PROGRAMMING LANGUAGES

PROJECT1

Lexical Analyzer for Psi++

Kerem Alp ÖZECİK
05170000049
Hasan ORAL
05170000073

15 MAYIS 2020

Programming Languages Project 1: Lexical Analyzer for Psi++

PROGRAMIN AMACI:

Lexical analiz karakter dizisini, token dizisine çevirme işlemine denir.
Token dediğimiz şey ise, bir veya daha fazla karakterden oluşan ve grup olarak önem taşıyan karakter dizisi anlamına gelir.
Bu analizi yapan kodlara da lexer denir.
Çoğu zaman, lexer'ların oluşturduğu token dizisi, parser tarafından işlenir.
Bunlar bir dil oluşturmak ve bu dili analiz etmek için kullanılır.
Biz de bu projede C dilinde program içinde tanımladığımız identifier, integer, string, brackets, yorum satırları,operatör ve keywordlerimizin olduğu bir lexical anaylyser yaptık.
Bu işlemleri yaparken de main in içinde kolaylık oluşturması için "isNumber", "checkKeywords", "isDelimiter", "isOperator" ve "EndOrBracket" fonksiyonlarını yazdık.

- "isNumber" fonksiyonumuzu "code.psi"'ın içindeki sayıları okurken "code.lex" dosyamıza "IntConst" şeklinde olacak outputlar için kullandık.

- "checkKeywords" fonksiyonumuzu "code.psi"'ın içindeki keywordleri okurken "code.lex" dosyamıza "Keyword" şeklinde olacak outputlar için kullandık.

- "isOperator" fonksiyonumuzu "code.psi"'ın içindeki operatörleri okurken "code.lex" dosyamıza "Operator" şeklinde olacak outputlar için kullandık.

- "isDelimiter" fonksiyonumuzu "code.psi"'ın içindekileri okurken "code.lex" dosyamıza operatörleri yazdırmak için elemanları bölmeye yarayan fonksiyondur.

- "EndOrBracket" fonksiyonumuzu "code.psi"'ın içindeki sayıları okuduktan sonra "EndOfLine" yazmaya yaramasının yanı sıra çeşitli parantezleri bastırma durumunda kullandığımız fonksiyondur.

PROGRAM İÇİN HARCANAN SÜRE:

Programı yazmadan önce planlama yapmamız yaklaşık 1 saatimizi aldı.
Programı yazma sürecimiz ise 6 gün sürdü. Bu 6 gün boyunca her gün yaklaşık 2 saat vaktimizi aldı.

Kaynak Kod:

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
```

```
#include <stdlib.h>

// bolucu fonksiyonumuz
bool isDelimiter(char ch)
{
if (ch == ' ' || ch == '+' || ch == '-' || ch == '*'
|| ch == '/'  || ch == ':' )
return (true);
return (false);
}

// Operatör kontrolu icin fonksiyonumuz
bool isOperator(char ch){

{
if (ch == '+' || ch == '-' || ch == '*' ||
ch == '/' || ch == '=' || ch == ':')
return (true);
return (false);
}
}


void EndOrBracket(char harf,FILE *fileLex){


if(harf == ';'){ // ; karakteri kontrolü yapiyoruz EndOfLine bastirmak icin
fputs("EndOfLine\n",fileLex);
}
//Parantez karakterleri bloklari
if(harf == '('){
fputs("LeftParantheses\n", fileLex);
}
if(harf == ')'){
fputs("RightParantheses\n", fileLex);
}
if(harf == '['){
fputs("LeftSquareBracket\n", fileLex);
}
if(harf == ']'){
fputs("RightSquareBracket\n", fileLex);
}
if(harf == '{'){
fputs("LeftCurlyBracket\n", fileLex);
}
if(harf == '}'){
fputs("RightCurlyBracket\n", fileLex);
}
}

//sayilari kontrol eden fonksiyonumuz
bool isNumber(char ch){
```

```
if (ch == '0' || ch == '1' || ch == '2' || ch == '3'
|| ch == '4'  || ch == '5'|| ch == '6'|| ch == '7'|| ch == '8'|| ch == '9' )
return (true);
return (false);

}

//keyword olup olamdigini kontrol eden fonksiyonumuz
int checkKeyword(char buffer[]){
char keywords[18][10] =
{"break","case","char","const","continue","do","else","enum","float","for","goto","if","int",
"long","record","return","static","while"};
int i, flag = 0;
for(i = 0; i < 18; ++i){
if(strcmp(keywords[i], buffer) == 0){
flag = 1;
break;
}
}

return flag;
}


int main(int argc, char *argv[]) {
int int_length=0;
int id_length=0;
char temp;
char words[20];
char strings[100];
int i=0;
int j=0;
int k=0;
int m=0;

int id_mi=1;
FILE *file, *fileLex;
file = fopen("code.psi", "r");//dosyamizi okumak icin actik
fileLex = fopen("code.lex", "w");//bu dosyamizi da alacagimiz sonuclari icine yazdirmak icin
olusturduk

char harf=getc(file);
if(harf==-1){
printf("Dosya Bulunamadi!!!");
}
while(!feof(file)){

if(harf == 32 || (8 < harf && harf < 14)){ //Bosluk karakterleri icin if blogu
harf = getc(file);
}
if((96 < harf && harf <123) || (64 < harf && harf < 91)){
```

```
i = 0;
j = 0;
id_length = 0;
id_mi=1;


while(isupper(harf) || islower(harf) || isNumber(harf)|| harf == 95){ //kucuk ve buyuk harfler,
rakamlar, "_" kontrolu


if(i > 20){//identifierin uzunlugunun 20'dan fazla olma durumu icin if blogumuz
printf("Identifier isimleri 20'den fazla olamaz! Program burada sonlandi!!");
return 1;
}
if((isNumber(harf))|| harf == '_'){ //alınan karakterin sayi ya da "_" olup olmadigini kontrol eder.
id_mi=0;    }
if(isupper(harf)){ //buyuk karakter kucuk karaktere donusturulur.
harf = tolower(harf);
}
words[i] = harf;
harf = getc(file);
i++;
}// iç while sonu


words[i] = '\0'; //Wordsun sonuna null karakter eklendi
id_length=i;



if(id_mi==1){//id_mi 1 donerse keywords bastirir

if(checkKeyword(words) == 1){

fputs("Keyword(", fileLex);

while(j< id_length){
fputc(words[j], fileLex);
j++;
}
fputs(")\n", fileLex);


}
else{
id_mi=0;//keyword kontrolunden bir sey bulamazsa id_mi 0 olur bu sayede identifierimizi
bastirabiliriz
}

if(id_mi==0) {//identifier bastirmak icin kod blogumuz
fputs("Identifier(", fileLex);
```

```
while(j< id_length){

fputc(words[j], fileLex);

j++;
}
fputs(")\n", fileLex);

}

}


}

if(isDelimiter(harf)){//oparatorleri kontrol etmek icin bolucu fonksiyonumuzu kullandik

if(harf==':'){
harf=getc(file);

if(harf=='='){//":=" oparatoru icin kontrol blogu
fputs("Operator(:=)\n", fileLex);

}}//if sonu :

else if(isOperator(harf))  {


if(harf=='+'){
harf=getc(file);
if(harf=='+'){
fputs("Operator(++)\n", fileLex);//"++" operator kontrolu icin

}
else{
fputs("Operator(+)\n", fileLex);
continue;
}

}
else if(harf=='-'){
harf=getc(file);
if(harf=='-'){// "--" operator kontrolu icin
fputs("Operator(--)\n", fileLex);

}
else{
fputs("Operator(-)\n", fileLex);
continue;
}

}
```

```
else if(harf == '/'){ //alinan karakter bolum isareti
temp=harf;//eger sadece bolum isareti varsa diye harfi tempe atadik
harf = getc(file);
if(harf == '*'){ //bolum isaretinden sonra * geldi mi gelmedi diye kontrol blogumuz
while(!feof(file)){
harf = getc(file);
if(harf == '*'){ //yildiz geldiyse ve sonraki karakterimiz bolum ise yorum sonu olduğunu belirten
kontrol blogumuz
harf = getc(file);
if(harf == '/')
break;
}
if(harf == EOF){//yorum satirinda sorun olmasi durumundaki if blogumuz
printf("Unterminated comment! Program burada sonlandi!");
return 1;
}
}
}

else{

fputs("Operator(", fileLex);
fputc(temp,fileLex);//sadece bolum isareti(yorum satiri kullanilmamissa) varsa tempi bastirir
fputs(")\n", fileLex);

}

}

else{

fputs("Operator(", fileLex);
fputc(harf,fileLex);
fputs(")\n", fileLex);

}

}//else if operator

}//delimeter sonu

if (harf=='"')
{
//Alinan karakter çift tirnak
harf = getc(file);
strings[k] = harf;
k++;
while(harf !='"' && harf != EOF){
harf = getc(file);
strings[k] = harf;
```

```
k++;
}
strings[k] = '\0';
if(harf == '"'){
fputs("StrConst(", fileLex);
while(m < (strlen(strings)-1)){
fputc(strings[m], fileLex);
m++;
}
fputs(")\n", fileLex);
}
else if(harf == EOF){ //tirnak isaretinde sorun olmasi durumundaki kosul blogumuz
printf("Missing terminating! Program burada sonlandi!");
return 1;
}
}//string if sonu

if(isNumber(harf)){

fputs("IntegerConst(", fileLex);

while(isNumber(harf)){
if(int_length > 10){//integerlarin uzunlugunun 10'dan fazla olma durumu icin if blogumuz
printf("Integers 10 karakterden fazla olamaz! Program burada Sonlandi!");
return 1;
}
fputc(harf, fileLex);
harf = getc(file);
int_length++;


}
fputs(")\n", fileLex);
int_length = 0;
continue;

}

EndOrBracket(harf,fileLex);//parantezler ve EndOfLine satirlarimiz icin fonksiyonumuzu calistirdik.
harf=getc(file);

}//ana while sonu
fclose(file);
fclose(fileLex);
fflush(stdin);
return 0;

}
```

Hatasız durum

Text file:

```
hi:=hello+25;
goto/57;
further++goo;
hasanOraL--;
"hasani(n)patenti";
(kerem);
/*burasionemli*/;
```

Çıktılar:

```
Identifier(hi)
Operetor(:=)
Identifier(hello)
Operator(+)
IntegerConst(25)
EndOfLine
Keyword(goto)
Operetor(/)
IntegerConst(57)
EndOfLine
Identifier(further)
Operetor(++)
Identifier(goo)
EndOfLine
Identifier(hasanoral)
Operetor(--)
EndOfLine
StrConst(hasani(n)patent
i)
EndOfLine
LeftParantheses
Identifier(kerem)
RightParantheses
EndOfLine
EndOfLine//yorum satırı
olduğu için code.lex e
yazılmadı ama end of
line verdi.
```

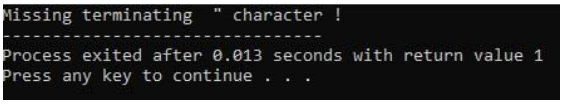
Tırnak hatası

Text file:

hi:=hello+25;
goto/57;
further++goo;
hasanOraL--;
"hasani(n)patenti;→Tırnak kapanmadı
(kerem);
/*burasionemli*/;

Çıktılar:

```
Identifier(hi)
Operetor(:=)
Identifier(hello)
Operator(+)
IntegerConst(25)
EndOfLine
Keyword(goto)
Operetor(/)
IntegerConst(57)
EndOfLine
Identifier(further)
Operetor(++)
Identifier(goo)
EndOfLine
Identifier(hasanoral)
Operetor(--)
EndOfLine
```



Program hata mesajı verip çalışmayı durdurdu.

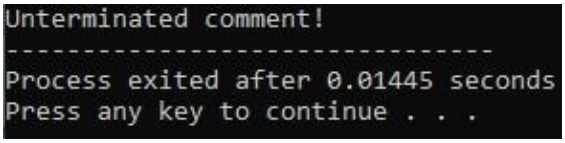
Yorum Satırı Hatası

Text file:

hi:=hello+25;
goto/57;
/*burasionemli;→*/ ifadesi ile bitmesi gerekiyordu.
further++goo;
hasanOraL--;
"hasani(n)patenti";
(kerem);

Çıktılar:

```
Identifier(hi)
Operetor(:=)
Identifier(hello)
Operator(+)
IntegerConst(25)
EndOfLine
Keyword(goto)
Operetor(/)
IntegerConst(57)
EndOfLine
```



Program hata mesajı verip çalışmayı durdurdu.

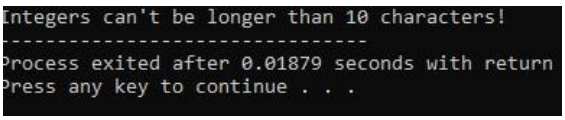
Int Boyutu Hatası

Text file:

hi:=hello+2500000000000000000000;→int 10 karakterden fazla
goto/57;
/*burasionemli*/;
further++goo;
hasanOraL--;
"hasani(n)patenti";
(kerem);

Çıktılar:

```
Identifier(hi)
Operetor(:=)
Identifier(hello)
Operator(+)
IntegerConst(25000000000
```



Program hata mesajı verip çalışmayı durdurdu.

String Boyutu
Hatası

Text file:

hi:=helloooooooooooooooooooooo
oooooooooooooooooooooooooooo
oooooooooooooooooooooooooooo
oooooooooooooooooooooooooooo
ooooooooooooooooooooooo+25;

id 20 karakterden fazla

goto/57;

/*burasionemli*/;

further++goo;

hasanOraL--;

(kerem);

Çıktılar:

```
Identifier(hi)
Operetor(:=)
```

```
Identifier names can't be longer than 20 characters!
--------------------------------
Process exited after 0.01592 seconds with return value 1
Press any key to continue . . .
```

Program hata mesajı verip çalışmayı durdurdu.