ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ
Εργασία 2
2022-2023
Σημασιολογική Ανάλυση και Παραγωγή Κώδικα

Μισαηλίδης Σάββας (ics21166)

Απαραίτητες προσθήκες

Λεκτικός αναλυτής

```
%{
        #include <stdlib.h>
        #include <string.h>
        int line = 1;
        %}

        DIGITS [0-9]
        LETTER [a-zA-Z]
        VARIABLE {LETTER}({LETTER}|{DIGITS}|_)*
        FLOAT {DIGITS}+\.{DIGITS}+
        newline \n|\x0A|\x0D\x0A
        %%

        "start" {return T_start;}
        "end" {return T_end;}
        "forall" {return T_forall;}
        "print" {return T_print;}
        "in" {return T_in;}

        "int" {yylval.tokentype=type_integer;return T_type;}
        "float" {yylval.tokentype=type_real;return T_type;}

        "*" {return '*';}
        "+" {return '+';}
        "(" {return '(';}
        ")" {return ')';}
        "[" {return '[';}
        "]" {return ']';}
        ".." {return T_float;}

        {DIGITS}+ {yylval.lexical=strdup(yytext);return T_int;}
        {FLOAT} {yylval.lexical=strdup(yytext);return T_float;}
        {VARIABLE} {yylval.lexical=strdup(yytext);return T_id;}

        {newline}  { line++;}
        [ \t] { /* nothing */ }
        . {
        printf("Lexical Analysis: Unexpected String! :: %s. in
line %d. \n",yytext,yylineno); }

%%
```

Συντακτικός αναλυτής

Δήλωση του union

```
%union {
char *lexical;
ParType tokentype;
struct {
ParType type;
char * place;} se;
RelationType relopIndex;
struct {
NUMBER_LIST_TYPE trueLbl;
NUMBER_LIST_TYPE falseLbl;
} condLabels;
```

Δηλώσεις των token

```
%token <tokentype> T_type

%token <lexical> T_int
%token <lexical> T_float
%token <lexical> T_id

%token T_start "start"
%token T_end "end"
%token T_forall "forall"
%token T_print "print"
%token T_in "in"

%token T_float ".."
%token '('
%token ')'
%token '['
%token ']'

%left '+'
%left '*'
```

Δηλώσεις μη-τερματικών συμβόλων

```
%type<se> term
%type<se> expr
%type<se> t_expr
%type<se> ut_expr
%type<se> non_par_expr
%type<se> par_expr
%type<se> pexpr
%type<se> operation
```

Καθορισμός Γραμματικής

(1) program

```
program:
"start" T_id
{
create_preample($2);
symbolTable=NULL;
}
stmts "end"
{
insertINSTRUCTION("return");
insertINSTRUCTION(".end method\n");
}
;
```

(2) stmts

```
stmts: /*empty*/
| stmts stmt
;
```

(3) stmt

```
stmt:
'(' cmd ')'
;
```

(4) command

```
command:
T_print pexpr
| assignments
;
```

(5) assignments

```
assignments:
T_id expr
{
addvar(&symbolTable,$1,$2.type);
insertSTORE($2.type, lookup_position(symbolTable,$1));
}
| T_id '[' T_forall T_id T_in T_int T_float T_int ']'
{

int start = atoi($6);
int end = atoi($8);
int size = end - start + 1;
pushInteger(size);

insertINSTRUCTION("newarray int");
addvar(&symbolTable, $1, type_int_array);
insertSTORE(type_int_array, lookup_position(symbolTable, $1));
```

```
pushInteger(0);
addvar(&symbolTable, $4, type_integer);
insertSTORE(type_integer, lookup_position(symbolTable, $4));
insertGOTO(2);

insertLabel(currentLabel());
Label();

insertLOAD(type_int_array, lookup_position(symbolTable, $1));
insertLOAD(type_integer, lookup_position(symbolTable, $4));

insertLOAD(type_integer, lookup_position(symbolTable, $4));
pushInteger(start);
insertOPERATION(type_integer, "add");
insertASTORE_ARRAY_ELEM(type_integer);
insertIINC(lookup_position(symbolTable, $4), 1);

insertLabel(currentLabel());

insertLOAD(type_integer, lookup_position(symbolTable, $4));
pushInteger(size-1);
insertINSTRUCTION("if_icmple #_1");
}
;
```

(6) print_expr

```
print_expr:
expr
{
insertINSTRUCTION("getstatic java/lang/System/out
Ljava/io/PrintStream;");
insertINSTRUCTION("swap");
insertINVOKEVITRUAL("java/io/PrintStream/println",$1.type,type_
void) ;
}
;
```

(7) expr

```
expr:
parenthesis_expr
| non_parenthesis_expr
;
```

(8) parenthesis_expr

```
parenthesis_expr:
'(' non_type_expr ')' { $$ = $2; }
| '(' type_expr')' { $$ = $2; }
;
```

(9) non_parenthesis_expr

```
non_parenthesis_expr:
non_type_expr
| type_expr
;
```

(10) type_expr

```
type_expr:
T_type expr
{
$$.type = $1;
if ($$.type != $2.type) {
if ($$.type == type_integer)
insertINSTRUCTION("f2i");
else
insertINSTRUCTION("i2f");
}
}
;
```

(11) non_type_expr

```
non_type_expr:
term
| operation
;
```

(12) operation

```
operation:
expr expr '+'
{
$$.type = typeDefinition($1.type, $2.type);
insertOPERATION($$.type, "add");
}
| expr expr '*'
{
$$.type = typeDefinition($1.type, $2.type);
insertOPERATION($$.type, "mul");}
;
```

(13) term

```
term:
T_id
{
if (!($$.type = lookup_type(symbolTable,$1))) {
ERR_VAR_MISSING($1,line);
}
$$.place = $1;
insertLOAD($$.type,lookup_position(symbolTable,$1));
}
| T_int
{
$$.place = $1;
$$.type = type_integer;
pushInteger(atoi($1));
}
| T_float
{
$$.place = $1;
$$.type = type_real;
insertLDC($1);
}
| T_id '[' T_int ']'
{
if (!(lookup_type(symbolTable,$1)==type_int_array)) {
ERR_VAR_MISSING($1, line);
}
$$.type = type_integer;
insertLOAD(type_int_array,lookup_position(symbolTable,$1));
pushInteger(atoi($3));
insertALOAD_ARRAY_ELEM(type_integer);
}
;
```

Παραδείγματα

ΠΑΡΑΔΕΙΓΜΑ	ΑΠΑΙΤΟΥΜΕΝΑ

`start simple1`
`  (print (3 4 +))`
`  end`
	Χρειάστηκε να δημιουργήσω το program, statements και statement έτσι ώστε να υλοποιηθεί το παράδειγμα (1)
	
	Αρχικα, το program, έπρεπε να αναγνωρίζει το start και το end ενός προγράμματος. Έπειτα, μέσω των statements, να τρέξει τα command τα οποία υπάρχουν στο αντίστοιχο παράδειγμα. Τέλος, θα πρέπει να αναγνωρίζει τους αριθμούς ή τις μεταβλητές καθώς και τα operations που πρέπει να γίνουν για να βγει το σωστό αποτέλεσμα.

`start simple2`
`  (print (3 4 + 7 *))`
`  (print (3.0 4.0 + 7.0 *))`
`  end`
	Στο συγκεκριμένο παράδειγμα, η επιπλέον προσθήκη στην γραμματική, είναι να αναγνωρίζει πολλαπλές πράξεις που μπορει να γίνουν μεταξύ των αριθμών και έξτρα η αναγνώριση δεκαδικών. Ουσιαστικά, ένας επιπλέον κανόνας ώστε να επιτραπτεί η αριστερή αναδρομή και ένας για την αναγνώρισει του νέου τύπου float.
	(stmts: stmts stmt) και (term: T_float).
	
	+ πράξη πολλαπλασιασμού (operation: expr expr *)

`start simple3`
`  (x 1)`
`  (print x)`
`  (x (3 4 +))`
`  (print x)`
`  (y (10))`
`  (print y)`
`  (y (10 x +))`
`  (print y)`
`  end`
	Στο παράδειγμα (3), έπρεπε να αναγνωρίζει ότι εκτός από σταθερές τιμές, χρειάστηκε να καταλάβει ότι υπάρχουν και δηλώσεις μεταβλητών. Τέλος, χρειάστηκε και η ανάθεση τιμής. (term: T_id + κώδικας για καταχώρηση τυπού και μεταβλητής) και στο command προστέθηκε ο κανόνας (command: assignments) όπου γίνεται η ανάθεση τιμής.

`start simple4`
`  (y (10))`
`  (x (3 y +))`
`  (print x)`
`  end`
	Δεν χρείαστηκε κάποια αλλαγή.

`start simple5`
`  (x (3 (int 4.0) +))`
`  (print x)`
`  end`
	Στο παράδειγμα (5), πρόσθεσα το typecasting(σε integer) και την δήλωση τύπων στο αναλυτή μου. Πιο συγκεκριμένα, προστέθηκαν τα type_expr και non_type_expr και άλλαξαν τα non_parenthesis_expr και parenthesis_expr, ώστε να υποστηρίζουν τις εκφράσεις (με και χωρίς τύπους).

start simple6 (x (1)) (x (int (3.0 4.0 +))) (print x) end	**Γίνονται σύνθετες πράξεις και στην συνέχεια ολόκληρη η πράξη να γίνει σε type casting. Άρα χρειάστηκε να προσθεθεί από μία πράξη να γίνει μετατροπή σε ακέραιο.**
start simple7 (x 1.0) (x (float 3 4 +)) (print x) (y 1) (y (int 1 2 +)) (print y) end	**Επιπλέον προσθήκη είναι ή μετατροπή type casting σε από ακέραιο σε float (i2f).** **Η αλλαγή έγινε στο type_expr όπου προστέθηκε επίσης ένας έξτρα έλεγχος για την μετατροπή (ένα else case).**
start simple8 (y [forall i in 3..4]) (print y[0]) (print y[1]) end	**Στο συγκεκριμένο παράδειγμα, χρειάστηκε η εισαγωγή του forall, όπου στα assignments επρεπέ να προστεθεί η περίπτωση του forall, όπου έπρεπε να δηλωθεί η διαδικασία ορισμού πίνακα, μεγέθους, εισαγωγή δεδομένων.** **Προσθήκες (term: T_id [T_int]) και (assignments : T_id '[' T_forall T_id T_in T_int T_float T_int ']')**
start simple9 (arr [forall i in 8..20]) (x (100 arr[0] +)) (print x) (y (100 arr[1] *)) (print y) end	**Δεν χρείαστηκε κάποια αλλαγή.**

Σημειώσεις

Για την υλοποιήση, χρειάστηκε να προστεθούν κανόνες που αναγνωρίζουν αν υπάρχουν παρενθέσεις ή όχι, καθώς και expressions τα οποία είτε είναι print ή οτιδήποτε σε πράξη. Για την επίτευξη των παραπάνω, χρειάστηκαν τα μη-τερματικά parenthesis_expr, non_parenthesis_expr και ένας κανόνας με το μη-τερματικό print_expr για την εκτύπωση των αποτελεσμάτων.