PROGRAMMING LANGUAGES PROJECT 2

05110000002 AHMET GÜL

05110000044 ONURHAN ÇELİK

05110000056 Ü.ANIL ÖZTÜRK

05110000082 CENGİZ BURSALIOĞLU

**HATA MESAJLARI VE ANLAMLARI**

Unrecognized token at Arithmetic Expresion > Aritmetik ifadede tanımsız karakter bulunması.

Unrecognized token at Output Clause > Output ifadesinde tanımsız karakter bulunması.

Unclosed Parenthesis > Kapanmamış parantez olması.

Unknown charachter > Dile ait olmayan karakter olması.

Unclosed String Literal > String sabitinin kapanmaması.

Identifiers must start with a letter > Identifier’lar harfle başlamalıdır.

**PROGRAMA DAİR ÖNEMLİ NOT !**

* **Programımızda yapılan aritmetik işlemlerde sayılar tek basamaklı olmalıdır.**

**BNF GRAMMER**

<assign> => <name>:=<expr>

<name>=><name>< any\_char>|<alphabetic>

<alphabetic> =>A|B|…|Z|a|b|….|z

<numeric>=>0|1|…|9

<any\_char>=> < alphabetic ><numeric>

<expr>=><name>+<expr>

|<name>-<expr>

|<name>\*<expr>

|<name>/<expr>

|(<expr>)

|<name>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <locale.h> //türkçe karakter yazdırmak için.

#include <stdbool.h> // to use bool

/\* Charachter classes \*/

#define LETTER 0

#define DIGIT 1

#define OTHER 2

/\* Token codes \*/

#define KEYWORD 10

#define IDENT 11

#define INT\_LIT 12

#define STR\_LIT 13

#define ASSIGN\_OP 14

#define LEFT\_PAREN 15

#define RIGHT\_PAREN 16

#define ADD\_OP 17

#define SUB\_OP 18

#define DIV\_OP 19

#define MULT\_OP 20

#define EOL\_SYM 21

/\*Exception Codes\*/

#define EX\_LONG\_ID 100

#define EX\_STRING\_UNCLOSED 101

#define EX\_ID\_DIGIT 102

#define EX\_UNKNOWN 103

#define EX\_UNCLOSED\_PAR 104

#define EX\_OUT\_UNR 105

#define EX\_EXPR\_UNR 106

#define MAX 50

typedef char \*string;

struct lexx

{

char str\_lit[100];

int token;

int deger;

};

struct lexx expr\_list[20][100];

struct lexx id\_list[20];

int expr\_no=0;

int tok\_no=0;

int out\_no=0;

struct lexx outs[100];

/\* Global Declarations \*/

struct lexx matrice[20][100]= {{}};

int charClass;

char lexeme[100];

char nextChar;

int lexLen;

int nextToken;

int lineNo = 1;

int columnNo = 0;

long int position;

FILE \*x\_cal, \*fopen();

/\*Function Declarations \*/

void addChar();

void getChar();

void getNonBlank();

int lookUp(char ch);

int lex();

int isKeyword();

void exceptionHandler(int exception);

void parser();

void expr();

void term();

void fact();

void outputSt();

void addOut();

string keyword\_list[1]= {"OUTPUT"};

string token\_list[12]= {"KEYWORD","IDENT","INT\_LIT","STR\_LIT","ASSIGN\_OP","LEFT\_PAREN","RIGHT\_PAREN","ADD\_OP",

"SUB\_OP","DIV\_OP","MULT\_OP","EOL\_SYM"

};

void push(int \*yigin, int \*tepe, int yeni);

int pop(int \*yigin, int \*tepe);

int tepe\_eleman(int \*yigin, int tepe);

int bos\_mu(int tepe);

int oncelik\_bul(char karakter);

void ortaek\_sonek\_donustur(struct lexx infix[20][100], char \*sonek, int rw);

int hesapla(char \*sonek);

int main()

{

setlocale(LC\_ALL,"Turkish");

char dosya\_adi[20];

printf("Lutfen '.cal' dosyasinin ismini uzantisiz olarak giriniz: ");

gets(dosya\_adi);

strcat(dosya\_adi,".cal");

if((x\_cal=fopen(dosya\_adi,"r"))== NULL)

printf("ERROR - cannot open fort.txt \n");

else

{

parser();

fclose(x\_cal);

}

int i;

for(i=0; i<expr\_no; i++)

{

char sonek[50];

ortaek\_sonek\_donustur(expr\_list,sonek,i);

id\_list[i].deger=hesapla(sonek);

}

for(i=0; i<out\_no; i++)

{

if(outs[i].token==STR\_LIT)

printf("%s ",outs[i].str\_lit);

else if(outs[i].token==IDENT)

{

int j = 0;

for(j=0; j<expr\_no; j++)

{

if(strcmp(outs[i].str\_lit,id\_list[j].str\_lit)==0)

{

printf(" %d ",id\_list[j].deger);

}

}

}

}

return 0;

}

void push(int \*yigin, int \*tepe, int yeni)

{

//hem donusturme hem hesaplamada ayni push fonksiyonu (tamsayi ekleyen) kullaniliyor

//donusturme isleminde yigina ekleme yapilirken karakterin ascii kodu eklenmektedir

if(\*tepe==MAX-1)

printf("yigin dolu!\n");

else

{

\*tepe=\*tepe+1;

yigin[\*tepe]=yeni;

}

return;

}

int pop(int \*yigin, int \*tepe)

{

//hem donusturme hem hesaplamada ayni pop fonksiyonu (tamsayi donduren) kullaniliyor

//donusturme isleminde yigindan cikarilan sayi karaktere cevrilerek kullanilmaktadir.

int gecici;

gecici=yigin[\*tepe];

\*tepe=\*tepe-1;

return gecici;

}

int tepe\_eleman(int \*yigin, int tepe)//tepedeki elemani dondur ama yigindan cikarma

{

return yigin[tepe];

}

int bos\_mu(int tepe)

{

return tepe==-1;//yigin bossa dogru(1) bos degilse yanlis(0) dondur

}

int oncelik\_bul(char karakter)

{

int oncelik=3;// \*/+- disindaki tum karakterler icin

switch (karakter)

{

case '\*':

case '/':

oncelik=2;

break;

case '+':

case '-':

oncelik=1;

break;

}

return oncelik;

}

void ortaek\_sonek\_donustur(struct lexx infix[20][100], char \*sonek,int rw)

{

int yigin[MAX];//yiginda karakterlerin ascii kodu saklanacak...

int tepe=-1;

int i,j=0;

push(yigin,&tepe,(infix[rw][0].str\_lit)[0]);//ilk karakter direk yigina ekleniyor

for(i=1; (infix[rw][i].str\_lit)[0]!='\0'; i++) //ortaek ifadenin sonuna kadar...

{

//yigin bos degilse ve tepedeki elemanin onceligi eldekinin onceligine esit

//ya da daha buyuk ise onu yigindan cikar ve sonek ifadenin sonuna ekle

while(!bos\_mu(tepe) && oncelik\_bul(tepe\_eleman(yigin,tepe))>=oncelik\_bul((infix[rw][i].str\_lit)[0]))

{

sonek[j]=pop(yigin,&tepe);

j++;

}

push(yigin,&tepe,(infix[rw][i].str\_lit)[0]);//eldekini yigina ekle

}

//yiginda kalanlari sonek ifadeye ekle

while(!bos\_mu(tepe))

{

sonek[j]=pop(yigin,&tepe);

j++;

}

sonek[j]='\0';//sonek ifade, string haline getiriliyor

return;

}

int hesapla(char \*sonek)

{

int yigin[MAX];

int tepe=-1;

int i,sayi,operand1,operand2;

for(i=0; sonek[i]!='\0'; ++i)

{

if(isdigit(sonek[i]))//karakter, rakam ise...

{

sayi=sonek[i]-48;//sayiya donustur(0-9 arasi rakamlarin ascii kodlari 48-57 arasinda

push(yigin,&tepe,sayi);//ve yigina ekle

}

else if(isalpha(sonek[i])) //karakter, harf ise...

{

int j;

for(j=0; j<expr\_no; j++)

{

if((id\_list[j].str\_lit)[0]==sonek[i])

{

sayi=id\_list[j].deger;

break;

}

}

push(yigin,&tepe,sayi);//ve yigina ekle

}

else//karakter, bir islem operatoru ise...

{

operand2=pop(yigin,&tepe);//ikinci operandi yigindan cikar

operand1=pop(yigin,&tepe);//ilk operandi yigindan cikar

switch(sonek[i])//islemi yap ve sonucunu yigina ekle

{

case '\*':

push(yigin,&tepe,operand1\*operand2);

break;

case '/':

push(yigin,&tepe,operand1/operand2);

break;//tam bolme yapiliyor!

case '+':

push(yigin,&tepe,operand1+operand2);

break;

case '-':

push(yigin,&tepe,operand1-operand2);

break;

}

}

}

return pop(yigin,&tepe);//yiginda kalan son eleman sonek ifadenin sonucudur

}

void addChar() //lexeme'e o karakteri ekliyor.

{

if(lexLen<=98)

{

lexeme[lexLen++] = nextChar;

lexeme[lexLen]=0;

}

else printf("Error - too long");

}

void getChar() //Bir sonraki karakteri alýyor.

{

if ((nextChar=getc(x\_cal))!=EOF)

{

if(isalpha(nextChar))

{

charClass = LETTER;

nextChar=toupper(nextChar);

}

else if (isdigit(nextChar))

charClass = DIGIT;

else

charClass = OTHER;

columnNo++;

}

else charClass = EOF;

if(nextChar=='\n')

{

lineNo++;

columnNo=0;

}

}

void getNonBlank() //Boþluklarý geçiyor.

{

while(isspace(nextChar))

getChar();

}

void exceptionHandler(int exception) //Hatalarý belirtiyor.

{

switch(exception)

{

case EX\_LONG\_ID:

printf("Error > Line %d, Column: %d: Identifier is too long! (%s)\n",lineNo,columnNo,lexeme);

break;

case EX\_ID\_DIGIT:

printf("Error > Line %d, Column %d: Identifiers MUST start with a letter!\n",lineNo,columnNo);

break;

case EX\_STRING\_UNCLOSED:

printf("Error > Line %d, Column %d: Unclosed String Literal!\n",lineNo,columnNo);

nextToken = 0;

break;

case EX\_UNKNOWN:

printf("Error > Line %d, Column %d: Unknown charachter! (%s)\n",lineNo,columnNo,lexeme);

nextToken = 0;

break;

case EX\_UNCLOSED\_PAR:

printf("Error > Line %d, Column %d: Unclosed Parenthesis!\n",lineNo,columnNo);

break;

case EX\_OUT\_UNR:

printf("Error > Line %d, Column %d: Unrecognized token at Output Clause!\n",lineNo,columnNo);

break;

case EX\_EXPR\_UNR:

printf("Error > Line %d, Column %d: Unrecognized token at Arithmetic Expresion!\n",lineNo,columnNo);

break;

}

}

int isKeyword() //Keyword olup olmadýðýna bakýyor ve ona göre atama yapýyor.

{

int i,flag=0;

for(i=0; i<1; i++)

{

if(strcmp(lexeme,keyword\_list[i])==0)

{

flag=1;

return KEYWORD;

}

}

if(flag==0 && strlen(lexeme)>25)

{

exceptionHandler(EX\_LONG\_ID);

return 0;

}

return IDENT;

}

int lookUp(char ch) //Sembol,Operatör türünü belirliyor.

{

switch (ch)

{

case '(':

nextToken=LEFT\_PAREN;

break;

case ')':

nextToken=RIGHT\_PAREN;

break;

case ';':

nextToken=EOL\_SYM;

break;

case '\*':

position=ftell(x\_cal);

getChar();

if (isdigit(nextChar) || isalpha(nextChar) || nextChar=='(')

{

nextToken=MULT\_OP;

fseek(x\_cal,position,SEEK\_SET);

}

break;

case ':':

position=ftell(x\_cal);

getChar();

if(nextChar=='=')

{

addChar();

nextToken=ASSIGN\_OP;

}

else

{

exceptionHandler(EX\_UNKNOWN);

fseek(x\_cal,position,SEEK\_SET);

}

break;

case '+':

nextToken=ADD\_OP;

break;

case '-':

nextToken=SUB\_OP;

break;

case '/':

nextToken=DIV\_OP;

break;

case '\"':

position=ftell(x\_cal);

do

{

getChar();

if(nextChar!='\"')

addChar();

}

while(nextChar!=';' && nextChar!='\"');

if(nextChar=='\"')

{

nextToken=STR\_LIT;

}

else

{

exceptionHandler(EX\_STRING\_UNCLOSED);

position=ftell(x\_cal);

}

break;

default:

exceptionHandler(EX\_UNKNOWN);

break;

}

return nextToken;

}

int lex () //Tokenlarý belirliyor,dosyaya ve ekrana yazýyor.

{

lexLen=0;

getNonBlank();

switch(charClass)

{

case LETTER:

addChar();

getChar();

while(charClass==LETTER || charClass == DIGIT || nextChar=='\_')

{

addChar();

getChar();

}

nextToken=isKeyword();

break;

case DIGIT:

addChar();

getChar();

while(charClass == DIGIT)

{

addChar();

getChar();

}

nextToken=INT\_LIT;

break;

case OTHER:

if(nextChar!='\"')

addChar();

lookUp(nextChar);

getChar();

break;

case EOF:

nextToken = EOF;

lexeme[0] = 'E';

lexeme[1] = 'O';

lexeme[2] = 'F';

lexeme[3] = 0;

break;

}

return nextToken;

}

void parser()

{

getChar();

lex();

do

{

if(nextToken==ASSIGN\_OP)

{

lex();

expr\_list[expr\_no][tok\_no].token=nextToken;

strcpy(expr\_list[expr\_no][tok\_no].str\_lit,lexeme);

tok\_no++;

expr();

expr\_no++;

tok\_no=0;

}

else if (nextToken==KEYWORD)

outputSt();

else

{

if(nextToken==IDENT)

{

id\_list[expr\_no].token=nextToken;

strcpy(id\_list[expr\_no].str\_lit,lexeme);

}

lex();

}

}

while(nextToken!=EOF);

}

void expr()

{

term();

while (nextToken == ADD\_OP || nextToken == SUB\_OP)

{

lex();

expr\_list[expr\_no][tok\_no].token=nextToken;

strcpy(expr\_list[expr\_no][tok\_no].str\_lit,lexeme);

tok\_no++;

term();

}

}

void term()

{

fact();

/\* As long as the next token is \* or /, get the

next token and parse the next factor \*/

while (nextToken == MULT\_OP || nextToken == DIV\_OP)

{

lex();

expr\_list[expr\_no][tok\_no].token=nextToken;

strcpy(expr\_list[expr\_no][tok\_no].str\_lit,lexeme);

tok\_no++;

fact();

}

}

void fact()

{

if (nextToken == IDENT || nextToken == INT\_LIT)

{

/\* Get the next token \*/

lex();

if(nextToken!=EOL\_SYM)

{

expr\_list[expr\_no][tok\_no].token=nextToken;

strcpy(expr\_list[expr\_no][tok\_no].str\_lit,lexeme);

tok\_no++;

}

}

else

{

if (nextToken == LEFT\_PAREN)

{

lex();

expr\_list[expr\_no][tok\_no].token=nextToken;

strcpy(expr\_list[expr\_no][tok\_no].str\_lit,lexeme);

tok\_no++;

expr();

if (nextToken == RIGHT\_PAREN)

{

lex();

expr\_list[expr\_no][tok\_no].token=nextToken;

strcpy(expr\_list[expr\_no][tok\_no].str\_lit,lexeme);

tok\_no++;

}

else

exceptionHandler(EX\_UNCLOSED\_PAR);

}

else

exceptionHandler(EX\_EXPR\_UNR);

}

}

void outputSt()

{

do

{

lex();

if(nextToken==IDENT || nextToken==STR\_LIT)

{

strcpy(outs[out\_no].str\_lit,lexeme);

outs[out\_no].token=nextToken;

out\_no++;

}

//mainde string sabitine en son yazdırmada kullanılack bişey yapılcak

else if(nextToken!=EOL\_SYM)

{

exceptionHandler(EX\_OUT\_UNR);

}

}

while(nextToken!=EOL\_SYM);

}