2.LEXICAL ANALYZER USING C

AIM

Design and implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and new lines**.**

Lex.c

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
void main()
{

FILE *fi,*fo,*fop,*fk; int flag=0,i=1;
char c,t,a[15],ch[15],file[20];
printf("\n Enter the File Name:");
scanf("%s",&file);
fi=fopen(file,"r");
fo=fopen("inter.c","w");
fop=fopen("oper.c","r");
fk=fopen("key.c","r"); c=getc(fi); while(!feof(fi))
{
if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1))
fputc(c,fo);
else
{
if(c=='\n') fprintf(fo,"\t$\t");
else fprintf(fo,"\t%c\t",c);
}
c=getc(fi);
}
fclose(fi); fclose(fo);
fi=fopen("inter.c","r"); printf("\n Lexical Analysis"); fscanf(fi,"%s",a);
printf("\n Line: %d\n",i++);
while(!feof(fi))
{
if(strcmp(a,"$")==0)
{
printf("\n Line: %d \n",i++); fscanf(fi,"%s",a);
}
fscanf(fop,"%s",ch); while(!feof(fop))
{
if(strcmp(ch,a)==0)
{
fscanf(fop,"%s",ch); printf("\t\t%s\t:\t%s\n",a,ch); flag=1;
} fscanf(fop,"%s",ch);
}
rewind(fop); fscanf(fk,"%s",ch);
while(!feof(fk))
{
if(strcmp(ch,a)==0)
{
```

```
fscanf(fk,"%k",ch); printf("\t\t%s\t:\tKeyword\n",a); flag=1;
}
fscanf(fk,"%s",ch);
}
rewind(fk); if(flag==0)
{
if(isdigit(a[0])) printf("\t\t%s\t:\tConstant\n",a); else printf("\t\t%s\t:\tIdentifier\n",a);
}
flag=0; fscanf(fi,"%s",a);
}
}
```

key.c

```
int void main char if for
while else printf scanf
FILE
Include stdio.h iostream.h
```

oper.c

```
( open para
) closepara
{ openbrace
} closebrace
< lesser
> greater
" doublequote ' singlequote
: colon
; semicolon
# preprocessor
= equal
== asign
% percentage
^ bitwise
& reference
* star
+ add
- sub
\ backslash
/ slash
```

input.c

```
#include "stdio.h"
void main()
{
int a=10,b,c;
a=b*c;
}
```

Input & Output:

```
[student@localhost]$ cc 1lex.c
[student@localhost]$ ./a.out

 Enter the File Name:input.c
```

Lexical Analysis
Line: 1
: preprocessor
include : Identifier
" : doublequote
stdio.h : Keyword
" : doublequote

Line: 2
void : Keyword
main : Keyword
(: open
) : closepara

Line: 3
{ : openbrace

Line: 4
int : Keyword
a : Identifier
= : equal
10 : Constant
, : Identifier
b : Identifier
, : Identifier
c : Identifier
; : semicolon

Line: 5
a : Identifier
= : equal
b : Identifier
* : star
c : Identifier
; : semicolon

Line: 6
} : closebrace

Line: 7
$: Identifier

3.REPLACE SPACE, TABS AND NEWLINES

AIM: - Implementation of lexical analyser using lex tool to ignore redundant spaces,tabs and new line

PROGRAM

```
%%
"  "*     {printf(" ");}
"\t"      {printf(" ");}
"\n"      {printf("");}
%%
int main()
{
extern FILE *yyin;
yyin=fopen("nw","r");
yylex();
}
int yywrap()
{
return 1;
}
```

OUTPUT

```
[student@localhost ]$lex replace.l
[student@localhost ]$ gcc -o replace lex.yy.c
[student@localhost ]$ ./replace
This   is          Compiler        Design Lab
This is Compiler Design Lab
```

4. IDENTIFICATION OF POSITIVE AND NEGATIVE INTEGERS

AIM:-

To create a lex program to identify positive and negative numbers, identifiers

PROGRAM

```
%%
[0-9]+ {printf("+ve no");}
-[0-9]+ {printf("-ve no");}
[a-zA-Z]+[A-Z0-9]* {printf("identifier");}
%%
int main()
{
yylex();
}

int yywrap()
{
return 1;
}
```

OUTPUT

```
student@145 ]$ lex posneg.l
[student@145 ]$ cc lex.yy.c
[student@145 ]$ ./a.out
34
+ve no
-34
-ve no
compiler
identifier
```

5.RECOGNISE A LANGUAGE

AIM:- To accept the following language L={0 ^{n}1^m, n>=1,m>=0}

PROGRAM

```
%{
#include<stdio.h>
int find=0;
%}
%%
0+1* {find=1;}
0*1+0+ {find=0;}
{}
\n {if(find==1)
printf("\n The string is accepted");
else
printf("\n The string is  not accepted");
exit(0);
}
%%
int main()
{
printf("Enter the string:");
yylex();
}
int yywrap()
{
return 1;
}
```

OUTPUT

```
[student@localhost ]$ lex arithexpr.l
[student@localhost ]$ cc lex.yy.c
[student@localhost ]$ ./a.out
Enter the string:00111
The string is accepted[student@localhost ]$ ./a.out
Enter the string:0110
The string is  not accepted
```

6. COUNTING OF CHARACTERS WORDS AND LINES

AIM: -To create a lex program that counts characters, words, lines.

PROGRAM

```
%{
#include<stdio.h>
int lines=0, words=0,s_letters=0,c_letters=0, num=0, spl_char=0,total=0;
%}

%%
\n { lines++; words++;}
[\t "] words++;
[A-Z] c_letters++;
[a-z] s_letters++;
[0-9] num++;
. spl_char++;
%%

main(void)
{
yyin= fopen("myfile.txt","r");
yylex();
total=s_letters+c_letters+num+spl_char;
printf(" This File contains ...");
printf("\n\t%d lines", lines);
printf("\n\t%d words",words);
printf("\n\t%d small letters", s_letters);
printf("\n\t%d capital letters",c_letters);
printf("\n\t%d digits", num);
printf("\n\t%d special characters",spl_char);
printf("\n\tIn total %d characters.\n",total);
}
int yywrap()
{
return(1);
}
```

OUTPUT

Contents of "myfile.txt" file

```
S7 CS
Complier Design Lab
[student@localhost ]$ lex cc.l
[student@localhost ]$ cc lex.yy.c
[student@localhost ]$ ./a.out
This File contains ...
2 lines
5 words
14 small letters
6 capital letters
1 digits
0 special characters
In total 21 characters.
```

7. RECOGNIZE VALID ARITHMETIC EXPRESSION

AIM: - To recognize a valid arithmetic expression that uses operator +, -, *, / etc.

PROGRAM

LEX

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[a-zA-Z]+ {return (VARIABLE);}
[0-9] {return (NUMBER);}
[\t]   {;}
[\n]  {return 0;}
.   {return yytext[0]; }
%%
int yywrap()
{
return 1;
}
```

YACC

```
%{
#include<stdio.h>
#include "lex.yy.c"
void yyerror();
%}
%token NUMBER
%token VARIABLE
%left  '+"-'
%left '*"/"%'
%left '("')'
%%
S: VARIABLE'='E{
        printf("\n Entered arithmetic expression is valid\n\n");
        return 0;}
E:E'+'E
| E'-'E
| E'*'E
| E'/'E
| E'%'E
```

```
| '('E')'
| NUMBER
| VARIABLE
;
%%
int main()
{
printf ("\n Enter any arithmetic expression which can have operations addition, subtraction, multiplication, division, modulus and rounded brackets:\n");
yyparse();
}
void yyerror()
{
printf("\n Entered arithmetic expression is invalid\n\n");
}
```

OUTPUT

```
[student@localhost ]$ lex validexpr.l
[student@localhost ]$ yacc -d validexpr.y
[student@localhost ]$ gcc -o validexpr y.tab.c
[student@localhost ]$ ./validexpr
Enter any arithmetic expression which can have operations addition, subtraction, multiplication, division, modulus and rounded brackets:
A=b+c*d
Entered arithmetic expression is valid
Enter any arithmetic expression which can have operations addition, subtraction, multiplication, division, modulus and rounded brackets:
a+b=c
Entered arithmetic expression is invalid
Enter any arithmetic expression which can have operations addition, subtraction, multiplication, division, modulus and rounded brackets:
S=(a+b)*c/d
Entered arithmetic expression is valid
```

8. RECOGNIZE A VALID VARIABLE

AIM: -

To recognize a valid variable which starts with a letter followed by any number of letters or digits.

PROGRAM

LEX

```
%{
#include "y.tab.h"
%}
%%
[0-9]  {return DIGIT;}
[a-zA-Z] {return LETTER;}
[\t] {;}
\n { return 0;}
. {return yytext[0];}
%%
int yywrap()
{
return 1;
}
```

YACC

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "lex.yy.c"
%}
%token DIGIT LETTER
%%
stmt: LETTER tail ;

tail:
     LETTER tail
   | DIGIT tail
   | /* empty */
   ;
%%
```

```
void main(){
printf("enter string \n");
yyparse();
printf("valid");
exit(0);
}
void yyerror()
{
printf("invalid");
exit(0);
}
```

OUTPUT

```
[student@145 ]$ lex validvar.l
[student@145 ]$ yacc -d validvar.y
[student@145 ]$ cc -o validvar y.tab.c
 [student@145 ]$ ./validvar
enter string
hello1
valid
[student@145 ]$ ./validvar
enter string
comp lab
invalid
[student@145 ]$ ./validvar
enter string
Hello 1
invalid
```