Практикум на ЭВМ. Интерпретатор. GOTO

Баев А.Ж.

Казахстанский филиал МГУ

24 марта 2022

Интерпретатор

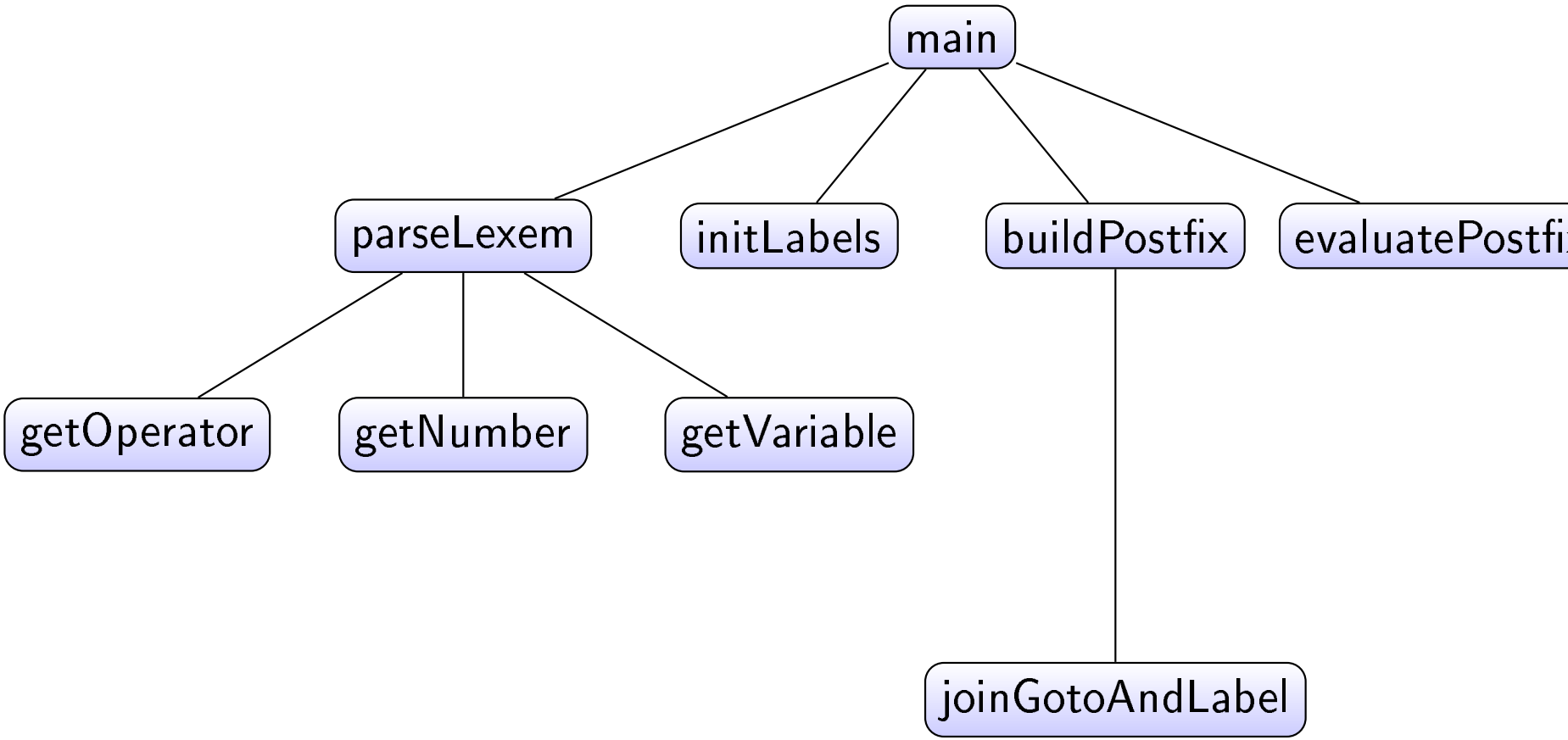
1. Арифметические операторы
2. Оператор присваивания
3. Логические операторы
4. **Оператор перехода (goto)**
5. Условный оператор
6. Цикл while
7. Массивы
8. Функции
9. Рекурсия (стек для вызова функций)

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

Реализация main

```
1 std::map<string, int> LabelTable;
```

Реализация main

```
 1  int main() {
 2    std::string codeline;
 3    std::vector< std::vector<Lexem *> > infixLines,
 4                                        postfixLines;
 5
 6    while (std::getline(std::cin, codeline))
 7      infixLines.push_back(parseLexem(codeline));
 8
 9    for (int row = 0; row < infixLines.size(); ++row)
10      initLabels(infixLines[row], row);
11
12    for (const auto &infix: infixLines)
13      postfixLines.push_back(buildPostfix(infix));
14
15    int row = 0;
16    while (0 <= row && row < postfixLines.size())
17      row = evaluatePostfix(postfixLines[row], row);
18    return 0;
```

Реализация parseLexem

```
1  std::vector<Lexem *> parseLexem(const std::string &co
2    std::vector<Lexem *> infix;
3    Lexem *ptr;
4    for (int i = 0; i < (int)codeline.size(); ) {
5      if (ptr = getOperator(codeline, i)) {
6        infix.push_back(ptr);
7        continue;
8      }
9      if (ptr = getNumber(codeline, i)) {
10       infix.push_back(ptr);
11       continue;
12     }
13     if (ptr = getVariable(codeline, i)) {
14       infix.push_back(ptr);
15       continue;
16     }
17     i++;
18   }
```

Реализация initLabels

```
1  void initLabels(std::vector<Lexem *> &infix, int row)
2    for (int i = 1; i < (int)infix.size(); i++) {
3      if (infix[i - 1]->type() == VARIABLE &&
4          infix[i]->type() == OPERATOR)
5      {
6        Variable *lexemvar = (Variable *)infix[i-1];
7        Operator *lexemop = (Operator *)infix[i];
8        if (lexemop->operType() == COLON) {
9          labels[lexemvar->getName()] = row;
10         delete infix[i - 1];
11         delete infix[i];
12         infix[i-1] = nullptr;
13         infix[i] = nullptr;
14         i++;
15       }
16     }
17   }
18 }
```

Реализация buildPostfix (goto)

```
 1 std::vector<Lexem *> buildPostfix(
 2   const std::vector<Lexem *> &infix)
 3 {
 4   std::vector<Lexem *> postfix;
 5   std::stack<Operator *> stack;
 6   for (const auto &lexem: infix) {
 7     if (lexem == nullptr)
 8       continue;
 9     if (lexem->type() == VARIABLE) {
10       Variable *lexemvar = (Variable *) lexem;
11       if (lexemvar->inLabelTable())
12         joinGotoAndLabel(lexemvar, stack);
13       else
14         postfix.push_back(lexem);
15     }
16         ...
17   }
18     ...
```

Реализация joinGotoAndLabel

```
1  void joinGotoAndLabel(Variable *lexemvar,
2                        std::stack<Operator *> &stack)
3  {
4    if (stack.top()->operType() == GOTO) {
5      Goto *lexemgoto = (Goto *)stack.top();
6      lexemgoto->setRow(lexemvar->getName());
7    }
8  }
```

Реализация evaluatePoliz

```
 1  int evaluatePostfix(const std::vector<Lexem *> &postfi
 2                      int row) {
 3    std::stack<int> stack;
 4    for (const auto &lexem: postfix) {
 5      if (lexem->type() == OPERATOR) {
 6        Operator *lexemop = (Operator *)lexem;
 7        if (lexemop->operType() == GOTO) {
 8          Goto *lexemgoto = (Goto *)lexemop;
 9          return lexemgoto->getRow();
10        } else if (...)
11          ...
12    }
13    return row + 1;
14  }
```

Реализация класса Goto

```
 1 class Goto: public Operator {
 2     int row;
 3 public:
 4     enum { UNDEFINED = -INT32_MAX };
 5     Goto(OPERATORTYPE optype): Operator(optype) {
 6         row = UNDEFINED;
 7     }
 8     void setRow(int row) {
 9         Goto::row = row;
10     }
11     int getRow() {
12         return row;
13     }
14     void print() {
15         std::cout << "[<row " << row << ">"
16                   << OPERTEXT[optype] << "] ";
17     }
18 };
```

Пример parseLexem

string

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

infix

```
1  0: [x] [:=] [1]
2  1: [y] [:=] [x] [+] [2]
3  2: [z] [:=] [3] [*] [4] [+] [5]
4  3: [goto] [L=?]
5  4: [x] [:=] [2]
6  5: [L] [:] [x] [:=] [3]
```

Пример initLabels

infix

```
1  0: [x] [:=] [1]
2  1: [y] [:=] [x] [+] [2]
3  2: [z] [:=] [3] [*] [4] [+] [5]
4  3: [goto] [L=5]
5  4: [x] [:=] [2]
6  5: [L] [:] [x] [:=] [3]
```

Пример buildPoliz

infix

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

postfix

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

Пример evaluatePoliz

result

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

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

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

3: [L=5] [goto]
variables: x=1 | y=3 | z=17

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