DOKUMENTACIJA

Programski prevodioci - predmetni zadatak

Osnovni podaci

Broj indeksa	Ime i prezime	Tema
SW 71/2019	Nikola Damjanović	Unije i type casting u miniC jeziku

Korišćeni alati

Naziv	Verzija
flex	2.6.4-6.2
bison	2:3.5.1+dfsg-1
make	4.2.1-1.2

Evidencija implementiranog dela

Za ovaj projekat dve stavke su implementirane u miniC jeziku. Unije i type casting.

Implementacija unija u miniC jeziku:

1) Sintaksa
2) Semantika
3) Generisanje koda

Implementacija type casting-a u miniC jeziku:

1) Sintaksa
2) Semantika

Detalji implementacije

Za parsiranje nije bilo potrebno širiti parser značajno. Radi mogućnosti rada sa unijama dva tokena su dodata u parser:

```
"union"    { return _UNION; }
"."        { return _DOT; }
```

Sintaksa i semantika za implementaciju unija će biti opisana sledeća.

Definisanje unija izgleda ovako u kodu.

```
program
  : union_list function_list…
  ;
```

```
union_list
  : /* empty */
  | union_list union_definition
  ;

union_definition
  : _UNION _ID
    {
      union_active = 1;
      union_name = $2;
      if (lookup_symbol(union_name, UNION_K) == NO_INDEX) {
        insert_symbol(union_name, UNION_K, NO_TYPE, ++union_counter, NO_ATR);
      }
      else {
        err("Union with name %s already exists", $2);
      }
    }
    _LBRACKET variable_list
    {
      if (union_var_num) {
        union_active = 0;
        union_var_num = 0;
      }
      else {
        err("Union %s doesn't have attributes.", $2);
      }
    }
    _RBRACKET _SEMICOLON
  ;
```

Niz tokena za definisanje unije je sledeći:

```
_UNION _ID _LBRACKET variable_list _RBRACKET _SEMICOLON
```

Što se semantike tiče, radi se provera da li unija sa istim imenom već postoji, da li su definisane varijable u uniji i takođe da li je definisana unija unutar unije (nije dozvoljeno).

“Instanciranje” unije i njeno korišćenje je implementirano na sledeći način.

```
variable
  : _TYPE _ID _SEMICOLON⋯
  | _UNION _ID _ID _SEMICOLON
    {
      if (union_active)
        err("Union cannot contain a union.");
      else {
        if (lookup_symbol($2, UNION_K) != NO_INDEX) {
          if (lookup_symbol($3, VAR) == NO_INDEX)
            insert_symbol_union($3, VAR, UNION, ++var_num, NO_ATR, $2);
          else
            err("Variable with name %s is already defined", $3);
        }
        else
          err("No union definition with name %s", $2);
      }
```

```
assignment_statement
  : _ID _ASSIGN num_exp _SEMICOLON⋯
  | _ID _DOT _ID _ASSIGN num_exp _SEMICOLON
    {
      int union_idx = lookup_symbol($1, VAR|PAR);
      if(union_idx == NO_INDEX)
        err("invalid lvalue '%s' in assignment", $1);
      else {
        // check if $3 _ID is variable in union definition and is valid type
        int union_var_idx = lookup_symbol_union_kind($3, UNION_VAR, get_union_name(union_idx));
        if(union_var_idx != NO_INDEX)
          if(cast_active) {
            if(get_type(union_var_idx) != cast_to_type)
              err("incompatible types in assignment. Value is casted to wrong type");
            cast_active = 0;
            cast_to_type = NO_TYPE;
          }
          else {
            if(get_type(union_var_idx) == get_type($5)) {
              // Sets union variable's active variable to the var_num in union definition
              set_active_variable(union_idx, get_atr1(union_var_idx));
            }
            else
              err("incompatible types in assignment");
          }
        else
          err("Union '%s' doesn't have variable with name %s", get_union_name(union_idx) ,$3);
      }
      if(get_kind($5) == UNION_VAR)
        gen_mov(lookup_symbol(variable_name, VAR|PAR), union_idx);
      else
        gen_mov($5, union_idx);
    }
```

```
exp
  : literal
  | _ID⋯
  | function_call⋯
  | _LPAREN num_exp _RPAREN⋯
  | _ID _DOT _ID
    {
      int union_idx = lookup_symbol($1, VAR|PAR);
      if(union_idx == NO_INDEX)
        err("variable '%s' undeclared", $1);
      else {
        variable_name = $1;
        $$ = lookup_symbol_union_kind($3, UNION_VAR, get_union_name(union_idx));
        if($$ == NO_INDEX)
          err("Union '%s' doesn't have variable with name %s", get_union_name(union_idx) ,$3);
        else {
          if (get_kind(union_idx) == VAR) {
            int union_var_idx = get_atr1($$);
            if(union_var_idx != get_active_variable(union_idx))
              err("Union member '%s' is not active.", $3);
          }
```

Pri definisanju varijable radi se provera da li je navedeni tip unije definisan i da li već varijabla sa istim imenom postoji.
Pri korišćenju unija proverava se da li varijabla ili parametar postoje. Zatim, da li navedeni atribut postoji, da li je “aktivan” i da li je adekvatnog tipa u trenutnoj situaciji.
Tip varijable/parametra je unija dok je pri pozivanju nekog od atributa pojmu dodeljen indeks definicije atributa unije koji se poziva kako bi znali kojeg tipa pojam treba biti.
Samim tim što je definition poziv atributa unije u *exp* pojmu, možemo ga koristiti i u izrazu za vraćanje vrednosti i možemo proslediti vrednost kao argument funkciji.

Za izraz vraćanja vrednosti iz funkcije ovako je urađena implementacija:

```
return_statement
  : _RETURN num_exp _SEMICOLON
    {
      if(cast_active) {
        if(get_type(fun_idx) != cast_to_type)
          err("incompatible types in return. Value is casted to wrong type");
        cast_active = 0;
        cast_to_type = NO_TYPE;
      }
      else
        if(get_type(fun_idx) != get_type($2))
          err("incompatible types in return");
      if(get_kind($2) == UNION_VAR) {
        gen_mov(lookup_symbol(variable_name, VAR|PAR), FUN_REG);
      }
      else
        if(get_type($2) == UNION) {
          if(strcmp(get_union_name(fun_idx), get_union_name($2)) == 0)
            gen_mov($2, FUN_REG);
          else
            err("incompatible union types in return.");
        }
        else
          gen_mov($2, FUN_REG);
        code("\n\t\tJMP \t@%s_exit", get_name(fun_idx));
    }
```

Implementacija za definisanje unije kao povratna vrednost funkcije izgleda ovako:

```
function
  : _TYPE _ID…
  | _UNION _ID _ID
    {
      fun_idx = lookup_symbol($3, FUN);
      if (lookup_symbol($2, UNION_K) != NO_INDEX) {
        if(fun_idx == NO_INDEX)
          fun_idx = insert_symbol_union($3, FUN, UNION, NO_ATR, NO_ATR, $2);
        else
          err("redefinition of function '%s'", $3);
      }
      else
        err("No union definition with name %s", $2);
      code("\n%s:", $3);
      code("\n\t\tPUSH\t%%14");
      code("\n\t\tMOV \t%%15,%%14");
    }
    _LPAREN parameter _RPAREN body
    {
      clear_symbols(fun_idx + 1);
      var_num = 0;
      code("\n@%s_exit:", $3);
      code("\n\t\tMOV \t%%14,%%15");
      code("\n\t\tPOP \t%%14");
      code("\n\t\tRET");
```

Na prvoj slici iznad se vidi da se radi provera da li je unija koja se vraća validne definicije. U slučaju da se vraća neki od atributa unije radi se provera da li su adekvatnog tipa.

Takođe bitno za spomenuti, tabela simbola je proširena za dve kolone:

1) Ime unije
2) "Aktivni" atribut unije

Za cast sintaksa i semantika izgleda ovako:

```
cast_exp
  : exp

  | _LPAREN _TYPE _RPAREN exp %prec CAST
    {
      cast_active = 1;
      cast_to_type = $2;
      $$ = $4;
    }
  ;
```

Sa slika iznad svuda se mogu videti if-ovi koji gledaju da li je izraz cast-ovan i da li treba da se gleda tip u koji je pojam castovan pri poređenju tipova.

Za generisanje koda implementirano je generisanje koda za unije. U suštini kod oponaša uniju ali drži sve u jednom registru, što nam i odgovara jer postoje samo dva primitivna tipa podataka, int i unsigned, gde oba zauzimaju istu količinu prostora u memoriji.

Ideje za nastavak

Što se unija tiče, funkcionišu kao što je očekivano. Međutim, nemaju nekog preteranog smisla s obzirom da u miniC postoje samo int i unsigned primitivni tipovi. Jedino mesto gde sam našao da ne radi intuitivno jeste kad se instanciranoj uniji dodeli unija koja se vraća iz poziva funkcije. U tom slučaju neće se moći koristiti neki od njenih atributa zbog provere u semantici za trenutnu aktivnu vrednost ako taj atribut nije poslednji korišćen od strane instacirane unije.

Cast je sopstvena priča i sama sintaksa C-a za type casting je dosta konfuzna i dvosmislena u nekim situacijama. Zbog opširnosti mogućnosti kod cast-a, odlučio sam se da samo implementiram cast pri dodeli jednog elementa ili vrednosti, cast povratne vrednosti i cast argumenta funkcije. U daljem radu definitvno bih uložio trud da omogućim cast i pri generisanju koda.

Za unapređenje ovog projekta u nekom značajnom smeru bila bi potrebna potpuna reimplementacija određenih delova miniC jezika, npr. tabele simbola. Mislim da bi unije imale benefit od postojanja drugih primitivnih i ne primitivnih tipova, samim tim predlažem u budućem radu implementaciju podrške za stringove, klase i sl.

Literatura

GCC GNU implementacija C kompajlera - ideja za cast sintaksu