Vedant Satarkar-180905336

CD Week9

Q1. Modify the Recursive Descent parser implemented in the previous lab to parse

decision making and looping statements with error reporting. Subset of grammar 7.1

is as follows:

statement->assign-stat|decision\_stat|looping-stat

decision-stat->if(expn){statement\_list}dprime

Code

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include "lex.c"

void program();

void declarations();

void datatype();

void idlist();

void idlistprime();

void assignstat();

void statementlist();

void statement();

void expn();

void eprime();

void simpleexp();

void seprime();

void term();

void tprime();

void factor();

void decisionstat();

void dprime();

void loopingstat();

void relop();

void addop();

void mulop();

struct token tkn;

FILE \*f1;

char \*rel[]={"==","!=","<=",">=",">","<"};

char \*add[]={"+","-"};

char \*mul[]={"\*","/","%"};

int isrel(char \*w)

{

int i;

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

{

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

{

return 1;

}

}

return 0;

}

int isadd(char \*w)

{

int i;

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

{

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

{return 1;

}

}

return 0;

}

int ismul(char \*w)

{

int i;

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

{

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

{

return 1;

}

}

return 0;

}

int main()

{

FILE \*fa, \*fb;

int ca, cb;

fa = fopen("./sampleIn9.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);

}

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;

}

}

printf("Compiled sucessfully\n");

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();

statementlist();

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

{

return;

}

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

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

{

loopingstat();

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

{

return;exit(0);

}

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

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

{

loopingstat();

}

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

{

decisionstat();

}

else

{

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

exit(1);

}

}

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

{

decisionstat();

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

{

return;

}

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

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

{

loopingstat();

}

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

{

decisionstat();

}

else

{

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

exit(1);

}

}

else

{

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

exit(1);

}

}

else

{

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

exit(1);

}

}

else{

printf(") missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("( missing at row=%d col=%d",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("; missing at row=%d col=%d",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("%s Missing datatype at row=%d col=%d",tkn.lexeme, tkn.row,tkn.col);

exit(1);

}

}

void idlist()

{

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

tkn=getNextToken(f1);

idlistprime();

}

else

{

printf("Missing IDENTIFIER at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

void idlistprime()

{

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

{

tkn=getNextToken(f1);

idlist();

}

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);

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

{

tkn=getNextToken(f1);

idlist();

}

else

{

return;

}

}

else

{

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

exit(1);

}

}

}

else

{

return;

}

}

void statementlist()

{

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

{return;

}

statement();

statementlist();

}

void statement()

{

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

{

assignstat();

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

{

tkn=getNextToken(f1);

return;

}

else

{

printf("; missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

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

{

decisionstat();

}

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

{

loopingstat();

}

}

void assignstat()

{

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

{

tkn=getNextToken(f1);

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

{

tkn=getNextToken(f1);

expn();

}

else

{

printf("= missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("Missing IDENTIFIER at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}void expn()

{

simpleexp();

eprime();

}

void eprime()

{

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

{

return;

}

relop();

simpleexp();

}

void simpleexp()

{

term();

seprime();

}

void seprime()

{

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

{

return;

}

addop();

term();

seprime();

}

void term()

{

factor();

tprime();

}

void tprime()

{

if(ismul(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;

}

}

void decisionstat()

{

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);

statementlist();

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

{

tkn=getNextToken(f1);

dprime();

}

else

{

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

exit(1);

}

}

else

{

printf("{} missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf(") missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("( missing at row=%d col=%d",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);

statementlist();

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

{

tkn=getNextToken(f1);

return;

}

else

{

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

exit(1);

}

}

else

{

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

exit(1);

}

}

else

{

return;

}

}

void loopingstat()

{

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);

statementlist();

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

{

tkn=getNextToken(f1);

return;

}

else{

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

exit(1);

}

}

else

{

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

exit(1);

}

}

else

{

printf(") missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("( missing at row=%d col=%d",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);

statementlist();

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

{

tkn=getNextToken(f1);

return;

}

else

{

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

exit(1);

}

}

else

{

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

exit(1);

}

}

else

{

printf(") missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("; missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("; missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

else

{

printf("( missing at row=%d col=%d",tkn.row,tkn.col);

exit(1);

}

}

}

void relop()

{

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);return;

}

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);

return;

}

}

void addop()

{

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);

return;

}

}

void mulop()

{

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);

return;

}

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

{

tkn=getNextToken(f1);

return;

}

}

Output

