Практикум на ЭВМ. Интерпретатор. IF WHILE

Баев А.Ж.

Казахстанский филиал МГУ

16 февраля 2019

Интерпретатор

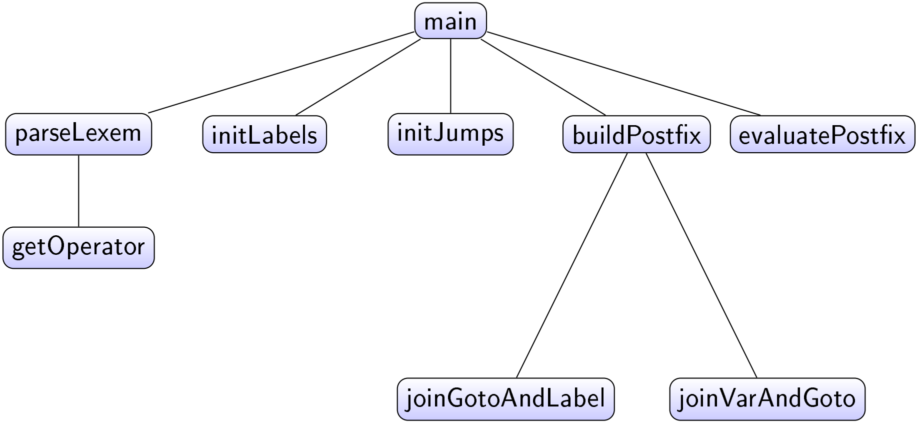
1. Арифметические операторы
2. Оператор присваивания
3. Логические операторы
4. Оператор перехода (goto)
5. **Условный оператор**
6. **Цикл while**
7. Массивы
8. Функции
9. Рекурсия (стек для вызова функций)

Текстовое представление

```
enum OPERATORTYPE {
  IF, THEN,
  ELSE, ENDIF,
  WHILE, ENDWHILE,
  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[] = {
  "if", "then",
  "else", "endif",
  "while", "endwhile",
  "goto", ":=", ":",
  "(", ")",
  "or",
  "and",
  "|",
  "^",
  "&",
  "==", "!=",
  "<<", ">>",
  "<=", "<", ">=", ">",
  "+", "-",
  "*", "/", "%"
};
```

Схема


main
parseLexem
initLabels
initJumps
buildPostfix
evaluatePostfix
getOperator
joinGotoAndLabel
joinVarAndGoto

Реализация main

```
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(); ++row)
    initLabels(infixLines[row], row);

  initJumps(infixLines);

  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 класса Goto

```
class Goto: public Operator {
    int row;
public:
    enum { UNDEFINED = -INT32_MAX };
    Goto(OPERATORTYPE optype): Operator(optype) {
        row = UNDEFINED;
    }
    void setRow(int row) {
        Goto::row = row;
    }
    void setRow(const std::string &labelname) {
        row = labels[labelname];
    }
    int getRow() {
        return row;
    }
    void print() {
        std::cout << "[<row␣" << row << ">"
                  << OPERTEXT[optype] << "]␣";
```

Реализация getOperator

```
Operator *getOperator(const std::string &codeline,
                      int &i) {
  ...
    if (op == GOTO || op == IF || op == ELSE ||
        op == WHILE || op == ENDWHILE)
      return new Goto(op);
    return new Operator(op);
  ...
  return nullptr;
}
```

Реализация initJumps для if-then-else-endif

```
std::stack<Goto *> stackIfElse;
for (int row = 0; row < (int)infixes.size(); row++) {
  for (int i = 0; i < (int)infix.size(); i++)
    if (infixLines[row][i]->type() == OPERATOR) {
      Operator *lexemoper = (Operator *)infixes[row][i];

      if (lexemoper->operType() == IF)
        stackIfElse.push((Goto *)lexemoper);

      if (lexemoper->operType() == ELSE) {
        stackIfElse.top()->setRow(row + 1);
        stackIfElse.pop();
        stackIfElse.push((Goto *)lexemoper);
      }

      if (lexemoper->operType() == ENDIF) {
        stackIfElse.top()->setRow(row + 1);
        stackIfElse.pop();
      }
```

Реализация initJumps для while-then-endwhile

```
std::stack<Goto *> stackWhile;
for (int row = 0; row < (int)infixes.size(); row++) {
  for (int i = 0; i < (int)infix.size(); i++)
    if (infixLines[row][i]->type() == OPERATOR) {
      Operator *lexemoper = (Operator *)infixes[row][i];

      if (lexemoper->operType() == WHILE) {
        Goto *lexemgoto = (Goto *)lexemoper;
        lexemgoto->setRow(row);
        stackWhile.push(lexemgoto);
      }

      if (lexemoper->operType() == ENDWHILE) {
        Goto *lexemgoto = (Goto *)lexemoper;
        lexemgoto->setRow(stackWhile.top()->getRow());
        stackWhile.top()->setRow(row + 1);
        stackWhile.pop();
      }
```

Реализация buildPostfix (endif)

```
    ...
    for (auto lexem: infix) {
        if (lexem->type() == OPERATOR) {
            Operator *lexemoper = (Operator *)lexem;
            if (lexemoper->operType() == ENDIF)
                continue;
            ...
        }
        ...
    }
    ...
```

Реализация evaluatePoliz (goto, else, endwhile)

```
int evaluatePostfix(const std::vector<Lexem *> &postfix,
                    int row) {
  std::stack<int> stack;
  for (const auto &lexem: postfix) {
    if (lexem->type() == OPERATOR) {
      Operator *lexemop = (Operator *)lexem;

      if (lexemop->operType() == GOTO ||
          lexemop->operType() == ELSE ||
          lexemop->operType() == ENDWHILE) {
        Goto *lexemgoto = (Goto *)lexemop;
        return lexemgoto->getRow();
      }

      ...
  }
  return row + 1;
}
```

Реализация evaluatePoliz (if, while)

```
int evaluatePostfix(const std::vector<Lexem *> &postfix,
                    int row) {
  std::stack<int> stack;
  for (const auto &lexem: postfix) {
    if (lexem->type() == OPERATOR) {
      Operator *lexemop = (Operator *)lexem;

      if (lexemop->operType() == IF ||
          lexemop->operType() == WHILE) {
        Goto *lexemgoto = (Goto *)lexemop;
        int rvalue = stack.top();
        stack.pop();
        if (!rvalue)
          return lexemgoto->getRow();
      }
      ...
  }
  return row + 1;
}
```

Пример parseLexem

string

```
i := 1
s := 0
while i < 4 then
    s := s + i
    i := i + 1
endwhile
i := 0
```

infix

```
0: [i] [<>:=] [1]
1: [s] [<>:=] [0]
2: [<row -2147483647>while] [i] [<] [4] [then]
3: [s] [<>:=] [s] [+] [i]
4: [i] [<>:=] [i] [+] [1]
5: [<row -2147483647>endwhile]
6: [i] [<>:=] [0]
```

Пример initJumps

infix

```
0:  [i]  [<>:=]  [1]
1:  [s]  [<>:=]  [0]
2:  [<row -2147483647>while]  [i]  [<]  [4]  [then]
3:  [s]  [<>:=]  [s]  [+]  [i]
4:  [i]  [<>:=]  [i]  [+]  [1]
5:  [<row -2147483647>endwhile]
6:  [i]  [<>:=]  [0]
```

infix

```
0:  [i]  [<>:=]  [1]
1:  [s]  [<>:=]  [0]
2:  [<row 6>while]  [i]  [<]  [4]  [then]
3:  [s]  [<>:=]  [s]  [+]  [i]
4:  [i]  [<>:=]  [i]  [+]  [1]
5:  [<row 2>endwhile]
6:  [i]  [<>:=]  [0]
```

Пример buildPoliz

infix

```
0: [i] [<>:=] [1]
1: [s] [<>:=] [0]
2: [<row 6>while] [i] [<] [4] [then]
3: [s] [<>:=] [s] [+] [i]
4: [i] [<>:=] [i] [+] [1]
5: [<row 2>endwhile]
6: [i] [<>:=] [0]
```

postfix

```
0: [1] [<i>:=]
1: [0] [<s>:=]
2: [i] [4] [<] [<row 6>while]
3: [s] [i] [+] [<s>:=]
4: [i] [1] [+] [<i>:=]
5: [<row 2>endwhile]
6: [0] [<i>:=]
```

Пример evaluatePoliz

```
0:  [1]  [<i>:=]
1:  [0]  [<s>:=]
2:  [i]  [5]  [<]  [<row 6>while]
3:  [s]  [i]  [+]  [<s>:=]
4:  [i]  [1]  [+]  [<i>:=]
5:  [<row 2>endwhile]
2:  [i]  [5]  [<]  [<row 6>while]
3:  [s]  [i]  [+]  [<s>:=]
4:  [i]  [1]  [+]  [<i>:=]
5:  [<row 2>endwhile]
2:  [i]  [5]  [<]  [<row 6>while]
3:  [s]  [i]  [+]  [<s>:=]
4:  [i]  [1]  [+]  [<i>:=]
5:  [<row 2>endwhile]
2:  [i]  [5]  [<]  [<row 6>while]
6:  [0]  [<i>:=]
```

```
i=1
i=1  s=0
i=1  s=0
i=1  s=1
i=2  s=1
i=2  s=1
i=2  s=1
i=2  s=3
i=3  s=3
i=3  s=3
i=3  s=3
i=3  s=6
i=4  s=6
i=4  s=6
i=4  s=6
i=0  s=6
```