

IMPLEMENTATION OF LEXICAL ANALYSER AND SYMBOL TABLE IN C

Date: 02/02/2021

Name: MADHUMITHA S

Register Number: 185001086

1) Aim:

To implement Lexical Analyser and Symbol Table in C language to identify and tokenise identifiers, constants, comments etc.

2) PROGRAM CODE:

```
/*MADHUMITHA S 185001086*/
/*CD LAB EXCERCISE-01*/
/*02-FEB-2021*/

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

char identifier_arr[10][10];
int val_arr[10];
int pos=0,pos1=0;

char arith["+-*/%"]; //done
char arithas[5][3]={"+=","-=", "*=", "/=", "%="};
char logi[3][3]={"&&","||","!"};
char rel[3][3]={"<","<=", ">",">=", "==", "!="}; //done
char bitwise[3][3]={"^","&","|","<<",">>"};
char unary[3][3]={"-", "++", "--" };
char spl[";,.[](){}[]"]; //done
char funccall[12]={"printf()", "scanf()", "getch()", "clrscr()"};
char comments[4]={"//", "/*", "*/"};
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"};
char var_val[10][10];
char val[10][10];
char op[10];
```

```

void expeval(char st[]){
int i=0,k=0,flag=0,j,f1=0,p=0,count;

char buff[20];
char numbuff[20];
for(i=0;i<strlen(st);i++){
    f1=0;count=0;
    if(isalnum(st[i])||st[i]=='_'){
        buff[k++]=st[i];flag=1;
    }

    else{
        if(flag){buff[k]='\0';
        if(strcmp(buff,"")!=0){
            printf("%s\t-Identifier\n",buff);
            strcpy(var_val[pos++],buff);
            strcpy(buff,"");flag=0;k=0;}}
        if(st[i]=='='){printf(" %c\t-Assignment operator\n",st[i] );continue;}
        for(j=0;j<5;j++){
            if(st[i]==arith[j]){
                printf("%c\t-Arithmetic operator\n",st[i] );op[pos1]=st[i];f1=1;break;
            }
        }
        if(f1)continue;
        for(j=0;j<11;j++){
            if(st[i]==spl[j]){
                printf("%c\t-Special Character\n",st[i] );f1=1;break;
            }
        }
        if(f1)continue;
        buff[k++]=st[i];
        for(j=0;j<5;j++){
            if(strcmp(buff,arithas[j])==0){
                printf("%s\t-Arithassign operator\n",buff );
            }
        }
    }
    for(j=0;j<3;j++){
        if(strcmp(buff,logi[j])==0){
            printf("%s\t-LOgical operator\n",buff );
        }
    }
    for(j=0;j<6;j++){
        if(strcmp(buff,rel[j])==0){
            printf("%s\t-RElational operator\n",buff );
        }
    }
    for(j=0;j<3;j++){
        if(strcmp(buff,unary[j])==0){
            printf("%s\t-Unary operator\n",buff );
        }
    }
}
}

```

```

for(j=0;j<5;j++){
    if(strcmp(buff,bitwise[j])==0){
        printf("%s\t-Bitwise operator\n",buff );
    }
}

}
//printf("Buff cont:%s\n",buff );
for(p=0;p<strlen(buff);p++){
    if(isdigit(buff[p])){
        count=count+1;

    }

}if(count==(strlen(buff))){printf("%s\t-Constant\n", buff);strcpy(val[pos1++],buff);
strcpy(buff," ");
}

}

}

}

void symboltab(){
    int i=0,addr=1000,bts=2;
    printf("Symbol\t||\tValue\t||\tAddress\t||\tBytes\n");
    printf("-----\n");
    for(i=0;i<pos;i++){
        printf("%s\t||\t",var_val[i] );
        printf("%s\t||\t%d\t||\t%d\n", val[i],addr,bts);
        addr+=bts;
    }
}

void lexanalyse(){
    FILE* fptr;
    char ch;int flag=0;
    char temp[15];char dum;
    char tstr[20];
    char s[30];int cnt=0;
    strcpy(temp," ");
    int f=0;
    int i=0,j=0,k=0;//counters or iterators

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

```

```

if(fptr==NULL){
    printf("\n\nERR:File not found!\nQuitting.....");exit(0);
}
while((ch=fgetc(fptr))!=EOF){

    for(i=0;i<5;i++){
        if(ch==arith[i]){

if((dum=getc(fptr))=='='){
    printf("%c%c\t-ArithAssign operator\n",ch,dum);
}
if(dum=='*'){
    printf("/");cnt=0;
    while((dum=getc(fptr))!='*'){
        printf("%c",dum );
        cnt=cnt+1;
        if(cnt==100){printf("ERR:Closing tag missing for comments!!\n");
            exit(0);}

    }printf("%c\t-Comments\n",dum);
    ch=getc(fptr);
    ch=getc(fptr);
}
else if(dum=='/'){

    printf("//");
    while((dum=getc(fptr))!='\n'){
        printf("%c",dum );
    }printf("\t-comments\n");

}
else if(dum=='-' || dum=='+'){
    printf("%c%c\t-Unary operator\n",dum,dum );
    ch=getc(fptr);

}

        else
            printf("%c\t-arithmetic operator\n", ch);

    }

}
for(i=0;i<12;i++)

```

```

{
    if(ch==spl[i]){
        printf("%c\t-special character\n",ch );
    }
}

if(ch=='#'){
    printf("%c",ch);
    while((ch=getc(fp))!='>'){
        printf("%c",ch );
    }printf("%c\t-preprocessor directive\n",ch );ch++;

}

//strinf cons
if(ch==''){
    printf("%c",ch );
    while((ch=getc(fp))!=''){
        printf("%c",ch );
    }printf("%c\t-String constant\n", ch );
}

//relational
if(ch=='<' || ch=='>' || ch=='=' || ch=='!'){
    printf("%c",ch );
    ch=getc(fp);
    if(ch=='<' || ch=='>'){
        printf("%c\t-Bitwise operator\n",ch );flag=-1;ch=getc(fp);
    }
    else{printf("%c\t-Relational operator\n",ch );flag=0;}
}

//logical
if(ch=='&' || ch=='|'){
    printf("%c",ch );
    ch=getc(fp);
    if(ch=='&' || ch=='|'){
        printf("%c\t-Logical operator\n",ch );
        flag=0;
    }
    else{printf("\t-Bitwise operator\n");}
}

```

```

//numbers
if(isdigit(ch)){
    printf("%c",ch );
    while(isdigit(ch=getc(fp))){
        printf("%c",ch );

    }printf("\t-INteger constant\n");

}

//identifier and keywords
if(ch=='\n'){
    continue;
}
if(isalnum(ch)){
    //printf("%c\n",ch );
    flag=0;
    j=0;tstr[0]=ch;j++;
    while((ch=getc(fp))!='\n'){
        tstr[j++]=ch;
    }tstr[j]='\0';
    for(i=0;i<33;i++){
        if(strcmp(keys[i],tstr)==0){
            flag=1;
            printf("%s\t-Keyword\n",tstr);
        }

    }
    for(i=0;i<5;i++){
        if(strcmp(funccall[i],tstr)==0){
            flag=1;
            printf("%s\t-Function call\n",tstr);
        }

    }
    if(flag==0){
        j=0;
        for(k=0;k<strlen(tstr);k++){
            if(isalnum(tstr[k])||tstr[k]=='_'){
                //printf("%c-Identifier",tstr[k]);
                temp[j++]=tstr[k];

            }
            else

```

```

                break;
            }
            expeval(tstr);
        }
    }

f=0;
}
fclose(fptr);
return ;
}

int main(){
lexanalyse();
symboltab(); //function calls
return 0;
}

```

3) OUTPUT SCREENSHOTS:

When the closing tags for comments not given: Program exits.

```

madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx:~/cd_lab$ gcc CD_lab_exp_01.c -o exp2
madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx:~/cd_lab$ ./exp2
/*unclosed comments reports error*****ERR:Closing tag missing for
comments!!
madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx:~/cd_lab$

```

```
madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx: ~/cd_lab/A_01
madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx:~$ cd cd_lab
madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx:~/cd_lab$ cd A_01
madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx:~/cd_lab/A_01$ gcc CD_lab_exp_01.c -o exp2
madhu@madhu-HP-Pavilion-x360-Convertible-14-dh1xxx:~/cd_lab/A_01$ ./exp2
#include<stdio.h>      -preprocessor directive
#include<stdlib.h>     -preprocessor directive
printf()              -Function call
//hello this is      -comments
/*This is the code*/ -Comments
a                     -Identifier
=                     -Assignment operator
8                     -Constant
;                     -Special Character
static               -Keyword
<<                   -Bitwise operator
!=                   -Relational operator
b                     -Identifier
=                     -Assignment operator
1                     -Constant
;                     -Special Character
>
-Relational operator
c                     -Identifier
=                     -Assignment operator
6                     -Constant
;                     -Special Character
Symbol || Value || Address || Bytes
-----
a || 8 || 1000 || 2
b || 1 || 1002 || 2
c || 6 || 1004 || 2
```

4) LEARNING OUTCOME:

- Implementing a lexical analyser that separates the given code into tokens and each token is parsed about and analysed.
- Maintenance of a symbol table in the compiler during the lexical analysis phase storing address and values of identifiers.
- Learnt briefly about the lexical analysis phase of a compiler.