SSN COLLEGE OF ENGINEERING

ASSIGNMENT 1 : LEXICAL ANALYSER USING C

-SRINITHYEE S K

185001166

Aim:

To write a program in C that simulates a Lexical Analyser.

Code:

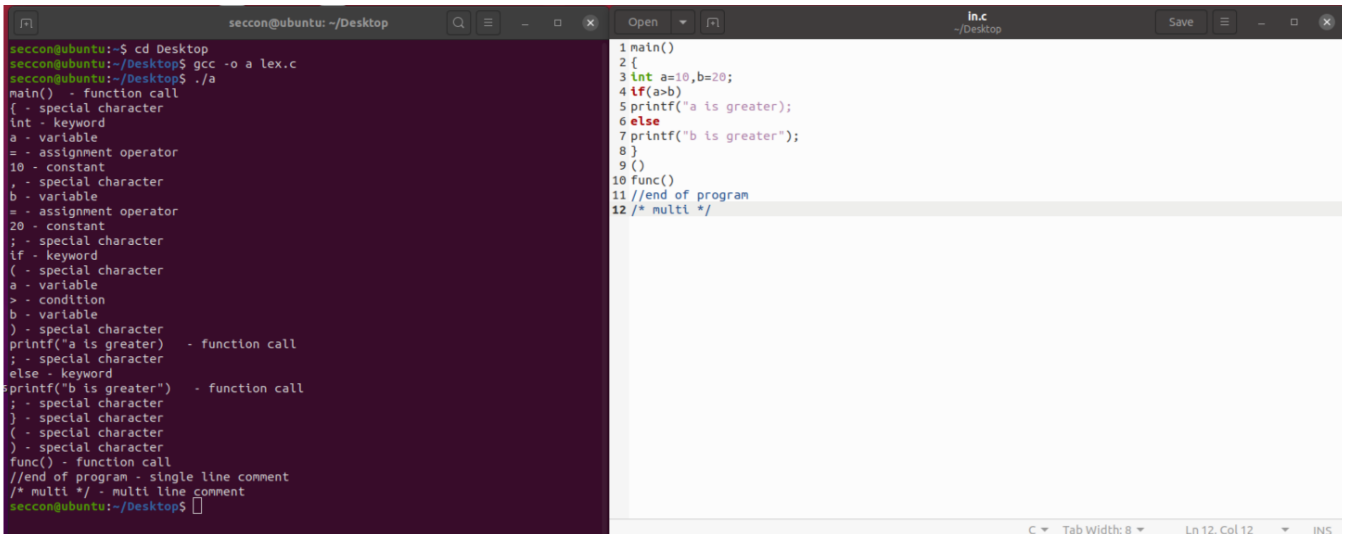
```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<ctype.h>
int main()
{
    FILE* fp;
    int count = 0;
    char* line = NULL;
    size_t len = 0;
    ssize_t linelen;
    char store1[10][100];
    char store2[10][100];
    fp = fopen("./in.c", "r");
    int dtype[10], cnt = 0;
    while((linelen = getline(&line, &len, fp)) != -1)
    {
        if(line[0] == '#')
        {
            for(int i = 0; i < strlen(line); i++)
            {
                if(line[i] != '\n') printf("%c", line[i]);
            }
            printf(" - preprocessor directive\n");
        }
        char* int1 = strstr(line,"int ");
        char* float1 = strstr(line,"float ");
        char* for1 = strstr(line,"for(");
        char* if1 = strstr(line,"if(");
        char* else1 = strstr(line,"else");
```

```
int declare = 0;
int conditional = 0;
if(int1 != NULL)
{
    declare = 1;
    printf("int - keyword\n");
    char* p = int1;
    char str[10];
    int slen = 0;
    char* t = p;
    int jumplen = strlen("int ");
    t = t + 4;
    while(*t != '\0')
    {
        char c = *t;
        str[slen++] = c;
        t = t + 1;
        if(*t == '=')
        {
            dtype[cnt++] = 0;
            t = t + 1;
            str[slen] = '\0';
            strcpy(store1[count], str);
            slen = 0;
            str[0] = '\0';
            while(isdigit(*t) || *t == '.')
            {
                char c = *t;
                str[slen++] = c;
                t = t + 1;
            }
            str[slen] = '\0';
            slen = 0;
            strcpy(store2[count], str);
        }
        if(*t ==',' | *t == ';')
        {
            count = count + 1;
            t = t + 1;
        }
    }
}
if(float1 != NULL)
{
    declare = 1;
    printf("float - keyword\n");
    char* p = float1;
    char str[10];
    int slen = 0;
    char* t = p;
    int jumplen = strlen("float ");
    t = t + 6;
    while(*t != '\0')
    {
        char c = *t;
        str[slen++]=c;
        t = t + 1;
        if(*t == '=')
        {

            dtype[cnt++] = 1;
            t = t + 1;
            str[slen] = '\0';
            strcpy(store1[count], str);
            slen = 0;
            str[0] = '\0';
            while(isdigit(*t) || *t == '.')
            {
                char c = *t;
                str[slen++] = c;
                t = t + 1;
            }
            str[slen] = '\0';
            slen = 0;
            strcpy(store2[count], str);
        }
        if(*t == ',' | *t == ';')
        {
            count = count + 1;
            t = t + 1;
        }
    }
}
if(for1 != NULL)
    printf("for - keyword\n");
if(if1 != NULL)
{
    printf("if - keyword\n");
    conditional = 1;
}
if(else1 != NULL)
    printf("else - keyword\n");
char* templine;
templine = line;
int first = 1;
if(declare == 1)
{
    while(templine != NULL)
    {
        if(first == 1)
        {
            templine = strstr(templine," ");
            first = 0;
        }
        else
        {
            printf(", - special character\n");
        }
        int equindex;
        for(int z = 0; z < strlen(templine); z++)
        {
            if(*(templine + z) == '=')
            {
                equindex = z;
                break;
            }
        }
        for(int j = 1; j < equindex; j++)
        {
            printf("%c", *(templine + j));
        }
        printf(" - variable\n");
        printf("= - assignment operator\n");
        templine = strstr(templine, "=");
        int commaindex;
        for(int z = 0; z < strlen(templine); z++)
        {
            if(*(templine + z) == ',')
            {
                commaindex = z;
                break;
            }
        }
        for(int j = 1; j < commaindex; j++)
        {
            printf("%c", *(templine + j));
        }
        printf(" - constant\n");
        templine = strstr(templine, ",");
    }
}
char* main1 = strstr(line, "main(");
char* printf1 = strstr(line, "printf(");
if(main1 != NULL || printf1 != NULL)
{
    for(int i = 0; i < strlen(line); i++)
    {
        if(line[i]=='\t' || line[i]==';' || line[i] == '\n')
        {
            printf(" ");
```

```
                }
                else
                {
                    printf("%c", line[i]);
                }
            }
            printf(" - function call\n");
        }
        char* popen = strstr(line, "{");
        if(popen != NULL) printf("{ - special character\n");
        char* semicolon = strstr(line, ";");
        if(semicolon != NULL) printf("; - special character\n");
        char* pclose = strstr(line, "}");
        if(pclose != NULL) printf("} - special character\n");
        char* bracket_open = strstr(line, "(");
        if(bracket_open != NULL && main1 == NULL && printf1 == NULL) printf("( - 
special character\n");
        char* tempvar;
        if(conditional == 1)
        {
            tempvar = strstr(line, "(");
            int i;
            int condition;
            for(int z = 0; z < strlen(tempvar); z++)
            {
                if(*(tempvar + z) == '<' || *(tempvar + z) == '>')
                {
                    condition = z;
                    break;
                }
            }
            for(int j = 1; j < condition; j++)
            {
                printf("%c", *(tempvar + j));
            }
            printf(" - variable\n");
            char* tempvar1 = strstr(tempvar, "<");
            char* tempvar2 = strstr(tempvar, ">");
            if(tempvar1 != NULL) tempvar = tempvar1;
            if(tempvar2 != NULL) tempvar = tempvar2;
            printf("%c - condition\n", *(tempvar));
            for(int z = 1; z < strlen(tempvar); z++)
            {
                if(*(tempvar + z) == ')')
                {
                    condition = z;
                    break;
                }
                else
                {
                    printf("%c", *(tempvar + z));
                }
            }
            printf(" - variable\n");
        }
        char* bracket_close = strstr(line, ")");
        if(bracket_close != NULL && main1 == NULL && printf1 == NULL) printf(") -
special character\n");
    }
    fclose(fp);
    return 0;
}
```

<u>Output:</u>



```
seccon@ubuntu:~$ cd Desktop
seccon@ubuntu:~/Desktop$ gcc -o a lex.c
seccon@ubuntu:~/Desktop$ ./a
main()  - function call
{ - special character
int - keyword
a - variable
= - assignment operator
10 - constant
, - special character
b - variable
= - assignment operator
20 - constant
; - special character
if - keyword
( - special character
a - variable
> - condition
b - variable
) - special character
printf("a is greater)   - function call
; - special character
else - keyword
printf("b is greater")   - function call
; - special character
} - special character
( - special character
) - special character
func() - function call
//end of program - single line comment
/* multi */ - multi line comment
seccon@ubuntu:~/Desktop$
```

```c
main()
{
int a=10,b=20;
if(a>b)
printf("a is greater);
else
printf("b is greater");
}
()
func()
//end of program
/* multi */
```

<u>Learning Outcome:</u>

- The role and operation of Lexical Analyser was understood.
- Implementation of Regular Expression has been learnt.
- Learnt to parse the program and token identification.
- Understood the role of a Lexical Analyser in compilation.
- Understood the significance of keywords and general structure of a C program.