

Логика

Мы

24 апреля 2023 г.

# Оглавление

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Аксиоматический метод</b>            | <b>2</b> |
| 1.1      | .....                                   | 2        |
| 1.2      | .....                                   | 2        |
| 1.3      | .....                                   | 2        |
| <b>2</b> | <b>Теория множеств</b>                  | <b>3</b> |
| 2.1      | Базовые понятия .....                   | 3        |
| 2.2      | Аксиомы .....                           | 3        |
| 2.2.1    | Равенства .....                         | 3        |
| 2.2.2    | Пары .....                              | 3        |
| 2.2.3    | Объединения .....                       | 3        |
| 2.2.4    | Степени .....                           | 3        |
| 2.2.5    | Выделения .....                         | 4        |
| 2.3      | Определения .....                       | 4        |
| <b>3</b> | <b>Программная инженерия</b>            | <b>5</b> |
| 3.1      | Изменяемость .....                      | 5        |
| <b>4</b> | <b>Трансляция кода</b>                  | <b>6</b> |
| 4.1      | Этапы компиляции .....                  | 7        |
| 4.1.1    | Лексический анализ (сканирование) ..... | 7        |
| 4.1.2    | Синтаксический анализ (парсинг) .....   | 7        |
| 4.1.3    | Семантический анализ .....              | 7        |
| 4.1.4    | Генерация промежуточного кода .....     | 8        |
| 4.1.5    | Оптимизация кода .....                  | 8        |
| 4.1.6    | Генерация кода .....                    | 8        |
| <b>5</b> | <b>Необработанное</b>                   | <b>9</b> |

# Глава 1

## Аксиоматический метод

Базовое понятие - это неопределяемое понятие.

### 1.1

Зафиксировать базовые понятия.

### 1.2

Зафиксировать аксиомы, связывающие понятия.

### 1.3

Выводить следствия по правилам логики.

## Глава 2

# Теория множеств

### 2.1 Базовые понятия

Принадлежность ( $x \in y$ ).

### 2.2 Аксиомы

#### 2.2.1 Равенства

$$x = y \longrightarrow \forall z \ (x \in z \longrightarrow y \in z)$$

#### 2.2.2 Пары

$$\exists z \ \forall u \ \left( u \in z \Leftrightarrow \begin{cases} u = x \\ u = y \end{cases} \right)$$

#### 2.2.3 Объединения

$$\exists y \ \forall u \ \left( u \in y \Leftrightarrow \exists z \ \begin{cases} u \in z \\ z \in x \end{cases} \right)$$

#### 2.2.4 Степени

$$\exists y \ \forall u \ (u \in y \Leftrightarrow u \subseteq x)$$

### 2.2.5 Выделения

$$\{x \in A \mid \varphi(x)\}$$

## 2.3 Определения

Класс =  $\{x \mid \varphi(x)\}$ . Не все классы являются множествами. Все множества являются классами.

$$x = y \Leftrightarrow (z \in x \Leftrightarrow z \in y)$$

$$x \subseteq y \Leftrightarrow \forall z (z \in x \longrightarrow z \in y)$$

## Глава 3

# Программная инженерия

Любую программу можно написать на низкоуровневом языке - языке, наиболее приближённом к устройству компьютера. Однако, тем не менее, разрабатывают всё новые языки программирования. Далее рассматриваются причины, по которым возникает нужда в высокоуровневых языках.

### 3.1 Изменяемость

Программистам в течение жизненного цикла разработки ПО приходится изменять программу. По причине изменений требований к продукту или для устранения ошибки.

Чтобы что-то изменить, нужно найти всю имплементацию этого чего-то в программе. С этим возникают две трудности: имплементация этого чего-то простирается в большой части кода - слишком много приходится править, чтобы внести нужное изменение - и код трудно читаем - сложно понять, где то, что нам нужно.

Языки программирования создают такими, чтобы они как можно более полно решали данные проблемы.

Часто добавляют "мультипарадигменные" конструкции, которые должны матчаться в нашем мозгу с устоявшимися паттернами. Однако всевозможных паттернов настолько много, что данные вводящиеся конструкции только капля в море.

Вторым способом бороться с данными проблемами, который работает всегда, является продолжающийся рефакторинг согласно сложности Джона.

## Глава 4

# Трансляция кода

**Компилятор** - это программа, переводящая текст программы с одного языка на другой.

**Интерпретатор** - это программа, выполняющая код программы, не переводя её на другой язык.

**Компоновщик (линкер)** - это программа, выполняющая разрешение внешних адресов памяти, по которым код из одного файла может обращаться к информации из другого файла.

**Загрузчик** - это программа, которая помещает все выполнимые объектные файлы в память для выполнения.

Компиляция состоит из анализа и синтеза.

В течение компиляции код может переводиться по цепочке в несколько промежуточных представлений.

Таблица символов содержит в себе информацию, которая накапливается на протяжении компиляции.

**Проход (pass)** - это этапы компиляции, преобразующие один файл в другой (необязательно в файл с целевым кодом).

## 4.1 Этапы компиляции

### 4.1.1 Лексический анализ (сканирование)

**Лексема** - это значащая последовательность символов кода.

**Токен** - это значение  $\langle \text{имя токена, значение атрибута} \rangle$ , представляющее лексему, где значение атрибута указывает на запись в таблице символов.

### 4.1.2 Синтаксический анализ (парсинг)

Синтаксический анализатор структурирует токены в синтаксическое дерево.

**Контекстно-свободная грамматика (КС-грамматика).**

**Терминальный символ** - это элементарный символ языка, определяемый грамматикой.

**Нетерминальный символ** - это множество строк терминалов, заданное продукцией.

**Продукция** - это определение конкретного нетерминального символа. Записывается как  $a \rightarrow b$ , где  $a$  - нетерминал, называемый заголовком (левой частью) продукции,  $b$  - последовательность (декартово произведение, если про множества) нетерминалов и/или (объединение, если про множества) терминалов (последовательность может быть пустой, что соответствует пустой строке или пустому множеству), называемая телом (правой частью) продукции.

Контекстно-свободная грамматика имеет четыре компонента:

1. Множество терминальных символов.
2. Множество нетерминальных символов.
3. Множество продукций.
4. Стартовый нетерминальный символ.

**Форма Бэкуса-Наура (BNF)** - это другая форма записи КС-грамматики.

### 4.1.3 Семантический анализ

Семантический анализатор проверяет синтаксическое дерево на корректность.



#### **4.1.4 Генерация промежуточного кода**

Генерация кода для абстрактной вычислительной машины.

#### **4.1.5 Оптимизация кода**

Оптимизация промежуточного кода.

#### **4.1.6 Генерация кода**

Генерация кода на целевом языке.

## Глава 5

# Необработанное

**Автонимный способ обозначения** - это способ обозначения, при котором формальные выражения обозначаются так же, как и их значения.

**Высказывательная форма.**

**Именная форма** - это выражение с переменной.

**Связанные переменные** - это переменные, вместо которых нельзя подставить значение.

**Основания математики** - это раздел (в книге сказано "аспект") математической логики, изучающий объекты математики, истинные свойства этих объектов, на основании которых можно вести рассуждения, а также "сохраняющие истину" способы рассуждений.