**CD Lab**

**Session 4 – Lab 9 (21/12/2020)**

Parthivi Choubey CSE – B - 5th semester

180905456 Roll. no. - 60

**Question 1**

**Code**

lex.c

#include <stdlib.h>

#include <stdio.h>

#include <ctype.h>

#include <string.h>

static int row=1,col=1;

char buf[2048];

char dbuf[128];

int ind=0;

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

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

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", "short", "unsigned", "printf", "scanf", "true", "false", "bool"};

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

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 isKeyword(char \*w)

{

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

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

return 1;

return 0;

}

int isdtype(char \*w)

{

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

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

return 1;

return 0;

}

void newLine()

{

++row; col=1;

}

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);

}

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

{

for(int 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 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';

}

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;

}

int sz(char \*w)

{

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

return sizeof(int);

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

return sizeof(char);

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

return 0;

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

return sizeof(float);

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;

}

lab9q1.c

#include "lex.c"

void program();

void declarations();

void datatype();

void idList();

void idListprime();

void idListprimePrime();

void stmtList();

void stmt();

void assignStat();

void expn();

void eprime();

void simpleExpn();

void seprime();

void term();

void tprime();

void factor();

void relop();

void addop();

void mulop();

void decStat();

void dPrime();

void loopStat();

struct token tkn;

FILE \*f1;

int main()

{

FILE \*fa, \*fb;

int ca, cb;

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

if (fa == NULL){

printf("Cannot open file \n");

exit(0);

}

fb = fopen("output.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("output.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);

}

}

putc(ca,fb);

ca = getc(fa);

}

fclose(fa);

fclose(fb);

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

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

ca = getc(fa);

while(ca != EOF){

putc(ca, fb);

ca = getc(fa);

}

fclose(fa);

fclose(fb);

remove("temp.c");

f1=fopen("output.c","r");

if(f1==NULL)

{

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

return 0;

}

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

{

if(strcmp(tkn.lexeme, "main") == 0)

{

program();

break;

}

}

fclose(f1);

}

void program()

{

if(strcmp(tkn.lexeme, "main") == 0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "(") == 0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, ")") == 0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "{") == 0)

{

tkn = getNextToken(f1);

declarations();

stmtList();

if(strcmp(tkn.lexeme, "}") == 0)

{

printf("Compilation successful\n");

return;

}

else

{

printf("ERROR: missing \"}\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"{\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \")\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"(\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"main\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void declarations()

{

if(isdtype(tkn.lexeme)==0)

return;

datatype();

idList();

if(strcmp(tkn.lexeme, ";") == 0)

{

tkn = getNextToken(f1);

declarations();

}

else

{

printf("ERROR: missing \";\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void datatype()

{

if(strcmp(tkn.lexeme, "int") == 0)

{

tkn = getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme, "char") == 0)

{

tkn = getNextToken(f1);

return;

}

else

{

printf("ERROR: missing datatype(int or char) at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void idList()

{

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

{

tkn = getNextToken(f1);

idListprime();

}

else

{

printf("ERROR: missing IDENTIFIER at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void idListprime()

{

if(strcmp(tkn.lexeme, ",") == 0)

{

tkn = getNextToken(f1);

idList();

}

else if(strcmp(tkn.lexeme, "[") == 0)

{

tkn = getNextToken(f1);

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

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "]") == 0)

{

tkn = getNextToken(f1);

idListprimePrime();

}

else

{

printf("ERROR: missing \"]\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing NUMBER at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

}

void idListprimePrime()

{

if(strcmp(tkn.lexeme, ",") == 0)

{

tkn = getNextToken(f1);

idList();

}

else

return;

}

void stmtList()

{

if(strcmp(tkn.type,"IDENTIFIER")==0 || strcmp(tkn.lexeme,"if") == 0 || strcmp(tkn.lexeme,"for") == 0 || strcmp(tkn.lexeme,"while") == 0)

{

stmt();

stmtList();

}

return;

}

void stmt()

{

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

{

assignStat();

if(strcmp(tkn.lexeme, ";") == 0)

{

tkn = getNextToken(f1);

return;

}

else

{

printf("ERROR: missing \";\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else if(strcmp(tkn.lexeme, "if")==0)

decStat();

else if((strcmp(tkn.lexeme, "while")==0) || (strcmp(tkn.lexeme, "for")==0))

loopStat();

else

{

printf("%d.%d : Expected \" statement \"\n",tkn.row,tkn.col);

exit(0);

}

}

void assignStat()

{

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

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "=") == 0)

{

tkn = getNextToken(f1);

expn();

}

else

{

printf("ERROR: missing \"=\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing IDENTIFIER at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void expn()

{

simpleExpn();

eprime();

}

void eprime()

{

if(strcmp(tkn.type,"RELATIONALOPERATOR")!=0)

return;

relop();

simpleExpn();

}

void simpleExpn()

{

term();

seprime();

}

void seprime()

{

if((strcmp(tkn.lexeme, "+") != 0) && (strcmp(tkn.lexeme, "-") != 0))

return;

addop();

term();

seprime();

}

void term()

{

factor();

tprime();

}

void tprime()

{

if((strcmp(tkn.lexeme, "\*") != 0) && (strcmp(tkn.lexeme, "/") != 0)&& (strcmp(tkn.lexeme, "%") != 0))

return;

mulop();

factor();

tprime();

}

void factor()

{

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);

return;

}

else

{

printf("ERROR: Expected IDENTIFIER or NUMBER at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void relop()

{

if(strcmp(tkn.lexeme,"==")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,"!=")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,"<=")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,">=")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,"<")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,">")==0)

{

tkn=getNextToken(f1);

return;

}

else

{

printf("ERROR: Expected RELATIONAL OPERATOR or NUMBER at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void addop()

{

if(strcmp(tkn.lexeme,"+")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,"-")==0)

{

tkn=getNextToken(f1);

return;

}

else

{

printf("ERROR: Expected \"+\" or \"-\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void mulop()

{

if(strcmp(tkn.lexeme,"\*")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,"/")==0)

{

tkn=getNextToken(f1);

return;

}

else if(strcmp(tkn.lexeme,"\*")==0)

{

tkn=getNextToken(f1);

return;

}

else

{

printf("ERROR: Expected \"\*\" or \"/\" or \"%%\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void decStat()

{

if(strcmp(tkn.lexeme, "if")==0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "(") == 0)

{

tkn = getNextToken(f1);

expn();

if(strcmp(tkn.lexeme, ")") == 0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "{") == 0)

{

tkn = getNextToken(f1);

stmtList();

if(strcmp(tkn.lexeme, "}") == 0)

{

tkn = getNextToken(f1);

dPrime();

return;

}

else

{

printf("ERROR: missing \"}\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"{\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \")\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"(\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"keyword\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

void dPrime()

{

if(strcmp(tkn.lexeme, "else")==0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "{") == 0)

{

tkn = getNextToken(f1);

stmtList();

if(strcmp(tkn.lexeme, "}") == 0)

{

tkn = getNextToken(f1);

return;

}

else

{

printf("ERROR: missing \"}\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"{\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

return;

}

void loopStat()

{

if(strcmp(tkn.lexeme, "while")==0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "(") == 0)

{

tkn = getNextToken(f1);

expn();

if(strcmp(tkn.lexeme, ")") == 0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "{") == 0)

{

tkn = getNextToken(f1);

stmtList();

if(strcmp(tkn.lexeme, "}") == 0)

{

tkn = getNextToken(f1);

return;

}

else

{

printf("ERROR: missing \"}\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"{\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \")\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"(\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else if(strcmp(tkn.lexeme, "for")==0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "(") == 0)

{

tkn = getNextToken(f1);

assignStat();

if(strcmp(tkn.lexeme, ";") == 0)

{

tkn = getNextToken(f1);

expn();

if(strcmp(tkn.lexeme, ";") == 0)

{

tkn = getNextToken(f1);

assignStat();

if(strcmp(tkn.lexeme, ")") == 0)

{

tkn = getNextToken(f1);

if(strcmp(tkn.lexeme, "{") == 0)

{

tkn = getNextToken(f1);

stmtList();

if(strcmp(tkn.lexeme, "}") == 0)

{

tkn = getNextToken(f1);

return;

}

else

{

printf("ERROR: missing \"}\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"{\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \")\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \";\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \";\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"(\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("ERROR: missing \"keyword\" at row=%d col=%d\n",tkn.row,tkn.col);

exit(1);

}

}

**Output**

