**Lab 4: CONSTRUCTION OF SYMBOL TABLE**

1.Using getNextToken( ) implemented in Lab No 3, design a Lexical Analyzer to implement local and global symbol table to store tokens for identifiers using array of structure.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

const char \*keywords[] = {"auto","double","int","struct","break","else",

"long","switch","case","enum", "register","typedef","char","extern","return",

"union","continue","for","signed","void","do","if","static","while","default",

"goto","sizeof","volatile","const","float","short","unsigned","printf","scanf",

"true","false"};

const char \*datypes[]={"int","char","void","float","bool"};

int isdtype(char \*w)

{

int i;

for(i=0;i<sizeof(datypes)/sizeof(char\*);i++)

{

if(strcmp(w,datypes[i])==0)

{

return 1;

}

}

return 0;

}

int isKeyword(char \*w)

{

int i;

for(i=0;i<sizeof(keywords)/sizeof(char\*);i++)

{

if(strcmp(w,keywords[i])==0)

{

return 1;

}

}

return 0;

}

struct token

{

char lexeme[128];

unsigned int row,col;

char type[64];

};

struct sttable

{

int sno;

char lexeme[128];

char dtype[64];

char type[64];

int size;

};

int findTable(struct sttable \*tab,char \*nam,int n)

{

int i=0;

for(i=0;i<n;i++)

{

if(strcmp(tab[i].lexeme,nam)==0)

{

return 1;

}

}

return 0;

}

struct sttable fillTable(int sno,char \*lexn,char \*dt,char \*t,int s)

{

struct sttable tab;

tab.sno=sno;

strcpy(tab.lexeme,lexn);

strcpy(tab.dtype,dt);

strcpy(tab.type,t);

tab.size=s;

return tab;

}

void printTable(struct sttable \*tab,int n)

{

for(int i=0;i<n;i++)

{

printf("%d %s %s %s %d\n",tab[i].sno,tab[i].lexeme,tab[i].dtype,tab[i].type,tab[i].size);

}

}

static int row=1,col=1;

char buf[2048];

char dbuf[128];

int ind=0;

const char specialsymbols[]={'?',';',':',','};

const char arithmeticsymbols[]={'\*'};

int charIs(int c,const char \*arr)

{

int len;

if(arr==specialsymbols)

{

len=sizeof(specialsymbols)/sizeof(char);

}

else if(arr==arithmeticsymbols)

{

len=sizeof(arithmeticsymbols)/sizeof(char);

}

for(int i=0;i<len;i++)

{

if(c==arr[i])

{

return 1;

}

}

return 0;

}

void fillToken(struct token \*tkn,char c,int row,int col, char \*type)

{

tkn->row=row;

tkn->col=col;

strcpy(tkn->type,type);

tkn->lexeme[0]=c;

tkn->lexeme[1]='\0';

}

void newLine()

{

++row;

col=1;

}

int sz(char \*w)

{

if(strcmp(w,"int")==0)

return 4;

if(strcmp(w,"char")==0)

return 1;

if(strcmp(w,"void")==0)

return 0;

if(strcmp(w,"float")==0)

return 8;

if(strcmp(w,"bool")==0)

return 1;

}

struct token getNextToken(FILE \*fa)

{

int c;

struct token tkn=

{

.row=-1

};

int gotToken=0;

while(!gotToken && (c=fgetc(fa))!=EOF)

{

if(charIs(c,specialsymbols))

{

fillToken(&tkn,c,row,col,"SS");

gotToken=1;

++col;

}

else if(charIs(c,arithmeticsymbols))

{

fseek(fa,-1,SEEK\_CUR);

c=getc(fa);

if(isalnum(c))

{

fillToken(&tkn,c,row,col,"ARITHMETICOPERATOR");

gotToken=1;

++col;

}

fseek(fa,1,SEEK\_CUR);

}

else if(c=='(')

{

fillToken(&tkn,c,row,col,"LB");

gotToken=1;

col++;

}

else if(c==')')

{

fillToken(&tkn,c,row,col,"RB");

gotToken=1;

col++;

}

else if(c=='{')

{

fillToken(&tkn,c,row,col,"LC");

gotToken=1;

col++;

}

else if(c=='}')

{

fillToken(&tkn,c,row,col,"RC");

gotToken=1;

col++;

}

else if(c=='[')

{

fillToken(&tkn,c,row,col,"LS");

gotToken=1;

col++;

}

else if(c==']')

{

fillToken(&tkn,c,row,col,"RS");

gotToken=1;

col++;

}

else if(c=='+')

{

int x=fgetc(fa);

if(x!='+')

{

fillToken(&tkn,c,row,col,"ARITHMETICOPERATOR");

gotToken=1;

col++;

fseek(fa,-1,SEEK\_CUR);

}

else

{

fillToken(&tkn,c,row,col,"UNARYOPERATOR");

strcpy(tkn.lexeme,"++");

gotToken=1;

col+=2;

}

}

else if(c=='-')

{

int x=fgetc(fa);

if(x!='-')

{

fillToken(&tkn,c,row,col,"ARITHMETICOPERATOR");

gotToken=1;

col++;

fseek(fa,-1,SEEK\_CUR);

}

else

{

fillToken(&tkn,c,row,col,"UNARYOPERATOR");

strcpy(tkn.lexeme,"++");

gotToken=1;

col+=2;

}

}

else if(c=='=')

{

int x=fgetc(fa);

if(x!='=')

{

fillToken(&tkn,c,row,col,"ASSIGNMENTOPERATOR");

gotToken=1;

col++;

fseek(fa,-1,SEEK\_CUR);

}

else

{

fillToken(&tkn,c,row,col,"RELATIONALOPERATOR");

strcpy(tkn.lexeme,"++");

gotToken=1;

col+=2;

}

}

else if(isdigit(c))

{

fillToken(&tkn,c,row,col++,"NUMBER");

int j=1;

while((c=fgetc(fa))!=EOF && isdigit(c))

{

tkn.lexeme[j++]=c;

col++;

}

tkn.lexeme[j]='\0';

gotToken=1;

fseek(fa,-1,SEEK\_CUR);

}

else if(c == '#')

{

while((c = fgetc(fa))!= EOF && c != '\n');

newLine();

}

else if(c=='\n')

{

newLine();

c = fgetc(fa);

if(c == '#')

{

while((c = fgetc(fa)) != EOF && c != '\n');

newLine();

}

else if(c != EOF)

{

fseek(fa, -1, SEEK\_CUR);

}

}

else if(isspace(c))

{

++col;

}

else if(isalpha(c) || c=='\_')

{

tkn.row=row;

tkn.col=col++;

tkn.lexeme[0]=c;

int j=1;

while((c=fgetc(fa))!=EOF && isalnum(c))

{

tkn.lexeme[j++]=c;

col++;

}

tkn.lexeme[j]='\0';

if(isKeyword(tkn.lexeme))

{

strcpy(tkn.type,"KEYWORD");

}

else

{

strcpy(tkn.type,"IDENTIFIER");

}

gotToken=1;

fseek(fa,-1,SEEK\_CUR);

}

else if(c=='/')

{

int d=fgetc(fa);

++col;

if(d=='/')

{

while((c=fgetc(fa))!= EOF && c!='\n')

{

++col;

}

if(c=='\n')

{

newLine();

}

}

else if(d=='\*')

{

do

{

if(d=='\n')

{

newLine();

}

while((c==fgetc(fa))!= EOF && c!='\*')

{

++col;

if(c=='\n')

{

newLine();

}

}

++col;

}while((d==fgetc(fa))!= EOF && d!='/' && (++col));

++col;

}

else

{

fillToken(&tkn,c,row,--col,"ARITHMETIC OPERATOR");

gotToken=1;

fseek(fa,-1,SEEK\_CUR);

}

}

else if(c=='"')

{

tkn.row=row;

tkn.col=col;

strcpy(tkn.type, "STRING LITERAL");

int k = 1;

tkn.lexeme[0] = '"';

while((c = fgetc(fa)) != EOF && c != '"')

{

tkn.lexeme[k++] = c;

++col;

}

tkn.lexeme[k] = '"';

gotToken = 1;

}

else if(c == '<' || c == '>' || c == '!')

{

fillToken(&tkn, c, row, col, "RELATIONALOPERATOR");

++col;

int d = fgetc(fa);

if(d == '=')

{

++col;

strcat(tkn.lexeme, "=");

}

else

{

if(c == '!')

{

strcpy(tkn.type, "LOGICALOPERATOR");

}

fseek(fa, -1, SEEK\_CUR);

}

gotToken = 1;

}

else if(c == '&' || c == '|')

{

int d = fgetc(fa);

if(c == d)

{

tkn.lexeme[0] = tkn.lexeme[1] = c;

tkn.lexeme[2] = '\0';

tkn.row = row;

tkn.col = col;

++col;

gotToken = 1;

strcpy(tkn.type, "LOGICALOPERATOR");

}

else

{

fseek(fa, -1, SEEK\_CUR);

}

++col;

}

else

{

++col;

}

}

return tkn;

}

int main()

{

FILE \*fa, \*fb;

int ca, cb;

fa = fopen("q1l1.c", "r");

if (fa == NULL)

{

printf("Cannot open file \n");

exit(0);

}

fb = fopen("sampleoutq1l1.c", "w+");

ca = getc(fa);

while (ca != EOF)

{

if(ca==' ')

{

putc(ca,fb);

while(ca==' ')

ca = getc(fa);

}

if (ca=='/')

{

cb = getc(fa);

if (cb == '/')

{

while(ca != '\n')

ca = getc(fa);

}

else if (cb == '\*')

{

do

{

while(ca != '\*')

ca = getc(fa);

ca = getc(fa);

} while (ca != '/');

}

else

{

putc(ca,fb);

putc(cb,fb);

}

}

else putc(ca,fb);

ca = getc(fa);

}

fclose(fa);

fclose(fb);

fa = fopen("sampleoutq1l1.c", "r");

if(fa == NULL)

{

printf("Cannot open file");

return 0;

}

fb = fopen("temp.c", "w+");

ca = getc(fa);

while (ca != EOF)

{

if(ca=='"')

{

putc(ca,fb);

ca=getc(fa);

while(ca!='"')

{

putc(ca,fb);

ca=getc(fa);

}

}

else if(ca=='#')

{

while(ca!='\n')

{

ca=getc(fa);

}

ca=getc(fa);

}

putc(ca,fb);

ca = getc(fa);

}

fclose(fa);

fclose(fb);

fa = fopen("temp.c", "r");

fb = fopen("sampleoutq1l1.c", "w");

ca = getc(fa);

while(ca != EOF)

{

putc(ca, fb);

ca = getc(fa);

}

fclose(fa);

fclose(fb);

remove("temp.c");

FILE \*f1=fopen("sampleoutq1l1.c","r");

if(f1==NULL)

{

printf("Error! File cannot be opened!\n");

return 0;

}

struct token tkn;

struct sttable st[10][100];

int flag=0,i=0,j=0;

int tabsz[10];

char w[25];

w[0]='\0';

while((tkn=getNextToken(f1)).row!=-1)

{

printf("<%s, %d, %d>\n",tkn.lexeme,tkn.row,tkn.col);

if(strcmp(tkn.type,"KEYWORD")==0)

{

if(isdtype(tkn.lexeme)==1)

{

strcpy(dbuf,tkn.lexeme);

}

}

else if(strcmp(tkn.type,"IDENTIFIER")==0)

{

strcpy(w,tkn.lexeme);

tkn=getNextToken(f1);

printf("<%s, %d, %d>\n",tkn.lexeme,tkn.row,tkn.col);

if((strcmp(tkn.type,"LB"))==0)

{

if(findTable(st[i],w,j)==0)

{

ind++;

st[i][j++]=fillTable(ind,w,dbuf,"func",-1);

}

}

if((strcmp(tkn.type,"LS"))==0)

{

if(findTable(st[i],w,j)==0)

{

tkn=getNextToken(f1);

printf("<%s, %d, %d>\n",tkn.lexeme,tkn.row,tkn.col);

int s=0;

if(strcmp(tkn.type,"NUMBER")==0)

{

s=atoi(tkn.lexeme);

}

ind++;

st[i][j++]=fillTable(ind,w,dbuf,"id",sz(dbuf)\*s);

}

}

else

{

if(findTable(st[i],w,j)==0)

{

ind++;

st[i][j++]=fillTable(ind,w,dbuf,"id",sz(dbuf));

}

}

}

else if(strcmp(tkn.type,"LC")==0)

{

flag++;

}

else if(strcmp(tkn.type,"RC")==0)

{

flag--;

if(flag==0)

{

tabsz[i]=j;

i++;

j=0;

ind=0;

}

}

}

int k=0;

for(k=0;k<i;k++)

{

printTable(st[k],tabsz[k]);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

}

fclose(f1);

}

Output







