

Sveučilište u Zagrebu  
Fakultet elektrotehnike i računarstva

Lea Suć

**Samostalni studentski projekt iz predmeta  
“Prevođenje programskih jezika”**

Zadatak broj 3004

Zagreb, siječanj 2011.

## **Samostalni studentski projekt iz predmeta “Prevođenje programskih jezika”**

**Student:** Lea Suć

**Matični broj studenta:** 0036444175

**Zadatak broj 3004:** Izgraditi gramatiku koja generira jezik u kojem su svi pravilno napisani aritmetički izrazi izgrađeni u postfiks notaciji pomoću operatora -, \*, +, i /. Na temelju izgrađene gramatike programski ostvariti jezični procesor koji pretvara pravilno napisane izraze u troadresne naredbe sa simboličkim registrima. Primjer ulaznog aritmetičkog izraza je:

34+-76-+\*

(značenje:  $(-(3+4)) * (7 + (-(6)))$ )

Napomena: operandi mogu biti isključivo jednoznamenaste cjelobrojne pozitivne konstante (1, 2, 3, 4, 5, 6, 7, 8, 9).

# Sadržaj

<b>1. Uvod.....</b>	<b>3</b>
1.1. Jezični procesor.....	3
1.2. Postfiks notacija.....	3
1.3. Troadresne naredbe.....	4
<b>2. Ostvarenje.....</b>	<b>5</b>
2.1. Leksička analiza.....	5
2.2. Sintaksna, semantička analiza i generiranje troadresnih naredbi.....	5
2.2.1. Izgradnja gramatike.....	5
2.2.2. Program Yacc.....	6
2.3. Programsko ostvarenje.....	8
2.4. Korištenje i primjer rada programa.....	9
<b>3. Zaključak.....</b>	<b>12</b>
<b>4. Literatura.....</b>	<b>13</b>

# 1. Uvod

## 1.1. Jezični procesor

Jezični procesor prevodi program višeg jezika opće namjene izravno u izvodivi strojni program, ili u jedan od oblika koje je potrebno dodatno doraditi prije samog izvođenja na računalu: program virtualnog stroja ili premjestivi strojni program, pri čemu je program virtualnog stroja strojno nezavisni program, a kod premjestivog strojnog programa naredbe nemaju do kraja izračunate numeričke vrijednosti memorijskih adresa.

Prevođenje izvornog programa u ciljni ostvaruje se u dvije osnovne faze rada jezičnog procesora: analiza izvornog i sinteza ciljnog programa. Faza analize prethodi fazi sinteze kako bi se ustanovilo ima li u izvornom kodu pogrešaka. Samo ispravni izvorni program moguće je prevesti u ciljni. Nema li pogrešaka u izvornom, izvodi se sinteza ciljnog programa. Faza analize rastavlja izvorni program u sastavne dijelove, provjerava pravila jezika, prijavljuje pogreške i zapisuje izvorni program primjenom različitih struktura podataka u memoriju računala. U fazi sinteze zapis izvornog programa, koji je generiran u fazi analize postupno se prevodi u ciljni program. Sinteza ciljnog programa izvodi se u sljedećim koracima: generiranje međukoda, strojno nezavisno optimiranje, generiranje strojnog programa, strojno zavisno optimiranje i priprema strojnog programa za izvođenje.

## 1.2. Postfiks notacija

Postfiks notacija je matematička notacija kod koje svaki operator slijedi nakon svojih operandada. Postfiksni sustav oznaka (obrnuta poljska notacija) definirao je krajem 20-ih godina 20. stoljeća J. Lukasiewicz. Kako se operatori u ovoj notaciji zapisuju iza operandada, zbroj dva broja  $x$  i  $y$  bit će zapisan u obliku " $xy+$ ", što je ekvivalentno zapisu " $x+y$ " u infiksnom sustavu. Kod postfiksne notacije nema potrebe za zagradama, jer su svi izrazi jednoznačni, a računanje se izvodi upravo onim redom kojim su napisane operacije. Aritmetički izrazi opisani postfiksним sustavom oznaka mogu se lako računati primjenom samo jednog potisnog stoga, zbog čega je, uz

činjenicu da ne koristi zagrade i da omogućuje definiranje operatora s više od dva operanda, postfiksni sustav važan za razvoj jezičnih procesora.

### **1.3. Troadresne naredbe**

Jedna od posljednjih faza rada jezičnog procesora, a prije generiranja ciljnog programa, generiranje je međukôda. Međukôd je prijelazni oblik izvornog programa koji se koristi tijekom sinteze ciljnog programa, a jednostavni jezični procesori ne generiraju međukôd, nego izravno ciljni program. Troadresni međukôd spada u linearne oblike međukôda. Naziv "troadresni" posljedica je rada s tri adrese, i to adresom lijevog, adresom desnog operanda i adresom rezultata. Pritom se koriste simbolički registri. Izvršava se operacija nad prvim i drugim operandom, a rezultat izvođenja se sprema u treći registar. Svaki se međurezultat sprema u novi simbolički registar. Moguće je zadati različite operacije nad operandima, kao što su aritmetičke ili logičke operacije. Operacije koje se koriste u ovom zadatku su zbrajanje (ADD), oduzimanje (SUB), množenje (MUL) i dijeljenje (DIV).

## 2. Ostvarenje

### 2.1. Leksička analiza

Leksička analiza je vrlo jednostavna, jer je dovoljno provjeriti pripadaju li svi znakovi ulaznog niza skupu {1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \*, /}. U slučaju da pripadaju, niz je leksički ispravan, a ako to nije slučaj, signalizira se leksička pogreška.

### 2.2. Sintaksna, semantička analiza i generiranje troadresnih naredbi

#### 2.2.1. Izgradnja gramatike

Izgradnja gramatike je osnovni dio konstrukcije sintaksnog analizatora. U zadanoj gramatici osnovni su tipovi podataka konstante, i to isključivo cijeli jednoznačenasti brojevi od jedan do devet. Koriste se aritmetički operatori +, -, \*, /, pri čemu se minus ponaša kao unarni operator, dok su svi ostali binarni. Radi se o postfiksnoj notaciji, pa operatori dolaze iza operandada.

Produkcije gramatike su prema tome:

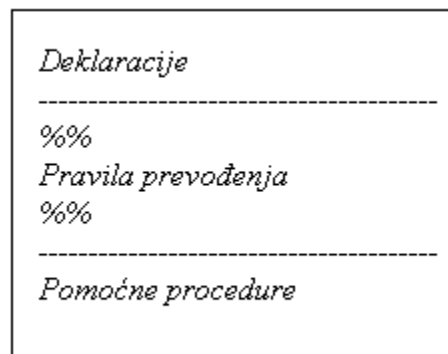
$$S \rightarrow S \ S \ +$$
$$S \rightarrow S \ -$$
$$S \rightarrow S \ S \ *$$
$$S \rightarrow S \ S \ /$$
$$S \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

### 2.2.2. Program Yacc

U ovom zadatku se kao glavni alat za automatizaciju sintaksne analize koristi *Yacc* (Yet another compiler-compiler), koji je primjer generatora parsera. Generator parsera je program koji na temelju produkcija gramatike gradi sintaksni analizator za zadani jezik. Program *Yacc* se, osim za izgradnju sintaksnog analizatora, upotrebljava i za rješavanje problema koji se mogu svesti na problem prihvatanja kontekсно neovisnih jezika.

*Yacc* koristi *LALR* (LookAhead-*LR*) postupak izgradnje tablice *LR* parsera. *LALR* postupak gradi puno manju tablicu *LR* parsera od kanonskog *LR* postupka, a temelji se na ideji grupiranja stanja koja su označena istim *LR*(0) stavkama u jedinstveno stanje. Nedostatak tog postupka je mogućnost nastajanja proturječja.

Ulazna datoteka programa ima nastavak *.y*, i u njoj se definira sintaksni analizator koji se potom predaje programu *Yacc*. Podijeljena je u tri cjeline, a to su deklaracije, pravila prevođenja i pomoćne C procedure, koje su u samoj datoteci obilježene u sljedećem obliku:



Deklaracije su dio definicije sintaksnog analizatora. Dio ograden oznakama *%{ i %}* predviđen je za deklaracije jezičnog procesora C, dok je drugi dio predviđen za deklaraciju završnih znakova gramatike.

```
%{  
  Deklaracije koje koristi C jezični procesor  
%}  
-----  
% Deklaracija završnih znakova gramatike
```

Leksičke se jedinice u dijelu deklaracija završnih znakova definiraju u sljedećem obliku: %token DIGIT, pri čemu je riječ "token" rezervirana riječ u programu *Yacc*. Ukoliko samo jedan znak čini završni znak, kao primjerice u slučaju operatora i specijalnih znakova, takve znakove ne treba posebno definirati. Osim toga, u dijelu deklaracija završnih znakova gramatike mogu se zadati posebne naredbe koje omogućuju razrješavanje *Pomakni/Reduciraj* proturječja na temelju prednosti i asocijativnosti završnih znakova gramatike. Redoslijed pisanja naredbi određuje prioritet završnih znakova – znak ima manju prednost ako je zadan prije nekog drugog, a ako su znakovi zadani u istoj deklaraciji, jednake su prednosti. Naredba %left '+' '-' definira da su operatori + i – lijevo asocijativni i da su jednake prednosti. Primjer deklaracija korištenih u zadatku :

```
%{  
    int _startRegNum=0;  
%}  
  
%start rules_list  
  
%token DIGIT  
  
%left '+' '-'  
%left '*' '/'
```



Središnji dio čine pravila prevođenja, a to su zapravo produkcije gramatike i akcije semantičkog analizatora koje su im pridružene, odnosno koje se izvršavaju nakon primjene produkcije. U programu *Yacc* se produkcija

$$\langle S \rangle \rightarrow \langle A \rangle \mid \langle B \rangle$$

zapisuje u obliku:

```
S : A {SemantičkaAkcija1}
    | B {SemantičkaAkcija2};
```

Semantičke akcije generiraju troadresne naredbe.

U trećem dijelu su zadane pomoćne procedure, koje čine razni pomoćni potprogrami sintaksnog analizatora, poput postupka oporavka od pogreške i potprograma leksičkog analizatora `yylex()`.

## 2.3. Programsko ostvarenje

U zadatku je korišten generator parsera *GPPG* (*Gardens Point Parser Generator*) za *LALR(1)* parsere, koji očekuje ulaznu datoteku u *Yacc* formatu, te generira izlaznu datoteku u programskom jeziku C#. Dizajniran je za korištenje s *GPLEX*-om (generatorom leksičkog analizatora), iako se mogu koristiti i odvojeno.

Generiranje parsera poziva se iz komandne linije (potrebno je pozicioniranje u direktoriju u kojem se nalazi `gppg.exe`) naredbom

```
gppg [opcije] ulazna_datoteka.y > izlazna_datoteka.cs
```

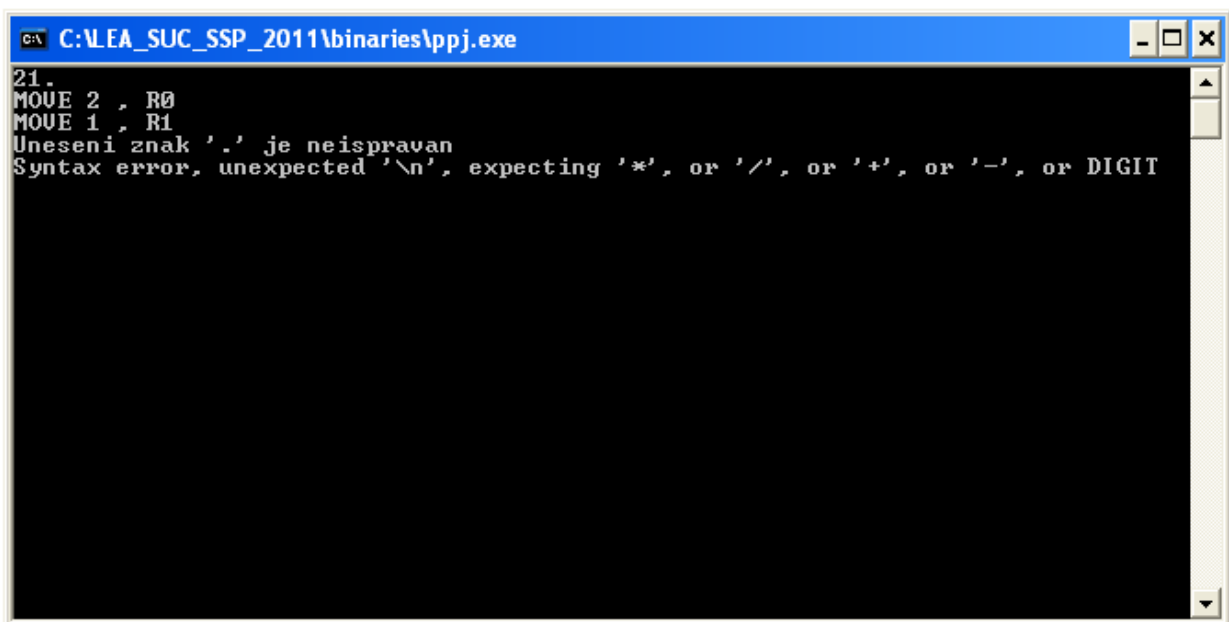
pri čemu se stvara C# datoteka (`izlazna_datoteka.cs`), koja predstavlja parser. Opcije su opisane u *GPPG* dokumentaciji. Nastala datoteka ubaci se u Visual Studio projekt, pri čemu je

potrebno referencirati `'ShiftReduceParser.dll'` biblioteku, čija se glavna klasa instancira s dva parametra koji određuju tip i definirani su u specifikaciji gramatike.

## 2.4. Korištenje i primjer rada programa

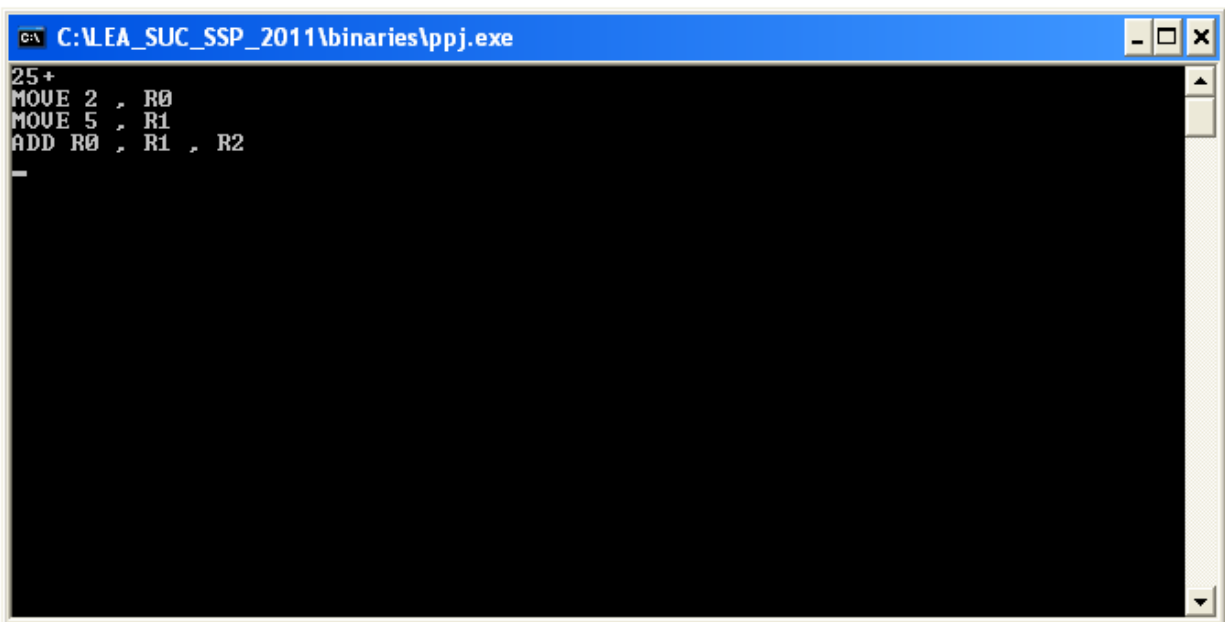
Program se može direktno pokrenuti dvostrukim klikom na `ppj.exe`, koji je priložen u ZIP datoteci. Nakon toga je potrebno upisati željeni izraz u postfiksnoj notaciji. U nastavku je navedeno nekoliko primjera ulaznog niza i izvođenja programa za taj unos.

Primjer izvođenja programa u slučaju leksički neispravnog niza:



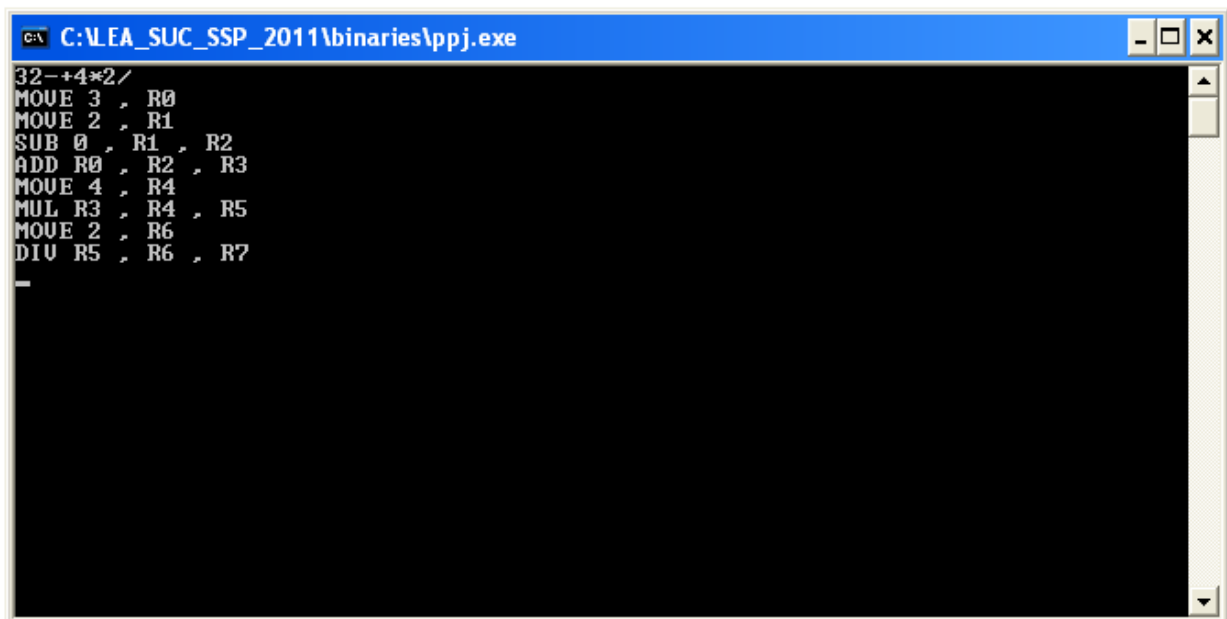
```
C:\LEA_SUC_SSP_2011\binaries\ppj.exe
21.
MOVE 2 , R0
MOVE 1 , R1
Uneseni znak '.' je neispravan
Syntax error, unexpected '\\n', expecting '*', or '/', or '+', or '-', or DIGIT
```

Primjer jednostavnog zbrajanja:



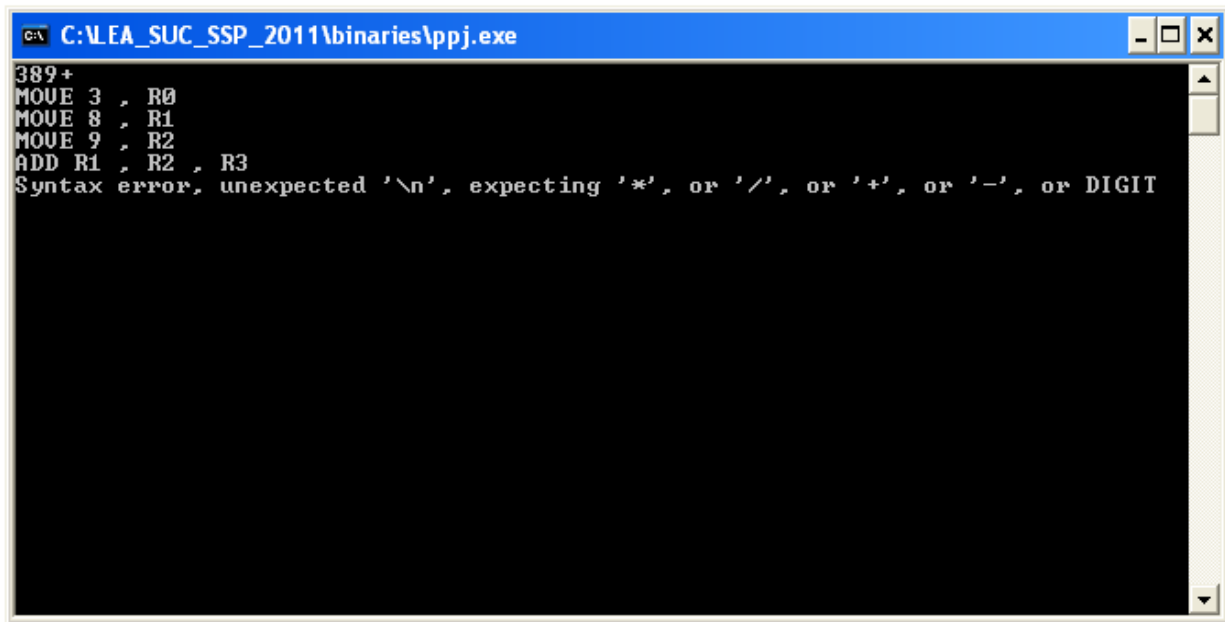
```
C:\V\EA_SUC_SSP_2011\binaries\ppj.exe
25+
MOVE 2 , R0
MOVE 5 , R1
ADD R0 , R1 , R2
-
```

Primjer izvođenja programa za ispravno zadan složeniji izraz "32-+4\*2/", koji ima značenje  $((3+(-2)) * 4) / 2$ :



```
C:\V\EA_SUC_SSP_2011\binaries\ppj.exe
32-+4*2/
MOVE 3 , R0
MOVE 2 , R1
SUB 0 , R1 , R2
ADD R0 , R2 , R3
MOVE 4 , R4
MUL R3 , R4 , R5
MOVE 2 , R6
DIV R5 , R6 , R7
-
```

U slučaju ulaznog niza u kojem nedostaje samo zadnji operator, program se ispravno izvodi sve do posljednjeg koraka, kad javlja grešku zbog očekivanih operandi ili operatora:



```
C:\MEA_SUC_SSP_2011\binaries\ppj.exe
389+
MOVE 3 , R0
MOVE 8 , R1
MOVE 9 , R2
ADD R1 , R2 , R3
Syntax error, unexpected '\n', expecting '*', or '/', or '+', or '-', or DIGIT
```

### 3. Zaključak

Iz gramatike izgrađene u postfiksnoj notaciji moguće je uz pomoć alata koji automatiziraju sintaksnu analizu jednostavno ostvariti jezični procesor koji pretvara pravilno napisane izraze u troadresne naredbe sa simboličkim registrima. *Yacc* alati su vrlo korisni i uvelike olakšavaju proces izgradnje jezičnog procesora, iako je za kvalitetno korištenje potrebno poznavanje prilično detaljno poznavanje njegove dokumentacije i bar osnovno znanje programiranja.

Zadatak se, osim uz pomoć *Yacc*-a, u gotovo svakom programskom jeziku može riješiti uz pomoć stoga, iako je takvo rješenje nešto zahtjevnije. Osim toga, postoji mnoštvo dostupnih algoritama koji opisuju kako se u *Yacc*-u programiraju kalkulatori, koji su također postfiksni, što je također olakšalo rješenje.

## **4. Literatura**

[1] Siniša Srblić, Prevođenje programskih jezika, Zagreb, 2007.