Практикум на ЭВМ. Интерпретатор. GOTO

Баев А.Ж.

Казахстанский филиал МГУ

31 марта 2021

Интерпретатор

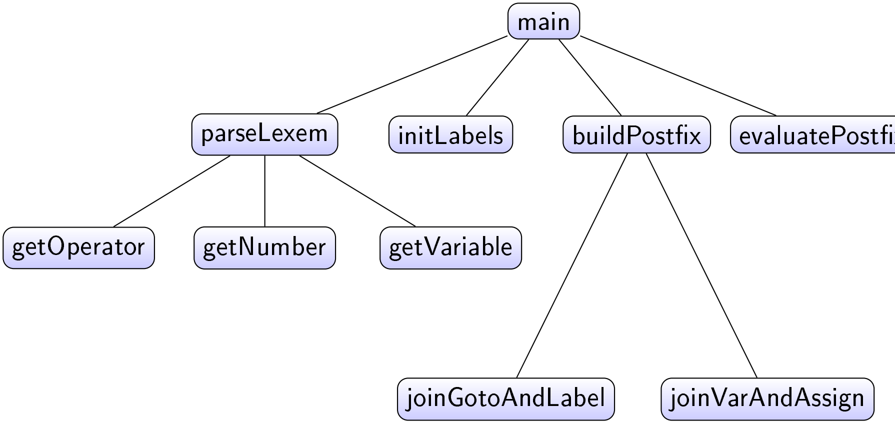
1. Арифметические операторы
2. Оператор присваивания
3. Логические операторы
4. **Оператор перехода (goto)**
5. Условный оператор
6. Цикл while
7. Массивы
8. Функции
9. Рекурсия (стек для вызова функций)

Текстовое представление

```
enum OPERATORTYPE {
  GOTO, ASSIGN, COLON,
  LBRACKET, RBRACKET,
  OR,
  AND,
  BITOR,
  XOR,
  BITAND,
  EQ, NEQ,
  SHL, SHR,
  LEQ, LT, GEQ, GT,
  PLUS, MINUS,
  MULT, DIV, MOD
};
```

```
std::string OPERTEXT[] = {
  "goto", ":=", ":",
  "(", ")",
  "or",
  "and",
  "|",
  "^",
  "&",
  "==", "!=",
  "<<", ">>",
  "<=", "<", ">=", ">",
  "+", "-",
  "*", "/", "%"
};
```

Схема


main
parseLexem
initLabels
buildPostfix
evaluatePostfi
getOperator
getNumber
getVariable
joinGotoAndLabel
joinVarAndAssign

Реализация main

```cpp
int main() {
  std::string codeline;
  std::vector< std::vector<Lexem *> > infixLines,
                                      postfixLines;

  while (std::getline(std::cin, codeline))
    infixLines.push_back(parseLexem(codeline));

  for (int row = 0; row < (int)infixLines.size(); ++r
    initLabels(infixLines[row], row);

  for (const auto &infix: infixLines)
    postfixLines.push_back(buildPostfix(infix));

  int row = 0;
  while (0 <= row && row < (int)postfixLines.size())
    row = evaluatePostfix(postfixLines[row], row);
  return 0;
}
```

Реализация parseLexem

```
std::vector<Lexem *> parseLexem(const std::string &co
  std::vector<Lexem *> infix;
  Lexem *ptr;
  for (int i = 0; i < (int)codeline.size(); ) {
    if (ptr = getOperator(codeline, i)) {
      infix.push_back(ptr);
      continue;
    }
    if (ptr = getNumber(codeline, i)) {
      infix.push_back(ptr);
      continue;
    }
    if (ptr = getVariable(codeline, i)) {
      infix.push_back(ptr);
      continue;
    }
    i++;
  }
  return infix;
```

Реализация initLabels

```
void initLabels(std::vector<Lexem *> &infix, int row)
  for (int i = 1; i < (int)infix.size(); i++) {
    if (infix[i - 1]->type() == VARIABLE &&
        infix[i]->type() == OPERATOR)
    {
      Variable *lexemvar = (Variable *)infix[i-1];
      Operator *lexemop = (Operator *)infix[i];
      if (lexemop->operType() == COLON) {
        labels[lexemvar->getName()] = row;
        delete infix[i - 1];
        delete infix[i];
        infix[i-1] = nullptr;
        infix[i] = nullptr;
        i++;
      }
    }
  }
}
```

Реализация buildPostfix (goto)

```cpp
std::vector<Lexem *> buildPostfix(
  const std::vector<Lexem *> &infix)
{
  std::vector<Lexem *> postfix;
  std::stack<Operator *> stack;
  for (const auto &lexem: infix) {
    if (lexem == nullptr)
      continue;
    if (lexem->type() == VARIABLE) {
      Variable *lexemvar = (Variable *) lexem;
      if (lexemvar->inLabelTable())
        joinGotoAndLabel(lexemvar, stack);
      else
        postfix.push_back(lexem);
    }
        ...
  }
    ...
  return postfix;
```

Реализация joinGotoAndLabel

```
void joinGotoAndLabel(Variable *lexemvar,
                      std::stack<Operator *> &stack)
{
  if (stack.top()->operType() == GOTO) {
    Goto *lexemgoto = (Goto *)stack.top();
    lexemgoto->setRow(lexemvar->getName());
  }
}
```

Реализация buildPostfix (assign)

```
std::vector<Lexem *> buildPostfix(
  const std::vector<Lexem *> &infix)
{
  std::vector<Lexem *> postfix;
  std::stack<Operator *> stack;
  for (const auto &lexem: infix) {
    ...
    if (lexem->type() == OPERATOR) {
      Operator *lexemoper = (Operator *)lexem;
      if (lexemoper->operType() == ASSIGN)
        joinVarAndAssign((Assign *)lexemoper, postfix
      ...
  }
  ...
  return postfix;
}
```

Реализация joinVarAndAssign

```
void joinVarAndAssign(Assign *lexemassign,
                      std::vector<Lexem *> &postfix)
  Lexem *previous = *postfix.rbegin();
  if (previous -> type() == VARIABLE) {
    Variable *var = (Variable *)previous;
    lexemassign->setVarName(var->getName());
    postfix.pop_back();
    delete var;
  }
}
```

Реализация evaluatePoliz

```
int evaluatePostfix(const std::vector<Lexem *> &postfi
                    int row) {
  std::stack<int> stack;
  for (const auto &lexem: postfix) {
    if (lexem->type() == OPERATOR) {
      Operator *lexemop = (Operator *)lexem;
      if (lexemop->operType() == GOTO) {
        Goto *lexemgoto = (Goto *)lexemop;
        return lexemgoto->getRow();
      } else if (lexemop->operType() == ASSIGN) {
        Assign *lexemassign = (Assign *)lexemop;
        int rvalue = stack.top();
        stack.pop();
        stack.push(lexemassign->evaluate(rvalue));
      } else
        ...
  }
  return row + 1;
}
```

Пример parseLexem

string

```
x := 1
y := x + 2
z := 3 * 4 + 5
goto L
x := 2
L: x := 3
```

infix

```
0: [x] [<>:=] [1]
1: [y] [<>:=] [x] [+] [2]
2: [z] [<>:=] [3] [*] [4] [+] [5]
3: [<row -2147483647>goto] [L]
4: [x] [<>:=] [2]
5: [L] [:] [x] [<>:=] [3]
```

Пример initLabels

infix

```
0:  [x]  [<>:=]  [1]
1:  [y]  [<>:=]  [x]  [+]  [2]
2:  [z]  [<>:=]  [3]  [*]  [4]  [+]  [5]
3:  [<row -2147483647>goto]  [L]
4:  [x]  [<>:=]  [2]
5:  [L]  [:]  [x]  [<>:=]  [3]
```

labels

```
L=5
```

Пример buildPoliz

infix

```
0:  [x]  [<>:=]  [1]
1:  [y]  [<>:=]  [x]  [+]  [2]
2:  [z]  [<>:=]  [3]  [*]  [4]  [+]  [5]
3:  [<row -2147483647>goto]  [L]
4:  [x]  [<>:=]  [2]
5:  [L]  [:]  [x]  [<>:=]  [3]
```

postfix

```
0:  [1]  [<x>:=]
1:  [x]  [2]  [+]  [<y>:=]
2:  [3]  [4]  [*]  [5]  [+]  [<z>:=]
3:  [<row 5>goto]
4:  [2]  [<x>:=]
5:  [3]  [<x>:=]
```

Пример evaluatePoliz

result

```
0:  [1]  [<x>:=]
variables:  x=1

1:  [x]  [2]  [+]  [<y>:=]
variables:  x=1 | y=3

2:  [3]  [4]  [*]  [5]  [+]  [<z>:=]
variables:  x=1 | y=3 | z=17

3:  [<row 5>goto]
variables:  x=1 | y=3 | z=17

5:  [3]  [<x>:=]
variables:  x=3 | y=3 | z=17
```