06,5. КВАНТОРЫ, ФАЙЛЫ

курс лекций по информатике и программированию
для студентов первого курса ИТИС КФУ (java-поток)
2023/2024

М.М. Абрамский

кандидат технических наук, доцент кафедры программной инженерии

ДРУГОЙ ПОДХОД К ПРОВЕРКЕ ПРАВИЛЬНОСТИ

ЖЗН

Вы хотите купить себе велосипед.

В городе 100 точек, где продаются велосипеды, удобные вам.

Куда вы пойдете?

Суть ответов

- Куда бы вы ни пошли, как бы ни обходили магазины, **вы остановитесь**, когда в очередном магазине его купите!
    - Из 100 магазинов это может быть и второй, и третий, и сотый магазин,
    - ВАЖНО: после покупки вы все равно остановитесь – в следующий магазин не пойдете.

Переносимся на урок

Дана строка s,

Необходимо проверить в ней наличие символа ‘a’.

```
for (int i = 0; i < s.length(); i++) {
    if (s.charAt(i) == 'a') {
        System.out.println(s.charAt(i));
    }
}
System.out.println("НЕТ!");
```

Формулировочка

- Нас не спрашивают
  - Ни сколько таких элементов
  - Ни какой индекс у нуля

- Нас спрашивают
  - Есть или нет.

! Предикат

Условие поиска – всегда предикат

Примеры:

- P(x) – x – нулевой
- P(x) – x – велосипед, который мне понравился

Но предикат отражает только условие над одним объектом.

- *А мы ведь ищем нулевой «среди элементов массива»*
- *Подходящий велосипед среди множества магазинов.*

Нужно уметь формулировать предикатное утверждение над множеством данных

Кванторы

- Не просто сокращенная запись «для любого» или «существует»
- Но оператор в математической логике
- $\forall x\ P(x)$, $\exists x\ P(x)$
  - для любого x верно P(x)
  - существует x, что для него верно P(x)

Кстати

- Не нужно писать *if (p == true)*
  - Если p - true, то (p == true) – тоже true
  - Если p - false, то (p == true) – тоже false
  - Значит, p и p == true – одно и то же выражение
- Аналогично if (p == false)
  - p - true – значит (p == false) – false
  - p - false – значит (p == false) – true
  - Значит p == false – это p наоборот
    - а p наоборот – это !p

Итак

«Ладно, я понял, я должен вернуть булевское значение – это и есть ответ квантора. Вот мой код»

```
boolean flag = false;
for (int i = 0; i < s.length(); i++) {
    if (s.charAt(i) == 'a') {
        flag = true;
    }
}
```

Долго. Нужно прерывание

! Смысл переменной flag – состояние «найдено». В начале оно – false
Если бы был квантор А – то начальное значение – true – “пока не опровергли, верно)

Что такое квантор по факту

$\forall$ i P(a[i]) = P(a[0]) & P(a[1]) &...& P(a[n-1])

$\exists$ i P(a[i]) = P(a[0]) V P(a[1]) V...V P(a[n-1])

Что мы знаем про И и ИЛИ на нескольких аргументах?

- Вот именно. Зачем лишний раз считать, если ответ уже понятен.
- Нужно прерывать обработку, если ответ понятен.

Прерывание. Способ 1. Классика

```
boolean flag = false;
for (int i = 0; i < s.length() && !flag; i++) {
    if (s.charAt(i) == 'a') {
        flag = true;
    }
}
```

Прерывание. Способ 2.

```
boolean flag = false;
for (int i = 0; i < s.length(); i++) {
    if (s.charAt(i) == 'a') {
        flag = true;
        break;
    }
}
```

break и булева переменная дублируют функции друг друга.

Прерывание. Еще один способ

```
public static boolean check(String s) {
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) == 'a')
            return true;
    }
    return false;
}

public static void main(String[] args) {

    String s = "Hello!";
    …
    System.out.println(check(s));

}
```

Квантерные задачи

- Частный случай ***метода состояний***.

- Для кванторов состояний всего 2:
  - «нет/есть» (пока нет – ищем, когда есть – даем ответ)
  - «все/не все» для A

- Метод состояний – фактическое обобщение квантора
  - Существует ли ошибка в задаче или нет

Метод состояний. Пример: проверка идентификатора

```
int state = 0;
String s = scanner.next();
i = 0
while (state != 2 && i < s.length()) {
    switch (state) {
        case 0:
            if (s.charAt(i) - буква || s.charAt(i) == "_")
                state = 1;
            else
                state = 2;
            break;
        case 1:
            if (s.charAt(i) - буква или цифра || s.charAt(i) == "_")
                state = 1;
            else
                state = 2;
            break;
    }
    i++;
}
if (state == 2) - все плохо
else - все хорошо
```

КОГДА НАБОР СИМВОЛОВ – НЕ ТОЛЬКО СТРОКА

Ввод через консоль

```
cmd - java FirstClass

F:\praxis\601>javac FirstClass.java

F:\praxis\601>java FirstClass
100500
```

«Что у него в программцах, моя прелесть?»

Что угодно

- `int x = scanner.nextInt();`
- `double z = -1 * scanner.nextDouble();`
- `String s = scanner.next();`
- `String s = scanner.nextLine()`
- …

Классическое считывание с текстового источника

При привычном считывании мы вводим информацию на экран – а это символы (текст).

Поведение при вводе данных:

- Некоторые операторы ввода при вызове операторов ввода конвертируют входные данные из строкового типа в тот тип данных, который необходим (x = read()).
  - Pascal: read
  - C, C++: scanf, cin
- Некоторые осуществляют первичный, «сырой» (raw) ввод, его результат надо сконвертировать в нужный тип
  - Python: input() – считывает строку, а дальше – int(…), float(…).
- Многие языки содержат средства для обоих способов:
  - Java: Scanner/DataInputStream конвертирует (nextInt / readInt), а, например, BufferedReader умеет считывать только строку до переноса (readLine)

But We Are Lazy

Чтобы не вводить данные каждый раз при запуске программы, могут пригодиться … файлы!

Кстати, это не единственная ли это причина для использования файлов

Важная причина использования файлов как источников данных в программе – необходимость хранить данные между запусками программы.

- «ААААА, я не засейвился»

Любое сохранение используется для этих целей!

Исторический контекст

- 1950, Radio Corporation of America (RCA) в журнале «Popular Science»: *"the results of countless computations can be kept 'on **file**' and taken out again. Such a **'file'** now exists in a 'memory' tube developed at RCA Laboratories. Electronically it retains figures fed into calculating machines, holds them in storage while it memorizes new ones – speeds intelligent solutions through mazes of mathematics."*

- Изначально – физическое хранилище данных (т.е. файл – это такая «память»).

- Начиная с 60-х, появляется концепция «виртуальных» файлов на одно физическом хранилище – так и появляется современный файл.

Файл

- Всегда набор битов!
  - Байтов, килобайтов, мегабайтов
  - У бита два возможных значения, поэтому файл называется **бинарным**.

Текстовые файлы

Частный случай бинарных файлов. Текстовые файлы – файлы из символов, файлы типа char.

Текстовый редактор интерпретирует содержимое как коды символов в кодировке – и отображает символы на экране!

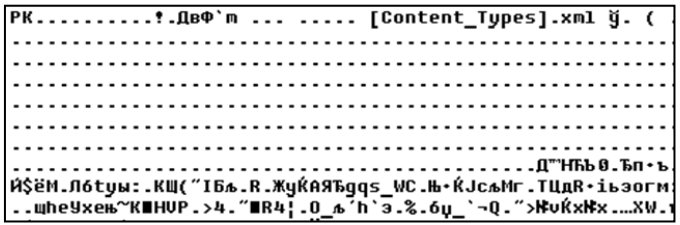
- Отсюда и «абракадабра», когда мы открываем не текстовые файлы в текстовом редакторе.

Примеры текстовых файлов

- .java, .py, .cpp, .cs, .ini, .txt, .html, .js, .css, .rtf, .fb2, .csv

Примеры нетекстовых файлов

- .jpg, .doc, .xlx, .exe, .djvu
- Могут содержать текстовые вставки



Почему не все источники данных текстовые

Взял текстовый редактор, написал туда 126

- Что я сделал по факту?
- Сколько занимает 126?

Текстовый файл

3 символа, размер файла = 3 х размер одного символа (зависит от кодировки, но не менее 1 байта) – т.е. не менее 3 байтов

Бинарный файл

8 битов = 1 байт

Возьмем уже 2 числа

65 и 66 – влезают в byte, значит занимают 2 байта.

Но если записать в текстовый файл `65 66`, то это минимум 5 байтов (считая пробел).

Чтение бинарное файлов

Каждое приложение знает свой формат файлов (смысл каждого байта на определенной позиции)

Пример – TIFF (отрывок):

«…после 8-байтового заголовка:

- Байты 0-1: Первое ***слово*** файла определяет порядок байтов, используемый в файле. Допустимыми его значениями являются: II (hex 4949) MM (hex 4D4D)
- Байты 2-3: Второе ***слово*** TIFF-файла - это номер версии. Число, равное 42 (2A hex), но оно не равно номеру редакции текущей спецификации TIFF (в данном случае номер редакции текущей спецификации - это 5.0). Но если это случится, то будет означать, что TIFF изменился настолько радикально, что программа чтения TIFF должна немедленно прекратить работу. Число 42 было выбрано из-за его глубокого философского смысла. Оно может и должно использоваться для дополнительной проверки того, что это действительно TIFF-файл.
- Байты 4-7: Это ***слово*** типа long, содержащее смещение в байтах первой директории файла (Image File Directory). Директория может располагаться в любом месте файла вслед за заголовком, но ее начало должно быть выровнено на границу слова. В частности, директория может следовать за данными изображения, которое она описывает. Программы чтения должны просто перемещаться по этому указателю, вне зависимости от того, куда он указывает. (Термин байтовое смещение (byte offset) всегда используется в этом документе, чтобы ссылаться на положение относительно начала файла.
- Директории файла (Image File Directory - IFD) состоят из 2-байтового счетчика числа элементов (т.е. числа тегов в данной директории), вслед за которым расположена последовательность 12-байтовых тегов и далее 4-байтовое смещение для следующей директории (или 0, если таковая отсутствует). Не забывайте записывать 4 нулевых байта в конце последней директории! Каждый 12-байтный элемент IFD имеет следующий формат: Байты 0-1 содержат Тег (Tag) поля. Байты 2-3 содержат Тип (Type) поля. Байты 4-7 содержат Длину (Length) поля (здесь, возможно, более удачным термином является Count - Счетчик). Байты 8-11 содержат Смещение для значения (Value Offset), т.е. байтовое смещение того места в файле, где расположено само значение. Предполагается, что это смещение должно быть выровнено на границу слова, т.е. Value Offset должно быть четным числом. Это смещение может указывать на любое место в файле. Элементы в IFD должны быть отсортированы в порядке возрастания поля Tag…»

Консоль и файлы. Разное

- Консоль – **интерактивна**.
  - Ждет событий пользователя
    - ввод чередуется с обработкой данных.
    - конец ввода определяется пользователем в реальном времени.
      - » введи n, n чисел
      - » суммируй, пока не ввели 0
        - обратите внимания завершение в обоих случаях

- Файл – **нет**
  - Введен заранее.
    - значит размер конечен – значит можно говорить о том, что конец ввода уже определен заранее
    - логично, что это позволяет обрабатывать файл по особому

EOF – когда файл закончится

В любом языке программирования – End Of File

- Пока файл не кончился, читаем данные
- While file is not over, read data
- While (!file.over()) { data = file.read(); }

Как работать с файлом

- *Открыть файл;*
- *Пока есть информация, читать из него или записать в него данные;*
- *Закрыть файл;*

Java. Класс File

- Класс в Java, собирающий понятие «файла» (и папки).

- `File f = new File("input.txt")`

  ! расширение
  ! абсолютный и относительный пути

hasNext…

```
Scanner scanner = new Scanner(new File("input.txt"));

while (scanner.hasNextInt()) {

    int x = scanner.nextInt();
    …
}
```