good ------*----- evening, Fiza Here.
2 : A
3 : B
4 : C
5 : D
6 : 11245579
7 : a=100
8 : 100-200
9 : a=10+3.5;
10 : b=3.5E+3;
11 : c=3E3565;
12 : d=3.3;
```

Output:

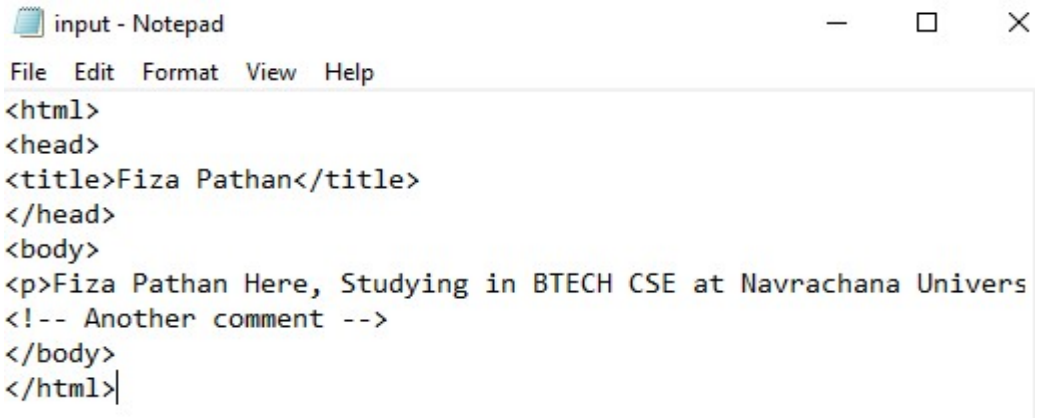
```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_program_4d>flex 4d.l

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_program_4d>gcc lex.yy.c

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_program_4d>a.exe
done
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_program_4d>
```

e) Write a Lex program to printout all markup open tags and HTML comments in file.

Input:

```
%{
#include <stdio.h>
int num=0;
%}
%%
"<"[A-Za-z0-9]+">" printf("%s is valid markup tag\n",yytext);
"<!--"(.|\n)*"-->" num++;
\n      ;
.       ;
%%
int main() {
   yyin = fopen("input.txt","r");
   yylex();
   printf("%d comment",num);
   return 0;
}
int yywrap() {
   return 1;
}
```

Input.txt file:



Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_4e>a.exe
<html> is valid markup tag
<head> is valid markup tag
<title> is valid markup tag
<body> is valid markup tag
<p> is valid markup tag
1 comment
```

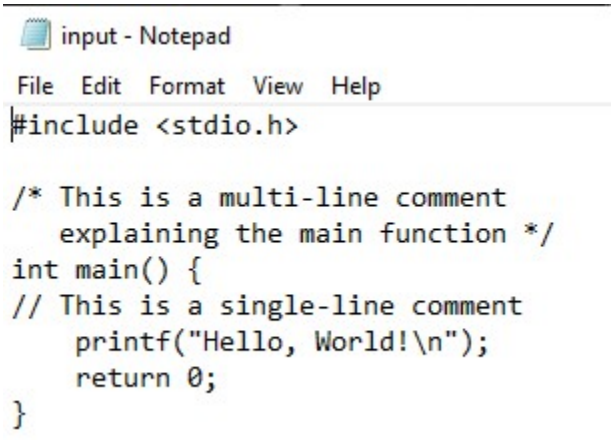
Practical-5

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.

Input:

```
%{
#include <stdio.h>
int c = 0;
%}
%%
"/*"[^*/]*"*/"  {fprintf(yyout, "");c++;}
"//".*  {fprintf(yyout,  " ");c++;}
.* fprintf(yyout, "%s" ,yytext);
%%
int main() {
   yyin = fopen("input.txt","r");
   yyout = fopen("output.txt", "w");
   yylex();
   printf("%d Comments", c);
}
int yywrap()  {return(1);}
```

input.txt:



Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_5a>flex 5a.l

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_5a>gcc lex.yy.c

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_5a>a.exe
2 Comments
```

Output.txt:

output - Notepad

File Edit Format View Help

```
#include <stdio.h>


int main() {

    printf("Hello, World!\n");
    return 0;
}
```

b) Write a Lex program to recognize keywords, identifiers, operators, numbers, special symbols, literals from a given C program.

Input:

```
%{
#include<stdio.h>
%}
digits [0-9]+
%%
if|else|while|do|switch|case|return|int  {printf("<%s, Keyword>\n", yytext);}
[a-zA-Z_][a-zA-Z0-9_]*  {printf("<%s, Identifier>\n", yytext);}
{digits}(\.{digits})?([Ee][+-]?{digits})?  {printf("<%s, Number>\n", yytext);}
[!@*&^%()<>,;={}.\[\]] {printf("<%s, special symbol>\n", yytext);}
[ \t\n]+   { /* Ignore Whitespace */ }
"/*"[^*/"]+"*/" ;  { /* Ignore C-style comments */ }
"//"[^\n]+ ; { /* Ignore C++-style comments */ }
\"[^\"]+\"  {printf("%s, string constant\n", yytext);}
. {printf("'%s' Not Recognized\n", yytext);}
%%
```

```
int main() {
   yyin = fopen("input.txt", "r");
   yylex();
   return 0;
}
int yywrap() { return 1; }
```

input.txt:

input - Notepad

File Edit Format View Help

```
#include <stdio.h>

/* This is a multi-line comment
   explaining the main function */
int main() {
    // This is a single-line comment
    printf("Hello, World!\n");
    return 0;
}
```

Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_5b>flex 5b.l

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_5b>gcc lex.yy.c

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Lex_Program\Lex_Program_5b>a.exe
'#' Not Recognized
<include, Identifier>
<<, special symbol>
<stdio, Identifier>
<., special symbol>
<h, Identifier>
<>, special symbol>
<int, Keyword>
<main, Identifier>
<(, special symbol>
<), special symbol>
<{, special symbol>
<printf, Identifier>
<(, special symbol>
"Hello, World!\n", string constant
<), special symbol>
<;, special symbol>
<return, Keyword>
<0, Number>
<;, special symbol>
<}, special symbol>
```

Practical-6

a) **Program to implement Recursive Descent Parsing in C.**

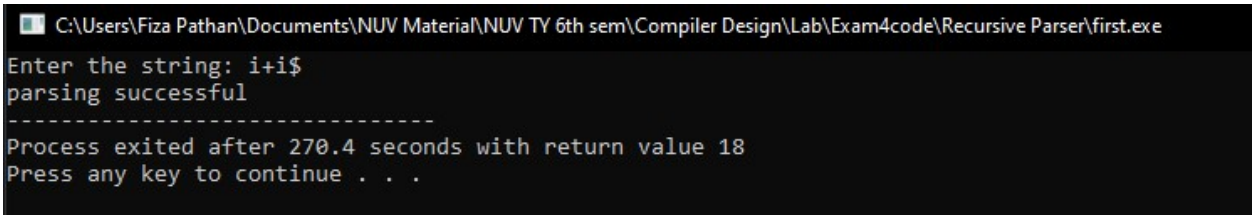
Input:

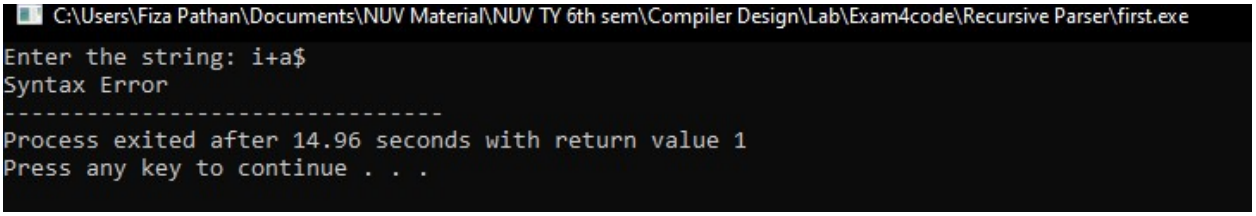
```
#include <stdio.h>
#include <stdlib.h>
char s[20];
int i = 1;
char l;
int match(char l);
int E1();
int E()
{
   if (l == 'i')
   {
      match('i');
      E1();
   }
   else
   {
      printf("Error parsing string");
      exit(1);
   }
   return 0;
}
int E1()
{
   if (l == '+')
   {
      match('+');
      match('i');
```

```
        E1();
    }
    else
    {
        return 0;
    }
}
int match(char t)
{
    if (l == t)
    {
        l = s[i];
        i++;
    }
    else
    {
        printf("Syntax Error");
        exit(1);
    }
    return 0;
}
void main()
{
    printf("Enter the string: ");
    scanf("%s", &s);
    l = s[0];
    E();
    if (l == '$')
    {
        printf("parsing successful");
```

```
    }
    else
    {
        printf("Error while parsing the string\n");
    }
}
```

Output:





Practical-7

a) To Study about Yet Another Compiler-Compiler(YACC).

- YACC (Yet Another Compiler-Compiler) is a tool that generates a parser in C for context-free grammars using LALR(1) parsing. It takes a grammar specification (e.g., input.y) with rules and actions, producing a parser (y.tab.c) that processes tokens from a lexical analyzer (often Lex). The grammar for the provided recursive descent parser (E → i E1, E1 → + i E1 | ε) can be easily written in YACC, automating parsing compared to manual coding.

- YACC File Structure:

```
%{
  C code
%}
Token definitions, %start
%%
Grammar rules: nonterminal : production { C action };
%%
C functions (e.g., main, yyerror)
```

- Key Differences from Recursive Descent:

  YACC: Bottom-up, automated, handles complex LALR(1) grammars.
  Recursive Descent: Top-down, manual, limited to LL(1) grammars.

- Use: Compilers, language processing (e.g., C, Pascal).

b) Create Yacc and Lex specification files to recognizes arithmetic expressions involving +, -, * and / .

Input:

sampleY.y:

```
%{
 #include<stdio.h>
 int yylex(void);
 void yyerror(char *);
%}
%token NUM
%token id
%left '+' '-' '*' '/'
```

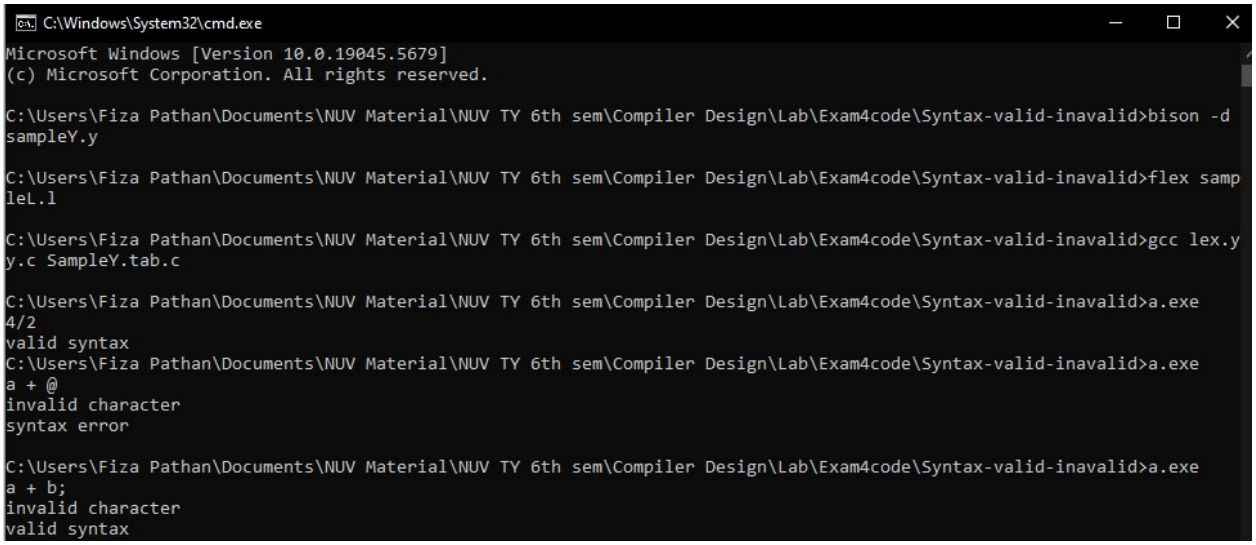
```
%%
S : E '\n' { printf("valid syntax");return 0; }
E : E '+' E { }
  | E '-' E { }
  | E '*' E { }
  | E '/' E { }
  | NUM { }
  | id { }
%%
void yyerror(char *s) {
 fprintf(stderr, "%s\n", s);
}
int main() {
 yyparse();return 0;
}
```

sampleL.l:

```
%{
#include <stdlib.h>
void yyerror(char *);
#include "sampleY.tab.h"
%}
%%
[0-9]+  return NUM;
[a-zA-Z_][a-zA-Z0-9_]* return id;
[+\-*/\n] return *yytext;
[ \t] ;
. yyerror("invalid character");
%%
int yywrap() {
 return 1;
```

```
}
```

Output:



c) Create Yacc and Lex specification files are used to generate a calculator which accepts integer type arguments.

Input:

Yacc.y:

```
%{
 #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; }
T : T '/' F { $$ = $1 / $3; }
```

```
 | F        { $$ = $1; }
F:NUM    { $$ = $1; }
%%
void yyerror(char *s) {
 fprintf(stderr, "%s\n", s);
}
int main() {
 yyparse();
 return 0;
}
```

Lex.l:

```
%{
#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;
}
```

Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7a>flex lex.l

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7a>gcc lex.yy.c yacc.tab.c

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7a>a.exe
4/2
2

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7a>a.exe
11*10+2
112

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7a>a.exe
10-4/2
8
```

d) Create Yacc and Lex specification files are used to convert infix expression to postfix expression.

Input:

7d.l

```
%{
   #include <stdlib.h>
   #include "7d.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;
}
```

7d.y

```
%{
   #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) {
 fprintf(stderr, "%s\n", s);
}
int main() {
 yyparse();
 return 0;
}
```

Output:

```
C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7d>bison -d 7d.y

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7d>flex 7d.l

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7d>gcc lex.yy.c 7d.tab.c

C:\Users\Fiza Pathan\Documents\NUV Material\NUV TY 6th sem\Compiler Design\Lab\Bison_Program\7d>a.exe
2+3*4
234*+
```

e) Write a Program for Three-Address-Code generation.

Input:

n.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
extern FILE *yyin;
extern FILE *yyout;
int yylex(void);
void yyerror(char *);
char* newTemp();
int tempCount = 1;
%}
%union {
   char* str;
}
%token <str> ID
%token <str> NUMBER
%type <str> E T F
%%
S : ID '=' E ';' {
       fprintf(yyout,"%s = %s\n", $1, $3);
   }
 ;
```

```
E : E '+' T {
        char* temp = newTemp();
        fprintf(yyout,"%s = %s + %s\n", temp, $1, $3);
        $$ = temp;
    }
  | T {
        $$ = $1;
    }
  ;
T : T '*' F {
        char* temp = newTemp();
        fprintf(yyout,"%s = %s * %s\n", temp, $1, $3);
        $$ = temp;
    }
  | F {
        $$ = $1;
    }
  ;
F : ID {
        $$ = $1;
    }
  | NUMBER {
        char* temp = newTemp();
        fprintf(yyout,"%s = %s\n", temp, $1);
        $$ = temp;
    }
  ;
%%
char* newTemp() {
    char buffer[10];
```

```
   sprintf(buffer, "t%d", tempCount++);
   return strdup(buffer);
}
void yyerror(char *s) {
   fprintf(stderr, "Error: %s\n", s);
}
int main() {
yyin= fopen("input.txt","r");
yyout= fopen("output.txt","w");
   yyparse();
   return 0;
}
```

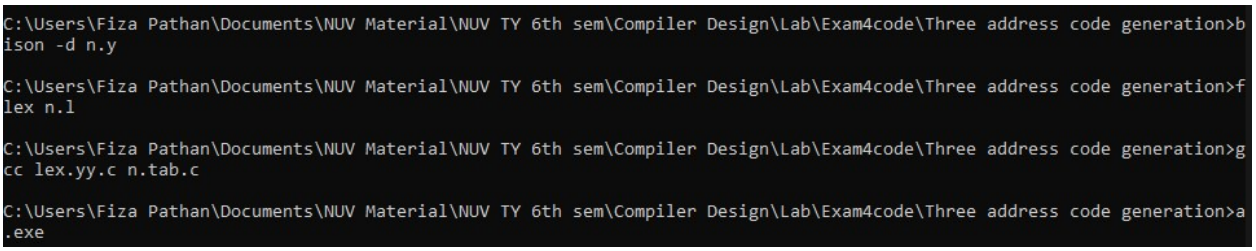
n.l:

```
%{
#include "n.tab.h"
#include <string.h>
%}
%%
[0-9]+       { yylval.str = strdup(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]*   { yylval.str = strdup(yytext); return ID; }
"="          return '=';
";"          return ';';
"+"          return '+';
"*"          return '*';
[ \t\n]      ;   // skip whitespace
.            { printf("Unknown character: %s\n", yytext); }

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

```
}
```

Input.txt:

Output:



Output.txt: