

# Практикум на ЭВМ. Интерпретатор. Полиз

Баев А.Ж.

Казахстанский филиал МГУ

01 марта 2022

# План на семестр

- 1 Интерпретатор
- 2 Web сервер
- 3 Параллельное программирование

# 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

- 1 Лексический анализ строка

10 + 2

- 2 Синтаксический анализ инфикс

10	+	2
----	---	---

- 3 Вычисление постфикс

10	2	+
----	---	---

инфикс

10	+	2
----	---	---

постфикс

10	2	+
----	---	---

значение

12

# Реализация функций

```
1  #include <string>
2  #include <vector>
3
4  std::vector<Lexem *> parseLexem(
5      std::string codeline);
6
7  std::vector<Lexem *> buildPoliz(
8      std::vector<Lexem *> infix);
9
10 int evaluatePoliz(
11     std::vector<Lexem *> poliz);
```

## Реализация основной функции

```
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 классов

```
1  class Lexem {  
2  public:  
3      Lexem();  
4  };  
5  
6  class Number: public Lexem {  
7      int value;  
8  public:  
9      Number();  
10     int getValue();  
11 };
```

## Реализация классов

```
1  enum OPERATOR {
2      LBRACKET, RBRACKET,
3      PLUS, MINUS,
4      MULTIPLY
5  };
6  int PRIORITY[] = {
7      -1, -1,
8      0, 0,
9      1
10 };
11 class Oper: public Lexem {
12     OPERATOR opertype;
13 public:
14     Oper();
15     OPERATOR getType();
16     int getPriority();
17     int getValue(const Number& left,
18                 const Number& right);
```



# Лексический анализ

- 1 склеить цифры в числа

```
1 number = 0;  
2 for (int i = 0; i < codeline.size(); i++)  
3     number := number * 10 + codeline[i] - '0';
```

- 2 игнорировать пробелы и табуляции
- 3 обрабатывать до конца строки

# Синтаксический анализ (алгоритм построения полиза «сортировочная станция»)

Пример:

$$1 + 2$$

символ входной строки

1

+

2

конец строки

стек операторов

+				
+				

выходная строка

1				
1				
1	2			
1	2	+		

# Синтаксический анализ (алгоритм построения полиза «сортировочная станция»)

Пример:

$$1 + 2 * 4$$

символ входной строки

1

+

2

\*

4

конец строки

стек операторов

+				
+				
+	*			
+	*			

выходная строка

1				
1				
1	2			
1	2			
1	2	4		
1	2	4	*	+

# Синтаксический анализ (алгоритм построения полиза «сортировочная станция»)

Пример:

$$(1 + 2) * 4$$

символ входной строки

(

1

+

2

)

\*

4

конец строки

стек операторов

(				
(				
(	+			
(	+			
*				
*				

выходная строка

1				
1				
1				
1	2			
1	2	+		
1	2	+		
1	2	+	4	
1	2	+	4	*

# Синтаксический анализ (алгоритм построения полиза «сортировочная станция»)

Пример:

$$1 + 2 * 3 * (9 - 4)$$

# Синтаксический анализ (алгоритм построения полиза «сортировочная станция»)

- 1 Читаем очередной символ.
- 2 Если символ является числом, добавляем его к выходной строке.
- 3 Если символ является открывающей скобкой, помещаем его в стек.
- 4 Если символ является закрывающей скобкой, то выталкиваем все элементы из стека в выходную строку, пока верхним элементом стека не станет открывающая скобка.
- 5 Если символ является бинарной операцией «орег», то пока на вершине стека операция на вершине стека имеет приоритет больше (или равен для левоассоциативной операции), чем «орег», то выталкиваем верхний элемент стека в выходную строку, в конце помещаем «орег» в стек.
- 6 Когда входная строка закончилась, выталкиваем все символы из стека (операторы) в выходную строку.

# Синтаксический анализ (реализация)

```
1  #include <stack>
2
3  //create
4  stack <int> stack;
5
6  //push value to top
7  stack.push(1);
8
9  //get top value
10 int x = stack.front();
11
12 //pop top value (no return value!)
13 stack.pop();
```

# Вычисление полиза

Пример:

$$1 + 2$$

Польская инверсная запись:

1	2	+
---	---	---

Стек для вычислений:

1	1		
2	1	2	
+	3		



# Вычисление полиза

Пример:

$$1 + 2 * 4$$

Польская инверсная запись:

1	2	4	*	+
---	---	---	---	---

Стек для вычислений:

1	1		
2	1	2	
4	1	2	4
*	1	8	
+	9		

# Вычисление полиза

Пример:

$$(1 + 2) * 4$$

Польская инверсная запись:

1	2	+	4	*
---	---	---	---	---

Стек для вычислений:

1	1		
2	1	2	
+	3		
4	3	4	
*	12		

# Вычисление полиза

Пример:

$$1 + 2 * 3 * (9 - 4)$$

Польская инверсная запись:

1	2	3	*	9	4	—	*	+
---	---	---	---	---	---	---	---	---

Стек для вычислений:

1	1				
2	1	2			
3	1	2	3		
*	1	6			
9	1	6	9		
4	1	6	9	4	
—	1	6	5		
*	1	30			
+	31				