Programski prevodioci

Vežbe 6

Sadržaj

1. Uvod

U dokumentu su data rešenja zadataka koji su rađeni na šestim vežbama.

2. Rešenja zadataka

Svi zadaci se rešavaju sledećim redosledom:

- Dodati nove tokene na vrh `.y` datoteke.
- Definisati regularne izraze u `.l` datoteci za nove tokene.
- Proširiti gramatiku jezika tako da sintaksno podržava novu konstrukciju.
- Dodati semantičke provere.

2.1. Zadatak 1: `postinkrement kao iskaz`

Dodati token _INC, Izmeniti gramatiku na sledeći način:

```
inc_statement
  : _ID
      {
          if(lookup_symbol($1, FUN) != NO_INDEX){
          err("Postincrement may be only used on variables, not functions.");
          }
          int idx = lookup_symbol($1, VAR|PAR);
          if(idx == NO_INDEX){
          err("%s is not declared previously as a variable", $1);
          }
      }
   _INC _SEMICOLON
  ;

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

```
  | inc_statement
  ;
```

2.2. Zadatak 2: `lokalne promenljive unutar bloka`

Dodati globalnu promenljivu koja će se koristiti kao brojač nivoa bloka:

```
%{
  int block;
%}
```

Proširiti gramatiku tako da se omogući deklarisanje lokalnih promenljivih unutar bloka:

```
compound_statement
  : _LBRACKET
  {
    block++; ③
    $<i>$ = get_last_element(); ①
  }
  variable_list statement_list _RBRACKET
  {
    //print_symtab();
    block--; ③
    clear_symbols($<i>2 + 1); ②
  }
  ;
```

① Na početku bloka treba sačuvati indeks poslednjeg elementa u tabeli simbola.

② Na kraju bloka treba obrisati sve lokalne promenljive iz tog bloka. Funkciji `clear_symbols` se prosleđuje indeks elementa od kog treba početi brisanje. Prema tome, prosleđuje se prethodno sačuvani indeks uvećan za 1, kako bi se izbrisale sve promenljive iz tog bloka.

③ Na početku bloka treba globalnu promenljivu `block` uvećati za 1. Analogno, na kraju bloka je treba umanjiti za 1. Prema tome, ova globalna promenljiva predstavlja "nivo" bloka u kojem parser trenutno nalazi.

```
variable
  : _TYPE_ _ID _SEMICOLON
  {
    int i = lookup_symbol($2, VAR|PAR);
    if( (i != -1) && (get_atr2(i) == block) ) ②
      err("redefinition of '%s'", $2);
    else {
```

```
      insert_symbol($2, VAR, $1, ++var_num, block); ①
    }
  }
  ;
```

① Prilikom dodavanja lokalne promenljive u tabelu simbola, poslednja kolona je ranije bila neiskorišćena. Imajući to u vidu, poslednju kolonu je uvek dozvoljeno iskoristiti za čuvanje neke dodatne informacije, ako je to potrebno za rešavanje zadataka. Konkretno u ovom zadatku, u kolonu `atr2` čuvamo "nivo" u kojem je promenljiva deklarisana.

② Ranije je *dovoljan* uslov za grešku bio to da u tabeli simbola već postoji promenljiva sa istim nazivom. Za ovaj zadatak treba modifikovati uslov tako da je greška *samo ako* već postoji promenljiva sa istim nazivom *i* to ako je ona deklarisana *baš u tom bloku u kojem se trenutno nalazimo*.

2.3. Zadatak 3: `branch` iskaz

Doati u specifikaciji skenera:

```
"branch"     { return _BRANCH;}
"do_start"   { return _DO_START;}
"do_end"     { return _DO_END;}
","          { return _COMMA;}
```

Na početku .y fajla dodati:

```
%token _COMMA
%token _DO_START
%token _DO_END
%token _BRANCH
```

Dodati globalne promenljive:

```
  int id_idx = -1;
  int branch_array[100];
  int branch_array_idx = 0;
```

Proširiti gramatiku na sledeći način:

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

branch_statement
  : _BRANCH _LPAREN _ID
  {
    id_idx = lookup_symbol($3, VAR|PAR);
```

```
    if(id_idx == NO_INDEX)
        err("not declared");
  } _SEMICOLON constants
  {
      branch_array_idx = 0;
  }
   _RPAREN do_part
  ;

constants
  : literal
  {
    if(get_type($1) != get_type(id_idx))
        err("ne poklapaju se tipovi");
    int i = 0;
    while(i < branch_array_idx) {
        if($1 == branch_array[i]) { //ako takva konstanta vec postoji u nizu
            err("duplicated constant");
            break;
        }
        i++;
    }
    if(i == branch_array_idx) { //ako nije duplikat
        branch_array[branch_array_idx] = $1; //ubaci konstantu u niz
        branch_array_idx++;
    }
  }
  | constants _COMMA literal
  {
     if(get_type($3) != get_type(id_idx))
        err("ne poklapaju se tipovi");

     int i = 0;
     while(i < branch_array_idx) {
        if($3 == branch_array[i]) { //ako takva konstanta vec postoji u nizu
            err("duplicated constant");
            break;
        }
        i++;
    }
    if(i == branch_array_idx) { //ako nije duplikat
        branch_array[branch_array_idx] = $3; //ubaci konstantu u niz
        branch_array_idx++;
    }
  }
  ;

do_part
  : _DO_START statement _DO_END
  | do_part _DO_START statement _DO_END
  ;
```

2.4. Zadatak 4: funkcija sa više parametara

```
%{
  int* parameter_map[128];
  int arg_counter = 0;
%}

...
%type <i> num_exp exp literal function_call rel_exp argument
...

function
  : _TYPE _ID
      {
        fun_idx = lookup_symbol($2, FUN);
        if(fun_idx == NO_INDEX){
          int* param_types = (int*) malloc(sizeof(int)*128);
          fun_idx = insert_symbol($2, FUN, $1, NO_ATR, NO_ATR);
          parameter_map[fun_idx] = param_types;
        }
        else
          err("redefinition of function '%s'", $2);
      }
    _LPAREN parameter_list _RPAREN body
      {
        clear_symbols(fun_idx + 1);
        var_num = 0;
      }
  ;

parameter_list
  : /* empty */
    { set_atr1(fun_idx, 0); }
  | parameters
  ;

parameters
  : parameter
  | parameters _COMMA parameter
  ;

parameter
  : _TYPE _ID
      {
        if(lookup_symbol($2, PAR) != -1){
          err("Redefinition of parameter %s ", $2);
        }
        insert_symbol($2, PAR, $1, 1, NO_ATR);
        int num_params = get_atr1(fun_idx);
        int* param_types = parameter_map[fun_idx];
```

```
        param_types[num_params] = $1;
        num_params += 1;
        set_atr1(fun_idx, num_params);
      }
  ;
```

Poziv funkcije:

```
function_call
  : _ID
      {
        fcall_idx = lookup_symbol($1, FUN);
        if(fcall_idx == NO_INDEX)
          err("'%s' is not a function", $1);

        arg_counter = 0;
      }
    _LPAREN argument_list _RPAREN
      {
        if(get_atr1(fcall_idx) != arg_counter)
          err("wrong number of args to function '%s'",
              get_name(fcall_idx));
        set_type(FUN_REG, get_type(fcall_idx));
        $$ = FUN_REG;
        arg_counter = 0;
      }
  ;

argument_list
  : /* empty */
  | arguments
  ;
```

```
arguments
  : argument
  | arguments _COMMA argument
  ;

argument
  : num_exp
    {
      if(parameter_map[fcall_idx][arg_counter] != get_type($1))
        err("incompatible type for argument in '%s'",
            get_name(fcall_idx));
      arg_counter += 1;
    }
  ;
```