Федеральное государственное автономное образовательное учреждение высшего образования "Национальный Исследовательский Университет ИТМО"
Мегафакультет Компьютерных Технологий и Управления
Факультет Программной Инженерии и Компьютерной Техники

Вариант №5

Задание

по дисциплине

'Разработка компиляторов'

Выполнил Студент группы P33102
Лапин Алексей Александрович
Преподаватель:
Лаздин Артур Вячеславович

г. Санкт-Петербург
2024г.

Задание

Разработать язык программирования, который должен реализовать следующие компоненты:

1. Присваивание (оператор или операция), арифметические и логические операции.
2. Ветвление, включая вариант факультативного else.
3. Цикл while.
4. Поддержка целочисленного и логического типа данных.
5. Многострочные комментарии в стиле Си-подобных языков.

Что добавит баллы:

1. Вместо ветвление if [then] else конструируется оператор if elif [elif]+ else
2. Вместо цикла while (или в дополнение к нему) конструируется цикл for.

1 Ход работы

Сделал лексический анализатор грамматики с помощью библиотеки FLEX.

```
%{
#include "ast.h"
#include <stdio.h>
#include "parser.tab.h"
%}

%option noyywrap

%x COMMENT

%%

[0-9][a-zA-Z][a-zA-Z0-9]* { yyerror("Names cannot start with a
   digit");}
[0-9]+          { yylval.i = atoi(yytext); return NUMBER; }


    /* tokens for arithmetic and logical operators. */

    /* single character ops */
"+" |
"-" |
"*" |
"/" |
"=" |
"," |
";" |
"(" |
")" |
"{" |
```

```
"}"             { return yytext[0]; }

    /* comparation ops, all are a CMP token */
">"                 { yylval.subtok = NT_GT; return CMP; }
"<"                 { yylval.subtok = NT_LT; return CMP; }
"!="                { yylval.subtok = NT_NEQ; return CMP; }
"=="                { yylval.subtok = NT_EQ; return CMP; }
">="                { yylval.subtok = NT_GTE; return CMP; }
"<="                { yylval.subtok = NT_LTE; return CMP; }

    /* keywords */
"if"                { return IF;     }
"else"              { return ELSE;   }
"while"             { return WHILE;  }
"for"               { return FOR;    }
"elif"              { return ELIF;   }

"&&"                { return AND;   }
"||"                { return OR;    }
"and"               { return AND;   }
"or"                { return OR;    }
"not"               { return NOT;   }
"!"                 { return NOT;   }


    /* booleans */
"true"              { yylval.i = 1; return NUMBER; }
"false"             { yylval.i = 0; return NUMBER;  }

     /* comments */

"/*"                { BEGIN(COMMENT); }
<COMMENT>"*/"       { BEGIN(INITIAL); }
<COMMENT>.          { /* consume anything inside comments */ }
<COMMENT>\n         { /* consume newlines in comments */ }
<COMMENT><<EOF>>    { yyerror("Unterminated comment"); }

"//".*              /* single line comment */

    /* names */
[a-zA-Z][a-zA-Z0-9]*    { yylval.s = strdup(yytext); return NAME; }


[ \t]+                  { /* ignore whitespace */ }
\\\n                    {printf("c> "); }  /* ignore line
   continuation */
\n                      {  }

.                       { printf("Unexpected character: %c\n",
   yytext[0]); }

%%
```

Сделал синтаксический анализатор грамматики с помощью библиотеки BISON.

```
%{
#include "ast.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
void yyerror(char *s, ...);
int yylex(void);
%}

%union {
    struct ast *a;
    int i;
    char *s; /* which symbol */
    ntype_t subtok;
}

/* declare tokens */
%token <i> NUMBER
%token <s> NAME
%token EOL YYEOF
%token IF ELSE WHILE FOR ELIF

%right '='
%left OR
%left AND
%left NOT
%left <subtok> CMP
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS
%type <a> exp list flow if_stmt elif_stmt else_stmt

%start prog
%%
prog: list YYEOF                        {
                                            if($1 != NULL) {
                                                print_ast(stdout,
                                                    $1, 0);
                                                FILE* fl =
                                                    fopen("out.S",
                                                    "w");
                                                FILE* fl2 =
                                                    fopen("tree",
                                                    "w");
                                                if(fl == NULL) {
                                                    printf("Error
                                                        opening
                                                        file\n");
                                                    exit(1);
                                                }
                                                if(fl2 == NULL) {
                                                    printf("Error
                                                        opening
```

```
                                                        file\n");
46                                                          exit(1);
47                                                      }
48                                                      print_asm(fl, $1);
49                                                      print_ast(fl2, $1,
                                                           0);
50                                                      fclose(fl);
51                                                      treefree($1);
52                                                  }
53                                                  printf("> ");
54                                              }
55      | error YYEOF                       { yyerrok; printf("> ");}
56      ;
57
58  flow: if_stmt
59      | WHILE exp '{' list '}'                            { $$ =
           newflow(NT_WHILE, $2, $4, NULL);  }
60      | FOR '(' exp ';' exp ';' exp ')' '{' list '}'      { $$ =
           newfor($3, $5, $7, $10); }
61      ;
62
63  if_stmt: IF exp '{' list '}' else_stmt                  { $$ =
       newflow(NT_IF, $2, $4, $6); }
64      | IF exp '{' list '}' elif_stmt                     { $$ =
           newflow(NT_IF, $2, $4, $6); }
65      ;
66  elif_stmt: ELIF exp '{' list '}' else_stmt              { $$ =
       newflow(NT_IF, $2, $4, $6); }
67      | ELIF exp '{' list '}' elif_stmt                   { $$ =
           newflow(NT_IF, $2, $4, $6); }
68      ;
69  else_stmt:  /* nothing */                               { $$ = NULL; }
70      | ELSE '{' list '}'                                 { $$ = $3; }
71      ;
72  list: /* nothing */                    { $$ = NULL; }
73      | exp ';' list                     {
74                                             if($3 == NULL) $$ = $1;
75                                             else $$ =
                                                  newast(NT_LIST, $1,
                                                  $3);
76                                         }
77      | flow list                        {
78                                             if($2 == NULL) $$ = $1;
79                                             else $$ =
                                                  newast(NT_LIST, $1,
                                                  $2);
80                                         }
81      ;
82
83  exp:  exp CMP exp                      { $$ = newcmp($2, $1, $3);
            }
84      | exp '+' exp                      { $$ = newast(NT_ADD, $1,
           $3);     }
```

```
85      | exp '-' exp                          { $$ = newast(NT_SUB, $1,
            $3);      }
86      | exp '*' exp                          { $$ = newast(NT_MUL, $1,
            $3);      }
87      | exp '/' exp                          { $$ = newast(NT_DIV, $1,
            $3);      }
88      | '(' exp ')'                          { $$ = $2;
                                    }
89      | '-' exp  %prec UMINUS                { $$ = newast(NT_UMINUS, $2,
            NULL);    }
90      | exp AND exp                          { $$ = newast(NT_AND, $1,
            $3);    }
91      | exp OR exp                           { $$ = newast(NT_OR, $1,
            $3);     }
92      | NOT exp                              { $$ = newast(NT_NOT, $2,
            NULL); }
93      | NUMBER                               { $$ = newnum($1);
                        }
94      | NAME                                 { $$ = newref($1);
                        }
95      | NAME '=' exp                         { $$ = newasgn($1, $3);
                    }
96      ;
97  %%
```

2 Проверка соответсвия требованиям

2.1 Присваивание (оператор или операция)

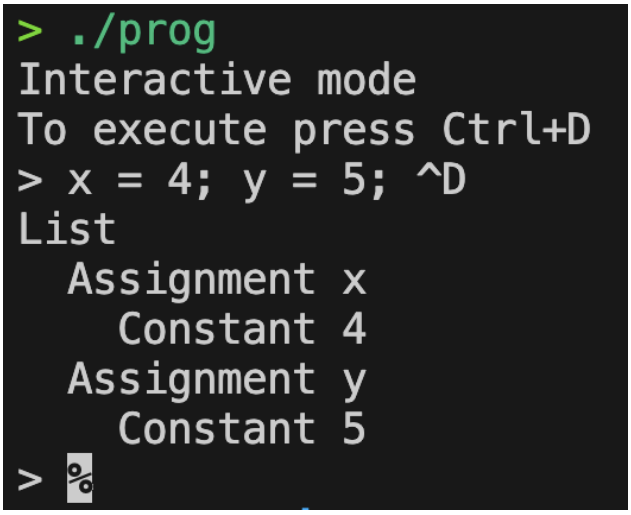


Рис. 1: Присваивание

out.S:

```
jal x1, MAIN
x:
data 0 * 1
```

```
    y:
    data 0 * 1
    MAIN:
    li x1, 4
    sw x0, 1, x1
    li x1, 5
    sw x0, 2, x1
    ebreak
```

2.2 Арифметические и логические операции. Ветвление, включая вариант факультативного elif.Поддержка целочисленного и логического типа данных.Многострочные комментарии в стиле Си-подобных языков.

Программа:

```
x = 4;
y = false;
z = 0;
if (y) {
    z = 1;
}
/* Some Comment
 Multi-line
 Should be ignored
*/
elif (x == 4) {
    z = 2;
}
else {
    z = 3;
}
```

tree:

```
List
Assignment x
    Constant 4
List
    Assignment y
    Constant 0
    List
    Assignment z
        Constant 0
    If
        Symbol y
        Assignment z
        Constant 1
        If
        Comparison ==
            Symbol x
            Constant 4
```

```
        Assignment z
            Constant 2
        Assignment z
            Constant 3
```

out.S:

```
jal x1, MAIN
x:
data 0 * 1
y:
data 0 * 1
z:
data 0 * 1
MAIN:
li x1, 4
sw x0, 1, x1
li x1, 0
sw x0, 2, x1
li x1, 0
sw x0, 3, x1
lw x2, x0, 2
beq x2, x0, ELSE0
li x1, 1
sw x0, 3, x1
jal x1, ENDIF0
ELSE0:
lw x2, x0, 1
li x3, 4
seq x2, x2, x3
beq x2, x0, ELSE1
li x1, 2
sw x0, 3, x1
jal x1, ENDIF1
ELSE1:
li x1, 3
sw x0, 3, x1
ENDIF1:
ENDIF0:
ebreak
```

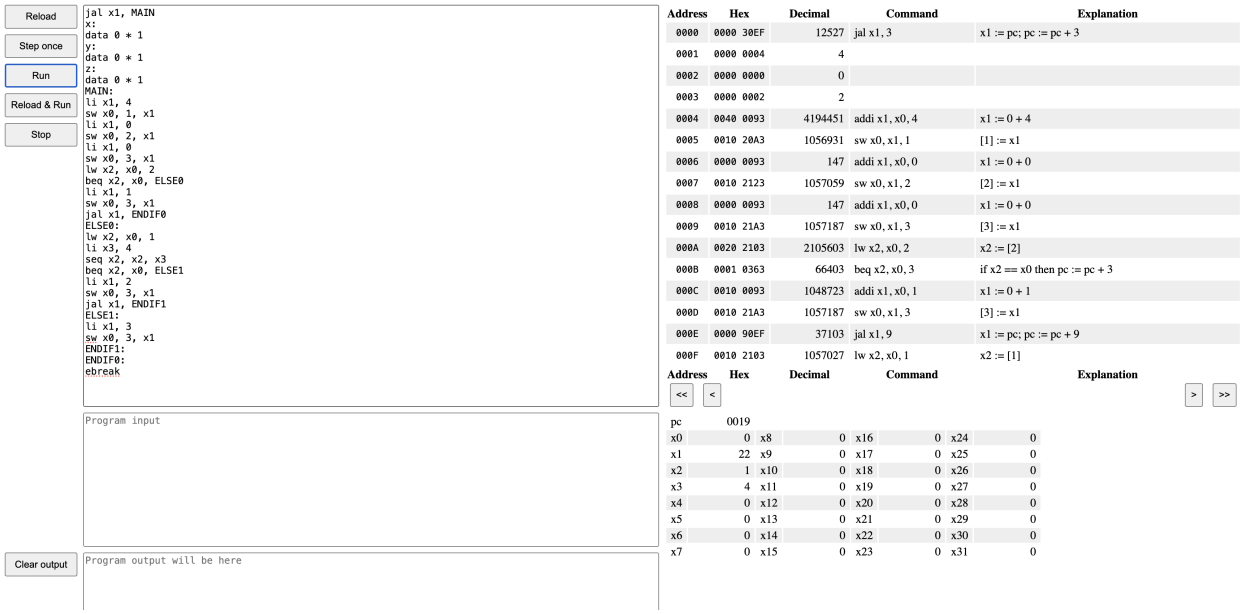


Рис. 2: Программа

2.3 Оператор For

Программа:

```
x = 4;
y = 32;
for (i = 1; i < 5; i = i + 1) {
    x = x * i;
    y = y / 2;
}
```

tree:

```
List
Assignment x
    Constant 4
List
    Assignment y
    Constant 32
    For
    Assignment i
        Constant 1
    Comparison <
        Symbol i
        Constant 5
    Assignment i
        Operator +
        Symbol i
        Constant 1
    List
        Assignment x
        Operator *
            Symbol x
```

```
            Symbol i
        Assignment y
        Operator /
            Symbol y
            Constant 2
```

out.S:

```
jal x1, MAIN
i:
data 0 * 1
x:
data 0 * 1
y:
data 0 * 1
MAIN:
li x1, 4
sw x0, 2, x1
li x1, 32
sw x0, 3, x1
li x1, 1
sw x0, 1, x1
WHILE0:
lw x2, x0, 1
li x3, 5
slt x2, x2, x3
beq x2, x0, ENDWHILE0
lw x1, x0, 2
lw x2, x0, 1
mul x1, x1, x2
sw x0, 2, x1
lw x1, x0, 3
li x2, 2
div x1, x1, x2
sw x0, 3, x1
lw x1, x0, 1
li x2, 1
add x1, x1, x2
sw x0, 1, x1
jal x1, WHILE0
ENDWHILE0:
ebreak
```

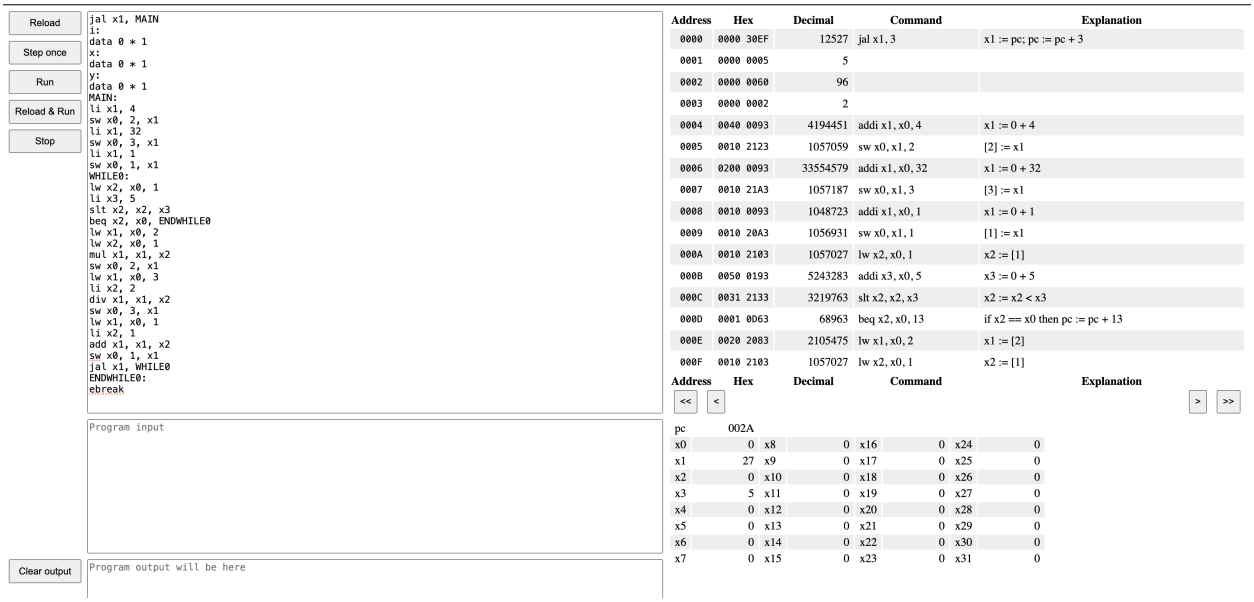
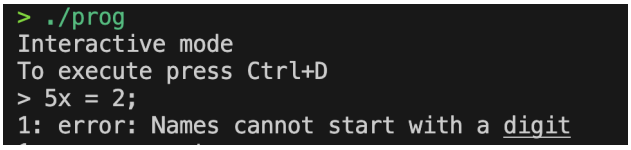
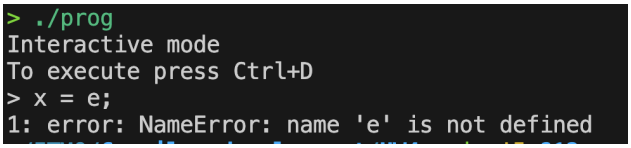
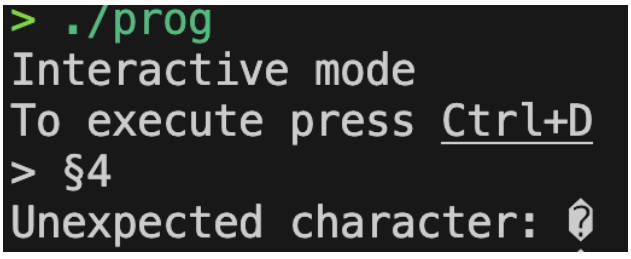


Рис. 3: Программа

3 Обработка ошибок







prog:

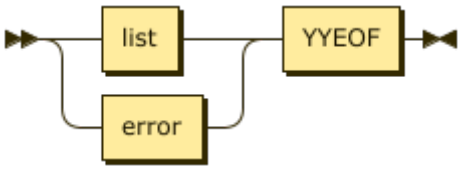


Figure 1: prog

flow:

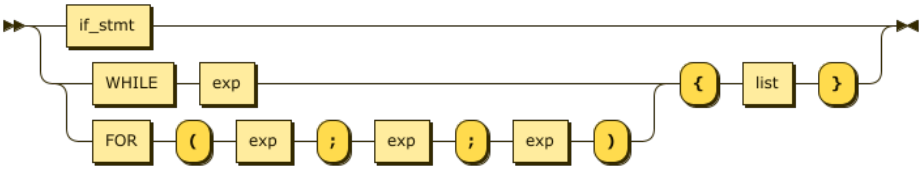


Figure 2: flow

referenced by:

- list

if_stmt:

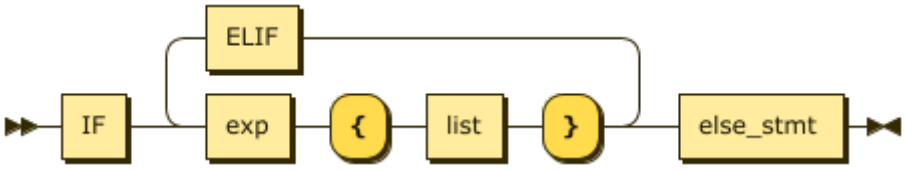


Figure 3: if_stmt

referenced by:

- flow

else_stmt:

referenced by:

- if_stmt

list:

referenced by:

- else_stmt
- flow

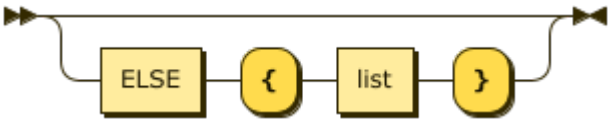


Figure 4: else_stmt

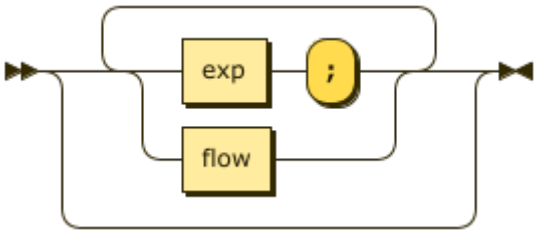


Figure 5: list

- if_stmt
- prog

exp:

referenced by:

- exp
- flow
- if_stmt
- list

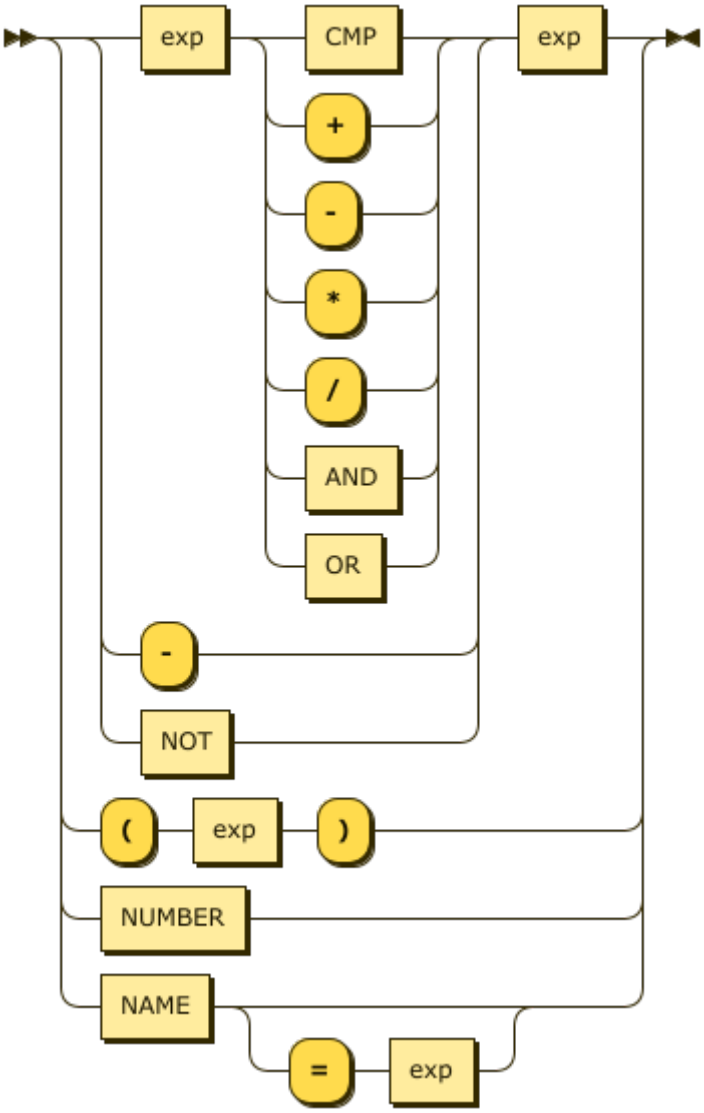


Figure 6: exp