

College Of Engineering Trivandrum

Compiler Design Lab

CS431



Submitted By:

Neethu S

S6 CSE Roll No:42

TVE18CS043

Department of Computer Science

January 4, 2021

Contents

1	Lexical Analyser	2
1.1	Aim	2
1.2	Theory	2
1.3	Algorithm	2
1.4	Code	3
1.5	Output	5
1.6	Result	5



Cycle 1 Experiment 5

1 Lexical Analyser

1.1 Aim

To write a program to perform lexical analysis.

1.2 Theory

Lexical analysis is the very first phase in the compiler designing. It 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 analysis 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 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. It reads character streams from the source code, checks for legal tokens, and pass the data to the syntax analyzer when it demands. The common type of tokens include:

- **Keyword:** A keyword is a word reserved by a programming language having a special meaning.
- **Identifier:** It is a user-defined name used to uniquely identify a program element. It can be a class, method, variable, namespace etc.
- **Operator:** It is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result.
- **Separator:** Separators are used to separate one programming element from the other.
- **Literals:** A literal is a notation for representing a fixed value and do not change during the course of execution of the program.

1.3 Algorithm

1. Start
2. Open the input file input.txt in read mode
3. If the file is not found . Display an error message
4. Else scan and analyse each component ch until the end of file
 - If ch is a keyword display 'kwd'
 - If ch is an identifier display 'id'
 - If ch is any arithmetic operator display 'op-plus/sub/div/mul..'
 - If ch is any other operators display the same
5. Close the file
6. Stop

1.4 Code

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

void keyw(char *p);
int i = 0, id = 0, kw = 0, num = 0, op = 0;
char keys[32][10] = {"auto", "break", "case", "char", "const", "continue",
    "default", "do", "double", "else", "enum", "extern", "float",
    "for", "goto", "if", "int", "long", "register", "return", "short",
    "signed", "sizeof", "static", "struct", "switch", "typedef",
    "union", "unsigned", "void", "volatile", "while"};

int main()
{
    char ch, str[25], seps[15]=" \t\n,;(){}[]#\"<>", oper[]="!%^&*--+=~|.<>/?";
    int j;
    char fname[50];
    FILE *f1;

    printf("Enter the file name: ");
    scanf("%s", fname);
    f1 = fopen(fname, "r");

    if (f1 == NULL)
    {
        printf("file not found");
        exit(0);
    }

    while ((ch = fgetc(f1)) != EOF)
    {
        for (j = 0; j <= 14; j++)
        {
            if (ch == oper[j])
            {
                printf("%c is an operator\n", ch);
                op++;
                str[i] = '\0';
                keyw(str);
            }
        }
        for (j = 0; j <= 14; j++)
        {
            if (i == -1)
                break;
            if (ch == seps[j])
            {
                if (ch == '#')
                {
```

```

        while (ch != '>')
        {
            printf("%c", ch);
            ch = fgetc(f1);
        }
        printf("%c is a header file\n", ch);
        i = -1;
        break;
    }
    if (ch == '"')
    {
        do
        {
            ch = fgetc(f1);
            printf("%c", ch);
        } while (ch != '"');
        printf("\b is an argument\n");
        i = -1;
        break;
    }
    str[i] = '\\0';
    keyw(str);
}
}
if (i != -1)
{
    str[i] = ch;
    i++;
}
else
    i = 0;
}
printf("Keywords: %d\nIdentifiers: %d\nOperators: %d\nNumbers: %d\n", kw, id, op, num);
}

```

```

void keyw(char *p)
{
    int k, flag = 0;
    for (k = 0; k <= 31; k++)
    {
        if (strcmp(keys[k], p) == 0)
        {
            printf("%s is a keyword\n", p);
            kw++;
            flag = 1;
            break;
        }
    }
    if (flag == 0)
    {
        if (isdigit(p[0]))

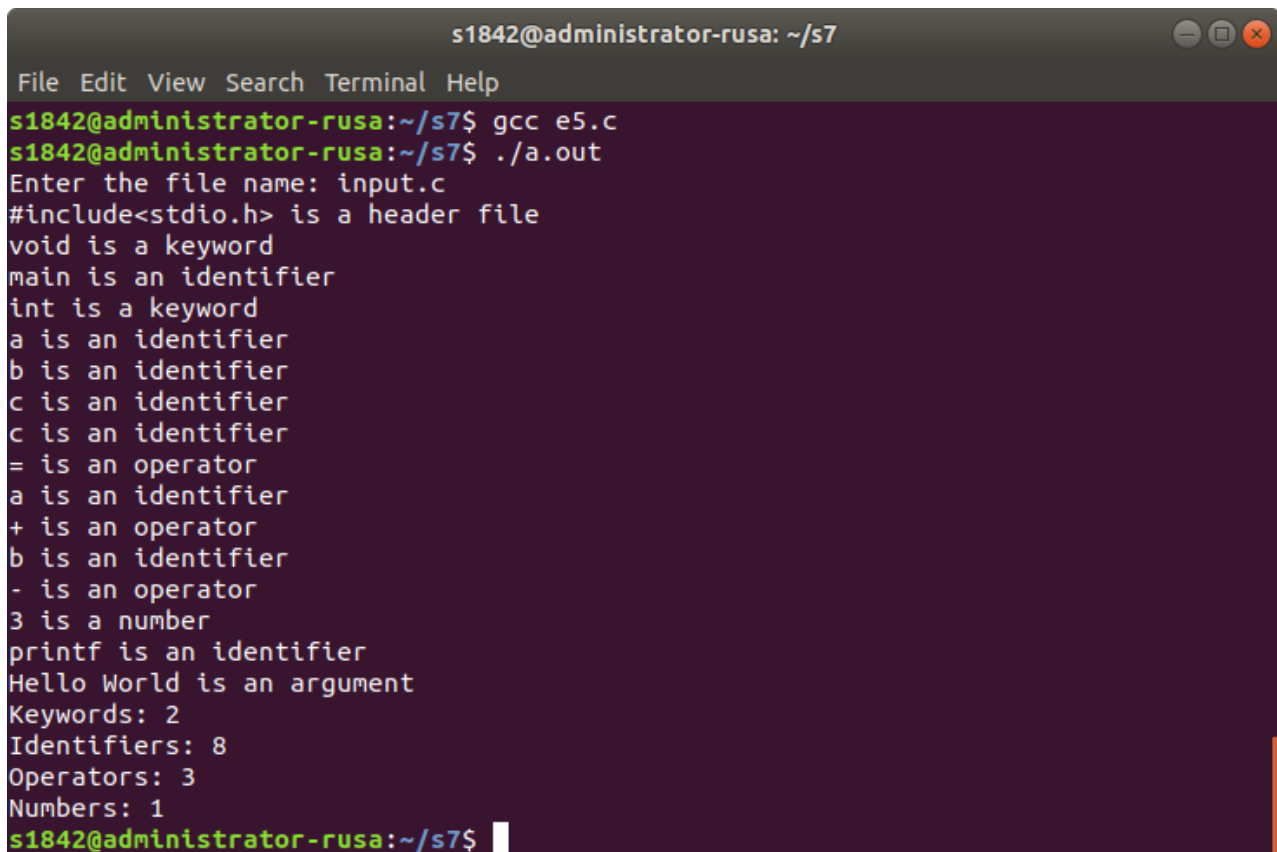
```

```

{
    printf("%s is a number\n", p);
    num++;
}
else
{
    //if(p[0]!=13&&p[0]!=10)
    if (p[0] != '\0')
    {
        printf("%s is an identifier\n", p);
        id++;
    }
}
}
i = -1;
}

```

1.5 Output



```

s1842@administrator-rusa: ~/s7
File Edit View Search Terminal Help
s1842@administrator-rusa:~/s7$ gcc e5.c
s1842@administrator-rusa:~/s7$ ./a.out
Enter the file name: input.c
#include<stdio.h> is a header file
void is a keyword
main is an identifier
int is a keyword
a is an identifier
b is an identifier
c is an identifier
c is an identifier
= is an operator
a is an identifier
+ is an operator
b is an identifier
- is an operator
3 is a number
printf is an identifier
Hello World is an argument
Keywords: 2
Identifiers: 8
Operators: 3
Numbers: 1
s1842@administrator-rusa:~/s7$

```

1.6 Result

Implemented the program for the lexical analyzer using C language in Ubuntu 20.04 with kernel and the above outputs were obtained.