# DOKUMENTACIJA

Programski prevodioci - predmetni zadatak

**Osnovni podaci**

| Broj indeksa | Ime i prezime | Tema |
|---|---|---|
| SW16/2019 | Miladin Momčilović | Nizovi i pokazivači |

**Korišćeni alati**

| Naziv | Verzija |
|---|---|
| Flex, Bison, Hipsim, GCC Compiler | |

## Evidencija implementiranog dela

Niz kao povratna vrednost funkcija – sintaksa i semantika
Inicijalizacija niza pri deklaraciji – sintaksa, semantika i generisanje koda
Indeksiranje nizova – sintaksa, semantika i generisanje koda
Dodela vrednosti nizu na određenom indeksu – sintaksa, semantika i generisanje koda
Deklaracija pokazivača – sintaksa i semantika
Dodela vrednosti pokazivaču – sintaksa i semantika
Pokazivač kao povratna vrednost funkcija – semantika i sintaksa

## Detalji implementacije

Detaljnije objasniti na koji način su implementirani zadaci. Delovi koda -može, ukoliko ima smisla. Slike - može (prikaz izgenerisanog stabla, izgled izmenjene tabele simbola,... ). Obavezno numerisati tabele, slike, grafike i referencirati ih pravilno u dokumentaciji.
Navesti koji test fajlovi su relevantni.

- Implementirani su celobrojni nizovi koje je moguće (ali nije potrebno) inicijalizovati literalima u deklaraciji. Veličina niza je statička i ona se određuje pri deklaraciji u okviru [] ili na osnovu broja elemenata prilikom inicijalizacije. Elementima niza može se pristupiti preko indexa korišćenjem literala, takođe na isti način se može raditi dodela vrednosti ovim nizovima. Vrednost koja se može dodeliti nizu može biti bilo koji numerički iskaz.
- Implementirani su i pokazivači, podržani su samo pokazivači na INT i UINT i odrađene su samo semantika i sintaksa. Moguća je dodela vrednosti pokazivaču pomoću '&' i pristup vrednosti pomoću ' * '.

Osnovna ideja za realizaciju nizova bila je da se ostavi mesta na steku, gde će se smeštati elementi kada se radi generisanje. U tabeli simbola čuvam samo referencu na početak niza, kao što je prikazano na *Slika 1*.

```
| _TYPE _ID array_size _SEMICOLON // Definiton of array
    {
      if(lookup_symbol($2, VAR|PAR|ARR|PTR) == NO_INDEX)
      {
          insert_symbol($2, ARR, $1, ++var_num, $3);
          code("\n\t\tSUBS\t %%15,$%d,%%15", 4 * $3);
      }
      else
         err("redefinition of '%s'", $2);
    }
| _TYPE _ID _LSBRACKET _RSBRACKET _ASSIGN _LBRACKET elements_list _RBRACKET _SEMICOLON
    {
      if(lookup_symbol($2, VAR|PAR|ARR|PTR) == NO_INDEX)
      {
          int idx = insert_symbol($2, ARR, $1, ++var_num, literal_list_count);
          code("\n\t\tSUBS\t %%15,$%d,%%15", 4 * literal_list_count);
          for (int i =0; i < literal_list_count; i++) {
              gen_mov_array(array_literals[i], i, idx);
          }
      }
      else
         err("redefinition of '%s'", $2);
    }
;
```

*Slika 1*

```
elements_list
  : literal
  {
    array_literals[literal_list_count] = $1;
    literal_list_count += 1;
  }
  | elements_list _COLON literal
  {
    array_literals[literal_list_count] = $3;
    literal_list_count += 1;
  }
  ;
```

*Slika 2*

Takođe, bilo je neophodno proširiti *exp*(*Slika 4*) i *num_exp*(*Slika 5*) čija vrednost nije više integer, već struktura koja sadrži dva integera, jedan integer je isti kao i pre i pokazuje na indeks, dok drugi sadrži vrednost offset-a za elemente niza.(*Slika 3*)

```
typedef struct exp_vals {
    int index;
    int value;
} exp_vals;
```

*Slika 3*

```
num_exp
  : exp
  | num_exp _AROP exp
      {
        if (!((get_type($1->index) == INT_PTR && get_type($3->index) == INT) || (get_type($1->index) == INT
        && get_type($3->index) == INT_PTR))) {
            if (!((get_type($1->index) == UINT_PTR && get_type($3->index) == UINT) || (get_type($1->index)
            == UINT && get_type($3->index) == UINT_PTR))) {
                if(get_type($1->index) != get_type($3->index))
                    err("invalid operands: arithmetic operation");
            }
        }
        int t1 = get_type($1->index);
        code("\n\t\t%s\t", ar_instructions[$2 + (t1 - 1) * AROP_NUMBER]);

        if (get_kind($1->index) == ARR)
            gen_sym_name_array_elem($1->index, $1->value);
        else
            gen_sym_name($1->index);
        code(",");
        if (get_kind($3->index) == ARR)
            gen_sym_name_array_elem($3->index, $3->value);
        else
            gen_sym_name($3->index);
        code(",");
        free_if_reg($3->index);
        free_if_reg($1->index);

        struct exp_vals *value = (struct exp_vals*) malloc(sizeof(struct exp_vals));
        value->index = take_reg();
        value->value = -1;
        $$ = value;
        if (get_kind($$->index) == ARR)
            gen_sym_name_array_elem($$->index, -1);
        else
            gen_sym_name($$->index);
        set_type($$->index, t1);
      }
  ;
```

*Slika 4*

```
exp
  : literal
      {
      struct exp_vals *value = (struct exp_vals*) malloc(sizeof(struct exp_vals));
      value->index = $1;
      value->value = -1;
      $$ = value;
      }
  | _ID array_size
      {
        int head = lookup_symbol($1, ARR);
        if (head == NO_INDEX)
            err("'%s' undeclared", $1);
        if ($2 >= get_atr2(head))
            err("'%s' index out of range", $1);
        struct exp_vals *value = (struct exp_vals*) malloc(sizeof(struct exp_vals));
        value->index = head;
        value->value = $2;
        $$ = value;
      }
  | _ID
      {
        int elem = lookup_symbol($1, VAR|PAR|ARR);
        if(elem == NO_INDEX)
          err("'%s' undeclared", $1);
        struct exp_vals *value = (struct exp_vals*) malloc(sizeof(struct exp_vals));
        value->index = elem;
        value->value = -1;
        $$ = value;
      }
  | function_call
      {
        int elem = take_reg();
        struct exp_vals *value = (struct exp_vals*) malloc(sizeof(struct exp_vals));
        value->index = elem;
        value->value = -1;
        $$ = value;
        gen_mov(FUN_REG, elem);
      }
  | _LPAREN num_exp _RPAREN
      {
        struct exp_vals *value = (struct exp_vals*) malloc(sizeof(struct exp_vals));
        value->index = $2->index;
        value->value = -1;
        $$ = value;
      }
  | _POINTER _ID
  {
        int idx = lookup_symbol($2, PTR);
        if (idx == NO_INDEX)
          err("'%s' undeclared", $2);
        struct exp_vals *value = (struct exp_vals*) malloc(sizeof(struct exp_vals));
        value->index = idx;
        value->value = -1;
        $$ = value;
  }
  ;
```

*Slika 5*

Dodate su i funkcije za generisanje koda specifično za nizove *gen_sym_name_array_elem*(*Slika 6*), *gen_mov_array*(*Slika 7*) i proširena je *gen_cmp*(*Slika 8*), da podrži poređenje kod nizova, zbog drugačije funkcije za generisanje *sym_name*-a.

Generisanje imena za nizove je slično kao i za obične varijable samo je dodat i offset pomnožen sa 4.

```
void gen_sym_name_array_elem(int index, int offset) {
  if(index > -1) {
    if (get_kind(index) == ARR) // -n*4(%14)
      code("-%d(%%14)", get_atr1(index) * 4 + (offset + 1) * 4);
    else if(get_kind(index) == VAR) // -n*4(%14)
      code("-%d(%%14)", get_atr1(index) * 4);
    else if(get_kind(index) == PAR) // m*4(%14)
      code("%d(%%14)", 4 + get_atr1(index) *4);
    else if(get_kind(index) == LIT)
      code("$%s", get_name(index));
    else //function, reg
      code("%s", get_name(index));
  }
}
```

*Slika 6*

```
void gen_mov_array(int input_index, int offset, int output_index) {
  code("\n\t\tMOV \t");
  gen_sym_name_array_elem(input_index, offset);
  code(",");
  gen_sym_name_array_elem(output_index, offset);

  //ako se smešta u registar, treba preneti tip
  if(output_index >= 0 && output_index <= LAST_WORKING_REG)
    set_type(output_index, get_type(input_index));
  free_if_reg(input_index);

}
```

*Slika 7*

```
void gen_cmp(struct exp_vals *op1_index, struct exp_vals *op2_index) {
  if(get_type(op1_index->index) == INT)
    code("\n\t\tCMPS \t");
  else
    code("\n\t\tCMPU \t");
  if(get_kind(op1_index->index) == ARR)
  {
      gen_sym_name_array_elem(op1_index->index, op1_index->value);
      code(",");
      gen_sym_name_array_elem(op2_index->index, op2_index->value);
  }
  else
  {
      gen_sym_name(op1_index->index);
      code(",");
      gen_sym_name(op2_index->index);
  }
  free_if_reg(op2_index->index);
  free_if_reg(op1_index->index);
}
```

*Slika 8*

Svi relevantni testovi nalaze se u korenskom direktorijumu projekta u folderima *tests_array, tests_pointers, tests_for*.

Primeri korišćenja:

- **Niz**
  - Deklaracija niza – *int niz[5];*
  - Inicijalizacija niza pri deklaraciji – *int niz[] = {1,2,3,4,5};*
  - Pristup elementima niza – *a = niz[1];*
  - Dodela vrednosti elementu niza – *niz[1] = 5;*
  - Element niza kao povratan vrednost – *return niz[1];*
  - Niz kao povratna vrednost – *int* foo() {… return niz; }*

- **Pokazivač**
  - Inicijalizacija pokazivača – *int* p;*
  - Dodela vrednosti pokazivaču – *p = &x;*
  - Preuzimanje vrednosti pokazivača – *c = *p;*
  - Vrednost pokazivača kao povratna vrednost - *return *p;*
  - Pokazivač kao povratna vrednost - *int* foo() {… return p; }*

# Ideje za nastavak

Pokušao sam da realizujem pristup elementima niza preko bilo kog numeričkog iskaza, nisam uspeo. Razlog je što se sama vrednost ne čuva nigde u okviru tabele simbola, nego na nivou generisanja koda, pa je problem izvršiti proveru da li je indeks van opsega niza. Ovo bi bila jedan od ideja za nastavak. Takođe omogućiti funkcionalnosti tipa Min, Max, Avg, Sum za nizove. Za pokazivače odraditi deo generisanja koda i omogućiti iteraciju kroz niz upotrebom pokazivača.

# Literatura

Predavanja/vežbe
https://stackoverflow.com
https://www.w3schools.com/