

Uvod

Aritmetički izraz pripada elementima programskih jezika. Aritmetički izrazi pojavljuju se u gotovo svakom programskom jeziku. Aritmetika je grana matematike koja proučava računske operacije s brojevima. Osnovne aritmetičke operacije su zbrajanje, oduzimanje, množenje i dijeljenje.

Postfiksna notacija (postfix notation) je matematička notacija u kojoj svaki operator slijedi nakon svojih operandi. Poznata je pod nazivom obrnuta poljska notacija (reverse polish notation, RPN), po analogiji sa srodnom poljskom notacijom (prefiksna notacija) koju je uveo poljski matematičar Jan Lukasiewicz 1920. godine. Postfiksnu notaciju je izmislio australski računalni znanstvenik i filozof Charles Hamblin, početkom pedesetih godina dvadesetog stoljeća. Kao što sam ranije navela, u postfiksnoj notaciji svaki operator slijedi nakon svojih operandi. To znači da ukoliko recimo želimo zbrojiti dva broja a i b , operacija se zapisuje " $ab+$ " a ne " $a+b$ ". Ukoliko postoje višestruke operacije, operator je dan neposredno nakon drugog operandi, odnosno izraz " $a+b-c$ " u postfiksnoj se notaciji zapisuje kao " $ab+c-$ ". Prednost postfiksne notacije u odnosu na infiksnu je ta što nema potrebe za zagradama. Da bi se to pojasnilo, pogledajmo primjer izraza " $3+4*5$ ". Nedostatkom zagrada ne možemo biti posve sigurni odnosi li se taj izraz na " $3+(4*5)=23$ " ili na izraz " $(3+4)*5=35$ ", naravno ukoliko nema zagrada, pretpostavljamo da se mislilo na prvi izraz radi prednosti operatora. U postfiks notaciji takvih nedoumica nema. Pisanjem izraza " $34+5*$ " jednoznačno dobivamo " $(3+4)*5=35$ ", a pisanjem izraza " $345*+$ " jednoznačno dobivamo " $3+(4*5)=23$ ".

Postfiksna i prefiksna notacija su od velikog značaja za razvoj jezičnih procesora, iz razloga što ne koriste zagrade, omogućavaju definiranje operatora s više od dva operandi i posebno jer je aritmetičke izraze opisane tim sustavom oznaka moguće lako izračunati primjenom samo jednog potisnog stoga. Također, aritmetički izrazi opisani tim sustavom oznaka predstavljaju lakši način računanja aritmetičkih izraza zadanih infiks sustavom oznaka: prvo se infiksni sustav prevede u postfiksni ili prefiksni, a nakon toga se vrijednost računa primjenom potisnog automata. Postfiksni i prefiksni sustav oznaka se često koristi kao osnova za izgradnju međukoda, jer omogućava učinkovitu pretvorbu stablaste hijerarhije strukture infiksnog sustava oznaka u linearan strojni program primjenom samo jednog potisnog automata.

Generiranje troadresnog međukoda je jedna od zadnjih faza rada jezičnog procesora prije one konačne, generiranja ciljnog programa. Ime dolazi od činjenice da se radi sa 3 adrese: adresom lijevog operanda, adresom desnog operanda i adresom rezultata. Registri su simbolički i prilikom generiranja koda koristi ih se proizvoljno mnogo. Svaki međurezultat sprema se u novi simbolički registar.

Troadresne naredbe imaju oblik :

operacija	adr_operanda1	adr_operanda2	adr_rezultat
-----------	---------------	---------------	--------------

To znači da izvršava se operacija nad prvim (registar `adr_operanda1`) i drugim operandom (registar `adr_operanda2`), a rezultat izvođenja operacije se sprema u treći registar (`adr_rezultat`). Zadaju se različite operacije nad operandima, na primjer aritmetičke operacije, logičke operacije, itd. U ovom zadatku, operacije koje će se koristiti su : ADD (zbrajanje), SUB (oduzimanje), MUL (množenje) i DIV (dijeljenje).

Opis

Kao i kod svakog jezičnog procesora, i u ovom primjeru pojavljuju se tri faze analize izvornog programa: leksička, sintaksna i semantička analiza. Na temelju semantičke analize radi se prva faza sinteze - generiranje troadresnog međukoda.

Leksička analiza je poprilično trivijalna. Provjerava jesu li svi znakovi ulaznog niza iz skupa {1,2,3,4,5,6,7,8,9,+,-,*,/}. Ukoliko jesu, niz je leksički ispravan, u suprotnom nije.

Sintaksnu analizu moguće je dijelom automatizirati pomoću generatora parsera. Generator parsera je program koji iz zadanih produkcija gramatike gradi sintaksni analizator za zadani jezik. YACC (Yet another compiler-compiler) je primjer generatora parsera. Osim za izgradnju sintaksnog analizatora, program YACC se koristi za rješavanje problema koje je moguće svesti na problem prihvaćanja kontekstno neovisnih jezika. Program YACC koristi LALR (LookAhead-LR) postupak gradnje tablice LR parsera. LALR postupak gradi znatno manju tablicu LR parsera od kanonskog LR postupka. Osnovna ideja LALR postupka je grupiranje stanja označenih istim LR(0) stavkama u jedinstveno stanje. Nedostatak ovog postupka je mogućnost nastanka proturječja. Sintaksni analizator definira se u datoteci sa nastavkom `.y` koja se predaje

programu YACC.

Definicija sintaksnog analizatora sastoji se od tri osnovne cjeline:

Deklaracije

%%

Pravila Prevođenja

%%

Pomoćni potprogrami

Prva cjelina su deklaracije koje se sastoje od dva dijela. Dio ograđen oznakama "%{" i "%}" predviđen je za deklaracije jezičnog procesora C. Drugi dio deklaracija predviđen je za definiranje završnih znakova gramatike. Središnji dio definicije sintaksnog analizatora su pravila prevođenja. Pravila prevođenja čine produkcije gramatike i njima pridružene akcije semantičkog analizatora. U trećem dijelu zadaju se razni pomoćni potprogrami sintaksnog analizatora kao što su postupak oporavka od pogreške i potprogram leksičkog analizatora yylex().

U dijelu deklaracija definiraju se leksičke jedinice u obliku tokena na sljedeći način: %token DIGIT. Ako se završni znak sastoji od samo jednog znaka, kao što su na primjer operatori, tada te znakove nije potrebno posebno definirati. Pod deklaracije se mogu zadati naredbe koje služe za razrješavanje proturječja. Na primjer, naredba %left '+' definira da je operator + lijevo asocijativan. Redoslijed pisanja naredbi određuje prioritet završnih znakova. Ako je završni znak zadan prije nekog drugog, tada je njegova prednost manja. Ako su zadani u istoj deklaraciji to znači da su znakovi jednake prednosti. Ako prednost produkcije nije moguće pravilno odrediti na temelju krajnjeg desnog završnog znaka, onda YACC dozvoljava izravno definiranje prednosti sa naredbom %prec <ZavršniZnak>. Prednost produkcije tada je jednaka prednosti završnog znaka koje je definiran u dijelu deklaracija. Pod deklaracije spada i naredba %start <PočetniZnak> kojom se definira početni nezavršni znak gramatike, što za ovaj zadatak izgleda:

```

%start list

%token DIGIT

%left '+' '-'
%left '*' '/' '%'
%left UMINUS

%%

list : /*empty */
    | list stat '\n'
    | list error '\n'
    {
        yyerrok();
    }
    ;

stat : expr
    {

```

Pravilo prevođenja sadrži produkciju gramatike i semantičku akciju koja se izvršava nakon primjene produkcije. Završni znakovi koji se sastoje od samo jednog simbola pišu se u jednostrukim navodnicima, na primjer '+'. Produkcija gramatike:

$$\langle \text{izraz} \rangle \rightarrow \langle \text{produkcija1} \rangle \mid \langle \text{produkcija2} \rangle$$

u programu YACC zapisuje se u sljedećem obliku:

```

izraz : produkcija1 {SemantičkaAkcija1}
      | produkcija2 {SemantičkaAkcija2} ;

```

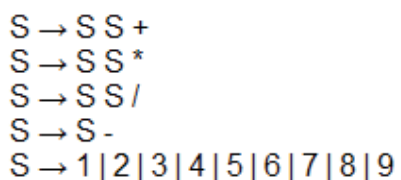
Ostvarenje

Sintaksna analiza pomoću generatora parsera YACC izvedena je korištenjem alata GPPG (Gardens Point Parser Generator). GPPG je generator za LALR (1) parsere koji

prihvaća ulaz u YACC formatu i generira izlaznu datoteku u programskom jeziku C#. GPPG parseri su dizajnirani za korištenje sa leksičkim analizatorom GPLEX (Gardens Point Scanner Generator), ali se također mogu koristiti uz ručno pisane analizatore ili analizatore napravljene sa drugim alatima. GPPG i parseri generirani njime implementirani su isključivo u C# te koriste generičke tipove podataka definirane u C# 2.0. Parser implementira parsiranje od dna prema vrhu te koristi tehniku parsiranja Pomakni-Reduciraj. LR parser gradi se LALR postupkom (Look Ahead LR) koji je programski manje zahtjevan od kanonskog LR postupka, obuhvaća manji skup jezika od kanonskog LR postupka, ali ga je moguće primijeniti na veći skup jezika od SLR postupka. Rad parsera oslanja na komponentu "ShiftReduceParser.dll" čija glavna klasa se instancira sa dva parametra koji odrađuju tip i koji su definirani u specifikaciji gramatike. Pozivom naredbe "gppg.exe [opcije] ulazna_datoteka.y > izlazna_datoteka.cs" kreira se C# datoteka (.cs) koja predstavlja generirani parser. Generirana datoteka ubačena je u Visual Studio projekt.

Gramatika

Glavni dio konstrukcije sintaksnog analizatora bio je izgradnja gramatike. U nastavku su opisane osnovna obilježja jezika implementirana u ovoj gramatici. Osnovni tipovi podataka su konstante koje mogu biti cijeli brojevi. Omogućeni su aritmetički operatori "+", "-", "*", i "/". Jedina razlika porodukcija gramatike u odnosu na zaključke koji se mogu donjeti na temelju gore opisane postfiksne notacije je ta da se minus (-) ponaša kao unarni operator. To sam zaključila na temelju primjera zadanog zadatkom. Dakle, ulazni niz je "34+-76-+*" a značenje niza je: $(-(3+4)) * (7 + (-(6)))$. S obzirom na gramatiku zadanu zadatkom koja ima produkcije :



```
S → S S +  
S → S S *  
S → S S /  
S → S -  
S → 1|2|3|4|5|6|7|8|9
```

Slijedi popis svih produkcija konstruirane gramatike koja se nalazi u datoteci parser.y:

```

expr :
    expr expr '*'
    {
        System.Console.WriteLine("MUL R"+($1-10)+" , R"+($2-10)+" , R"+(_base-10));

        $$ = _base++;
    }
    | expr expr '/'
    {
        System.Console.WriteLine("DIV R"+($1-10)+" , R"+($2-10)+" , R"+(_base-10));

        $$ = _base++;
    }
    | expr expr '+'
    {
        System.Console.WriteLine("ADD R"+($1-10)+" , R"+($2-10)+" , R"+(_base-10));

        $$ = _base++;
    }
    | expr '-'
    {
        System.Console.WriteLine("SUB "+0+" , R"+($1-10)+" , R"+(_base-10));

        $$ = _base++;
    }
    | number
    ;

number : DIGIT

    {
        System.Console.WriteLine("MOVE "+$1+" R"+(_base-10));

        $$ = _base++;
    }

```

```
}  
  
;  
  
%%
```

Način korištenja programa i primjer izvođenja

Program se pokreće iz konzole *Visual Studio 2008 Command Prompt*. Sa naredbom *cd* potrebno se je pozicionirati u datoteku *binaries* koja se nalazi u datoteci *gppg*. Za pokretanje jezičnog procesora potrebno je upisati naredbu "ivana". Nakon toga se može upisati ulazni niz napisan u postfiks notaciji.

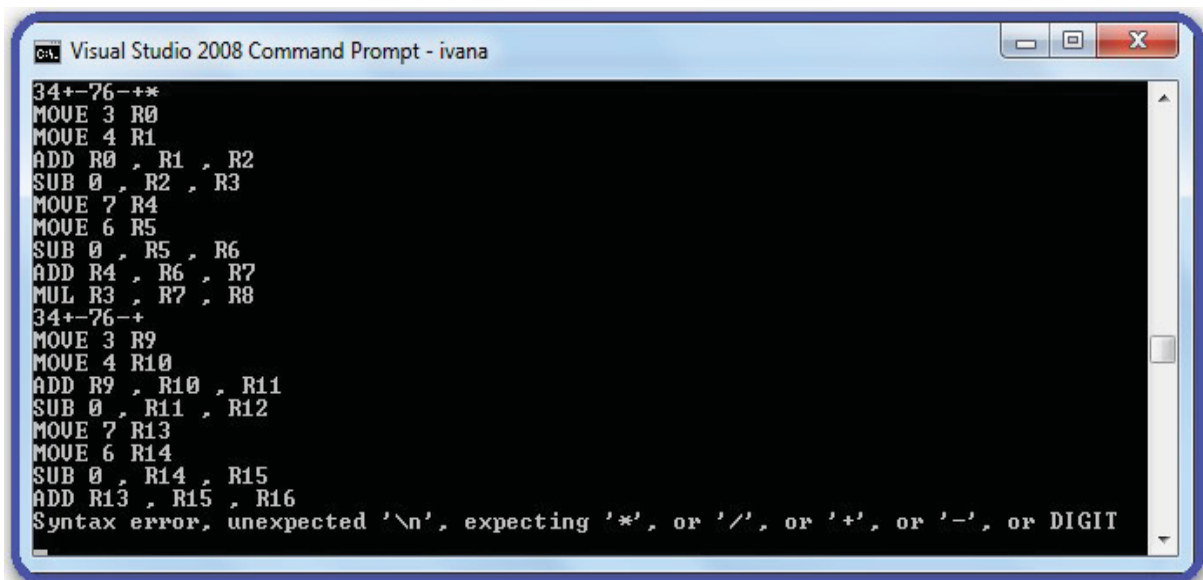
Za primjer izvođenja programa, navedeni su rezultati za jedan ispravan i jedan neispravan unos ulaznog niza.

Primjer ispravnog ulaznog niza može biti primjer zadan u zadatku, odnosno :

```
"34+-76-+*"
```

a primjer neispravnog ulaznog niza neka bude isti kao i ispravan, ali bez zadnjeg operatora, odnosno:

```
"34+-76-+"
```



```
Visual Studio 2008 Command Prompt - ivana
34+-76-+*
MOVE 3 R0
MOVE 4 R1
ADD R0 , R1 , R2
SUB 0 , R2 , R3
MOVE 7 R4
MOVE 6 R5
SUB 0 , R5 , R6
ADD R4 , R6 , R7
MUL R3 , R7 , R8
34+-76-+
MOVE 3 R9
MOVE 4 R10
ADD R9 , R10 , R11
SUB 0 , R11 , R12
MOVE 7 R13
MOVE 6 R14
SUB 0 , R14 , R15
ADD R13 , R15 , R16
Syntax error, unexpected '\n', expecting '*', or '/', or '+', or '-', or DIGIT
```

Ispis za ispravan ulazni niz je i sam ispravan. Ispis za neispravan ulazni niz do zadnjeg operatora je također ispravan, no nakon toga izbacuje grešku. Problem je u tome što nije trebao sljediti kraj niza, odnosno, po pravilima gramatike u postfiksnoj notaciji, trebao je sljediti ili još jedan operator, ili još operandi i operatora.

Zaključak

Na temelju gramatike izgrađene u postfiks notaciji je moguće ostvariti jezični procesor koji pretvara pravilno napisane izraze u troadresne naredbe sa simboličkim registrima. Na početku se je činilo nemoguće za izvesti. Većinu problema je riješio GPPG. Inače su postfiksni (reverse polish notation) kalkulatori, koji dakle računaju izraze napisane u postfiksnoj notaciji programiraju pomoću YACC-a, pa ima puno dostupnih algoritama na internetu koji su bili od velike pomoći za rješavanje problema. Postoji i knjiga naziva *Compiler design using FLEX and YACC*, koja većinu primjera objašnjava preko infiksne, postfiksne i prefiksne notacije, te troadresnih naredbi, i premda nema konkretno riješen ovaj zadatak, lijepo se može naučiti, uz pomoć drugih primjera, kako rješavati ovakve tipove problema.

Mogao se zadatak riješiti i bez pomoći YACC-a u gotovo bilo kojem programskom jeziku. Najlakši (i najpravičniji) način bi bilo rješenje pomoću stoga. Takvo rješenje zahtjevalo je nešto više posla, a s obzirom na to da je većina primjera postfiksne, infiksne i prefiksne notacije riješeno pomoću YACC-a, sam odlučila da bi možda bilo najbolje da i moje rješenje koristi takav alat.