Programski prevodioci

04 miniC

Fakultet tehničkih nauka, Novi Sad
22-23/Z
Dunja Vrbaški

miniC i micko

- programski jezik miniC razvijen u okviru nastave na FTN
- kompajler za jezik je: micko
- autori: doc dr Zorica Suvajdžin Rakić, prof dr Miroslav Hajduković, prof dr Žarko Živanov

- resursi:
- https://github.com/zsuvajdzin/micko
- zbirke (acs – prevodioci)
    - Flex&Bison
    - miniC
        - gramatike
        - micko ukratko
        - leksička analiza
        - sintaksna analiza
        - semantika
        - generisanje koda

```
//OPIS: ABS funkcija
//RETURN: 5

int abs(int i) {
   int res;

   if(i < 0)
      res = 0 - i;
   else
      res = i;

   return res;
}

int main() {
   return abs(-5);
}
```

tokeni:

ID,

INT_NUMBER, UINT_NUMBER,

LPAREN, RPAREN, LBRACKET,

RBRACKET, SEMICOLON,

AROP, RELOP

ASSIGN,

TYPE,

IF, ELSE

RETURN

skip:

beline i linijski komentari

```
[ \t\n]+                { /* skip */ }

"int"                   { return _TYPE; }
"unsigned"              { return _TYPE; }
"if"                    { return _IF; }
"else"                  { return _ELSE; }
"return"                { return _RETURN; }

"("                     { return _LPAREN; }
")"                     { return _RPAREN; }
"{"                     { return _LBRACKET; }
"}"                     { return _RBRACKET; }
";"                     { return _SEMICOLON; }
"="                     { return _ASSIGN; }

"+"                     { return _AROP; }
"-"                     { return _AROP; }

"<"                     { return _RELOP; }
">"                     { return _RELOP; }
"<="                    { return _RELOP; }
">="                    { return _RELOP; }
"=="                    { return _RELOP; }
"!="                    { return _RELOP; }

[a-zA-Z][a-zA-Z0-9]*    { return _ID; }
[+-]?[0-9]{1,10}        { return _INT_NUMBER;}
[0-9]{1,10}[uU]         { return _UINT_NUMBER;}

\/\/.*                  { /* skip */ }
```

Leksika (flex, jezik.l)

identifikator (ime)

sastoji se od 1 ili više slova ili cifara, ali ne sme početi cifrom

```
[a-zA-Z][a-zA-Z0-9]*
```

Token:

_ID

Napomena:

regularni izraz: iznad

BNF: `id ::= letter ( letter | digit )*` (zbirka)

celobrojni označeni literal

1 ili više cifara

```
[+-]?[0-9]{1,10}
```

Token: _INT_NUMBER

celobrojni neoznačeni literal

1 ili više cifara iz kojih sledi oznaka: malo ili veliko U

```
[0-9]{1,10}[uU]
```

Token: _UINT_NUMBER

sintaksni elementi

```
"("
")"
"{"
"}"
";"
```

Tokeni: _LPAREN, _RPAREN, _LBRACKET, _RBRACKET, _SEMICOLON

operatori

```
"+"
"-"
```

Token: _AROP

```
"<"
">"
"<="
">="
"=="
"!="
```

Token: _RELOP

```
"="
```

Token: _ASSIGN

Razdvojeni RE iako se radi o istom tokenu (značajno za sintaksu).
Imaće drugačije vrednosti (značajno za semantiku i gk).

tip

```
"int"
"unsigned"
```

Token: _TYPE

if-else

```
"if"
"else"
```

Token: _IF, _ELSE

return

```
"return"
```

Token: _RETURN

Sintaksa (bison, jezik.y)

program, function_list, function

type, parameter,

body, variable_list, variable,

statement_list, statement,

compound_statement, assignment_statement,

num_exp, exp, literal,

function_call, argument,

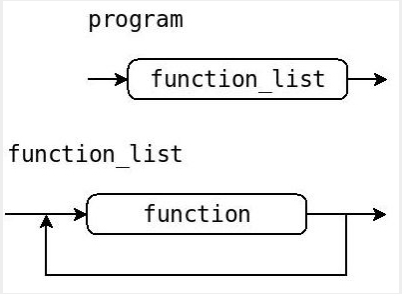
if_statement, if_part, rel_exp,

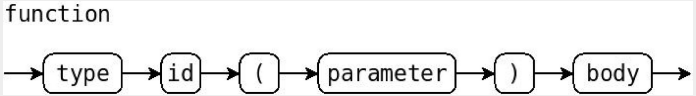
return_statement

```
%{
  #include <stdio.h>
  #include "defs.h"

  int yyparse(void);
  int yylex(void);
  ...
%}

%token _TYPE
%token _IF
...


%%

program
  : function_list
  ;

function_list
  : function
  | function_list function
  ;

function
  : type _ID _LPAREN parameter _RPAREN body
  ;

type
  : _TYPE
  ;

parameter
  : /* empty */
  | type _ID
  ;

body
  : _LBRACKET variable_list statement_list _RBRACKET
  ;

variable_list
  : /* empty */
  | variable_list variable
  ;

variable
  : _TYPE _ID _SEMICOLON
  ;
```

```
program
    : function_list
    ;

function_list
   : function
   | function_list function
   ;

function
   : type _ID _LPAREN parameter _RPAREN body
   ;
```

sintaksni dijagrami





Program ima bar jednu funkciju.
Obratiti pažnju:

- nigde ne piše da je to funkcija main().

```
type
  : _TYPE
  ;

parameter
  : /* empty */
  | type _ID
;

body
  : _LBRACKET variable_list statement_list _RBRACKET
  ;
```

```
✔ int f1(int p) {
    ...
  }

✔ int f2() {
    int x;
    int y;
    ...
  }

✖ int f(int p, int q) {
    ...
  }
```

Obratiti pažnju:

- nema liste parametara. Funkcija ima 0 ili 1 parametar.
- nema void. Funkcija mora vratiti int ili uint.
- u telu prvo idu deklaracije, a onda iskazi.

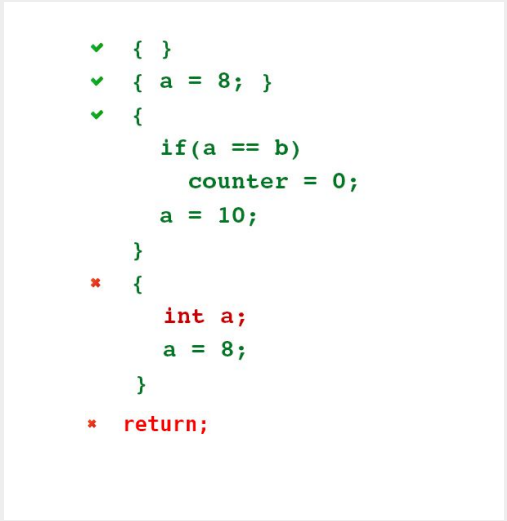
```
variable_list
  : /* empty */
  | variable_list variable
;

variable
  : _TYPE _ID _SEMICOLON
;
```

```
✔ int counter;
✔ int line;
✔ unsigned n;

✖ int counter, line;
```

Obratiti pažnju:
- nema višestruke deklaracije

```
statement_list
  : /* empty */
  | statement_list statement
  ;

statement
  : compound_statement
  | assignment_statement
  | if_statement
  | return_statement
  ;
```

```
compound_statement
  : _LBRACKET statement_list _RBRACKET
  ;

assignment_statement
  : _ID _ASSIGN num_exp _SEMICOLON
  ;

return_statement
  : _RETURN num_exp _SEMICOLON
  ;
```



```
%nonassoc ONLY_IF
%nonassoc _ELSE

%%


if_statement
  : if_part %prec ONLY_IF
  | if_part _ELSE statement
  ;

if_part
  : _IF _LPAREN rel_exp _RPAREN statement
  ;
```

```
num_exp
   : exp
   | num_exp _AROP exp
   ;

exp
   : literal
   | _ID
   | function_call
   | _LPAREN num_exp _RPAREN
   ;

literal
   : _INT_NUMBER
   | _UINT_NUMBER
   ;
```

```
✔ a = b;
✔ counter = 0;
✔ line = line + 1;
✔ c = j + (k – 4);
✔ m = f();
✖ func();
✖ b = m * n;
✖ i = j++;
```

num_exp se pojavljivao u:
assignment_statement i return_statement

```
function_call
  : _ID _LPAREN argument _RPAREN
  ;

argument
  : /* empty */
  | num_exp
  ;
```

Primetiti rekurziju koja nije direktna:
num_exp → function_call → argument → num_exp

```
rel_exp
  : num_exp _RELOP num_exp
  ;
```

rel_exp se pojavljivao u:
if_part

```
%nonassoc ONLY_IF
%nonassoc _ELSE

%%


if_statement
  : if_part %prec ONLY_IF
  | if_part _ELSE statement
  ;

if_part
  : _IF _LPAREN rel_exp _RPAREN statement
  ;
```

Postoje sledeće deklaracije (*navode se umesto %token*)

```
%left
%right
%nonassoc
%precedence
```

Token koji je kasnije definisan ima veći prioritet

Postoji i modifikator (*navodi se kod pravila*)

```
%prec
```

`%left`	levo asocijativan (token/operator)
`%right`	desno asocijativan (token/operator)
`%nonassoc`	nije asocijativan (token/operator) Ako je OP posmatrani token: *a OP b OP c* - sintaksna greška
`%precedence`	Definiše samo prioritet, ne definiše asocijativnost. Za slučajeve kada asocijativnost nije potrebna i kad predstavlja višak informacija i zapravo može i da napravi/sakrije drugi problem.
`%prec`	Pravilo podrazumevano preuzima prioritet poslednjeg tokena. Kada postoji modifikator - pravilo preuzima prioritet navedenog tokena. Navodi se na kraju pravila. Za potrebe promene prioriteta u nekom kontekstu.

Razrešenje shift/reduce konflikta se može postići na osnovu prioriteta.

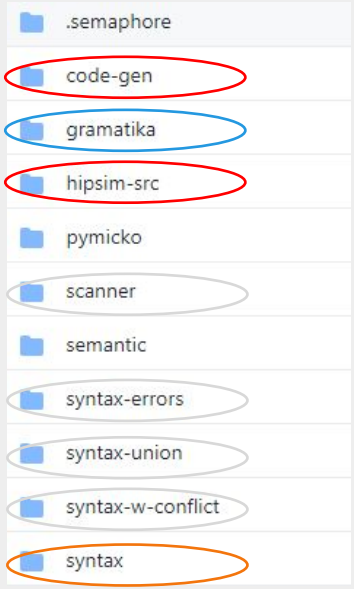
Reduce:
Ako je prioritet pravila veći od prioriteta LA tokena
Ako imaju isti prioritet i asocijativnost pravila je left

Shift – u ostalim slučajevima
Ako je prioritet pravila manji od prioriteta LA tokena
Ako token ili pravilo nemaju prioritet

5 + 2 + 3	→	(5 + 2) + 3	reduce želimo
5 + 2 * 3	→	5 + (2 * 3)	shift želimo
5 * 2 + 3	→	(5 * 2) + 3	reduce želimo

```
%nonassoc ONLY_IF
%nonassoc _ELSE

%%


if_statement
  : if_part %prec ONLY_IF
  | if_part _ELSE statement
  ;

if_part
  : _IF _LPAREN rel_exp _RPAREN statement
  ;
```

Pošto se odnose na tokene (terminale) – može se uvesti pomoćni token koji koristimo isključivo za prioritet.



- micko kompajler (GH)
- razdvojene su faze, poslednja code-gen

Pitanja i zadaci

Implementirati kalkulator (+, *).

Da li IF/ELSE moze da se reši uz pomoć %precedence umesto %nonasoc deklaracije? Probati.