Programski prevodioci

Vežbe 5

Sadržaj

1. Uvod

Na ovim vežbama biće prikazani prioriteti kao i detekcija semantičkih grešaka.

2. Problem "visećeg" `else`

Posmatrajmo sledeću konstrukciju:

```
if (/* uslov */)
  if (/* uslov */)
    /* iskaz */
  else
    /* iskaz */
```

Uzmimo da je definisana sledeća gramatika za ovakvu konstrukciju:

```
if_statement
  : if_part
  | if_part _ELSE statement
  ;

if_part
  : _IF _LPAREN rel_exp _RPAREN statement
  ;
```

Kada parser naiđe na token `_ELSE`, dolazi do `shift/reduce` konflikta:

1. Parser može izvršiti redukciju i tako završiti parsiranje unutrašnjeg `if` iskaza. Ovo bi značilo da `else` deo pripada spoljašnjoj petlji.
2. Parser može nastaviti sa preuzimanjem tokena i tek nakon kompletnog preuzimanja `else` dela izvršiti redukciju. Ovo bi značilo da `else` deo pripada unutrašnjoj petlji.

Parser ne može samostalno da odluči koji od ova 2 navedena slučaja je ispravan, pa stoga dolazi do konflikta.

Razrešavanje konflikta definisanjem prioriteta:

```
...

%nonassoc ONLY_IF ①
%nonassoc _ELSE

%%

if_statement
  : if_part %prec ONLY_IF ②
  | if_part _ELSE statement
  ;

if_part
  : _IF _LPAREN rel_exp _RPAREN statement
  ;

%%

...
```

① Bison omogućava navođenje prioriteta i asocijativnosti tokena.

> *Moguće deklaracije su:*
>
> `%left`
> : Operatori su levo asocijativni.
>
> `%right`
> : Operatori su desno asocijativni.
>
> `%nonassoc`
> : Definiše samo prioritete, a ne i asocijativnost.

Posle svake deklaracije navodi se lista tokena na koje se deklaracija odnosi.

Prioriteti operatora su kontrolisani redosledom navođenja deklaracija. Token koji je prvi naveden u `.y` datoteci ima najmanji prioritet, dok poslednji token ima najveći prioritet. Prema tome, u listingu iznad, navedeno je da token `ONLY_IF` ima manji prioritet od `_ELSE` tokena.

Bitno je naglasiti da se prioriteti i asocijativnosti odnose samo na tokene, a ne i na pojmove.

② Imajući u vidu da se prioriteti odnose samo na tokene, dolazimo do problema. Prvo `if_statement` pravilo ne sadrži nijedan token, već samo pojam `if_part`. Dakle, nije moguće specificirati prioritete tokena, kada token ne postoji. U ovakvim slučajevima može se navesti `%prec` modifikator praćen nazivom nekog izmišljenog tokena. Ovo znači da token `ONLY_IF` ne postoji zapravo u ulaznom tekstu, već da je to izmišljen pomoćni token koji nam omogućava da specificiramo da `_ELSE` ima veći prioritet od tog izmišljenog tokena.

Na ovaj način, razrešena je dvosmislenost gramatike. Token `_ELSE` ima veći prioritet, pa će parser

nedvosmisleno odlučiti da nastavi preuzimanje tokena. Ovako podešeni prioriteti imaju za posledicu da `else` uvek pripada unutrašnjem `if` iskazu, što i jeste način na koji C i miniC programi funkcionišu.

3. Zadaci

3.1. Zadatak 1: `void` tip

Proširiti gramatiku novim tipom podatka `void`. Tip `void` se može pojaviti samo kao povratni tip funkcije, ali ne i kao tip promenljive ili parametra (u tom slučaju treba prijaviti semantičku grešku).

Slično kao što je rađeno za `int` i `unsigned`.

3.2. Zadatak 2: `return ;`

Proširiti gramatiku novim `return` iskazom koji ima oblik:

```
"return" ";"
```

Realizovati sledeće semantičke provere:

1. Ako se u `void` funkciji nađe `return exp ;`, treba prijaviti semantičku grešku, jer funkcija ne bi trebalo da vrati vrednost.
2. Ako se u `int` ili `unsigned` funkciji nađe `return ;`, treba prijaviti *warning* jer se očekuje da funkcija vrati neku vrednost.
3. Ako se u `int` ili `unsigned` funkciji nijednom ne pojavi `return` naredba, treba prijaviti *warning* jer se očekuje da funkcija vrati neku vrednost.

Tabela 1. Matrica semantičkih provera:

	`void`	`int`/`unsigned`
`return exp ;`	error	OK
`return ;`	OK	warning
Bez `return`	OK	warning

3.3. Zadatak 3: `for` iskaz

Proširiti miniC gramatiku `for` iskazom koji ima sledeći oblik:

```
"for" "(" <type> <id1> "=" <lit> ";" <relation> ";" <id2> "++" ")"
  <stmt>
```

gde je:

`<type>`
: Tip podatka (`int` ili `unsigned`)

`<id1>` i `<id2>`
: Identifikatori

`<relation>`
: Relacioni izraz

`<stmt>`
: Statement

Realizovati sledeće semantičke provere:

1. `<id1>` treba da bude lokalna promenljiva za `for` iskaz (sledeći `for` iskaz može da definiše iterator sa istim imenom).
2. Tip literala `<lit>` treba da bude isti kao tip promenljive `<id1>`.
3. `<id1>` i `<id2>` treba da budu ista promenljiva.

DODATNO: Ispravno tretirati ugnježdene `for` iskaze.

Primer ispravnog `for` iskaza:

```
int x;
x = 0;
for (int i = 0; i < 8; i++)
  x = x + i;
```

3.4. Zadatak 4: `branch` iskaz

```
"branch" "(" <var> ";" <const1> "," <const2> "," <const3> ")"
    "first" <statement1>
    "second" <statement2>
    "third" <statement3>
    "otherwise" <statement4>
```

Gde:

- <var> predstavlja ime promenljive ili parametra
- <const1>, <const2> i <const3> predstavljaju konstante
- <statement1>, <statement2>, <statement3> i <statement4> predstavljaju iskaze

Uvek mora postojati tačno tri konstante i tačno četiri iskaza

Realizovati sledeće semantičke provere:

1. Promenljiva var mora biti prethodno deklarisana.

2. Konstante const1, const2 i const3 moraju biti istog tipa kao i var.

Primer:

```
    branch ( a ; 1 , 3 , 5 )
        first a = a + 1;
        second a = a + 3;
        third a = a + 5;
        otherwise a = a - 3;
```

3.5. Zadatak 5: `switch` iskaz

Proširiti miniC gramatiku pojednostavljenim `switch` iskazom. Sintaksa `switch` iskaza ima oblik:

```
"switch" "(" <switch_expression> ")" "{"
    "case" <constant_expression> ":" <case_body> [ "break" ";" ]
     ...
    [ "default" ":" <default_statement> ]
"}"
```

gde je:

`<switch_expression>`
: Ime promenljive

`<constant_expression>`
: Konstanta

`<case_body>`
: Iskaz (statement)

`<default_statement>`
: Iskaz (statement)

Pri tome:

- Mora postojati bar jedna `case` naredba.
- `break` naredba se opciono može pojaviti samo na kraju `case` naredbe.
- `default` naredba je opciona i moze se pojaviti samo posle svih `case` naredbi.

Realizovati sledeće semantičke provere:

1. Promenljiva u `<switch_expression>` mora biti prethodno deklarisana.
2. Tip konstante u `case` naredbi mora biti isti kao tip promenljive u `<switch_expression>`.
3. Konstante u svim `case` iskazima moraju biti jedinstvene.

Prilikom testiranja ne treba testirati ugnježdene `switch` iskaze.

Primer:

```
switch(state) {
  case 1: x = 1; break;
  case 2: { x = 5;} break;
  default: x = 10;
}
```