Αναφορά Εργαστηριακής Άσκησης - Αρχές Γλωσσών Προγραμματισμού και Μεταγλωττιστών

Παναγιώτης Μπακλέσης (Α.Μ. 6136), Παναγιώτης Νομικός (Α.Μ. 6154)

Σεπτέμβριος 2017

Αρχικά, οι δηλώσεις μας ακολουθούν τις εντολές τις γλώσσας και μεταφράζουν την EBNF σε έκρφαση γλώσσας προγραμματισμού. Η αρχική μας έκφραση είναι το "programm" το οποίο καλεί τον εαυτό του και ξεκινάει από εκεί η όλη διαδικασία της μετάφρασης. Αυτό που επιτρέπει το "programma" στον κανόνα του είναι μια "statement" δηλαδή μια δήλωση οποιοδήποτε είδους. Σε εκείνο τον κανόνα είναι που έχουμε προβλέψει το πιάσιμο τον λαθών που μπορεί να προκύψουν ώστε αν κάτι δεν αναγνωρίζεται ως μια δηλωτική μονάδα αυτό σημαίνει ότι δεν μπορεί να ταιριάξει με κανέναν κανόνα του προγράμματος οπότε ενεργοποιείται και ο κανόνας του λάθους ο οποίος μετράει τα λάθη και παρουσιάζει την γραμμή του λάθους χρησιμοποιώντας την εσωτερική μεταβλητή.

Για την δήλωση μεταβλητών καλεί έμμεσα αναδρομικά τον εαυτό της ώστε να διαβάσει όλες τις μεταβλητές που μπορεί να δηλωθούν στην ίδια γραμμή. Ο τύπος δεδομένων μπορεί να είναι απλός ή δείκτης. Μια μεταβλητή εδώ μπορεί να είναι μια απλή μεταβλητή ή μια δήλωση πίνακα.Στο τέλος κλείνουμε με το σύμβολο semicolon ; .

Μετά έχουμε την δήλωση συνάρτησης όπου αναφέρουμε τον τύπο επιστροφής, το id της και την λίστα παραμέτρων της αν έχει, τέλος έχουμε το σύμβολο semicolon ; . Οι παράμετροι της λίστας αποτελούν λίστες μεταβλητών και δεικτών. Έπειτα έχουμε τον ορισμό της συνάρτησης όπου τώρα έχουμε το τύπο επιστρεφόμενης τιμής, το id , την λίστα παραμέτρων της, αν έχει, και μετά έχουμε το σώμα της συνάρτησης.
Αρχικά ορίζουμε τις δηλώσεις, αν υπάρχουν, και μετά τις προτάσεις, αν υπάρχουν. Η πρόταση μπορεί να περιλαμβάνει διάφορα τμήματα κώδικα όπως: το ελληνικό ερωτηματικό που τερματίζει τις εντολές (semicolon), την δομή if , τις δεσμευμένες λέξεις break , continue, return, for και κάποιες άλλες αναδρομικές μορφές με χρήση παρενθέσεων. Οι εκφράσεις χρησιμοποιούνται για τις δεσμευμένες λέξεις NEW, DELETE, αναδρομικές μορφές έμμεσες και άμεσες με χρήση παρενθέσεων για προτεραιότητα.

Τέλος, ξεκινάει η διαδικασία αποτίμησης μια έκφρασης καταχώρησης. Στην έκφραση καταχώρησης χρησιμοποιούμε επίπεδα τα οποία θα πρέπει να προσπελάσουμε για να βεβαιωθούμε ότι ικανοποιούνται οι προτεραιότητες. Άρα κάθε έκφραση ξεκινά να ελέγχει την υψηλότερη σε βαθμό έκφραση: αν δεν ικανοποιηθεί, τότε κοιτάει στο αμέσως επόμενο επίπεδο προτεραιότητας, μέχρι να φτάσουμε

στις αρχικές εκφράσεις, όπου εκεί κλείνει η έκφρασή μας, δηλαδή με το όνομα που γίνεται η καταχώρηση ή αν δηλώθηκε τώρα η μεταβλητή τελειώνουμε με τον τύπο της μεταβλητής.

Τώρα, για να αναγνωριστεί αν οι μεταβλητές έχουν δηλωθεί πριν χρησιμοποιηθούν, χρησιμοποιούμε ένα πίνακα τύπου char * 1000 θέσεων(universal) όπου και καταχωρούμε τις μεταβλητές και συναρτήσεις που ορίζονται. Σε κάθε αναφορά μεταβλητής ή συνάρτησης χρησιμοποιούμε μια συνάρτηση η οποία τσεκάρει τον πίνακα αν υπάρχει αυτή η μεταβλητή: αν ναι, είναι εντάξει. Σε διαφορετική περίπτωση, βγαίνει ένα μήνυμα λάθους που λέει ότι αυτή η μεταβλητή δεν είναι δηλωμένη.

Ακολουθεί ο κώδικας των αρχείων flex, bison.

```
bison
%{
#include <string.h>
#include <cstdio>
#include <iostream>
#include <stdio.h>

//#define YYDEBUG 1

using namespace std;


extern "C" int yylex();
extern "C" int yyparse();
extern "C" FILE *yyin;
extern FILE *yyin;
extern int yylineno;
char* declared[1000];
int   index1=0;

void yyerror(const char *s);
void isitdeclared(char* s) {
        for(int i=0;i<index1;i++){
                if(!strcmp(s,declared[i]) || !strcmp(s,"main")){
                        return ;
                }
        }
cout<<"Oops. Its seems that you haven't declared that variable .\n"<< s <<endl;
}

%}

%locations
```

```
%union {
        int     val ;
        int     ival;
        double dval;
        char  *sval;
        char  *cval;
        char  *idval;
}

%start programm
%token COLON
%token BOOLEAN
%token INTEGER
%token DOUBLES
%token CHARACTER
%token BYREF
%token <ival> INT;
%token <dval> DOUBLE;
%token <sval> STRING;
%token  BOOLV;
%token <cval> CHAR;
%token <idval> ID;
%token VOID
%token IF
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE
%token FOR
%token CONTINUE
%token BREAK
%token RETURN
%token SEMICOLON
%token LPARE
%token RPARE
%token QMARK
%token NEW
%token DELETE
%token NULLS
%token LBRACKET
%token RBRACKET
%token LBRACE
%token RBRACE
%right  COMMA
%left '=' MINUSAS PLUSAS MODAS DIVAS MULAS
%left '<' '>' EQ NE LE GE LA LO
```

```
%left '+' '-'
%left '*' '/' '%'
%left '!' '&'
%left PLUSPLUS MINUSMINUS
%%
programm:
        statement | programm statement
        ;
statement:
        define_function
        | function_statement
        | variable_statement
        | error  { /*cout<<"Hello , we have an error\n"<<endl;*/ yyerror;}
        ;
variable_statement:
        variable_statement_tail SEMICOLON
        ;

variable_statement_tail:
         variable_statement_tail COMMA stating
        | data_types stating
        ;
data_types:
        basic_data_types | data_types '*'
        ;

basic_data_types:
        INTEGER
        | DOUBLES
        | CHARACTER
        | BOOLEAN
        | VOID
        ;

stating:
        ID { declared[index1++]=$1; }
        |ID LBRACKET const_expression RBRACKET {  declared[index1++]=$1; }
        ;
function_statement:
        data_types ID LPARE RPARE SEMICOLON {  declared[index1++]=$2; }
        | data_types ID LPARE parameter_list RPARE SEMICOLON {
declared[index1++]=$2; }
        ;
parameter_list:
        parameter | parameter_list COMMA parameter
        ;
```

```
parameter:
        data_types ID { declared[index1++]=$2; }
        | BYREF data_types ID { declared[index1++]=$3; }
        ;
define_function:
         data_types ID LPARE RPARE LBRACE body1 body2 RBRACE
{ isitdeclared($2); }
        | data_types ID LPARE parameter_list RPARE LBRACE body1 body2 RBRACE { i
        ;
body1:
         %empty
        | body1 statement
        ;
body2:
        %empty
        | body2 sentence
        ;
sentence:
        SEMICOLON
        | expression SEMICOLON
        | LBRACE RBRACE
        | LBRACE newsentence RBRACE
        | IF LPARE expression RPARE sentence %prec LOWER_THAN_ELSE
        | IF LPARE expression RPARE sentence ELSE sentence
        | FOR LPARE forarguments SEMICOLON forarguments SEMICOLON forarguments R
        | ID COLON FOR LPARE forarguments SEMICOLON forarguments SEMICOLON forarg
        | CONTINUE SEMICOLON
        | CONTINUE ID SEMICOLON { isitdeclared($2); }
        | BREAK SEMICOLON
        | BREAK  ID SEMICOLON { isitdeclared($2); }
        | RETURN SEMICOLON
        | RETURN expression SEMICOLON
        ;
newsentence:
        sentence | newsentence sentence
        ;
forarguments:
         %empty
        | expression
        ;
expression:
         expression LBRACKET expression RBRACKET
        | assignment_expression
        | NEW data_types
        | NEW data_types LBRACKET expression RBRACKET
```

```
        | DELETE expression
        | LPARE expression_list RPARE
        | ID LPARE expression_list RPARE { isitdeclared($1); }
        ;

primary_expression:
        ID { isitdeclared($1); }
        | LPARE expression RPARE
        | BOOLV
        | NULLS
        | INT
        | CHAR
        | DOUBLE
        | STRING
;
postfix_expression
        : primary_expression
        | postfix_expression PLUSPLUS
        | postfix_expression MINUSMINUS
        //| postfix_expression LPARE RPARE
        | postfix_expression LPARE parameter_list RPARE
        //| postfix_expression LBRACKET RBRACKET
;
unary_expression
        : postfix_expression
        | PLUSPLUS unary_expression
        | MINUSMINUS unary_expression
        | unary_operator unary_expression
unary_operator
        : '&'
        | '*'
        | '+'
        | '-'
        | '~'
        | '!'
        ;
multiplicative_expression
        : unary_expression
        | multiplicative_expression '*' unary_expression
        | multiplicative_expression '/' unary_expression
        | multiplicative_expression '%' unary_expression
        ;
additive_expression
        : multiplicative_expression
        | additive_expression '+' multiplicative_expression
        | additive_expression '-' multiplicative_expression
```

```
        ;

relational_expression
        : additive_expression
        | relational_expression '<' additive_expression
        | relational_expression '>' additive_expression
        | relational_expression LE additive_expression
        | relational_expression GE additive_expression
        ;
equality_expression
        : relational_expression
        | equality_expression EQ relational_expression
        | equality_expression NE relational_expression
        ;
logical_and_expression
        : equality_expression
        | logical_and_expression LA equality_expression
        ;
logical_or_expression
        : logical_and_expression
        | logical_or_expression LO logical_and_expression
        ;
conditional_expression
        : logical_or_expression
        | logical_or_expression QMARK expression COLON expression
        ;
assignment_expression:
        conditional_expression
        | unary_expression assignment_operator assignment_expression
        ;
assigngment_operator
        : '='
        | MULAS
        | DIVAS
        | MODAS
        | PLUSAS
        | MINUSAS
;
expression_list:
        expression | expression_list COMMA expression | %empty
        ;
const_expression:
        expression
        ;
%%
```

```
int main(int argc, char **argv){
        // open a file handle to a particular file:
        FILE *myfile = fopen(argv[1], "r");
        int status;
        char input;
        //make sure it's valid:
        if(!myfile) {
                cout << "I can't open file!" << endl;
                return -1;
        } else {
                do
                        {
                                status = fscanf(myfile, "%c", &input);


                               printf("%c ", input);

                }while(status != -1 );
        }
        rewind(myfile);
        // set flex to read from it instead of defaulting to STDIN:
        yyin = myfile;
        //yydebug = 3;
        // parse through the input until there is no more
        do {
                yyparse();
        } while(!feof(yyin));

}
void yyerror(const char *s) {
        static int counter   = 0;
        static int prerrline = 0;


        if(prerrline!=yylineno){
                counter++;
                prerrline = yylineno;
        }
        cout << "\nError in line: "<< yylineno << "Total number of errors: "<< c
        // might as well halt now:

}
```

```
flex
%{
#include <iostream>
#include <stdio.h>
using namespace std;
#define YY_DECL extern "C" int yylex()
//#define YYDEBUG 1

#include "bisonfile.tab.h"
int count = 0;
%}

%option yylineno

%%
":"             { return COLON; }
","             { return COMMA; }
"int"           { return INTEGER; }
"char"          { return CHARACTER; }
"bool"          { return BOOLEAN; }
"double"        { return DOUBLES; }
"true"          { return BOOLV;   }
"false"         { return BOOLV;   }
"void"          { return VOID; }
"if"            { return IF;   }
"else"          { return ELSE; }
"for"           { return FOR; }
"continue"      { return CONTINUE; }
"break"         { return BREAK; }
"return"        { return RETURN; }
";"             { return SEMICOLON; }
"("             { return LPARE; }
")"             { return RPARE; }
"?"             { return QMARK; }
"new"           { return NEW; }
"delete"        { return DELETE; }
"NULL"          { return NULLS; }
"["             { return LBRACKET; }
"]"             { return RBRACKET; }
"{"             { return LBRACE; }
"}"             { return RBRACE; }
"&"             { return '&' ; }
"*"             { return '*' ; }
"+"             { return '+' ; }
"-"             { return '-' ; }
"!"             { return '!' ; }
```

```
"/"             { return '/' ; }
"%"             { return '%' ; }
"<"             { return '<' ; }
">"             { return '>' ; }
"<="            { return LE  ; }
">="            { return GE  ; }
"=="            { return EQ  ; }
"!="            { return NE  ; }
"&&"            { return LA  ; }
"||"            { return LO  ; }
"++"            { return PLUSPLUS; }
"--"            { return MINUSMINUS; }
"="             { return '=' ; }
"*="            { return MULAS ; }
"/="            { return DIVAS ; }
"%="            { return MODAS ; }
"+="            { return PLUSAS ; }
"-+"            { return MINUSAS ; }
\'[a-zA-Z]\'            { yylval.cval  =  strdup(yytext); return CHAR;
}
[0-9]+\.[0-9]           { yylval.dval  =  atof(yytext);
return DOUBLE; }
[0-9]+                  { yylval.ival  =  atoi(yytext);
return INT   ;  }
\"[a-zA-Z0-9]+\"        { yylval.sval  =  strdup(yytext); return STRING; }
[a-zA-Z][a-zA-Z_0-9]*   { yylval.idval =  strdup(yytext); return ID;
}
[ \t\n]           ;
"//".*   ;
[/][*][^*]*[*]+([^*/][^*]*[*]+)*[/]
%%
```