04.
МАССИВЫ
СТРОКИ

Лекции по информатике для студентов
первого курса Высшей школы ИТИС
2019 год

МИХАИЛ АБРАМСКИЙ
старший преподаватель
Высшая школа ИТИС КФУ

Now we know

- Примитивные типы данных
- Структуры управления (последовательная, условия, циклы)
- Знание того, как качественно писать код.

- Погружаемся дальше.

?

- *Достаточно ли нам существующих типов?*
- *Достаточно ли нам существующих операций?*

- Мы знаем, что такое массивы.
- Но теперь мы можем доказать их необходимость с точки зрения сложности.

Имея только переменные примитивного типа, мы не можем управлять размером памяти, выделяемого на задачу

- Сложность почти всех алгоритмов, использующих только примитивы – O(1)
  - » *понятно, почему?*

Пример

- Вычислить определитель матрицы (методом Гаусса)
  - зависит от размера матрицы.

```
if (n == 2) {
    a11 = sc.nextInt();
    a12 = sc.nextInt();
    a21 = sc.nextInt();
    a22 = sc.nextInt();
} else if (n == 3) {
    a11 = sc.nextInt();
    a12 = sc.nextInt();
    a13 = sc.nextInt();
    a21 = sc.nextInt();
    …
}
…
```

бред

Необходимость

1. Структура данных, хранящая вход для целей обработки.

2. Размер структуры должен задаваться динамически во время работы.

    - А не как в Pascal – сначала объявляем 10 000-й массив, а затем вводим размер массива 5.

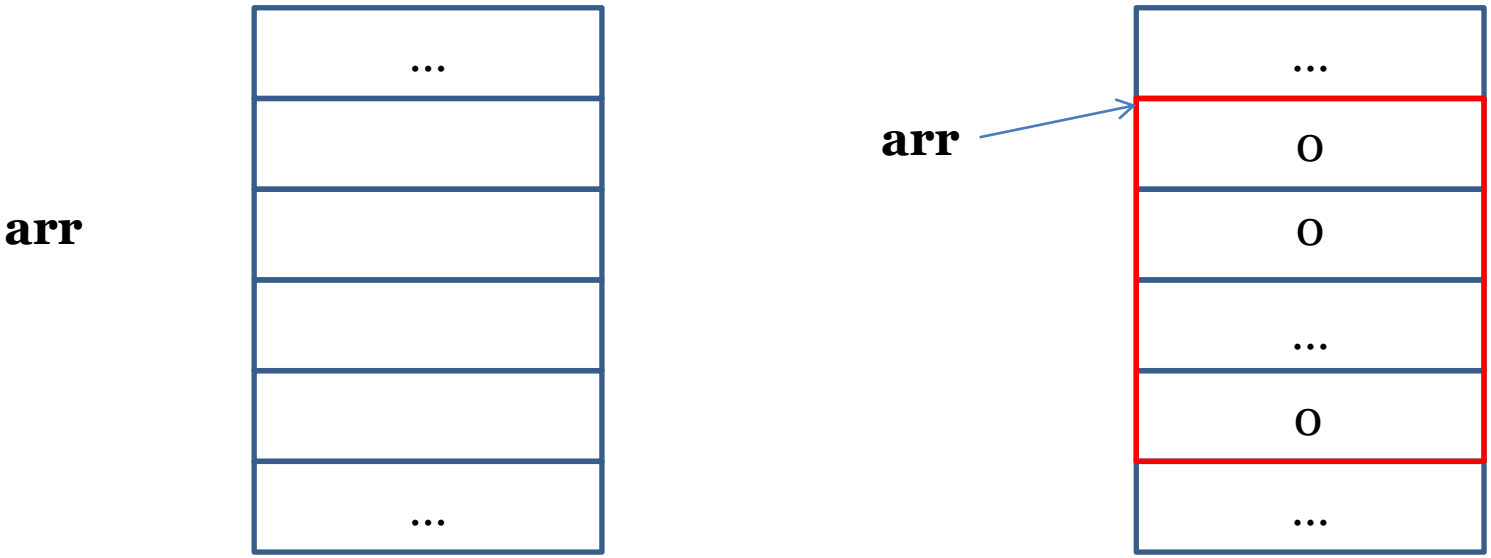
Массив

- *Набор* данных **одного** типа;
- Объявление: ТИП [ ] ИМЯ;
- Выделение памяти: arr = new ТИП[РАЗМЕР]
  - размер – целочисленная переменная, может быть вычислена заранее
  - каждый элемент массива получает значение типа по умолчанию
    - » нулевое значения – для boolean, char, ссылочного типа?
- a[i] – обращение к элементу под номером i;
- Если массив размера n, то индексы его элементов от **0** до **n-1.**

Как массив хранится в памяти

int [] arr

arr = new int[5]



Обратите внимание

1. Адрес массива совпадает с адресом его первого элемента
2. Ячейки массива – одного размера (т.к. одного типа)
3. Адреса вообще говоря – числа (0xfab3123)

Как получить адрес i-го элемента?

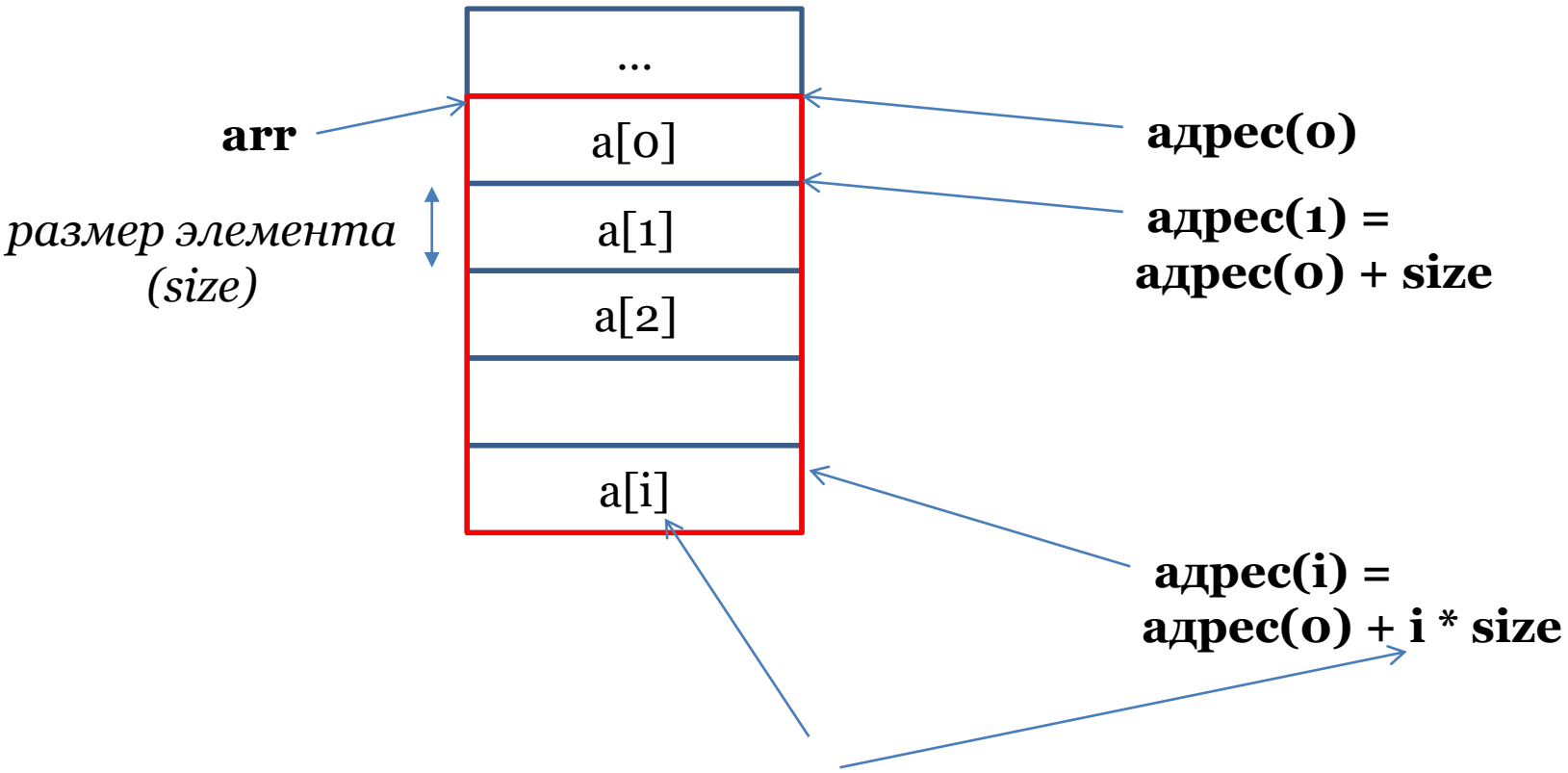
Что стоит за a[i]

a[i] – обращение к содержимому элемента массива a под номером i

обратить надо по адресу, а адрес легко считается:

- **`адрес(i) = адрес(0) + i * size`**
  - » адрес(0) = адрес массива = a
  - » size – размер типа данных массива

Что стоит за оператором a[i]



Сложность доступа к любому элементу массива?

Прямая адресация

Цикл прохода по массиву

- arr.length – длина массива
  - Не всегда вы располагаете переменной ее длины
- Учимся считать с нуля
  - Длина: n, первый элемент 0, второй – 1, …, последний – n – 1.

```
for (int i = 0; i < n; i++) {
    обработка a[i]
}
```

for each

- ТОЛЬКО ЧТЕНИЕ

```
for (int x : a) {
    обработка с x
    x = … - бесполезный оператор
}
```

Ошибки работы с массивом. Присваивание

```
int[] b = a;
```

// ждем тут, что все типа скопировалось в b

Но вспомните, что такое a и b?

Ошибки работы с массивом

- Выход за пределы массива (ошибки с его индексами):

```
for (int i = 1; i <= n; i++)
```

или:

```
for (int i = 0; i < n; i++) {
        a[i+1] = …;
```

Полезные штуки

Инициализация в коде.

```
int[] arr = new int[]{4, 8, 15, 16, 23, 42};
```

```
int[] arr = {4, 8, 15, 16, 23, 42};
```

Зачем?

Как узнать размер?

Библиотека (класс) java.util.Arrays

Вывод массива на экран одной строчкой:
`System.out.println(Arrays.toString(массив));`
Сравнение массивов
`Arrays.equals(массив1, массив2)`
Сортировка (быстрая, Dual-Pivot)
`Arrays.sort(массив)`
Копирование массива
`Arrays.copy(массив1, массив2)`

...

Еще много методов, позволяющих делать то, что обычно приходится делать вручную циклом.

Многомерные массивы

- Пример:

```
int n = 10;
int m = 20;
int [][] arr = new int[m][n];
```

- Обращение к элементу: a[i][j]

- Если a – двумерный массив, то a[i] – это что?

Ступенчатые массивы

Не всегда все подмассивы в многомерном массиве должны быть одного размера

```
int n = 10;
int [][] a = new int[n][];
a[0] = new int[n];
a[1] = new int[n-1];
//...
```

Хардкод

```
int n = 10;
int[] arr = new int[n];
for (int i = 0; i <= 9; i++) {
    //ввод arr
}
arr[9] = arr[1];
for (int i = 0; i < 5; i++) {
    // обнуление первой половины
    arr[i] = 0;
}
```

исправить

Массив символов

- char [] – в С это и называлось строкой.
- Java: Строка – отдельный тип (ссылочный), у которого должно быть много полезных функций (методов)
  - Есть целый класс задач, который решается на тестовых данных (текст – массив символов).
- *Еще раз: по смыслу строка – массив символов, но с точки зрения реализации – нет.*

Символ

- `char c = 'a';`
- Все символы имеют свой код
  - ASCII (american standard code for information interchange) – сначала 7 бит, затем 8 бит.
    - » Поэтому char и считают целочисленным типом.
  - Но 8 бит – это 1 байт, а мы знаем, что char в Java – 2 байта.
    - » Это все потому, что…

Unicode, UTF

UTF (unicode transformation format) - стандартная кодировка Java программы.

К символу можно обратиться по его коду (16-ный): ‘\u0053’ – (буква S)

Как бы мы не записали программу, символы, компилятор переводит их все в Unicode.

- **UTF-8, UTF-16**

Не только English

- Юникод разрешает вот такие идентификаторы:

```
public class ЭтоЧтоКласс {

    public static void main(String[] args) {

        final int МОЯ_КОНСТАНТА = 23;

    }

}
```

```java
 * 11-401
 * 045
 */

import java.util.Scanner;

public class Task045 {
    public static void main(String[] args) {
        Scanner 入力 = new Scanner(System.in);
        String[] サッカーチーム = 入力.nextLine().split(" ");
        int[] 成果 = new int[サッカーチーム.length];
        int サッカーの試合 = Integer.parseInt(入力.nextLine());
        for (int カウント = 0; カウント < サッカーの試合; カウント++) {
            String[] 文字列 = 入力.nextLine().split(" ");
            int 最初, 第2;
            for (最初 = 0; !サッカーチーム[最初].equals(文字列[0]) && (最初 < サッカーチーム.length); 最初++);
            for (第2 = 0; !サッカーチーム[第2].equals(文字列[1]) && (第2 < サッカーチーム.length); 第2++);
            String[] アカウント = 文字列[2].split(":");
            int 違い = Integer.parseInt(アカウント[0]) - Integer.parseInt(アカウント[1]);
            成果[最初] += 違い;
            成果[第2] -= 違い;
        }
        for (int カウント = 0; カウント < サッカーチーム.length; カウント++) {
            System.out.println(サッカーチーム[カウント] + "\t" + 成果[カウント]);
        }
    }
}
```

Escape Characters

- Если в символе \, значит у него есть особый смысл:
  - \n – перенос строки
  - \t – табуляция
  - \b – отмена предыдущего символа
- Также \ применяется, чтобы вывести символы, которые тяжело вывести обычным способом:
  - \\
  - \”
    - Т.к. “””” неправильно понимается компилятором.
  - \’

Класс String

- Неизменяемая строка
  - **immutable**, нельзя s[0] = ‘a’
  - есть изменяемые: StringBuffer, StringBuilder
- Доступ к символу: метод charAt(i)
- Длина: метод str.length()
  - не путать с массивом!
- Соединение строк: +
  - `String hi = “Hello + “, “ + “ITIS”`

Объявление

- Как ссылочный тип по хорошему строку нужно было бы создавать вот так:
  - String str = new String(“Hello!”);

- Но только для строки введено сокращение:
  - String str = “Hello!”;

Но не все так просто.

Что происходит в первом случае

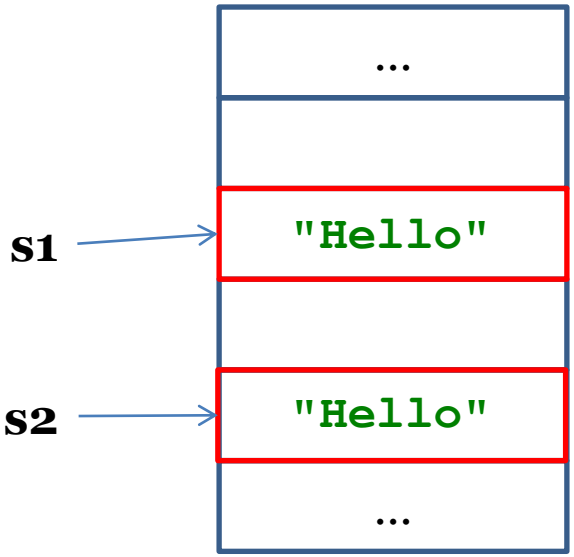
```
String s1 = new String("Hello");
```

- Объявляется ссылочная переменная (ссылка) s1 типа String
- new String(…) – создается объект класса String на основании содержимого строковой константы “Hello”
- Объект присваивается ссылке (теперь она на него указывает).

Правда об операции == на ссылочных типах данных!

== проверяет равенство ссылок (в одно и то же ли мы место ссылаемся или нет)

содержимое по ссылке на равенство не проверяется!

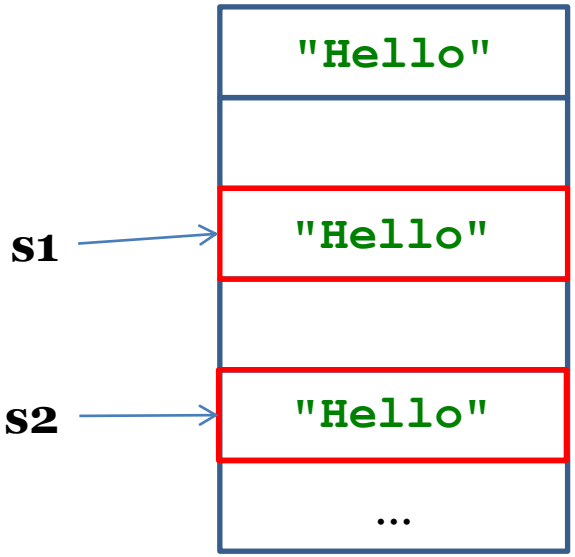



```
String s1 = new String("Hello");
String s2 = new String("Hello");
```

Чему равно s1 == s2?

Более того

- Строковые константы (“Hello” в нашем примере) создаются в памяти как отдельный объект:




```
String s1 = new String("Hello");
String s2 = new String("Hello");
System.out.println(s1 == "Hello");
System.out.println(s1 == s2);
```

что увидим?

Правильная проверка

`s1.equals(s2), s1.equals("Hello")`

- Проверяет содержимое
  - Верно не только для строк:
    - пусть arr1, arr2 – массивы.
    - В чем разница между
      - arr1 == arr2
      - Arrays.equals(arr1, arr2)?

Ho!

```
String s1 = "Hello";
String s2 = "Hello";
System.out.println(s1 == "Hello");
System.out.println(s1 == s2);
```

Но!

```
String s1 = "Hello";
String s2 = "Hello";
System.out.println(s1 == "Hello");
System.out.println(s1 == s2);
```

true
true

String s1 = “Hello”; // Создается строковый объект “Hello” и он присваивается ссылке s1. Строковые константы создаются 1 раз – следующее их упоминание – уже созданный объект. Поэтому тот же объект “Hello” присваивается и s2. Поэтому все 3 ссылки указывают на один и тот же объект.

Ho! [2]

```
String s1 = "Hello";
String s2 = "Hell" + "o";
System.out.println(s1 == s2);
System.out.println(s2 == "Hello");
```

Но! [2]

```
String s1 = "Hello";
String s2 = "Hell" + "o";
System.out.println(s1 == s2);
System.out.println(s2 == "Hello");
```

true
true

Hell + o создает должен порождать новый объект – строку Hello, но она уже есть, поэтому в s2 присваивается существующая Hello

System.out.println (?)

```
int x = 2, y = 5;
char a = 'a', b = 'b';
System.out.println(x + y);
System.out.println(x + y + "");
System.out.println("" + x + y);
System.out.println(x - y + "");
System.out.println(a + b);
System.out.println(a + b + "");
System.out.println("" + a + b);
System.out.println(a + y);
System.out.println("" + a + y);
```