Практикум на ЭВМ. Интерпретатор. Все бинарные операторы

Баев А.Ж.

Казахстанский филиал МГУ

12 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)

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)

Битовые операторы

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

Приоритет

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

```
1  int PRIORITY[] = {
2      -1,
3      0, 0,
4      1,
5      2,
6      3,
7      4,
8      5,
9      6, 6,
10     7, 7, 7, 7,
11     8, 8,
12     9, 9,
13     10, 10, 10
14 };
```

Текстовое представление

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

```
1  string OPERTEXT[] = {
2      ":=",
3      "(", ")",
4      "or",
5      "and",
6      "|",
7      "^",
8      "&",
9      "==", "!=",
10     "<=", "<", ">=", ">",
11     "<<", ">>",
12     "+", "-",
13     "*", "/", "%"
14 };
```

Реализация

```
int main() {
    std::string codeline;
    std::vector<Lexem *> infix;
    std::vector<Lexem *> postfix;
    int value;

    while (std::getline(std::cin, codeline)) {
        infix = parseLexem(codeline);
        postfix = buildPostfix(infix);
        value = evaluatePostfix(postfix);
        std::cout << value << std::endl;
    }
    return 0;
}
```

Реализация parseLexem

```
std::vector<Lexem *> parseLexem(
    const std::string &codeline)
{
    std::vector<Lexem *> lexems;
    int n = sizeof(OPERTEXT) / sizeof(std::string);
    for (int i = 0; i < codeline.size(); i++) {
        for (int op = 0; op < n; op++) {
            int len = OPERTEXT[op].size();
            if (OPERTEXT[op] == codeline.substr(i, len))
                lexems.push_back(new Operator(op));
        }
    }
}
```