

Практикум на ЭВМ. Интерпретатор. Бинарные операторы

Баев А.Ж.

Казахстанский филиал МГУ

10 февраля 2020

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

```
1 enum OPERATOR {  
2     LBRACKET, RBRACKET,  
3     ASSIGN,  
4     OR,  
5     AND,  
6     BITOR,  
7     XOR,  
8     BITAND,  
9     EQ, NEQ,  
10    LEQ, LT,  
11    GEQ, GT,  
12    SHL, SHR,  
13    PLUS, MINUS,  
14    MULT, DIV, MOD  
15 };
```

```
1 int PRIORITY[] = {  
2     -1, -1,  
3     0,  
4     1,  
5     2,  
6     3,  
7     4,  
8     5,  
9     6, 6,  
10    7, 7,  
11    7, 7,  
12    8, 8,  
13    9, 9,  
14    10, 10, 10  
15 };
```

Текстовое представление

```
1 enum OPERATOR {
2     LBRACKET, RBRACKET,
3     ASSIGN,
4     OR,
5     AND,
6     BITOR,
7     XOR,
8     BITAND,
9     EQ, NEQ,
10    LEQ, LT,
11    GEQ, GT,
12    SHL, SHR,
13    PLUS, MINUS,
14    MULT, DIV, MOD
15 };
```

```
1 string OPERTEXT[] = {
2     "(", ")",
3     ":",
4     "or",
5     "and",
6     "|",
7     "^",
8     "&",
9     "=", "!",
10    "<=", "<",
11    ">=", ">",
12    "<<", ">>",
13    "+", "-",
14    "*", "/", "%
15 };
```

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  std::vector<Lexem *> parseLexem(  
2      const std::string &codeline)  
3  {  
4      std::vector<Lexem *> lexems;  
5      int n = sizeof(OPERTEXT) / sizeof(std::string);  
6      for (int i = 0; i < codeline.size(); i++) {  
7          for (int op = 0; op < n; op++) {  
8              string subcodeline =  
9                  codeline.substr(i, OPERTEXT[op].size());  
10             if (OPERTEXT[op] == subcodeline) {  
11                 lexems.push_back(new Operator(op));  
12                 break;  
13             }  
14         }  
15     }  
16 }
```

Рекурсивный спуск)

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