

LAB MANUAL
of
Compiler Design Laboratory
(CSE606)
By
HardevSinh Parmar(22000834)
From
BTech. CSE, Semester 6th
Course In-charge: Prof. Vaibhavi Patel



**NAVRACHANA
UNIVERSITY**

a UGC recognized University

**Department of Computer Science and
Engineering**
School Engineering and Technology
Navrachana University, Vadodara
Autumn Semester
(2025)

1. Write a program to recognize the string starting from 'a' over {a,b}. Code:

```
#include<stdio.h>
int main()
{
    char string[100];
    printf("Enter the string: ");
    scanf("%s", string);
    int i=0, state=0;
    while(string[i]!='\0')
    {
        switch(state)
        {
            case 0: if(string[i]=='a') state=1; else
                    if(string[i]=='b')state=2; else
                    state=3;
                    break;
            case 1: if(string[i]=='a')
                    state=1;
                    else if(string[i]=='b')
                    state=1;
                    else
                    state=3;
                    break;
            case 2: if(string[i]=='a')
                    state=2;
                    else if(string[i]=='b')
                    state=2;
                    else
                    state=3;
                    break;
            case 3: break;
        }
        i++;
    }
    if(state==1)
        printf("String accepted\n");
    else if(state==2)
        printf("String not
accepted\n"); else
        printf("String not recognized\n");
}
```

```
return 0;
}
```

Output:

```
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab>p1.exe
Enter the string: abbaab
String accepted

C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab>p1.exe
Enter the string: baab
String not accepted

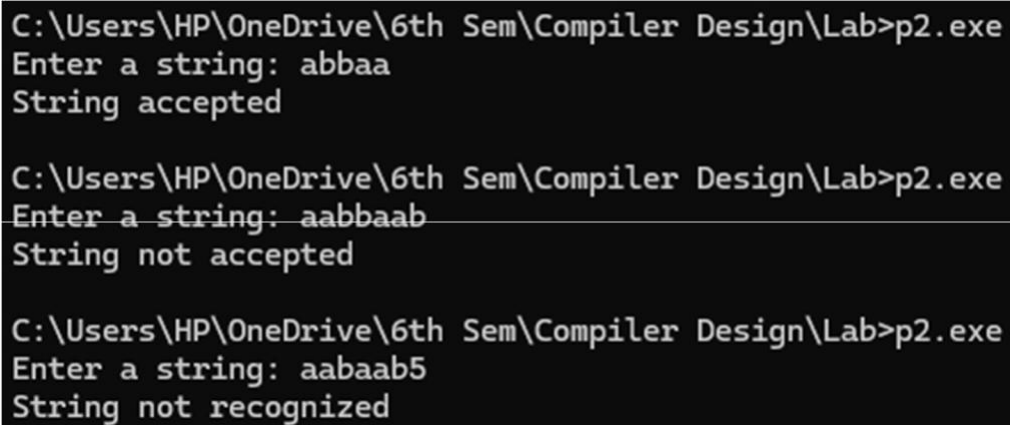
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab>p1.exe
Enter the string: aab4aab
String not recognized
```

2. Write a program to recognize the string ending on 'a' over {a,b}. Code:

```
#include<stdio.h>
int main()
{
    char string[100]; int
    state=0, i=0;
    printf("Enter a string: ");
    scanf("%s", string);

    while (string[i]!='\0')
    {
        switch(state)
        {
            case 0:
                if(string[i]=='a') state=1;
                else if(string[i]=='b') state=0;
                else state=2;
                break;
            case 1:
                if(string[i]=='a') state=1;
                else if(string[i]=='b') state=0;
                else state=2;
                break;
            case 2:
                break;
        }
    }
}
```

```
    }  
    i++;  
}  
if(state==1)  
    printf("String accepted\n");  
else if(state==0)  
    printf("String not accepted\n");  
else  
    printf("String not recognized\n");  
  
return 0;  
}
```

Output:

```
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab>p2.exe  
Enter a string: abbaa  
String accepted  
  
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab>p2.exe  
Enter a string: aabbaab  
String not accepted  
  
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab>p2.exe  
Enter a string: aabaab5  
String not recognized
```

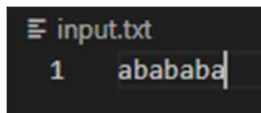
3. Write a program to recognize strings end with 'ab'. Take the input

from text file. Code:

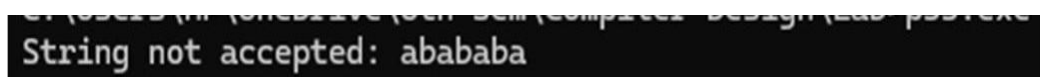
```
#include <stdio.h>  
  
#include <string.h>  
  
int main() {  
  
    char string[100];  
  
    FILE *file = fopen("input.txt",  
  
    "r"); if (file == NULL) {
```

```
printf("Could not open file input.txt\n");  
  
return 1;  
  
}  
  
while (fgets(string, sizeof(string), file))  
{  
  
    int i = 0, state = 0;  
  
    while (string[i] != '\0')  
    {  
  
        switch (state) {  
  
            case 0:  
  
                if (string[i] == 'a') state = 1;  
  
                else if (string[i] == 'b') state = 0;  
  
                else state = 3;  
  
                break;  
  
            case 1:  
  
                if (string[i] == 'a') state = 1;  
  
                else if (string[i] == 'b') state = 2;  
  
                else state = 3;  
  
                break;  
  
            case 2:  
  
                if (string[i] == 'a') state = 1;  
  
                else if (string[i] == 'b') state = 0;  
  
                else state = 3;  
  
                break;  
  
            case 3:
```

```
        state = 3;
        break;
    }
    i++;
}
if (state == 2)
    printf("String accepted: %s\n", string);
else if (state == 0 || state == 1)
    printf("String not accepted: %s\n", string);
else
    printf("String not recognized: %s\n", string);
}
fclose(file);
return 0;
}
```

Input:

```
≡ input.txt
1 abababa|
```

Output:

```
String not accepted: abababa
```



```
String accepted: abababab
```

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

Code:

```
#include<stdio.h>
int main()
{
    char string[100];
    FILE *file = fopen("4th.txt",
    "r"); if (file == NULL) {
        printf("Could not open file
        input.txt\n"); return 1;
    }
    while(fgets(string, sizeof(string), file))
    {
        int i=0, state=0;
        while(string[i]!='\0')
        {
            switch(state)
            {
                case 0: if(string[i]=='a')
                    state=1;
                    else if(string[i]=='b')
                    state=0; break;
                case 1: if(string[i]=='a')
                    state=1;
                    else if(string[i]=='b')
                    state=2; break;
                case 2: if(string[i]=='a')
                    state=1;
                    else if(string[i]=='b')
                    state=2; break;
            }
            i++;
        }
        if(state==2)
            printf("String accepted: %s\n",
            string); else if(state==0 || state==1)
            printf("String not accepted: %s\n",
            string); else
            printf("String not recognized: %s\n", string);
    }
}
```

```
}
```

Input:

```
4th.txt
1 aaababbbab
```

Output:

```
C:\Users\HP\OneDrive\6th Sem\Computer Design\Lab>p4.exe
String accepted: aaababbbab
```

5. Single line comment

Code:

```
#include <stdio.h>

#include <string.h>

int main()
{
    char string[100];
    FILE *file = fopen("comment.txt",
    "r"); if (file == NULL) {
        printf("Could not open file
        input.txt\n"); return 1;
    }
    while (fgets(string, sizeof(string), file))
    {
        int i=0, state=0;
        while(string[i]!='\0')
        {
            switch(state)
            {
                case 0: if(string[i]=='/')
                    state=1; else state=2;
                    break;
                case 1:
                    if(string[i]=='/') state=3;
                    else state=2;
                    break;
                case 2:
```



```
        break;
    case 3:
        break;
    }
    i++;
}
if(state==3)
    printf("Comment valid: %s\n", string);
else
    printf("Comment not valid: %s\n", string);
}
}
```

Output:

```
C:\Users\Hardev Sinh\OneDrive\Documents\Sem-6\CD\CD-LAB\
Comment valid: // my name is hardev.
```

6. Multiline Comment

Code:

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

int main()
{
    char string[1000];
    FILE *file = fopen("comment2.txt",
    "r"); if (file == NULL) {
        printf("Could not open file
        input.txt\n"); return 1;
    }
    while (fgets(string, sizeof(string), file))
    {
        int i=0, state=0;
        while(string[i]!='\0')
        {
            switch(state)
            {
                case 0: if(string[i]=='/')
                    state=1;
```

```
        else state=2;
        break;
    case 1:
        if(string[i]=='/') state=3;
        else if (string[i]=='*') state=4;
        else state=2;
        break;
    case 2:
        break;
    case 3:
        break;
    case 4:
        if (string[i]=='*') state=5;
        else state=4;
    case 5:
        if(string[i]=='/') state=6;
        else state=4;
    case 6:
        break;
    }
    i++;
}
if(state==3)
    printf("Singleline Comment valid: %s\n", string);
else if(state==6)
    printf("Multiline Comment valid: %s\n", string);
else
    printf("Comment not valid: %s\n", string);
}
}
```

Output:

```
C:\Users\Hardev Sinh\OneDrive\Documents\Sem-6\CD\CD-LAB\xy.
multiline Comment valid: /* hello * my name is hardev. */
```

2(a). Write a program to recognize the valid identifiers.

Code:

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

int main()
{
    char string[1000];
    printf("Enter the string: ");
    scanf("%s", string);    // Prevents buffer overflow
    int i = 0, state = 0;
    int num=1;
    while (num>0)
    {
        switch (state)
        {
            case 0: //A
                if (string[i] == 'i') state = 1;
                else if (isalpha(string[i]) || string[i] == '_') state = 5; break;

            case 1: //B
                if (string[i] == 'n') state = 2;
                else if (isalpha(string[i]) || string[i] == '_' ||
isdigit(string[i])) state = 5;
                else state = 6;
                break;

            case 2: //C
                if (string[i] == 't') state = 3;
                else if (isalpha(string[i]) || string[i] == '_' ||
isdigit(string[i])) state = 5;
                else state = 6;
                break;

            case 3: //D
                if (string[i] == '\0') state = 4;
                else if (isalpha(string[i]) || isdigit(string[i]) || string[i] ==
'_' ) state = 5;
                num--;
                break;
```

```
        case 4: //E
            // No transition needed, final accepting state break;

        case 5: //F
            if (isalpha(string[i]) || isdigit(string[i]) || string[i] == '_')
                state = 5;
            else state = 6;    // No need for checking '\0' here, it will end
                                the loop
            break;

        case 6: //G
            // No transition needed, final rejecting state break;

    }
    i++;
}
if (state == 3 || state==4)
    printf("int is keyword.");
else if (state == 6)
    printf("%s is valid identifier.", string);
else
    printf("%s is invalid identifier.", string);

return 0;
}
```

Output:

```
Enter the string: int a
int is keyword.
```

2(b). Write a program to recognize the valid operators.

Code:

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

int main()
{
    char string[1000];
    printf("Enter the string: ");
    scanf("%s", string);    // Prevents buffer overflow
    int i = 0, state = 0;
    int num=1;
    while (num>0)
    {
        switch(state)
        {
            case 0:
                if (string[i] == '+') state = 51;
                else if(string[i] == '*') state = 51;
                else if(string[i] == '/') state = 60;
                else if(string[i] == '=') state = 53;
                else if(string[i] == '-') state = 54;
                else if(string[i] == '?') state = 55;
                else if(string[i] == '<') state = 56;
                else if(string[i] == '>') state = 59;
                else if(string[i] == '!') state = 57;
                else if(string[i] == '&') state = 58;
                else if(string[i] == '|') state = 61;
                else if(string[i] == '~' || string[i] == '^') state = 62; break;

                case 51: // Arithmetic
                    if (string[i] == '\0') state = 51;
                    else if (string[i] == '+') state = 52;
                    else if (string[i] == '=') state = 53;
                    else state = 0;
                    num--;
                    break;

                case 52: // Unary
                    break;

                case 53: // Assignment
```

```
        if(string[i] == '=') state = 56;
        else state = 0;
        num--;
        break;

    case 54: // -
        if (string[i] == '\0') state = 51;
        else if (string[i] == '-') state = 52;
        else if (string[i] == '=') state = 53;
        else state = 0;
        num--;
        break;

    case 55: // Ternary
        if (string[i] == ':') state = 55;
        else if(string[i] == '\0') state = 0;
        else state = 0;
        num--;
        break;

    case 56: //Relational
        if (string[i] == '=') state = 56;
        else if(string[i] == '<') state = 58;
        else state = 0;
        num--;
        break;

    case 59: // >
        if (string[i] == '>') state = 58;
        else if (string[i] == '=') state = 56;
        else state = 0;
        num--;
        break;

    case 57: // Logical
        if (string[i] == '=') state = 56;
        else if(string[i] == '\0') state = 57;
        else state = 0;
        num--;
        break;

    case 58: // Bitwise
        if (string[i] == '&') state = 57;
        else if(string[i] == '\0') state = 58;
        else state = 0;
```

```
        num--;  
        break;  
  
    case 60: // "/"  
        if (string[i] == '\\0') state = 51;  
        else state = 0;  
        num--;  
        break;  
  
    case 61: // "|"  
        if (string[i] == '|') state = 57;  
        else if(string[i] == '\\0') state = 58;  
        else state = 0;  
        num--;  
        break;  
  
    case 62: // "~ ^"  
        if (string[i] == '\\0') state = 58;  
        else state = 0;  
        num--;  
        break;  
    }  
    i++;  
}  
  
if (state == 51) printf("%s is an Arithmetic operator.", string); else if(state == 52)  
printf("%s is an Unary operator.", string); else if(state == 53) printf("%s is an  
Assignment operator.", string); else if(state == 55) printf("%s is ternary or  
conditional operator.",  
string);  
else if(state == 56) printf("%s is a Relational Operator.", string); else if(state ==  
57) printf("%s is a Logical operator.", string); else if(state == 58) printf("%s is a  
Bitwise operator.", string); else  
  
    printf("Processing....");  
  
return 0;  
}
```

Output:

```
Enter the string: <=  
<= is a Relational Operator.
```

2(c). Write a program to recognize the valid number.

Code:

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

int main() {
    FILE *file;
    char buffer[100];
    char lexeme[100];
    char c;
    int f, i, state;

    file = fopen("numbers.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    while (fgets(buffer, 100, file)) {
        buffer[strcspn(buffer, "\n")] = 0;
        f = 0; i = 0;
        state = 0;

        while (buffer[f] != '\0')
            switch (state) {
                case 0:
                    c = buffer[f];
                    if (isdigit(c)) { state = 40; lexeme[i++] = c; }
                    else { state = 0; }
                    break;

                case 40:
                    c = buffer[f];
                    if (isdigit(c)) { state = 40; lexeme[i++] = c; } else if (c == '.') {
                        state = 41; lexeme[i++] = c; }
                    else if (c == 'E' || c == 'e') { state = 43; lexeme[i++] = c; }

                else {
                    lexeme[i] = '\0';
                    printf("%s is a valid number\n", lexeme);
                }
            }
        f++;
    }
}
```



```
        i = 0;
        state = 0;
        f--;
    }
    break;

case 41:
    c = buffer[f];
    if (isdigit(c)) { state = 42; lexeme[i++] = c; } else {

        lexeme[i] = '\0';
        printf("%s is an invalid number (expected digit after
decimal)\n", lexeme);

        i = 0;
        state = 0;
        f--;
    }
    break;

case 42:
    c = buffer[f];
    if (isdigit(c)) { state = 42; lexeme[i++] = c; }
    else if (c == 'E' || c == 'e') { state = 43; lexeme[i++] = c;

    }

    else {
        lexeme[i] = '\0';
        printf("%s is a valid number\n", lexeme);
        i = 0;
        state = 0;
        f--;
    }
    break;

case 43:
    c = buffer[f];
    if (c == '+' || c == '-') { state = 44; lexeme[i++] = c; } else if (isdigit(c))
    { state = 45; lexeme[i++] = c; } else {

        lexeme[i] = '\0';
        printf("%s is an invalid number (expected digit or sign after
'E'/'e')\n", lexeme);

        i = 0;
        state = 0;
        f--;
    }
}
```

```
        break;
    case 44:
        c = buffer[f];
        if (isdigit(c)) { state = 45; lexeme[i++] = c; } else {

            lexeme[i] = '\0';
            printf("%s is an invalid number (expected digit after sign in
exponent)\n", lexeme);

            i = 0;
            state = 0;
            f--;
        }
        break;

    case 45:
        c = buffer[f];
        if (isdigit(c)) { state = 45; lexeme[i++] = c; } else {

            lexeme[i] = '\0';
            printf("%s is a valid number\n", lexeme);
            i = 0;
            state = 0;
            f--;
        }
        break;
    }
    f++;
}

if (state == 40 || state == 41 || state == 42 || state == 45) { lexeme[i] = '\0';

    printf("%s is a valid number\n", lexeme);
} else {
    printf("%s is an invalid number\n", buffer);
}

}

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

Output:

```
31.1e31 is a valid number
123.123 is a valid number
12.42e-2 is a valid number
3E+2 is a valid number
3e2 is a valid number
```

2(d). Write a program to recognize the valid comments.

Code:

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

int main()
{
    char string[1000];
    FILE *file = fopen("comment2.txt", "r");
    if (file == NULL) {
        printf("Could not open file input.txt\n"); return 1;
    }
    while (fgets(string, sizeof(string), file))
    {
        int i=0, state=0;
        while(string[i]!='\0')
        {
            switch(state)
            {
                case 0:
                    if(string[i]=='/') state=1;
                    else state=2;
                    break;
                case 1:
                    if(string[i]=='/') state=3;
                    else if (string[i]=='*') state=4;
                    else state=2;
                    break;
                case 2:
                    break;
                case 3:
                    break;
```

```
        case 4:
            if (string[i]=='*') state=5;
            else state=4;
        case 5:
            if(string[i]=='/') state=6;
            else state=4;
        case 6:
            break;
    }
    i++;
}
if(state==3)
    printf("Singleline Comment valid: %s\n", string); else
if(state==6)
    printf("Multiline Comment valid: %s\n", string); else

    printf("Comment not valid: %s\n", string);
}
return 0;
}
```

Output:

```
C:\Users\Hardev Sinh\OneDrive\Documents\Sem-6\CD\CD-LAB\>
multiline Comment valid: /* hello * my name is hardev. */
```

2(e) . Program to implement Lexical Analyzer.**Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define BUFFER_SIZE 1000
void check(char *lexeme);

int main() {
    FILE *f1;
    char buffer[BUFFER_SIZE], lexeme[50]; // Static buffer for input and lexeme storage

    char c;
    int f = 0, state = 0, i = 0;
    f1 = fopen("Input.txt", "r");
    fread(buffer, sizeof(char), BUFFER_SIZE - 1, f1);
    buffer[BUFFER_SIZE - 1] = '\0'; // Null termination
    fclose(f1);

    while (buffer[f] != '\0') {
        c = buffer[f];
        switch (state) {
            case 0:
                if (isalpha(c) || c == '_') { state = 1;
                    lexeme[i++] = c;
                }
                else if (c == ' ' || c == '\t' || c == '\n') { state = 0;
                }
                else if (isdigit(c)) { state
                    = 13; lexeme[i++] =
                    c;
                }
                else if (c == '/') {
                    state = 11; // For comment
                }
                else if (c == ';' || c == ',' || c == '{' || c == '}') { printf(" %c is a
                    symbol\n", c);
                    state = 0;
                }
                else if (strchr("+-*/= %?<> !& | ~ ^", c)) {
```

```
        state = 50;
        lexeme[i++] = c;
    }
    else {
        state = 0;
    }
    break;

case 1:
    if (isalpha(c) || isdigit(c) || c == '_' ) { state = 1;

        lexeme[i++] = c;
    } else {
        lexeme[i] = '\0'; // Null-terminate the lexeme
        check(lexeme);    // Check if it's a keyword or identifier
        state = 0;
        i = 0;
        f--; // Step back to reprocess the current non-alphanumeric
character
    }
    break;

case 13:
    if(isdigit(c)) {
        state = 13;
        lexeme[i++] = c;
    }
    else if(c=='.') {
        state=14;
        lexeme[i++]=c;
    }
    else if(c=='E'||c=='e') {
        state=16;
        lexeme[i++]=c;
    }
    else {
        lexeme[i]='\0';
        printf("%s is a valid number\n", lexeme);
        i=0;
        state=0;
        f--;
    }
    break;

case 50: // Operator Handling
```

```
switch (lexeme[0]) {  
    case '+':  
        if (c == '+') {  
            printf("%s is a Unary operator\n", lexeme); state  
                = 0;  
        }  
        else if (c == '=') {  
            printf("%s is an Assignment operator\n", lexeme); state  
                = 0;  
        }  
        else {  
            printf("%s is an Arithmetic operator\n", lexeme);  
            state = 0;  
            f--;  
        }  
        break;  
  
    case '-':  
        if (c == '-') {  
            printf("%s is a Unary operator\n", lexeme); state  
                = 0;  
        }  
        else if (c == '=') {  
            printf("%s is an Assignment operator\n", lexeme); state  
                = 0;  
        }  
        else {  
            printf("%s is an Arithmetic operator\n", lexeme);  
            state = 0;  
            f--;  
        }  
        break;  
  
    case '*':  
    case '/':  
    case '%':  
        if (c == '=') {  
            printf("%s is an Assignment operator\n", lexeme); state  
                = 0;  
        }  
        else {  
            printf("%s is an Arithmetic operator\n", lexeme);  
            state = 0;  
            f--;  
        }  
}
```

```
        break;

    case '=':
        if (c == '=') {
            printf("%s is a Relational operator\n", lexeme); state =
                0;
        }

    else {
        printf("%s is an Assignment operator\n", lexeme);
        state = 0;
        f--;
    }
    break;

    case '<':
    case '>':
        if (c == '=' || c == lexeme[0]) {
            printf("%s is a Relational operator\n", lexeme); state =
                0;
        }

    else {
        printf("%s is a Relational operator\n", lexeme);
        state = 0;
        f--;
    }
    break;

    case '!':
    case '&':
    case '|':
        if (c == '=') {
            printf("%s is a Logical operator\n", lexeme); state =
                0;
        }

    else if (c == lexeme[0]) {
        printf("%s is a Logical operator\n", lexeme); state =
            0;
    }

    else {
        printf("%s is a Logical operator\n", lexeme);
        state = 0;
        f--;
    }
    break;
```



```

        case '~':
        case '^':
            printf("%s is a Bitwise operator\n", lexeme);
            state = 0;
            f--;
            break;

        case '?':
            if (c == ':') {
                printf("%s is a Ternary or conditional operator\n",
lexeme);

                state = 0;
            }
            else {
                state = 0;
                f--;
            }
            break;

        default:
            state = 0;
            break;
    }
    lexeme[0] = '\0';
    i = 0;
    break;

    default:
        state = 0;
        break;
    }
    f++;
}

}

void check(char *lexeme) {
    char *keywords[] = {
        "auto", "break", "case", "char", "const", "continue", "default", "do", "double",
        "else", "ef", "extern", "float", "for", "goto", "if",
        "inline", "int", "long", "register", "restrict", "return", "short", "signed", "sizeof", "static",
        "struct", "switch", "typedef", "union", "unsigned",
        "void", "volatile", "while"
    };
};

    for (int 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);  
}
```

Output:

```
int is a keyword  
main is an identifier  
{ is a symbol  
int is a keyword  
a is an identifier  
= is an Assignment operator  
10 is a valid number  
; is a symbol  
int is a keyword  
b is an identifier  
= is an Assignment operator  
20 is a valid number  
int is a keyword  
sum is an identifier  
=um is an Assignment operator  
0 is a valid number  
; is a symbol  
sum is an identifier  
=um is an Assignment operator  
a is an identifier  
+ is an Arithmetic operator  
b is an identifier  
; is a symbol  
printf is an identifier  
sum is an identifier  
a is an identifier  
+ is a Unary operator  
; is a symbol  
a is an identifier  
+ is an Assignment operator  
b is an identifier  
; is a symbol  
} is a symbol  
C is an identifier
```

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.

Code:

```
%{  
  
#include<stdio.h>  
  
int words=0,characters=0,no_of_lines=0;  
  
%}  
  
%%  
  
\n {no_of_lines++,words++;}  
  
. characters++;  
  
[t ]+ words++;  
  
%%  
  
void main(){  
  
yyin = fopen("4_1.txt","r");  
  
yylex();  
  
printf("This file is containing %d words.\n",words); printf("This file is  
  
containing %d characters.\n",characters); printf("This file is  
  
containing %d no_of_lines.\n",no_of_lines);  
  
}  
  
int yywrap(){  
  
return(1);}
```

Output:

```
This file is containing 7 words.  
This file is containing 57 characters.  
This file is containing 7 no_of_lines.
```

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

Code:

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

Output:


```
This file is containing 12 vowels.  
This file is containing 22 consonants.
```

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

Code:

```
%{  
    #include<stdio.h>  
}%  
%%  
[0-9]+(.[0-9]+)?([eE][+-]?[0-9]+)? printf("%s is valid number \n",yytext);  
  
\n    ;  
.  
%%  
  
void main() {  
    yyin = fopen("input.txt","r");  
    yylex();  
}  
int yywrap(){return(1);}
```

Output:



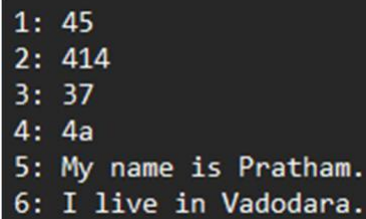
```
45 is valid number  
414 is valid number  
37 is valid number  
4 is valid number
```

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

Code:

```
%{
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);}
```

Ouput:



```
1: 45
2: 414
3: 37
4: 4a
5: My name is Pratham.
6: I live in Vadodara.
```

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

Code:

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

Output:

```
<html> is valid markup tag
<head> is valid markup tag
<title> is valid markup tag
</title> is valid markup tag
</head> is valid markup tag
<body> is valid markup tag
<h1> is valid markup tag
</h1> is valid markup tag
<p> is valid markup tag
</p> is valid markup tag
<p> is valid markup tag
</p> is valid markup tag
</div> is valid markup tag
</body> is valid markup tag
</html> is valid markup tag
2 comment
```

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.

Code:

```
%{
#include <stdio.h>
int comment_count = 0;
FILE *outfile;
%}
%%
"/".*          { comment_count++; /* Skip single-line comment */ }
"/"([^\n]*\n)*?/" { comment_count++; /* Skip multi-line comment */ }
.\n           { fputc(yytext[0], outfile); }
%%
int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Usage: %s <input_file>\n", argv[0]);
        return 1;
    }
    FILE *infile = fopen("sample.c", "r");
    if (!infile) {
        perror("Cannot open input file");
        return 1;
    }
    outfile = fopen("cleaned_code.c", "w");
    if (!outfile) {
        perror("Cannot open output file");
        return 1;
    }
    yyin = infile;
    yylex();
    fclose(infile);
    fclose(outfile);
    printf("Total number of comments: %d\n",
        comment_count); return 0;
}
int yywrap(){
    return 1;
}
```


Sample.c :

```
#include <stdio.h>

int main() {
    // This is a single-line
    comment int x = 10;
    float y = 20.5;

    /*
       This is a multi-line comment
       It should be removed
    */

    if (x < y) {
        printf("x is less than y\n");
    } else {
        printf("x is not less than y\n");
    }

    char c = 'A'; // Character literal
    return 0;
}
```

Output:

```
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab\LEX\5\>a.exe sample.c
Total number of comments: 3
```

```
#include <stdio.h>

int main() {
    int x = 10;
    float y = 20.5;

    if (x < y) {
        printf("x is less than y\n");
    } else {
        printf("x is not less than y\n");
    }

    char c = 'A';
    return 0;
}
```

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

Code:

```
%{  
  
#include <stdio.h>  
  
#include <string.h>  
  
#include <ctype.h>  
  
FILE *outfile;  
  
// C keywords list  
  
char *keywords[] = {  
  
    "int", "float", "return", "if", "else", "while", "for", "char", "double", "do",  
  
    "switch", "case", "break", "continue", "void", "long", "short", "unsigned",  
  
    "signed", "static", "struct", "union", "typedef", "const", "goto", "enum",  
  
    "default", "sizeof", "volatile", "register", NULL  
  
};  
  
int is_keyword(const char *word) { for  
  
    (int i = 0; keywords[i]; i++) {  
  
        if (strcmp(keywords[i], word) ==  
  
            0) return 1;  
  
    }  
  
    return 0;  
  
}  
  
%}
```

```

%%

\"([^\"]|\\.)*\\"      { fprintf(outfile, "String literal: %s\n", yytext); }

\'([^\']|\\.)\'        { fprintf(outfile, "Character literal: %s\n", yytext); }

[0-9]+\.[0-9]+        { fprintf(outfile, "Float number: %s\n", yytext); }

[0-9]+                { fprintf(outfile, "Integer number: %s\n", yytext); }

[a-zA-Z_][a-zA-Z0-9_]* {
    if (is_keyword(yytext))
        fprintf(outfile, "Keyword: %s\n", yytext);
    else
        fprintf(outfile, "Identifier: %s\n", yytext);
}

"=="|"!="|"<="|">="|"="|"+"| "-"|"*"|" /"|" "<"|" ">" { fprintf(outfile,
    "Operator: %s\n", yytext);
}

[{}()[];,:]          { fprintf(outfile, "Special symbol: %s\n", yytext); }

; // Skip whitespace

[ \t\n]+ .
    { fprintf(outfile, "Unknown token: %s\n", yytext); }

%%

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Usage: %s <input_file>\n", argv[0]);
        return 1;
    }
}

```

```
FILE *infile = fopen("sample.c", "r");  
if (!infile) {  
    perror("Cannot open input file");  
    return 1;  
}  
outfile = fopen("tokens.txt", "w");  
if (!outfile) {  
    perror("Cannot open output file");  
    return 1;  
}  
yyin = infile;  
yylex();  
fclose(infile);  
fclose(outfile);  
printf("Tokenization complete. Output written to  
tokens.txt\n"); return 0;  
}  
int yywrap() {  
    return 1;  
}
```

Sample.c :

```
#include <stdio.h>

int main() {
    int a = 10;
    float b = 20.5;
    char c = 'Z';
    const char *str = "Hello, World!";

    if (a < b) {
        printf("a is less than b\n");
    } else {
        printf("a is not less than b\n");
    }

    return 0;
}
```

Output:

```
C:\Users\hp\OneDrive\Documents\Computer Design\Lab\LEX\0\0-a.exe Sample.c
Tokenization complete. Output written to tokens.txt
```

Unknown token: #

Identifier: include

Operator: <

Identifier: stdio

Unknown token: .

Identifier: h

Operator: >

Keyword: int

Identifier: main

Special symbol: (

Special symbol:)

Special symbol: {

Keyword: int

Identifier: a

Operator: =

Integer number: 10

Special symbol: ;

Keyword: float

Identifier: b

Operator: =

Float number: 20.5

Special symbol: ;

Keyword: char

Identifier: c

Operator: =

Character literal: 'Z'

Special symbol: ;

Keyword: const

Keyword: char

Operator: *

Identifier: str

Operator: =

String literal: "Hello, World!"

Special symbol: ;

Keyword: if

Special symbol: (

Identifier: a

Operator: <

Identifier: b

Special symbol:)

Special symbol: {

Identifier: printf

Special symbol: (

String literal: "a is less than b\n"

Special symbol:)

Special symbol: ;

Special symbol: }

Keyword: else

Special symbol: {

Identifier: printf

Special symbol: (

String literal: "a is not less than b\n"

Special symbol:)

Special symbol: ;

Special symbol: }

Keyword: return

Integer number: 0

Special symbol: ;

Special symbol: }

6. Program to implement Recursive Descent Parsing in C.

$$\begin{aligned} E &\rightarrow iE' \\ E' &\rightarrow +iE' / -iE' / \epsilon \end{aligned}$$

Code:

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

char inp[100];
int l = 0;

void match(char t) {
    if (inp[l] == t) {
        l++;
    } else {
        printf("Error\n");
        exit(0);
    }
}

void E();
void E_prime();

void E() {
    if (inp[l] == 'i') {
        match('i');
        E_prime();
    }
}

void E_prime() {
    if (inp[l] == '+') {
        match('+');
        match('i');
        E_prime();
    } else if (inp[l] == '-') {
        match('-');
        match('i');
        E_prime();
    } else {
        return; // epsilon case
    }
}
```



```
    }  
}  
int main() {  
    printf("Enter expression: ");  
    scanf("%s", inp);  
  
    E();  
  
    if (inp[l] == '$') { // End of input  
        printf("Success\n");  
    } else {  
        printf("Error\n");  
    }  
  
    return 0;  
}
```

Output:

```
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab\CD_LAB_EndSem_22000794\Recursive Descent\output>G1.exe  
Enter expression: ii  
Error
```

```
C:\Users\HP\OneDrive\6th Sem\Compiler Design\Lab\CD_LAB_EndSem_22000794\Recursive Descent\output>G1.exe  
Enter expression: i+i-i$  
Success
```

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

YACC (Yet Another Compiler Compiler) is a parser generator tool used in compiler design to produce syntax analyzers for context-free grammars. It works closely with Lex, where Lex handles lexical analysis and YACC performs syntax analysis based on grammar rules. YACC generates efficient LALR(1) parsers and allows grammar specification in a structured format with declarations, grammar rules, and associated C actions. It interprets token streams from Lex to understand program structure and syntax. Widely used in building compilers and interpreters, YACC simplifies the implementation of parsing logic for programming languages.

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

Code:

Lex:

```
%{  
  
#include<stdlib.h>  
  
void yyerror(char *);  
  
#include "1.tab.h"  
  
%}  
  
%%  
  
[0-9]+ return num;  
[-/+*\n] return *yytext;  
  
[ \t] ;  
  
. yyerror("invalid");  
  
%%  
  
int yywrap(){  
  
    return 1;  
  
}
```

Yacc:

```
%{  
  
#include<stdio.h>  
  
int yylex(void);  
  
void yyerror(char *);  
  
%}  
  
%token num  
  
%%  
  
S:E'\n' {printf("Valid syntax."); return 0;}  
  
E:E'-T {}  
  
|E'+T {}  
  
|T {}  
  
T:T'/F {}  
  
|T'*F {}  
  
|F {}  
  
F:num {}  
  
%%  
  
void yyerror(char *s){  
  
    printf("%s\n",s);  
  
}  
  
int main(){  
  
    yyparse(); return 0;  
  
}
```

Output:

```
5*4/3-4+2  
Valid syntax.
```

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

Code:

Lex:

```
%{  
  
#include<stdlib.h>  
  
#include "1.tab.h"  
  
void yyerror(char *);  
  
%}  
  
%%  
  
[0-9]+ {yylval=atoi(yytext); return num;}  
  
[-+*/\n] {return *yytext;}  
  
[(\)/] {return *yytext;}  
  
[ \t] ;  
  
. {yyerror("invalid");}  
  
%%  
  
int yywrap(){  
  
    return 1;  
  
}
```

Yacc:

```
%{  
  
#include<stdio.h>  
  
void yyerror(char  
  
*); int yylex(void);  
  
%}  
  
%token num
```

```
%%  
S:E'\n' {printf("%d\n",$1); return 0;}  
E:E'-T {$$=$1-$3;}  
|T {$$=$1;}  
T:T'+F {$$=$1+$3;}  
|F {$$=$1;}  
F:F'*G {$$=$1*$3;}  
|G {$$=$1;}  
G:G/'H {$$=$1/$3;}  
|H {$$=$1;}  
H:('(E)') {$$=$2;}  
|num {$$=$1;}  
%%  
void yyerror(char *s){  
    printf("%s\n",s);  
}  
int main(){  
    yyparse(); return 0;  
}
```

Output:

```
C:\Users\Hardev Sinh\OneDrive\Documents\Sem-6\CD\CD-LAB\xyz\Calculator>a.exe  
100/5+3-2*8  
7
```

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

Code:

Lex:

```
%{  
  
#include<stdlib.h>  
  
#include "1.tab.h"  
  
void yyerror(char *);  
  
%}  
  
%%  
  
[0-9]+ {yylval.num=atoi(yytext); return INTEGER;}  
[A-Za-z_][A-Za-z_0-9]* {yylval.str=yytext; return ID;}  
[-+*/\n] {return *yytext;}  
  
[ \t] ;  
  
. {yyerror("Invalid character.");}  
  
%%  
  
int yywrap(){  
  
    return 1;  
  
}
```

Yacc:

```
%{  
  
#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("-");}

|T {}

T:T'+'F {printf("+");}

|F {}

F:F'*'G {printf("*");}

|G {}

G:G'/'H {printf("/");}

|H {}

H:INTEGER {printf("%d", $1);}

|ID {printf("%s", $1);}

%%

void yyerror(char *s){

    printf("%s\n", s);

}

int main(){

    yyparse(); return 0;

}
```

Output:

```
C:\Users\Hardev Sinh\OneDrive\Documents\Sem-6\CD\abc>a.exe
8*3+4
8 3 * 4 +
```