Name: Adwait Hegde
Roll No: 2019130019
TE Comp (Batch-A)

EXPERIMENT 5

Aim:

Implement lexical analyzer
To write Lexical analyzer using flex

Theory:

What is Lexical Analysis?

Lexical Analysis is the very first phase in the compiler designing. A Lexer takes the modified source code which is written in the form of sentences. In other words, it helps you to convert a sequence of characters into a sequence of tokens. The lexical analyzer breaks this syntax into a series of tokens. It removes any extra space or comment written in the source code.
Programs that perform Lexical Analysis in compiler design are called lexical analyzers or lexers. A lexer contains tokenizer or scanner. If the lexical analyzer detects that the token is invalid, it generates an error. The role of Lexical Analyzer in compiler design is to read character streams from the source code, check for legal tokens, and pass the data to the syntax analyzer when it demands.

What's a lexeme?

A lexeme is a sequence of characters that are included in the source program according to the matching pattern of a token. It is nothing but an instance of a token.

What's a token?

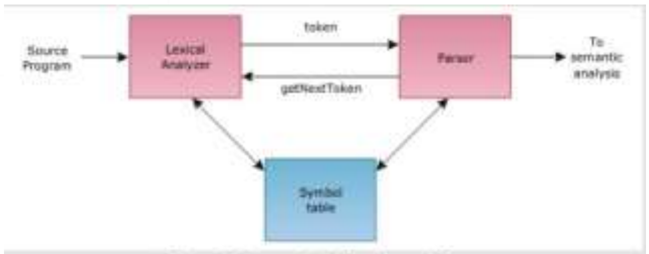
Tokens in compiler design are the sequence of characters which represents a unit of information in the source program.

What is Pattern?

A pattern is a description which is used by the token. In the case of a keyword which uses as a token, the pattern is a sequence of characters.

Lexical Analyzer Architecture:

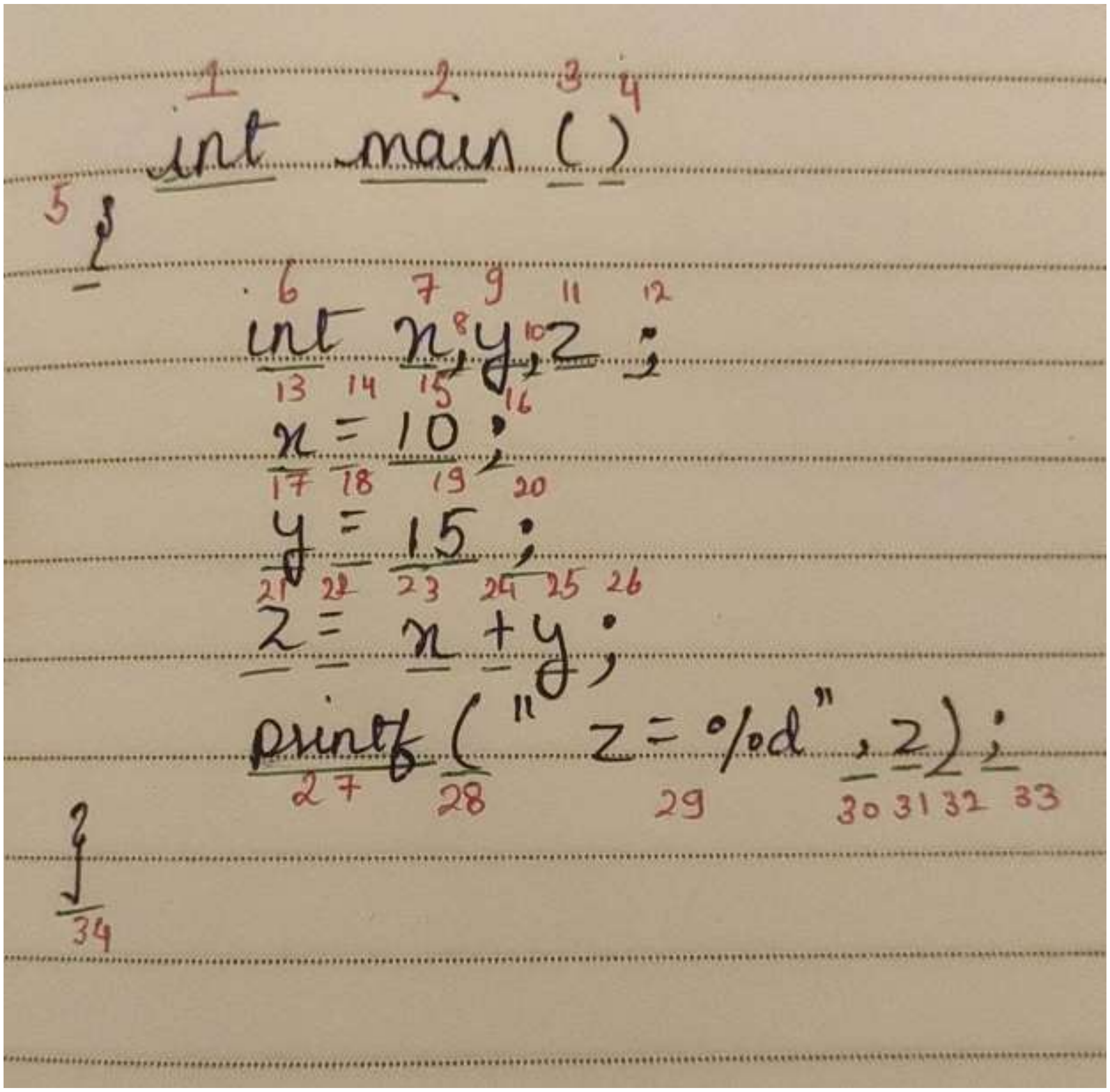
The main task of lexical analysis is to read input characters in the code and produce tokens. Lexical analyzer scans the entire source code of the program. It identifies each token one by one. Scanners are usually implemented to produce tokens only when requested by a parser. Here is how recognition of tokens in compiler design works-



Interaction between Lexical Analyzer and Parser

1. “Get next token” is a command which is sent from the parser to the lexical analyzer.
2. On receiving this command, the lexical analyzer scans the input until it finds the next token.
3. It returns the token to Parser.

Solved Problem:



```
int main ()
{
    int x,y,z;
    x = 10;
    y = 15;
    z = x + y;
    printf ("z= %d", z);
}
```

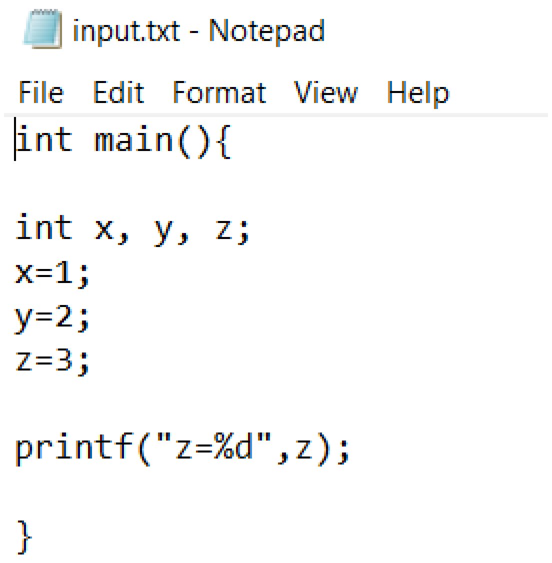
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

Implementation

Code:

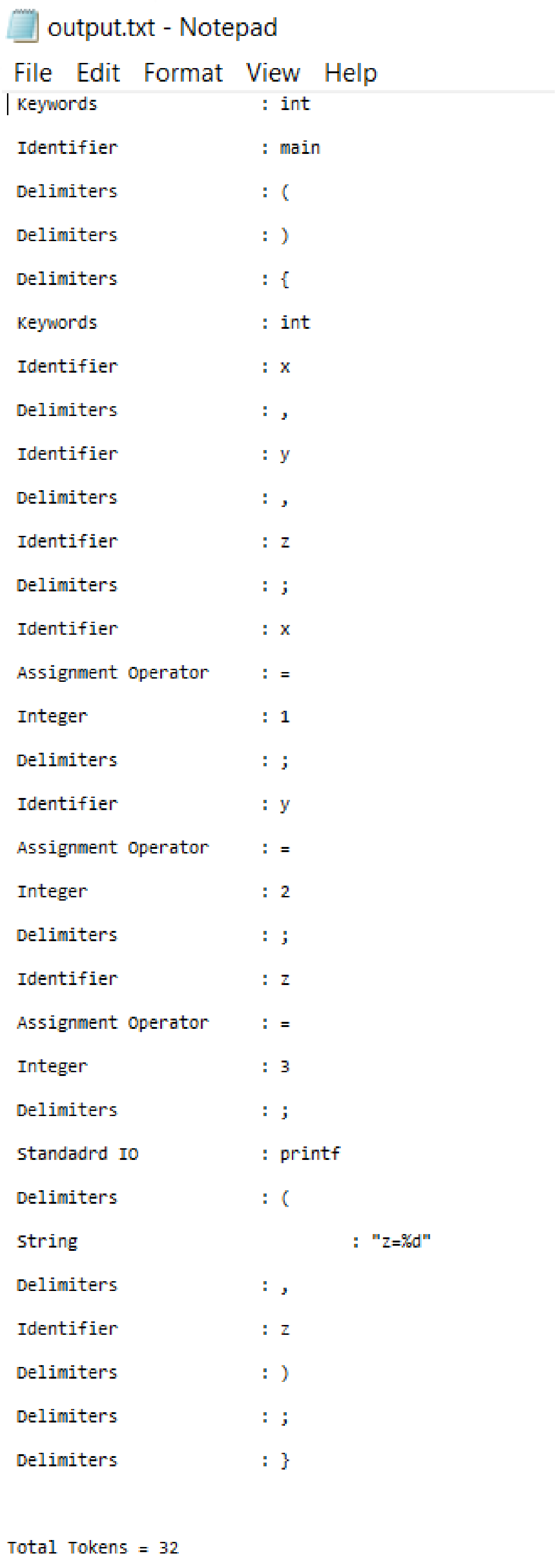
```
%{
int total=0;
%}

%option noyywrap

%%

#.                                      {total++; fprintf(yyout," Preprocessor Directive  : %s\n\n",yytext);}
[''|,|;(|)|{|}|.|_]                     {total++; fprintf(yyout," Delimiters              : %s\n\n",yytext);}
[[]]                                    {total++; fprintf(yyout," Delimiters              : %s\n\n",yytext);}
[-+*/%]                                 {total++; fprintf(yyout," Arithmetic Operator     : %s\n\n",yytext);}
([<>]=?)|([!=]=)                        {total++; fprintf(yyout," Relational Operator     : %s\n\n",yytext);}
(&&)|(\|\|)|(!)                         {total++; fprintf(yyout," Logical Operator        : %s\n\n",yytext);}
("if")|("else")|("switch")|("case")|("default")|("break")|("int")|("float")|("char")|("double")|("long")|("for")|
("while")|("do")|("void")|("goto")|("auto")|("signed")|("const")|("extern")|("register")|("unsigned")|("return")|
("continue")|("enum")|("sizeof")|("struct")|("typedef ")|("union")|("volatile")     {total++; fprintf(yyout,"
Keywords                : %s\n\n",yytext);}
"printf"                                {total++; fprintf(yyout," Standadrd IO            : %s\n\n",yytext);}
"//".*                                  {total++; fprintf(yyout," Single line comment     : %s\n\n",yytext);}
"/*"([^\*]*|[\*]*)*"*/"                 {total++; fprintf(yyout," Multi line comment      : %s\n\n",yytext);}
[a-zA-Z_][a-zA-Z0-9_]*                  {total++; fprintf(yyout," Identifier              : %s\n\n",yytext);}
[0-9]*"."[0-9]+                         {total++; fprintf(yyout," Floating point value    : %s\n\n",yytext);}
[-][0-9]*"."[0-9]+                      {total++; fprintf(yyout," Negative Floating point : %s\n\n",yytext);}
[0-9]+                                  {total++; fprintf(yyout," Integer                 : %s\n\n",yytext);}
"-"[0-9]+                               {total++; fprintf(yyout," Negative Integer        : %s\n\n",yytext);}
=                                       {total++; fprintf(yyout," Assignment Operator     : %s\n\n",yytext);}
["]([^"\\\n]|\\.|\\\n)*["]              {total++; fprintf(yyout," String                  : %s\n\n",yytext);}
[ \t\n"]+                                ;
%%

int main()
{
    extern FILE *yyin, *yyout;
    yyin = fopen("input.txt","r");
    yyout = fopen("output.txt","w");
    yylex();
    fprintf(yyout,"\n\nTotal Tokens = %d", total);
    return 0;
}
```

Input.txt



```
int main(){

int x, y, z;
x=1;
y=2;
z=3;

printf("z=%d",z);

}
```

output.txt



```
Keywords                : int
Identifier              : main
Delimiters              : (
Delimiters              : )
Delimiters              : {
Keywords                : int
Identifier              : x
Delimiters              : ,
Identifier              : y
Delimiters              : ,
Identifier              : z
Delimiters              : ;
Identifier              : x
Assignment Operator     : =
Integer                 : 1
Delimiters              : ;
Identifier              : y
Assignment Operator     : =
Integer                 : 2
Delimiters              : ;
Identifier              : z
Assignment Operator     : =
Integer                 : 3
Delimiters              : ;
Standadrd IO            : printf
Delimiters              : (
String                          : "z=%d"
Delimiters              : ,
Identifier              : z
Delimiters              : )
Delimiters              : ;
Delimiters              : }


Total Tokens = 32
```

Code:

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

int state;
char code[1000];
char c2;
char start;
int start_index;
char buff[1000];
int buff_index;

char keys[26][20] = {
    "auto",
    "double",
    "main",
    "struct",
    "long",
    "case",
    "enum",
    "char",
    "register",
    "return",
    "union",
    "const",
    "short",
    "for",
    "void",
    "default",
    "goto",
    "while",
    "int",
    "char",
    "if",
    "else",
    "switch",
    "break",
    "enum",
    "printf"
};
int key_store[26][2];
int key_count;
int extra_count;
```

```
void lexical_analyzer(char c[], int n)
{
    int i=0;
    if (c[n-1] == '\n')
        n-=1;

    while(i<n)
    {
        switch(state)
        {

            case 0:

                c2 = c[i];
                i++;

                //operators
                if ((c2 == '+') || (c2 == '-'))
                {
                    state = 1;
                }
                else if (c2 == '/')
                {
                    if (i<n){
                        if (c[i]=='/')
                        {
                            printf("Single line comment\n\n");
                            i=n;
                        }
                        else if (c[i]=='*')
                        {
                            i++;
                            printf("multi line comment\n\n");
                            state=6;
                        }
                        else
                            state = 1;
                    }
                    printf("< arithmetic operator : %c > recognized\n\n",c2);
                }
                else if ((c2 == '%') || (c2 == '*'))
                    state = 1;
                else if ((c2 == '&') || (c2 == '|'))
                    state = 2;
                else if (c2 == '^')
                    state = 3;
                else if (c2 == '=')
                    state = 4;
                else if ((c2 == '>') || (c2 == '<'))
                    state = 5;
                else if (c2 == '!')
                    if (i<n && c[i]=='=')
                    {
                        i++;
                        printf("< relational operator : %c= >
recognized\n\n",c2);
                    }
                    else
                        printf("< logical operator : %c > recognized\n\n",c2);

                //header
                else if(c2 == '#' && i<n)
                {
                    if (c[i]=='i')
                    {
                        printf("header file included\n\n");
                        i=n;
                    }
                    else if(c[i]=='d')
                    {
                        printf("constant defined\n\n");
                        i+=6;
                    }
                }

                //separators
                else if(c2 == ',' || c2 == ';')
                    printf("< separator : %c > recognized\n\n",c2);

                //braces, brackets, parenthesis
                else if(c2 == '(' || c2 == ')')
                    printf("< parenthesis : %c > recognized\n\n",c2);

                else if(c2 == '[' || c2 == ']')
                    printf("< brackets : %c > recognized\n\n",c2);
```

```
else if(c2 == '{' || c2 == '}')
    printf("< braces : %c > recognized\n\n",c2);

//constants
else if (isdigit(c2)>0)
{
    start = c2;
    start_index=0;
    buff_index=0;
    buff[buff_index] = c2;
    buff_index++;
    state = 7;
}

//string
else if(c2 == '"')
{
    buff_index=0;
    buff[buff_index++] = c2;
    state = 8;
}

//keyword and identifier
else if(isalpha(c2)>0 || c2=='_')
{
    key_count=0;
    for (int j=0;j<26;j++)
    {
        if(c2 == keys[j][0])
        {
            key_store[key_count][0] = j;
            key_store[key_count][1] = strlen(keys[j]);
            key_count++;
        }
    }
    extra_count = key_count;

    if (key_count>0) //check for keywords
        state=9;
    else
        state=10;

    start = c2;
    buff[0] = c2;
```

```
                    buff_index=1;
                }

                break;



            case 1:
                if (i<n)
                {
                    if (c2 == '+' || c2=='-')
                    {
                        if (c2 == c[i])
                        {
                            i++;
                            printf("< arithmetic operator : %c%c > 
recognized\n\n",c2,c2);
                            state=0;
                            break;
                        }
                    }

                    if(c[i] == '=')
                    {
                        i++;
                        printf("< assignment operator : %c= > 
recognized\n\n",c2);
                    }
                    else
                        printf("< arithmetic operator : %c > recognized\n\n",c2);
                }

                state=0;
                break;

            case 2:
                if (i<n)
                {
                    if(c[i] == c2)
                    {
                        i++;
                        printf("< logical operator : %c%c > 
recognized\n\n",c2,c2);
                    }
                    else
```

```
                        printf("< bitwise operator : %c > recognized\n\n",c2);
                }

                state=0;
                break;

            case 3:
                printf("< bitwise operator : %c > recognized\n\n",c2);

                state=0;
                break;

            case 4:
                if (i<n)
                {
                    if (c2 == c[i])
                    {
                        i++;
                        printf("< relational operator : %c%c >
recognized\n\n",c2,c2);
                    }
                    else
                        printf("< assignment operator : %c > recognized\n\n",c2);

                }

                state=0;
                break;

            case 5:
                if (i<n)
                {
                    if (c2 == c[i])
                    {
                        i++;
                        printf("< bitwise operator : %c%c >
recognized\n\n",c2,c2);
                    }
                    else
                    {
                        if(c[i] == '=')
                        {
                            i++;
                            printf("< relational operator : %c= >
recognized\n\n",c2);
```

```
                        }
                        else
                            printf("< relational operator : %c >
recognized\n\n",c2);
                    }
                }

                state=0;
                break;

            case 6:

                if (i<n)
                {
                    c2 = c[i];
                    i++;

                    if (i<n && c2=='*' && c[i]=='/')
                    {
                        i++;
                        state=0;
                    }
                }

                break;

            case 7:
                if (i<n)
                {
                    c2 = c[i];
                    i++;

                    if (isdigit(c2)>0)
                    {
                        buff[buff_index] = c2;
                        buff_index++;
                    }
                    else if(c2=='.')
                    {
                        if(isdigit(start)>0)
                        {
                            start='.';
                            start_index = buff_index;
                            buff[buff_index] = c2;
                            buff_index++;
                        }
                        else//dot repeats ifafter . just after e, i-2, +,- just
+,-: i-3
                        {
                            i--;
                            printf("< Constant : ");
                            for(int k = 0;k<buff_index;k++)
                                printf("%c",buff[k]);
                            printf(" > recognized\n\n");
                            state=0;
                            break;
                        }
                    }
                    else if(c2=='E')
                    {
                        if(start == '.')
                        {
                            start='E';
                            start_index=buff_index;
                            buff[buff_index] = c2;
                            buff_index++;
                            if(i<n)
                            {
                                if (c[i] == '+' || c[i] == '-')
                                {
                                    start = c[i];
                                    start_index = buff_index;
                                    buff[buff_index] = c[i];
                                    i++;
                                    buff_index++;
                                }
                                else
                                {
                                    i--;
                                    printf("< Constant : ");
                                    for(int k = 0;k<buff_index-1;k++)
                                        printf("%c",buff[k]);
                                    printf(" > recognized\n\n");
                                    state=0;
                                    break;
                                }
                            }
                            else//ends with E
                            {
                                i -= 1;
```

```
                printf("< Constant : ");
                for(int k = 0;k<buff_index-1;k++)
                    printf("%c",buff[k]);
                printf(" > recognized\n\n");


                state=0;
                break;
            }
        }
        else //E repeats
        {
            i = (i - 2) - (buff_index-start_index);
            if(buff_index-start_index>1)
            {
                printf("< Constant : ");
                for(int k = 0;k<buff_index;k++)
                    printf("%c",buff[k]);
                printf(" > recognized\n\n");
            }
            else
            {
                printf("< Constant : ");
                for(int k = 0;k<buff_index-2;k++)
                    printf("%c",buff[k]);
                printf(" > recognized\n\n");
            }

            state=0;
            break;
        }
    }
    else
    {
        printf("< Constant : ");
        for(int k = 0;k<buff_index;k++)
            printf("%c",buff[k]);
        printf(" > recognized\n\n");i--;
        state=0;

    }
}
else
{
    if(isdigit(start)>0 || start=='.')
```

```
                    {
                        printf("< Constant : ");
                        for(int k = 0;k<buff_index;k++)
                            printf("%c",buff[k]);
                        printf(" > recognized\n\n");
                    }
                    else if(start=='+' || start == '-'){
                        if ((buff_index-start_index) > 1)
                        {
                            printf("< Constant : ");
                            for(int k = 0;k<buff_index;k++)
                                printf("%c",buff[k]);
                            printf(" > recognized\n\n");
                        }
                        else
                        {
                            i -= 2;
                            printf("< Constant : ");
                            for(int k = 0;k<buff_index-2;k++)
                                printf("%c",buff[k]);
                            printf(" > recognized\n\n");
                        }
                    }

                    state=0;
                }
                break;

            case 8:
                if(i<n)
                {
                    c2 = c[i];
                    i++;

                    if(c2 == '"')
                    {
                        if (buff_index==1)
                        {
                            printf("< string : empty string > recognized\n\n");
                            state=0;
                            break;
                        }
                        else
                        {
                            if(buff[buff_index-1]!='\\')
                            {
                                printf("< string : ");
                                for(int k = 1;k<buff_index;k++)
                                    printf("%c",buff[k]);
                                printf(" > recognized\n\n");
                                state=0;
                                break;
                            }
                            else
                            {
                                buff[buff_index]=c2;
                                buff_index++;
                            }
                        }

                    }
                    else
                    {
                        buff[buff_index]=c2;
                        buff_index++;
                    }
                }
                else//abrupt end to string
                {
                    i -= (buff_index-1);
                    state=0;
                }
                break;

            case 9:
                if (i<n)
                {
                    c2 = c[i];
                    i++;

                    if(isalpha(c2)>0 && islower(c2)>0)
                    {

                        extra_count=0;
                        for(int x = 0;x<key_count;x++)
                        {
                            if(key_store[x][0]!=-1 && key_store[x][1]!=-1)
                            {
                                if(buff_index < key_store[x][1] &&
c2==keys[key_store[x][0]][buff_index])
```

```
                {
                    extra_count++;
                }
                else
                {
                    key_store[x][0] = -1;
                    key_store[x][1] = -1;
                }
            }
        }

        buff[buff_index++] = c2;

        if(extra_count==0)
        {
            state=10;
        }

    }
    else
    {
        i--;
        if(extra_count==1)
        {

            for(int f=0;f<key_count;f++)
            {
                if(key_store[f][0]!=-1 && key_store[f][1]!=-1)
                {
                    if(buff_index == key_store[f][1])
                    {
                        printf("< Keyword : ");
                        for(int k = 0;k<buff_index;k++)
                            printf("%c",buff[k]);
                        printf(" > recognized\n\n");
                        state=0;
                        break;
                    }
                }
            }
            if (state==9)
                state=10;
            break;
        }
        else
```

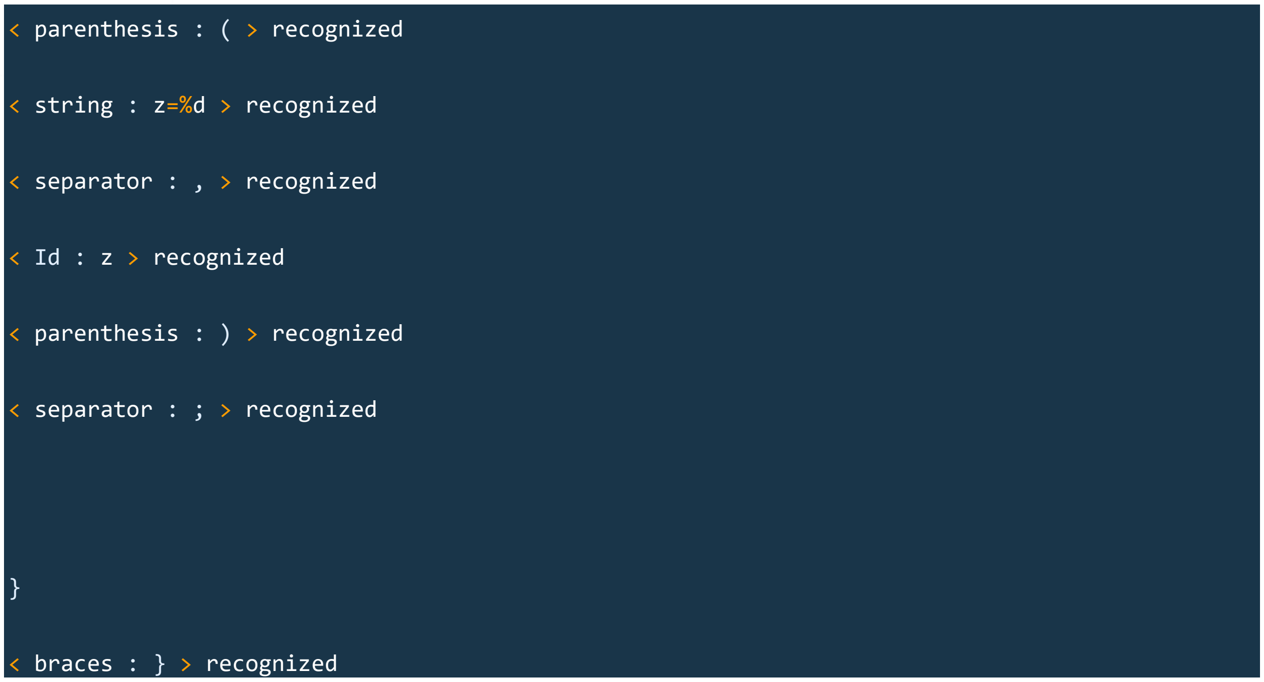
```
                        {
                            if(isalpha(c[i])>0 || isdigit(c[i])>0 || c[i]=='_')
                                state=10;
                            else
                                state=0;
                        }
                    }

                }
                else
                {
                    if (extra_count == 1)
                    {
                        printf("< Keyword : ");
                        for(int k = 0;k<buff_index;k++)
                            printf("%c",buff[k]);
                        printf(" > recognized\n\n");
                    }
                    else
                    {
                        printf("< Id : ");
                        for(int k = 0;k<buff_index;k++)
                            printf("%c",buff[k]);
                        printf(" > recognized\n\n");
                    }

                    state=0;
                }

                break;

            case 10:
                if(i<n)
                {
                    c2=c[i];
                    i++;

                    if(c2=='_' || isalpha(c2)>0 || isdigit(c2)>0)
                    {
                        buff[buff_index++] = c2;
                    }
                    else
                    {
                        i--;
                        printf("< Id : ");
                        for(int k = 0;k<buff_index;k++)
                            printf("%c",buff[k]);
                        printf(" > recognized\n\n");
                        state=0;
                    }
                }
                else
                {
                    printf("< Id : ");
                    for(int k = 0;k<buff_index;k++)
                        printf("%c",buff[k]);
                    printf(" > recognized\n\n");
                    state=0;
                }

                break;
        }
    }
}




int main()
{
    FILE *fptr;
    char ch;

    fptr = fopen("code.txt", "r");

    if (fptr == NULL)
    {
        printf("File is not available \n");
    }
    else
    {
        state = 0;
        while(fgets(code,sizeof(code),fptr))
        {
            printf("\n%s\n",code);
            lexical_analyzer(code , strlen(code));
        }
    }
    fclose(fptr);
    return 0;
}
```

Input:

```
int main(){

int x, y, z;
x=1;
y=2;
z=3;

printf("z=%d",z);

}
```

Output:

```
int main(){

< Keyword : int > recognized

< Keyword : main > recognized

< parenthesis : ( > recognized

< parenthesis : ) > recognized

< braces : { > recognized




int x, y, z;

< Keyword : int > recognized

< Id : x > recognized

< separator : , > recognized

< Id : y > recognized

< separator : , > recognized
< Id : z > recognized

< separator : ; > recognized


x=1;

< Id : x > recognized

< assignment operator : = > recognized

< Constant : 1 > recognized

< separator : ; > recognized


y=2;

< Id : y > recognized

< assignment operator : = > recognized

< Constant : 2 > recognized

< separator : ; > recognized


z=3;

< Id : z > recognized

< assignment operator : = > recognized

< Constant : 3 > recognized

< separator : ; > recognized




printf("z=%d",z);

< Keyword : printf > recognized
```



```
< parenthesis : ( > recognized

< string : z=%d > recognized

< separator : , > recognized

< Id : z > recognized

< parenthesis : ) > recognized

< separator : ; > recognized



}

< braces : } > recognized
```

Conclusion:

In this Experiment, I learned about the concept of Lexical Analysis stage of a compiler. I understood how a lexical analyzer reads input characters and groups them into lexemes and produces a sequence of tokens for each lexeme. Also the working of flex tool was understood and a lex program which identifies lexemes in a C++ program and produce tokens was implemented.