Sveučilište u Zagrebu
Fakultet Elektrotehnike i Računarstva

Telefunken

Samostalni studentski projekt

''Uvod u teoriju računarstva''

Zadatak broj 2020

Zagreb, lipanj 2011.

Samostalni studentski projekt

2020. zadatak, UTR SSP, Srednja inačica	Napisati gramatiku i parser tehnikom rekurzivnog spusta za aritmetičke izraze. Izrazi uključuju operatore zbrajanja, oduzimanja, množenja i dijeljenja, znakove jednakosti, nejednakosti, i usporedbe (manje, veće, manje ili jednako i veće ili jednako) te zagrade. Primjer ispravno napisanih izraza: (a < b) ((a-b+f) != c*e) NAPOMENA: U izrazima se mogu pojavljivati samo varijable. Varijable moraju započinjati slovom.

Student: Telefunken

JMBAG:

Zadatak broj 2020: Parser tehnikom rekurzivnog spusta za aritmetičke izraze

Sadržaj:

1.Uvod	**3. Stranica**
2. Priprema	**4. Stranica**
2.1. Priprema izraza	**4. Stranica**
2.2. Priprema gramatike	**4. stranica**
3. Ostvarenje programa	**6. Stranica**
4. Vizualizacija programa	**7. Stranica**
5. Zaključak	**9. Stranica**

1. Uvod

Tehnika rekurzivnog spusta dobila je ime zbog rekurzivnog poziva potprograma i parsiranja od vrha prema dnu. Tehnika rekurzivnog spusta ostvaruje se programskim jezikom (u ovom slučaju Pythonom) koji ima svojstvo rekurzivnog poziva potprograma. Gramatika se prilagodi tehnici rekurzivnog spusta. Nakon toga se svakom nezavršnom znaku gramatike pridružuje potprogram. Potprogram ispituje da li podniz zadanog niza zadovoljava strukturu zadanu desnom stranom one produkcije u kojoj je nezavršni znak pridružen tom potprogramu na lijevoj strani. Završni znakovi na desnoj strani produkcije uspoređuju se sa znakovima u nizu. Za nezavršne znakove na desnoj strani produkcije pozivaju se potprogrami koji provjeravaju strukturu za te nezavršne znakove. Nađe li se jedan te isti nezavršni znak na lijevoj i desnoj strani produkcije, potprogram pridružen tom nezavršnom znaku poziva se rekurzivno. Potprograme je moguće (i nužno u ovom studentskom projektu) rekurzivno pozivati i pomoću drugih potprograma. U ovom studentskom projektu, funkcije „glume“ potprograme.

Cilj ovog projekta je napisati gramatiku, a potom program koji će tehnikom rekurzivnog spusta ispisati je li neki izraz aritmetički. Aritmetički izraz je široki pojam, ali zadatak ograničava mogućnost pisanja na „(“ i „)“ zagrade te operatore „+ - * / < > = <= >= !=“. U zadatku su navedeni primjeri koje bi program trebao prihvatiti. Primjer „((a-b+f)!=c*e)“ je složeniji i bolji za demonstraciju te će se koristiti u programu i za vizualizaciju tehnike rekurzivnog spusta.

2. Priprema

2.1. Priprema izraza

Prije bilo kakvog pisanja programa izraz se mora prilagoditi za program tehnikom rekurzivnog spusta. Autor ovog projekta se odlučio za polje kao „spremnika“ izraza. Ubrzo se došlo do problema kako da program zna što su varijable, a što su zagrade i operatori. Problem je riješen tako da se koristi ASCII tablica za sve znakove osim operatora jednakosti koji se skupljaju u jednu varijablu. Kad znak bude jednak jednom od operatora ili zagradama to je znak programu da prestane skupljati varijablu i varijabla se provjerava da li varijabla započinje slovom. Ako je varijabla ispravna, varijabla se zamjenjuje znakom „v“[1] zbog unificiranosti svih varijabli (u aritmetičkom izrazu nije bitno kako se zove varijabla i koliko se puta pojavljuje u izrazu). Nakon toga se ta varijabla dodaje spremniku. Nakon što se spremila varijabla, gleda se koja zagrada ili operator je učitan i navedenim postupkom se pohranjuje u spremnik. Na kraju se dodaje znak „$“ u spremnik koji označava kraj i koji će biti ključan za prihvatljivost izraza.
Time se ostvaruje izraz prilagođen za tehniku rekurzivnog spusta.[2]

2.2. Priprema gramatike

Za aritmetički izraz najjednostavnije bi bilo napisati gramatiku:

G = ({A},{+,-,*,/,>,<,>=,<=,=,!=,(,),varijabla},P,A)

Produkcije:

A→A+A	A→A-A	A→A*A	A→A/A
A→A>A	A→A<A	A→A>=A	A→A<=A
A→A=A	A→A!=A	A→(A)	A→varijabla

Definicija tehnike rekurzivnog spusta ostvarene programskim jezikom nam nalaže da se svakom nezavršnom znaku gramatike pridruži potprogram koji, kao što je već navedeno, ispituje da li podniz zadanog niza zadovoljava strukturu zadanu desnom stranom one produkcije u kojoj je nezavršni znak pridružen tom potprogramu na lijevoj strani. Potprogram bi trebao provjeravati na način da ako je s desne strane završni znak, onda treba usporediti dobiveni znak i ako su jednaki pročitati sljedeći znak u nizu i poslati

[1] Razlog pretvorbe varijable i slučaj parsiranja same varijable objašnjen u dokumentu Priprema_izraza.pdf
[2] Detaljnije objašnjenje (uključujući i programsko rješenje) pripreme objašnjeno u dokumentu Priprema_izraza.pdf

ga sljedećem potprogramu ili se vratiti. Ako je s desne strane nezavršni znak, potprogram treba otići u pripadni potprogram i nastaviti navedenim tokom.

No pojavljuje se problem lijeve rekurzije. Već zbog prve produkcije gramatike potprogram **A** (jedini u ovoj gramatici jer postoji samo jedan nezavršni znak **A**) poziva samog sebe i ulazi u beskonačnu petlju. Potrebno je uvesti nove nezavršne znakove kako bi se gramatika prilagodila programu i izbjegla lijeva rekurzija.

Nezavršni znak **A** ostaje početni nezavršni znak i služi kao provjera prihvaća li se izraz. Potprogram **A** (dalje u tekstu pod potprogram se misli na potprogram **X** pridružen nezavršnom znaku **X**) poziva potprogram **B** i provjera se vrši na način da potprogram **B** mora vratiti završni znak **$**.

A→B$

Potprogram **B** poziva potprogram **C** i provjerava je li povratni znak iz potprograma **C** jednak jednom od završnih znakova **+ - * /** . Ako je, potprogram **B** poziva samog sebe (time se omogućava beskonačan broj varijabli i operatora).

B→C+B **B→C-B** **B→C*B** **B→C/B**

Potprogram **C** poziva potprogram **D** i provjerava je li povratni znak jednak jednom od završnih znakova **< > <= >= = !=.** Ako je, poziva potprogram **B**

C→D<B **C→D>B** **C→D<=B** **C→D>=B**

C→D=B **C→D!=B**

Kako bi se osigurala mogućnost da potprogram ne treba vratiti završni znak (primjer je kraj izraza poput …+varijabla ili …+(…)) dodaje se produkcija

C→D

Koja se programski rješava na način da se u slučaju kad ne pročita završni znak jednostavno vrati znak (bez čitanja novog znaka) prethodnom potprogramu kao kod završetka svih potprograma.

Potprogram **D** prihvaća znak i provjerava je li on jednak završnom znaku **(** ili **varijabla**. Ako je **(** to znači da započinje aritmetički izraz unutar zagrade (potprogram čita sljedeći znak u izrazu i šalje ga potprogramu **B**) i sukladno tome očekuje povratni završni znak **)** nakon čega čita sljedeći znak i vraća ga

prethodnom potprogramu. Ako je prihvaćeni znak **varijabla**, potprogram čita sljedeći znak i vraća ga prethodnom potprogramu.

D→(B) **D→varijabla**

Ovim procesom su produkcije kompletne i pokrivaju sve slučajeve. Gramatika za aritmetičke izraze prilagođena tehnici rekurzivnog spusta:

G = ({A, B, C, D},{+,-,*,/,>,<,>=,<=,=,!=,(,),varijabla},P,A)

Produkcije:

A→B$	**B→C+B**	**C→D<B**	**D→(B)**
	B→C-B	**C→D>B**	**D→varijabla**
	B→C*B	**C→D<=B**	
	B→C/B	**C→D>=B**	
		C→D=B	
		C→D!=B	
		C→D	

3. Ostvarenje programa

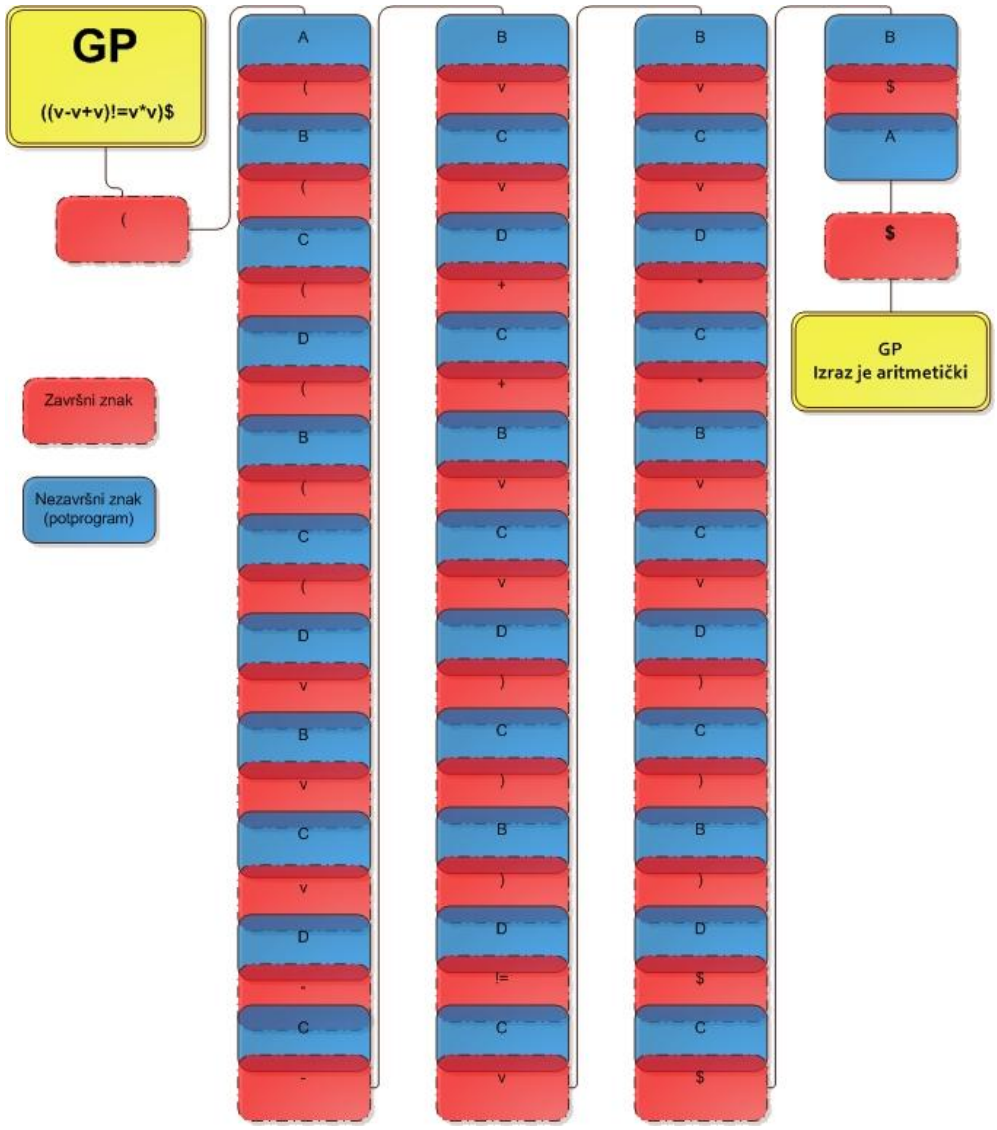
Prilagodbom gramatike je zapravo objašnjeno funkcioniranje programa. Funkcije A, B, C i D imaju funkciju potprograma **A**, **B**, **C** i **D**. Svaka od tih funkcija prihvaća znak (Ulaz u funkciji) i trenutni izraz (Izraz u funkciji). Kao što je već navedeno izraz je napisan u obliku polja. Izraz se čita s lijeva na desno bez povratka i zato se prvi znak u izrazu „reže“[3] i šalje i(li) provjerava kako je navedeno u prethodnom poglavlju. Na taj način se uklanja mogućnost da kad potprogram čita sljedeći znak u izrazu da ne pročita isti znak. Pored navedenih potprograma dodan je potprogram tj. funkcija GlavniProgram[4] koja radi inicijalizaciju gramatike. Funkcije koje očekuju neki završni znak[5], a dobiju neočekivani, vraćaju nulu čime se automatski sprječava daljnje čitanje znakova iz izraza i ispisuje da izraz nije aritmetički.

[3] Naredbom Ulaz = Izraz.pop(0)
[4] Program se mogao napisati i bez te funkcije tako da je mogla funkcija A imati ulogu funkcije GlavniProgram, ali postupak iz knjige *Jezični procesori 1, Siniša Srbljić*, navodi pisanje takvog potprograma.
[5] Funkcije A i D

4. Vizualizacija programa

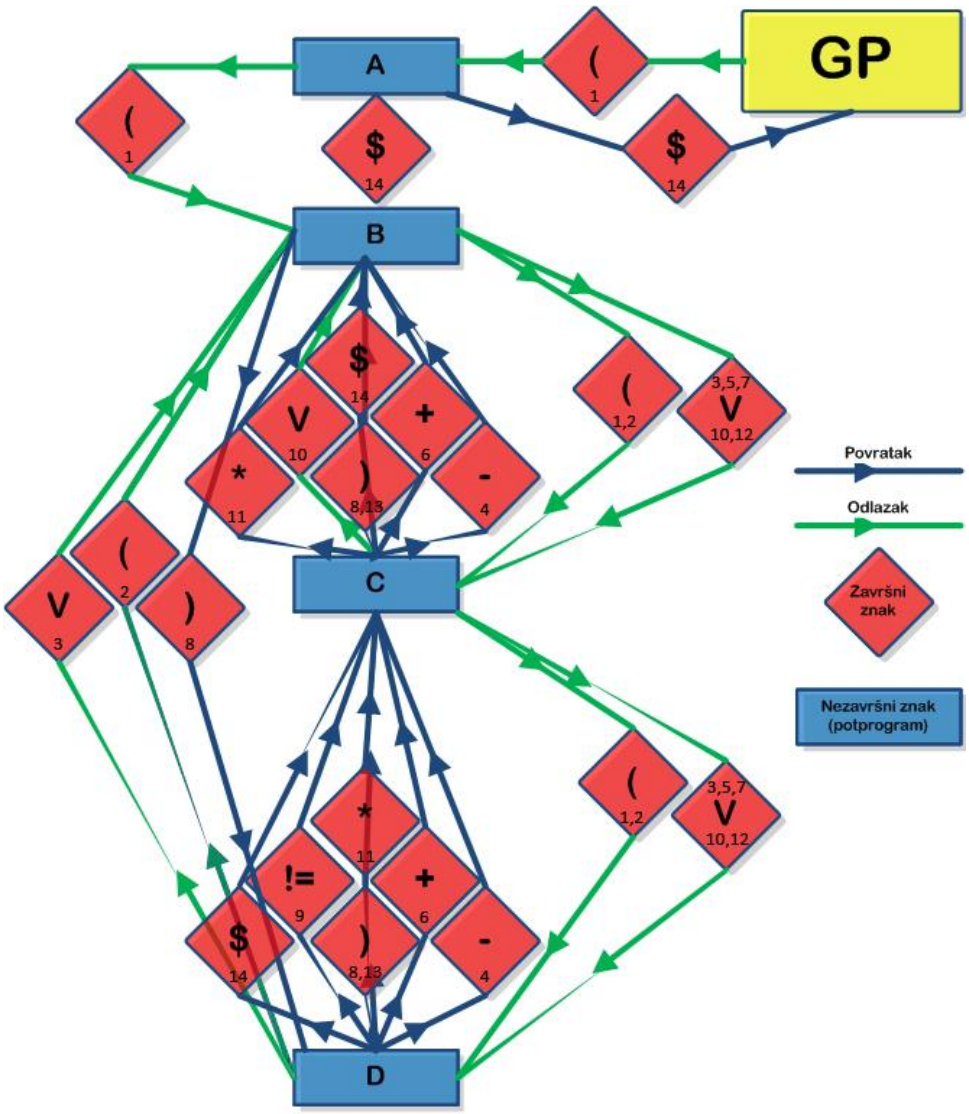
Dijagram prilagođen lakšem čitanju programa.

Najbolje pratiti uz program s koracima koji ispisuje svaki korak (u koju funkciju se ulazi, koji znak se predaje ili vraća itd.)[6] . Složeni dijagram je na sljedećoj stranici.



[6] UTR_SSP_2020_Py2.7_Telefunken_S_koracima.py
Pokrenuti ga u Python Shellu zbog preglednosti

Dijagram



1	2	3	4	5	6	7	8	9	10	11	12	13	14
(	(	a	-	b	+	f	)	!=	c	*	e	)	$

5. Zaključak

Pokretanjem programa bez koraka[7] i korištenjem istog za provjeru je li neki izraz aritmetički ubrzo shvatimo zašto je tehnika rekurzivnog spusta vrlo važna. Zahvaljujući njoj program je izrazito brz (u programu s koracima se vrijeme troši na ispisivanje koraka). Čak i relativno kompliciran aritmetički izraz
((a+a+a)/(a+a+a))!=(a/a)*(a-a)+a
se provjeri gotovo trenutno.

Dok bi se za jednostavnije izraze moglo napisati jednostavniji program s provjerama, što su izrazi duži i kompliciraniji, program bi jednostavno bio sve sporiji.

Za program tehnikom rekurzivnog spusta koristi se relativno jednostavan algoritam prezentiran u knjizi *Jezični procesori 1, Siniša Srbljić*. Veći problem je bila prilagodba izraza i gramatike navedena u 2. poglavlju.

[7] UTR_SSP_2020_Py2.7_Telefunken_Bez_koraka