1. **Implement Lexical analyzer using C**

**AIM:** Implement Lexical analyzer / Scanner using C

**HARDWARE REQUIREMENTS:** Pentium III, 256MBRAM, 80GB HDD,keyboard, mouse, monitor.

**SOFTWARE REQUIREMENTS:** Windows XP, Turbo C.

**DESCRIPTION:** Lexical analysis is the process of converting a sequence of characters into a sequence of tokens. A program or function which performs lexical analysis is called a **lexical analyzer**, **lexer** or **scanner.** A **token** is a string of characters, categorized according to the rules as a symbol (e.g. IDENTIFIER, NUMBER, COMMA, etc.). The process of forming tokens from an input stream of characters is called **tokenization** and the lexer categorizes them according to a symbol type. A token can look like anything that is useful for processing an input text stream or text file.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

#define SIZE 128

#define NONE -1

#define EOS '\0'

#define NUM 256

#define KEYWORD 257

#define PAREN 258

#define ID 259

#define ASSIGN 260

#define REL\_OP 261

#define DONE 262

#define MAX 999

char lexemes[MAX];

char buffer[SIZE];

int lastchar=-1;

int lastentry =0;

int tokenval=NONE;

int lineno=1;

struct entry

{

char \*lexptr;

int token;

}

symtable[100];

/\*stores the list of keywords\*/

struct entry keywords[]=

{

"if",KEYWORD,"else",KEYWORD,"FOR",KEYWORD,"INT",KEYWORD,"FLOAT",KEYWORD,"DOUBLE",KEYWORD,"CHAR",KEYWORD,"STRUCT",KEYWORD,"RETURN",KEYWORD,

0,0};

/\*THIS FUNCTION ISSSUE A COMPILER ERROR\*/

void Error\_message(char \*m)

{

fprintf(stderr,"line %d: %s\n",lineno,m);

exit(1);

}

/\*this function is used to search the symbol table for particular entry\*/

int look\_up(char s[])

{

int k;

for(k=lastentry;k>0;k=k-1)

if(strcmp(symtable[k].lexptr,s)==0)

return k;

return 0;

}

int insert(char s[],int tok)

{

int len;

len=strlen(s);

if((lastentry+1)>=MAX)

Error\_message("Symbol table is full");

if((lastchar+len+1)>=MAX)

Error\_message("lexemas array is full");

lastentry=(lastentry +1);

symtable[lastentry].token=tok;

symtable[lastentry].lexptr=&lexemes[lastchar+1];

lastchar=lastchar + len+1;

strcpy(symtable[lastentry].lexptr,s);

return lastentry;

}

void Initialize()

{

struct entry \*ptr;

for(ptr=keywords;ptr->token;ptr++)

insert(ptr->lexptr,ptr->token);

}

int lexer()

{

int t;

int val,i=0;

while(1)

{

t=getchar();

if(t==' '||t=='\t')

;

else

if(t=='\n')

lineno=lineno+1;

else if(t=='('||t==')')

return PAREN;

else if(t=='<'||t=='>'||t=='>='||t=='<='||t=='!=')

return REL\_OP;

else if(t=='=')

return ASSIGN;

else if(isdigit(t))

{

ungetc(t,stdin);

scanf("%d",&tokenval);

return NUM;

}

else if(isalpha(t))

{

while(isalnum(t))

{

buffer[i]=t;

t=getchar();

i=i+1;

if(i>=SIZE)

Error\_message("compiler error");

}

buffer[i]=EOS;

if(t!=EOF)

ungetc(t,stdin);

val=look\_up(buffer);

if(val==0)

val= insert(buffer,ID);

tokenval=val;

return symtable[val].token;

}

else if(t==EOF)

return DONE;

else

{

tokenval=NONE;

return t;

}

}

}

void main()

{

int lookahead;

char ans;

clrscr();

printf("\n\t\t program for lexical analysis \n");

Initialize();

printf("\n Enter the expresssion and put ; at the end");

printf("\n press ctrl z to terminate ....\n");

lookahead=lexer();

while(lookahead!=DONE)

{

if(lookahead==NUM)

{

printf("\n NUmber:");

printf("%d",tokenval);

}

if(lookahead=='+'||lookahead=='-'||lookahead=='\*'||lookahead=='/')

printf("\n operator");

if(lookahead==PAREN)

printf("\n parenthesis");

if(lookahead==ID)

{

printf("\n Identifier:");

printf("%s",symtable[tokenval].lexptr);

}

if(lookahead==KEYWORD)

printf("\n keyword");

if(lookahead==ASSIGN)

printf("\n Assignment operator");

if(lookahead==REL\_OP)

printf("\n relational operator");

lookahead=lexer();

}

}

**OUTPUT:**

Program for lexical analysis

Enter the expression and put ; at the end

Press ctrl Z to terminate…

Else;

Keyword

1. **Simple Lex Programs**

**AIM:**. Lex programs to recognize Keywords, String ending with 00, starts and end with ‘k’ ,numbers with 1 in its 5th position from right, and To assign line numbers for source code

**HARDWARE REQUIREMENTS:** Pentium III, 256MBRAM, 80GB HDD,keyboard, mouse, monitor.

**SOFTWARE REQUIREMENTS:** Windows XP, Turbo C.

**DESCRIPTION:** Lex helps write programs whose control flow is directed by instances of regular expressions in the input stream. Lex source is a table of regular expressions and corresponding program fragments. The table is translated to a program which reads an input stream, copying it to an output stream and partitioning the input into strings which match the given expressions. As each such string is recognized the corresponding program fragment is executed. The recognition of the expressions is performed by a deterministic finite automaton generated by Lex. The program fragments written by the user are executed in the order in which the corresponding regular expressions occur in the input stream.

**SOURCE CODE:**

**1.//program to recognize keywords**

%{

#include<stdio.h>

%}

%%

int|float|char{ printf(" data type: %s",yytext); }

%%

main()

{

yylex();

return(0);

}

int yywrap()

{

return;

}

**OUTPUT:**

int

datatype: int

**2.//program to assign line numbers for source code.**

%{

#include<stdio.h>

int lineno=0;

%}

line .\*\n

%%

{line} { printf("%d .%s",lineno++,yytext); }

%%

main()

{

yylex();

return (0);

}

**OUTPUT:**

abc

def

ghi

1.abc

2.def

3.ghi

**3. //program to recognize the strings which are ending with 00**

%{

#include<stdio.h>

%}

%%

[a-z A-Z 0-9]++00 { printf("string is acepted",yytext);}

.\* { printf("not acepted",yytext);}

%%

main()

{

yylex();

return(0);

}

yywrap()

{

return;

}

**OUTPUT:**

As100

String is accepted

**4. Program to recognize the numbers which has 1 in its 5th position from right**

%{

#include<stdio.h>

%}

%%

[0-9]\*1[0-9]{4} { printf("acepted:");}

[0-9]\*1[0-9] { printf("NOT acepted:");}

%%

main()

{

yylex();

return(0);

}

**OUTPUT:**

3410001

accepted

123456

NOT accepted

**5. Program to recognize the strings which are starting or ending with ‘k’**

%{

#include<stdio.h>

%}

begin-with-k k.\*

end-with-k .\*k

%%

{begin-with-k} { printf(" %s is a word that begin with k",yytext);}

{end-with-k} { printf(" %s is a word that end with k",yytext);}

%%

main()

{

yylex();

return(0);

}

**OUTPUT:**

kishore

kishore is a word that begin with k

shaik

shaik is a word that end with k