

Proiect Limbaje formale si translatoare
- Custom programming language -
Grupa 30232
Nedelcu Ioan-Andrei Zbughin Cezar

Cuprins

1)	Descriere teoretica
2)	Decizii de implementare
3)	Exemple de rulare
4)	Links

Descriere teoretica

Proiectul realizat constă în dezvoltarea unei limbaj de programare, un translator utilizând Lex și Yacc și o mașină virtuală pentru a rula codul generat. Acest are scopul de a explora procesul de dezvoltare a unui limbaj de programare, precum și aspectele legate de analiză lexicală, sintactică și execuție a codului.

Am proiectat un limbaj de programare personalizat. Am creat sintaxa și regulile specifice acestui limbaj, care trebuie respectate

Am dezvoltat un translator folosind Lex și Yacc (sau Bison) pentru a traduce codul sursă scris în limbajul personalizat într-un format pe care mașina mea virtuală îl poate înțelege și executa. Am utilizat Lex pentru a genera analizatoare lexicale, care analizează textul în componente mai mici numite "token-uri". De asemenea, am folosit Yacc pentru a genera analizatoare sintactice, care verifică dacă secvența de token-uri respectă regulile sintactice ale limbajului.

Am creat o mașină virtuală în limbajul C++ care poate rula codul generat de translator. Această mașină virtuală oferă un mediu de execuție eficient și gestionarea memoriei. Am implementat, de asemenea, instrucțiuni și optimizări specifice pentru a asigura o execuție cât mai rapidă și eficientă a codului generat în limbajul personalizat.

1) Masina virtuala

În cadrul acestui proiect, a fost dezvoltată o mașină virtuală pentru a rula codul generat de translatorul limbajului de programare. Mașina virtuală este implementată în limbajul C++ și este responsabilă de interpretarea și execuția codului intermediar produs de translator.

Pe baza codului intermediar, masina virtuala construieste o structura arborescenta echivalenta cu cea a arborelui de executie, urmand apoi sa o ruleze.

2) Translatorul

Am dezvoltat un translator folosind Lex și Yacc pentru a realiza analiza lexicală și sintactică a codului sursă scris în limbajul de programare personalizat. Translatorul preia codul sursă și generează un cod intermediar, care poate fi apoi executat în mașina virtuală.

Translatorul folosește Lex pentru a realiza analiza lexicală a codului sursă. Fișierul Lex conține reguli care descriu cum sunt recunoscute și clasificate diversele tokenuri din codul sursă. Aceste

reguli pot fi definite prin expresii regulate și acțiuni asociate, care sunt executate atunci când un token este identificat.

După ce analiza lexicală este finalizată, translatorul utilizează Yacc pentru a realiza analiza sintactică a codului sursă. Fișierul Yacc conține reguli gramaticale care descriu structura sintactică a limbajului de programare și definește cum sunt combinate tokenurile recunoscute pentru a forma construcții gramaticale valide.

Atunci când codul sursă este analizat sintactic, Yacc poate construi un arbore sintactic, care reprezintă structura ierarhică a construcțiilor gramaticale din cod. Acest arbore sintactic poate fi utilizat ulterior pentru generarea codului intermediar.

Detalii de implementare:

Codul intermediar are urmatorul format:

```
1 PUSH_CONTEXT 10
10 NEW_STR str1 2 "Str1"
2 PUSH_CONTEXT 3
3 NEW_STR str1 4 "Str2"
40 GET_STR str1
4 PRINT 40 5
5 POP_CONTEXT 6
60 GET_STR str1
6 PRINT 60 0
```

1 PUSH_CONTEXT 10: Această instrucțiune indică mașinii virtuale să creeze un nou context de execuție și să îl adauge în stiva de contexte. Acest context poate fi folosit pentru a stoca variabile și alte informații relevante pentru execuție.

10 NEW_STR str1 2 "Str1": Această instrucțiune creează o nouă variabilă de tip șir de caractere (string) cu numele "str1" și o inițializează cu valoarea "Str1". Variabila este creată în cadrul contextului de mai sus.

2 PUSH_CONTEXT 3: Similar cu instrucțiunea anterioară, aceasta creează un nou context de execuție cu și îl adaugă în stiva de contexte.

3 NEW_STR str1 4 "Str2": Această instrucțiune creează o altă variabilă de tip șir de caractere cu numele "str1" și o inițializează cu valoarea "Str2". Variabila este creată în cadrul ultimului context

40 GET_STR str1: Această instrucțiune obține valoarea variabilei "str1".

4 PRINT 40 5: Această instrucțiune afișează valoarea nodului 40, adică a variabilei str1.

5 POP_CONTEXT 6: Această instrucțiune elimină contextul curent din stiva de contexte. Toate variabilele și alte informații asociate acestui context sunt eliminate.

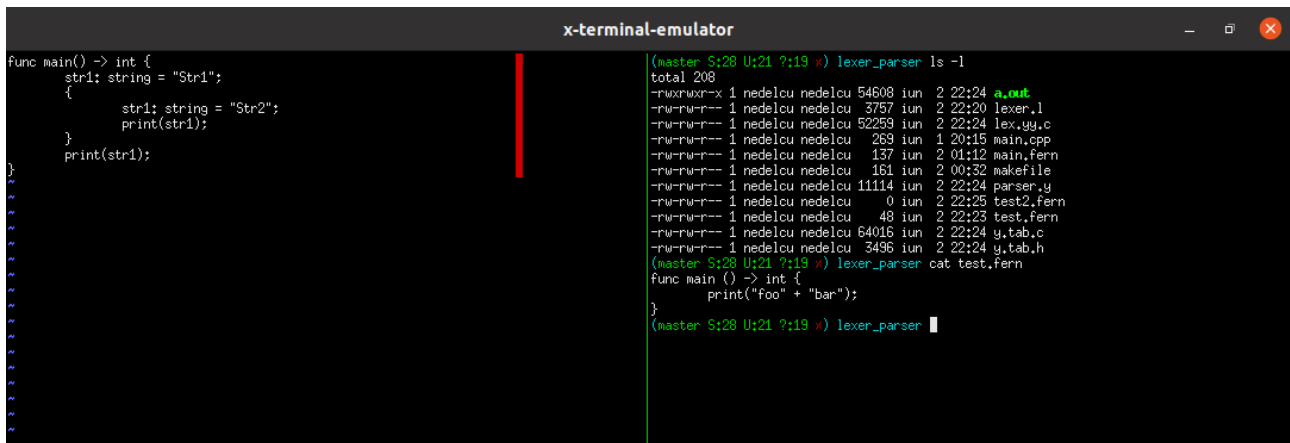
60 GET_STR str1: Această instrucțiune obține valoarea variabilei "str1" din contextul curent și o plasează pe stiva de operanzi.

6 PRINT 60 0: similar cu nodul 4, printează str1, de data asta din primul context creat.

Iar codul sursa care corespunde acestui cod intermediar este:

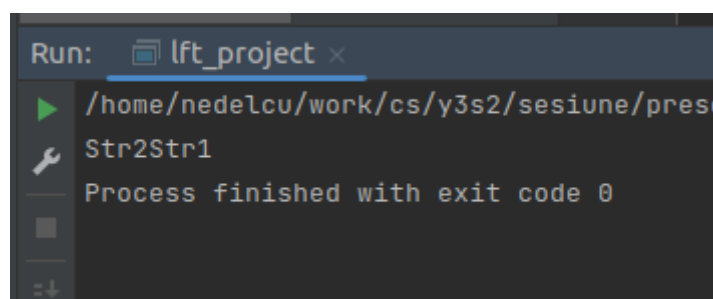
```
func main() -> int {
    str1: string = "Str1";
    {
        str1: string = "Str2";
        print(str1);
    }
    print(str1);
}
```

Iar output-ul asteptat este: Str1Str2



```
func main() -> int {
    str1: string = "Str1";
    {
        str1: string = "Str2";
        print(str1);
    }
    print(str1);
}

(master S:28 U:21 ?;19 x) lexer_parser ls -l
total 208
-rwxrwxr-x 1 nedelcu nedelcu 54608 iun  2 22:24 a.out
-rw-rw-r-- 1 nedelcu nedelcu  3757 iun  2 22:20 lexer.l
-rw-rw-r-- 1 nedelcu nedelcu 52259 iun  2 22:24 lex.yy.c
-rw-rw-r-- 1 nedelcu nedelcu   269 iun  1 20:15 main.cpp
-rw-rw-r-- 1 nedelcu nedelcu   137 iun  2 01:12 main.fern
-rw-rw-r-- 1 nedelcu nedelcu   161 iun  2 00:32 makefile
-rw-rw-r-- 1 nedelcu nedelcu 11114 iun  2 22:24 parser.y
-rw-rw-r-- 1 nedelcu nedelcu    0 iun  2 22:28 test2.fern
-rw-rw-r-- 1 nedelcu nedelcu    48 iun  2 22:23 test.fern
-rw-rw-r-- 1 nedelcu nedelcu 64016 iun  2 22:24 y.tab.c
-rw-rw-r-- 1 nedelcu nedelcu  3496 iun  2 22:24 y.tab.h
(master S:28 U:21 ?;19 x) lexer_parser cat test.fern
func main () -> int {
    print("foo" + "bar");
}
(master S:28 U:21 ?;19 x) lexer_parser
```



```
Run: lft_project x
/home/nedelcu/work/cs/y3s2/sesiune/pres
Str2Str1
Process finished with exit code 0
```



```
(master S:28 U:21 ?;19 x) lexer_parser ./
a.out ./idea/
(master S:28 U:21 ?;19 x) lexer_parser ./a.out < test.fern

2 CONST_STR "foo"
3 CONST_STR "bar"
4 SUM_STR 2 3
1 PRINT 4 0

5 PUSH_CONTEXT 4
(master S:28 U:21 ?;19 x) lexer_parser
```