UCS1602 - Compiler Design Lab
AY: *2022-23*

Name: *Krithika Swaminathan*
Roll No.: *205001057*

Assignment 1 – Implementation of Lexical Analyser and Symbol Table

Date: **10/02/2023**

Aim:

To implement a lexical analyser for a C program using C.

Code:

```
/* C program to implement a lexical analyser for C programs*/

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

#define maxbuffsize 1000

int startadd=1000;

struct symboltable{
char identifier_name[30],type[10],value[10];
int bytes,address;
}symbols[10];

int ind=0;

int issplchar(char buff)
{
   char splchars[]={';', ',' ,'.','[', ']', '(', ')', '{', '}', '[',']'};
   for(int i=0;i<11;i++)
   {
   if(buff==splchars[i]) return 1;
   }
   return 0;
}

int isfuncall(char buff[])
{
   char funcalls[][30]={"printf","main","scanf","getch","clrscr"};
   for(int i=0;i<5;i++)
   {
   if(strcmp(funcalls[i],buff)==0)
   return 1;
```

```
    }
    return 0;
}

int iskeyword(char buff[])
{
    char
keywords[][30]={"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"};
    for(int i=0;i<32;i++)
    {
    if(strcmp(keywords[i],buff)==0)
    return 1;
    }
    return 0;
}

int isdatatype(char buff[])
{
    char datatypes[][30]={"int","float","double","char"};
    for(int i=0;i<4;i++)
    {
    if(strcmp(datatypes[i],buff)==0)
    return 1;
    }
    return 0;
}

int numbytes(char buff[])
{
    if(strcmp("int",buff)==0) return 2;
    if(strcmp("float",buff)==0) return 4;
    if(strcmp("double",buff)==0) return 8;
    if(strcmp("char",buff)==0) return 1;
    return 0;
}

int isoperand(char buff)
{
    if(buff=='+' || buff=='-')return 1;
    if(buff=='*' || buff=='/' || buff=='%')return 2;
    if(buff=='<' || buff=='>' || buff=='!')return 3;
    if(buff=='=')return 4;
    if(buff=='&' || buff=='|')return 5;
    return 0;
}

int main()
{
    //Reading file into buffer
    char buff[maxbuffsize+1];
    FILE *fp=fopen("code.txt","r");
    if(!fp){printf("No such file found!\n");return 0;}
    int size=fread(buff,sizeof(char),maxbuffsize,fp);
    buff[size++]='\0';
    fclose(fp);
    printf("Sample code:\n%s\n\n",buff);

    printf("Lexical analysis: \n");
    for(int i=0;i<size;i++)
    {
        while(buff[i]!='\0' && buff[i]==' ' && buff[i]=='\n') i++;
        char temp[200]; int j=0;
        if(issplchar(buff[i]))
        {
        printf("%c - special character\n",buff[i]);
        continue;
        }
        if(isoperand(buff[i]))
        {
        if(isoperand(buff[i])==1)
        {
        if(buff[i]==buff[i+1]){
        printf("%c%c - unary operator\n",buff[i],buff[i+1]);
        i++;
        }
        else if(buff[i+1]=='='){
        printf("%c%c - arthmetic assignent operator\n",buff[i],buff[i+1]);
        i++;
        }
        else{
        printf("%c - arithmetic operator\n",buff[i]);
        }
        continue;
        }
        else if(isoperand(buff[i])==2)
        {
        if(buff[i+1]=='='){
        printf("%c%c - arithmetic assignment operator\n",buff[i],buff[i+1]);
        i++;
        }
        else
        printf("%c - arithmetic operator\n",buff[i]);
        continue;
        }
        else if(isoperand(buff[i])==3)
        {
        if(buff[i+1]=='=')
        {
        printf("%c%c - relational operator\n",buff[i],buff[i+1]);
        i++;
        }
        else{
        printf("%c - relational operator\n",buff[i]);
        }
        continue;
        }
        else if(isoperand(buff[i])==4)
        {
        if(buff[i]==buff[i+1]){
        printf("%c%c - relational operator\n",buff[i],buff[i+1]);
        i++;
        }
        else{
        printf("%c - assignment operator\n",buff[i]);
        }
        continue;
        }
        else if(isoperand(buff[i])==5)
        {
        if(buff[i]==buff[i+1]){
        printf("%c%c - logical operator\n",buff[i],buff[i+1]);
        i++;
        }
        else{
        printf("%c - bitwise operator\n",buff[i]);
        }
        continue;
        }
        }
        if(buff[i]=='#')
        {
        while(buff[i]!='\0' && buff[i]!='\n')
        {
```

```
temp[j++]=buff[i++];
}
temp[j]='\0';
printf("%s - preprocessor directive\n",temp);
continue;
}
if(isalpha(buff[i])||buff[i]=='_')
{
temp[j++]=buff[i++];
while(isalnum(buff[i]) || buff[i]=='_')
temp[j++]=buff[i++];
temp[j]='\0';
if(isfuncall(temp))
{
if(buff[i]=='('){
temp[j]=buff[i];
do{
i++;
j++;
temp[j]=buff[i];
}while(buff[i]!=')');
j+=1;
temp[j]='\0';
printf("%s - function call\n",temp);
continue;
}
}
else if(iskeyword(temp))
{
printf("%s - keyword\n",temp);
int store=i-1;
if(isdatatype(temp))
{
strcpy(symbols[ind].type,temp);
symbols[ind].bytes=numbytes(temp);
symbols[ind].address=startadd;
startadd+=numbytes(temp);
strcpy(temp,"");
j=0;
while(buff[i]==' ')i++;
if(isalpha(buff[i]) || buff[i]=='_')
{
temp[j++]=buff[i++];
while(isalnum(buff[i]) || buff[i]=='_')
temp[j++]=buff[i++];
```

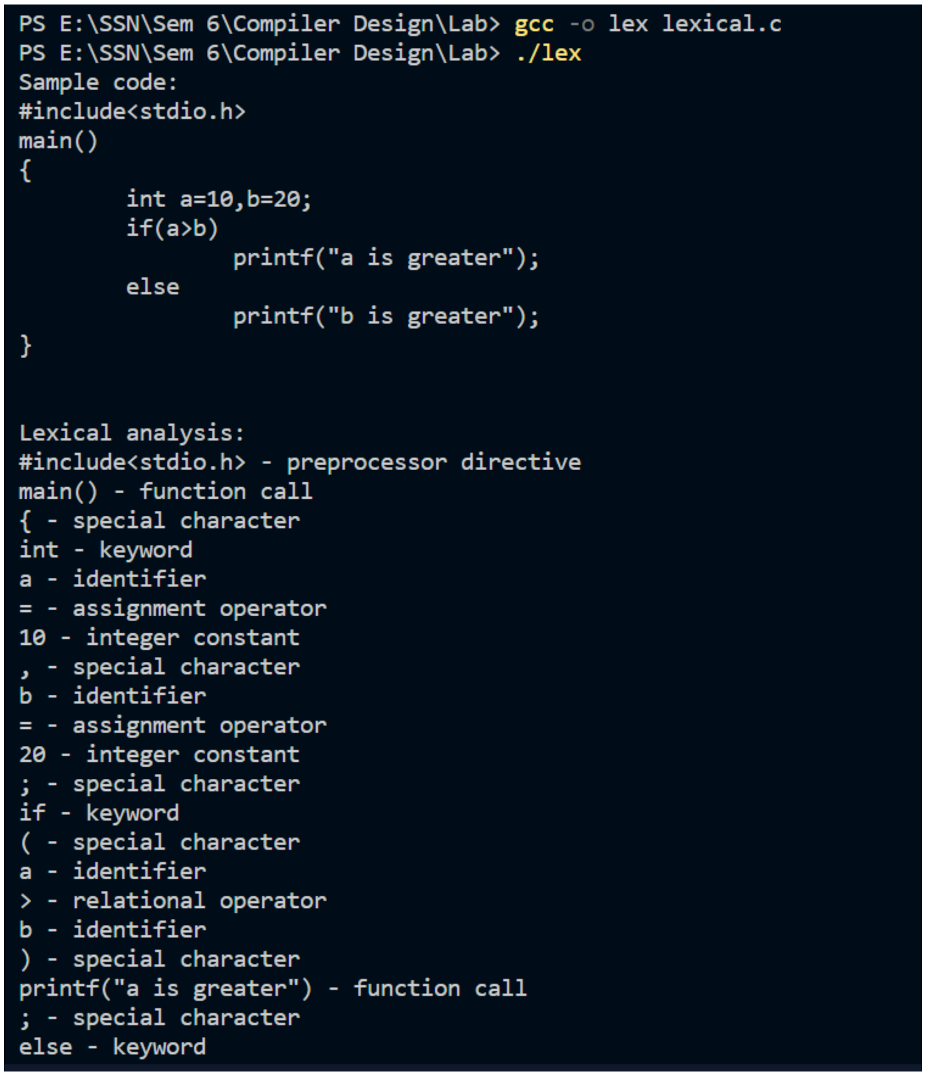
```
        temp[j]='\0';
        strcpy(symbols[ind].identifier_name,temp);
        }
        strcpy(temp,"");
        j=0;
        while(buff[i]==' ' || buff[i]=='=')i++;
        while(isdigit(buff[i]))
        temp[j++]=buff[i++];
        temp[j]='\0';
        strcpy(symbols[ind].value,temp);
        ind+=1;
        i=store;
        continue;
        }
        i-=1;
        continue;
        }
        else{
        printf("%s - identifier\n",temp);
        i-=1;
        continue;
        }
        }
        if(isdigit(buff[i]))
        {
        while(isdigit(buff[i]))
        temp[j++]=buff[i++];
        temp[j]='\0';
        i-=1;
        printf("%s - integer constant\n",temp);
        continue;
        }
    }
    printf("\nSymbol Table: \n\nname\ttype\tbytes\taddress\tvalue\n");
    for(int i=0;i<ind;i++)
    {

printf("%s\t%s\t%d\t%d\t%s\n\n",symbols[i].identifier_name,symbols[i].type,symbols[i].bytes,s
ymbols[i].address,symbols[i].
    value);
    }
    return 0;
}
```

Input: (Sample code)

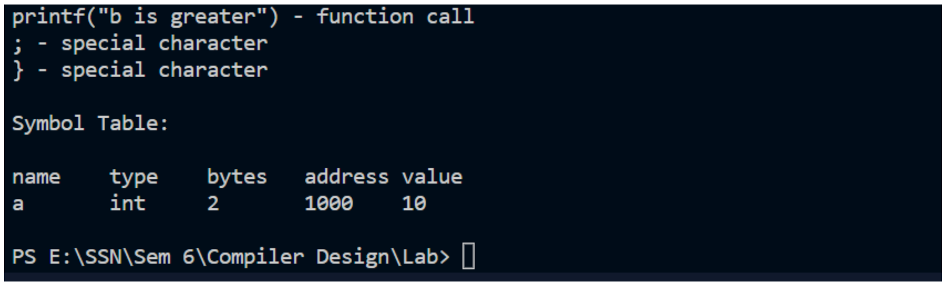
```
#include<stdio.h>
main()
{
	int a=10,b=20;
	if(a>b)
		printf("a is greater");
	else
		printf("b is greater");
}
```

Output:



```
PS E:\SSN\Sem 6\Compiler Design\Lab> gcc -o lex lexical.c
PS E:\SSN\Sem 6\Compiler Design\Lab> ./lex
Sample code:
#include<stdio.h>
main()
{
        int a=10,b=20;
        if(a>b)
                printf("a is greater");
        else
                printf("b is greater");
}


Lexical analysis:
#include<stdio.h> - preprocessor directive
main() - function call
{ - special character
int - keyword
a - identifier
= - assignment operator
10 - integer constant
, - special character
b - identifier
= - assignment operator
20 - integer constant
; - special character
if - keyword
( - special character
a - identifier
> - relational operator
b - identifier
) - special character
printf("a is greater") - function call
; - special character
else - keyword
```



```
printf("b is greater") - function call
; - special character
} - special character

Symbol Table:

name    type    bytes   address value
a       int     2       1000    10

PS E:\SSN\Sem 6\Compiler Design\Lab> 
```

Learning outcomes:

- The internal working of a compiler was analysed and understood.
- The concept of tokens and parsing for tokens in C was understood and implemented.
- A lexical analyser was implemented for a C program using C.