DOKUMENTACIJA

Programski prevodioci - predmetni zadatak

Osnovni podaci

Broj indeksa	Ime i prezime	Tema
SW 4 2019	Dušan Lazić	Nizovi i foreach petlje

Korišćeni alati

Naziv	Verzija
GCC	8.3.0
Flex	2.6.4
Bison	3.3.2

Evidencija implementiranog dela

Na najvišem nivou apstrakcije, miniC jezik proširen je nizovima i foreach petljama. Time je u ovom projektu urađeno nekoliko zadataka. Prvi zadatak bio je deklaracija nizova, za šta su urađene sintaksna i semantička analiza, kao i generisanje koda. Nakon toga je urađena dodela vrednosti elementima niza (na osnovu indeksa ili navođenjem svih vrednosti niza), kao i pristupanje elementima niza preko indeksa. Za obe stavke urađene su sintaksna i semantička analiza, i generisanje koda. Poslednji zadatak se tiče foreach petlji za iteraciju kroz niz i delimično je urađen. Ovom zadatku nedostaje deo koji se tiče generisanja koda. Zbog ovog nedostatka nije omogućen pristup trenutnom elementu niza iz tela foreach petlje, o čemu će biti više reči u nastavku. Kao i kod prethodnih zadataka, sintaksa i semantika za ovaj zadatak su odrađene.

Detalji implementacije

Svi zadaci imaju isti prvi korak, a to je pisanje testova. Pomoću testova je opisano kako sintaksa i semantika treba da izgledaju, i navedena je povratna vredonst main funkcije kako bi se opisalo i ponašanje koda (provera generisanja koda). Nakon toga rađene su redom leksička analiza, sintaksna, semantička i na kraju generisanje koda.

Deklaracija niza

Za potrebe deklaracije niza, dodata su dva nova tokena *_LBRACKET* i *_RBRACKET* koji predstavljaju levu i desnu ugaonu zagradu. Bitno je napomenuti da već postoje tokeni ovog naziva, ali predstavljaju vitičaste zagrade zbog čega sam ih preimenovao u *_LBRACE* i *_RBRACE*.

Prošireno je pravilo *variable* (kod 1) što znači da će se deklaracija nizova navoditi na istom mestu gde i deklaracija drugih promenljivih — na početku funkcije.

```
variable
 : _TYPE _ID _SEMICOLON
     {
       if(lookup_symbol($2, VAR|PAR) == NO_INDEX)
          insert_symbol($2, VAR, $1, ++var_num, NO_ATR);
       else
          err("redefinition of '%s'", $2);
     }
 | _TYPE _ID _LBRACKET _INT_NUMBER _RBRACKET _SEMICOLON
     {
       if(lookup_symbol($2, VAR|PAR) == NO_INDEX) {
         insert_symbol($2, VAR, toggle_array[$1], ++var_num, atoi($4));
         var_num += atoi($4);
       }
     }
 ;
```

kod 1.

Tipovi podatka su prošireni sa dva nova tipa koji označavaju kojeg je tipa niz posmatrajući tip njegovih elemenata (kod 2).

```
//tipovi podataka
enum types { NO_TYPE, INT, UINT, INT_ARRAY, UINT_ARRAY };
```

kod 2.

Kako bi se na osnovu tipa elementa dobio tip niza ili obrnuto, uvedeni su nizovi koji se mogu koristiti kao mape, i služe za pronalaženje odgovarajućeg tipa (kod 3). Jedna takva mapa jeste *toggle_array* koja se koristi u primeru iznad (kod 1), a njeno ponašanje opisano je u komentaru u kodu ispod (kod 3). Postoje i mape *force_array*, *ignore_array* i *is_array*.

```
       toggle_array[1] = 3; // INT         => INT_ARRAY
       toggle_array[3] = 1; // INT_ARRAY   => INT
       toggle_array[2] = 4; // UINT        => UINT_ARRAY
       toggle_array[4] = 2; // UINT_ARRAY => UINT
```

kod 3.

Vrednost prvog atributa simbola niza jeste *var_num* uvećan za jedan, i on označava početak niza u odnosu na pokazivač stek frejma. Odmah nakon upisa simbola u tabelu simbola, *var_num* se uvećava za dužinu niza (kod 1) čime se efektivno ove lokacije ostavljaju nizu na raspolaganje. Drugi atribut simbola jeste dužina niza (kod 1) i koristi se kasnije prilikom pristupa elementu ili dodeli vrednosti.

Na narednom primeru prikazan je izvorni kod sa leve, i generisani kod sa desne strane.

```
int main() {
   int niz[7];
```

```
main:
        PUSH    %14
        MOV     %15,%14
        SUBS    %15,$32,%15
```

kod 4.

Dodela vrednosti elementima niza

Dodavanje vrednosti elementima niza može se izvršiti na dva načina: putem indeksa elementa niza, ili navođenjem svih vrednosti elemenata niza. Za drugi pristup koriste se već postojeći tokeni preimenovani u *_LBRACE* i *_RBRACE,* koji su pomenuti ranije.

Za oba pristupa prošireno je pravilo *assignment_statement* (kod 5) kako bi se omogućilo da na levoj strani znaka dodele navedemo ciljani element niza (**niz[3]** = 5), ili da na desnoj strani navedemo sve vrednosti elemenata niza (niz = **{1, 2, 3}**).

```
// niz[0] = 5;
| _ID _LBRACKET _INT_NUMBER _RBRACKET _ASSIGN num_exp _SEMICOLON //
    {
      int idx = lookup_symbol($1, VAR|PAR);
      if (idx == NO_INDEX)
        err("invalid lvalue '%s' in assignment", $1);
      else
        if(ignore_array[get_type(idx)] != get_type($6))
          err("incompatible types in assignment");
      gen_mov_offset($6, idx, 0, atoi($3));
    }
```

kod 5.

Radi olakšanog generisanja MOV instrukcija koje se tiču nizova, uvedena je nova funkcija *gen_mov_offset* (kod 5) koja prima dva dodatna parametra — *offset* za ulazni indeks i *offset* za izlazni indeks. U primeru iznad, kao offset za izlaz dodaje se vrednost trećeg tokena, koji predstavlja indeks ciljanog elementa. *Offset* u ovom slučaju služi za računanje adrese ciljanog elementa na osnovu adrese niza (element sa indeksom 0) i prosleđenog indeksa.

Za drugi pristup uvedeno je pravilo *array*, koje kaže da vrednosti pri dodeli moraju biti odgovarajućeg tipa, u odgovarajućem broju, razdvojene zarezom i obuhvaćene vitičastim

zagradama. Broj elemenata prati se pomoću globalne promenljive num_of_elements i poredi se sa drugim atributom simbola niza radi utvrđivanja da li elemenata ima onoliko koliko je navedeno deklaracijom niza. Za svaki element niza generiše se kod pomoću pomenute *gen_mov_offset* funkcije.

```
int main() {
   int niz[7];

   niz = { 100, 101, 102, 103,
104, 105, 106 };

   return niz[3];
}
```

```
main:
        PUSH    %14
        MOV     %15,%14
        SUBS    %15,$32,%15
@main_body:
        MOV     $100,-4(%14)
        MOV     $101,-8(%14)
        MOV     $102,-12(%14)
        MOV     $103,-16(%14)
        MOV     $104,-20(%14)
        MOV     $105,-24(%14)
        MOV     $106,-28(%14)
        MOV     -16(%14),%13
        JMP     @main_exit
@main_exit:
        MOV     %14,%15
        POP     %14
        RET
```

kod 6.

Pristup elementima niza

Ključna komponenta za pristup elementima niza je, kao i za dodelu, *gen_mov_offset* funkcija. Prošireno je pravilo *exp* pri čemu se pored provere redosleda tokena i provere da li postoji simbol u tabeli simbola, proverava i da li prosleđeni indeks ne prevazilazi veličinu niza. Kad god se radi o *exp* pojmu, za njega se radi provera da li je u pitanju niz, i ukoliko jeste, koristi se *gen_mov_offset* funkcija umesto običnog *gen_mov* (kod 7). Kao offset za čitanje koristi se globalna promenljiva offset, koja dobija svoju vrednost unutar *exp* pravila i predstavlja prosleđeni indeks.

```
      if(is_array[get_type($3)])
        gen_mov_offset($3, idx, offset, 0);
      else
        gen_mov($3, idx);
```

kod 7.

Foreach petlja — iteriranje

Dodati su novi tokeni *_FOREACH*, *_COLON*, *_CONTINUE* i *_BREAK.* Redosled tokena i pojmova određen je uvođenjem novih pravila *foreach_statement*, *break_statement* i *continue_statement.* Kako *break* i *continue* ne bi mogli da se pojave izvan foreach petlje, uvedena je globalna promenljiva *inside_foreach* koja označava da li se trenutno obrađuje telo *foreach* petlje, tj. da li su *break* i *continue* dozvoljeni.

Najinteresantniji i najproblematičniji deo za implementaiju bilo je generisanje koda. Iako su skokovi urađeni, pristup elementu trenutne iteracije nije uspešno realizovan, o čemu će biti reči na kraju.

U nastavku je prikazano generisanje koda za inicijalizaciju *foreach* petlje, kao i početak svake petlje. U dva slobodna registra smeštaju se indeks trenutne iteracije (na početku je 0) i maksimalan indeks niza koji se iterira (veličina niza umanjena za 1). Kreira se labela koja se sastoji od reči foreach i rednog broja (identifikatora) petlje, koji se pamti u globalnoj promenljivoj. Prva instrukcija petlje jeste uvećanje indeksa trenutne iteracije za 1 (kod 8).

```
iter_index_reg_idx = take_reg();
set_type(iter_index_reg_idx, INT);

iter_max_reg_idx = take_reg();
set_type(iter_max_reg_idx, INT);

code("\n\t\tMOV \t$0,");
gen_sym_name(iter_index_reg_idx);

code("\n\t\tMOV \t$%d,",
get_atr2(idx) - 1);
gen_sym_name(iter_max_reg_idx);;

code("\n@foreach%d:",
++foreach_num);
code("\n\t\tADDS\t");
gen_sym_name(iter_index_reg_idx);
code(",$1,");
gen_sym_name(iter_index_reg_idx);
```

```
        MOV     $0,%0

        MOV     $6,%1

@foreach1:

        ADDS    %0,$1,%0
```

kod 8.

Na kraju petlje proverava se da li su trenutni indeks i maksimalan indeks izjednačeni. Ukoliko nisu, izvršava se skok na početak petlje i ulazi se u narednu iteraciju. Ukoliko jesu, izlazi se iz petlje i nastavlja izvršavanje koda (kod 9).

```
gen_cmp(iter_index_reg_idx,
iter_max_reg_idx);
code("\n\t\tJLES \t@foreach%d",
foreach_num);
code("\n@break%d:", foreach_num);
```

```
        CMPS   %0,%1

        JLES     @foreach1

@break1:
```

kod 9.

Na kraju petlje proverava se da li je trenutni indeks manji ili jednak maksimalnom indeksu. Ukoliko jeste, izvršava se skok na početak petlje i ulazi se u narednu iteraciju. Ukoliko nije, izlazi se iz petlje i nastavlja se izvršavanje koda (kod 9).

Na samom kraju petlje nalazi se *break* labela za odgovarajuću foreach petlju. Ono što *break_statement* radi jeste da samo generiše instrukciju bezuslovnog skoka na tu labelu. Slično tako, *continue_statement* radi bezuslovni skok na početak petlje i time ulazi u novu iteraciju.

Ono što je bitno napomenuti jeste da je moguće ući u beskonačnu petlju upotrebom continue iskaza, jer se pomoću njega može preskočiti poređenje trenutnog i maksimalnog indeksa. Rešenje bi bilo pomeriti tu proveru sa kraja petlje na početak, tako da se petlja u svakom slučaju vraća na početak, a izvršava se samo pod uslovom da je trenutni indeks manji ili jednak maksimalnom. U suprotnom se pravi skok na *break* labelu i izlazi iz petlje.

Foreach petlja — pristup trenutnom elementu

Ovaj zadatak je ostao neurađen jer nisam našao dobar način za dinamički pristup elementima niza. Pod dinamičkim pristupom misli se na malo drugačiji pristup elementima u odnosu na onaj opisan u prethondim zadacima. Do sada sam morao da implementiram samo pristup elementu niza na osnovu već zadatog indeksa koji je literal, tako da je generisanje koda bilo trivijalno. U slučaju pristupa elementu trenutne iteracije, indeks nije unapred zadat već je promenljiv, što bi značilo da generisani kod mora da računa novu adresu pri svakoj iteraciji. Koliko sam imao prilike da istražim, HipSim nema podršku za ovakav rad sa adresama, a drugačiji pristup koji bi se uklopio u ostatak moje implementacije nisam uspeo da nađem.

Prema tome, dok se ne reši ovaj problem kao i problem dinamičkog pristupa elementima niza, foreach petlje se mogu koristiti samo kao obične for petlje.

Ideje za nastavak

Prvi problem koji je potrebno rešiti jeste naravno već pomenuti nedostatak mogućnosti pristupa elementu u foreach petlji. Rešavanje ovog problema dozvolilo bi da se foreach petlja koristi onako kao što je i osmišljena, čime bi se ovaj zadatak rešio do kraja. Time bi se omogućio i dinamički pristup elementima niza (npr. *a = niz[b]*) što bi učinilo nizove mnogo korisnijim i pružilo bi veće mogućnosti za operacije nad istim.

Literatura

- Slajdovi sa vebi
- Programski jezik miniC specifikacija i kompajler http://www.acs.uns.ac.rs/sites/default/files/miniC-A4.pdf