Практикум на ЭВМ. Интерпретатор. Бинарные операторы

Баев А.Ж.

Казахстанский филиал МГУ

10 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)

Интерпретатор (битовые операторы)

Битовые операторы

1. `a & b;`
2. `a | b;`
3. `a ^ b;`
4. `a >> b;`
5. `a << b;`

Интерпретатор (операторы сравнения)

Операторы сравнения

1. `a > b;`
2. `a >= b;`
3. `a < b;`
4. `a <= b;`
5. `a == b;`
6. `a != b;`

Результаты вычислений: 1 — истина, 0 — ложь.

Интерпретатор (логические операторы)

Логические операторы

1. `a and b`;
2. `a or b`;

Приоритет

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

```
int PRIORITY[] = {
    -1, -1,
    0,
    1,
    2,
    3,
    4,
    5,
    6, 6,
    7, 7,
    7, 7,
    8, 8,
    9, 9,
    10, 10, 10
};
```

Текстовое представление

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

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

Реализация

```
1  int main() {
2      std::string codeline;
3      std::vector<Lexem *> infix;
4      std::vector<Lexem *> postfix;
5      int value;
6
7      while (std::getline(std::cin, codeline)) {
8          infix = parseLexem(codeline);
9          postfix = buildPostfix(infix);
10         value = evaluatePostfix(postfix);
11         std::cout << value << std::endl;
12     }
13     return 0;
14 }
```

Реализация parseLexem

```
1  Lexem* get_oper(...) {
2    for (int op = 0; op < n; op++) {
3      string subcodeline =
4        codeline.substr(i, OPERTEXT[op].size());
5      if (OPERTEXT[op] == subcodeline) {
6        i += OPERTEXT[op].size();
7        return new Operator(op);
8      }
9    }
10   return nullptr;
11 }
```

Реализация parseLexem

```
1  std::vector<Lexem *> parseLexem(
2      const std::string &codeline)
3  {
4    Lexem* lexem;
5    for (int pos = 0, next = 0; pos < codeline.size();)
6      lexem = get_oper(codeline, pos, &next);
7      if (oper != nullptr) {
8          lexem = new_pos;
9          continue;
10     }
11     lexem = get_num(codeline, pos, &next);
12     if (num != nullptr) {
13         lexem = new_pos;
14         continue;
15     }
16     lexem = scan_var(codeline, pos, &next);
17     if (var != nullptr) {
18         lexem = new_pos;
19         continue;
```

Рекурсивный спуск)

Для построения лексического и синтаксического анализатора можно использовать рекурсивный спуск – прямое описание грамматики языка.