Introduction to Computer Science
Reichman University

Recitation 4

Overview

- Arrays
- Call by value
- Mutable Vs Immutable
- ASCII

Question 1 – Length of Last Word

- Given a string which consists only of words* and spaces, return the length of the last word in the string.
  - Word is defined as a sequence of letters with strictly positive length.
- Note: you may assume that there is at least one word

- Examples:
  - java LengthOfLastWord "Hello World"
    - 5
  - java LengthOfLastWord "Intro to computer science"
    - 7
  - java LengthOfLastWord " Strings   are great   "
    - 5
  - java LengthOfLastWord " test "
    - 4
  - java LengthOfLastWord "   a"
    - 1

Question 1 – Solution

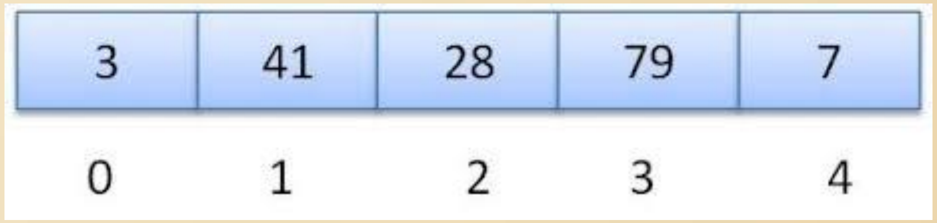
```
public static int lengthOfLastWord(String sentence){

     int end = sentence.length() - 1;

          while (end >= 0) {
               if (sentence.charAt(end) != ' ') {
                    break;
               }
               end--;
          }
     int start = end;
     while (start >= 0) {
               if (sentence.charAt(start) == ' ') {
                    break;
               }
          start--;
     }
     return end - start;
}
```

Recitation 4

Arrays



Question 2 - Array Print

- Similar to the case of strings, we can think about arrays as referring to some location in physical memory.
- Consider the next bit of code:

```
int[] array = new int[0];

System.out.println(array);

//[I@120d62b
```

- This curious number is actually an address in the computer's memory
- To compensate we will build a function which prints an array which will print the array in the following format :

  {arr[0], arr[1] ,..., arr[arr.length-1]};

Question 2 - Solution

```
public static void printArray(int[] array){

     System.out.print('{');
     for (int i = 0; i < array.length; i++){
          System.out.print(array[i]);
          char c = i != array.length - 1 ? ',' : '}';
          System.out.print(c);
     }
     System.out.println();
}
```

Note: string.length() has different semantic than array.length
it has different meanings but for now take note and ensure you are not confused.

Question 3 - Comparing arrays

- Let's use '==' to compare arrays.
- Why it happens?

```java
int[] a = {7};
int[] b = {7};
System.out.println(a == b);        // false
System.out.println(a[0] == b[0]);    // true
```

- To compensate we will build a function which compares if a couple of arrays contain the same elements in the same order:

Question 3 - Solution

```
public static boolean equalsArray(int[] arr1, int[] arr2){
     if (arr1.length != arr2.length) {
          return false;
     }
     for (int i = 0; i < arr1.length; i++){
          if (arr1[i] != arr2[i]) {
               return false;
          }
     }
     return true;
}
```

Command Line Arguments - Revisited

- Now that you know all about arrays, you may have noticed something familiar.
- You will recall that every program starts with:

```
public static void main (String[] args){
```

- Now, we may see that 'args' is simply an array of strings entered by the user.
- So, for example: 'Integer.parseInt(args[0]);' parses the first element located in args.
- Having taken out some of the mystery, this gives rise to a very natural way to check how many command line arguments were entered.
- args.length gives the requested number.

Recitation 4

ArrayOps

Question 4 – Sum Array

- Given an integer array arr, find the sum of the elements of the given array.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - sumArray({1,2,3})
    - 6
  - sumArray({-51,82,-20})
    - 11
  - sumArray({3,3,5,1})
    - 12
  - sumArray({1})
    - 1
  - sumArray({1,2,3,1,2,3,0,-23})
    - -11

Question 4 – Solution

```
public static int sumArray(int [] arr){

     int result = 0;

     for (int i = 0; i < arr.length; i++){

          result += arr[i];

     }

     return result;

}
```

Question 5 – Max Element Array

- Given an integer array arr, find the maximum value between the element of the given array.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - maxElement({1,2,3})
    - 3
  - maxElement({-51,82,-20})
    - 82
  - maxElement({3,3,5,1})
    - 5
  - maxElement({1})
    - 1
  - maxElement({1,2,3,1,2,3,0,-23})
    - 3
  - maxElement({-1,-2,-3})
    - -1

Question 5 – Solution

```
public static int maxElement(int [] arr){

    int result = Integer.MIN_VALUE;

    for (int i = 0; i < arr.length; i++){

        result = Math.max(arr[i], result);

    }

    return result;

}
```

Question 5, Expansion 1 – Min Element Array

- Given an integer array arr, find the minimum value between the element of the given array.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - minElement({1,2,3})
    - 3
  - minElement({-51,82,-20})
    - 82
  - minElement({3,3,5,1})
    - 5
  - minElement({1})
    - 1
  - minElement({1,2,3,1,2,3,0,-23})
    - 3
  - minElement({-1,-2,-3})
    - -1

Question 5, Expansion 1 – Solution

```
public static int minElement(int [] arr){

     int result = Integer.MAX_VALUE;

     for (int i = 0; i < arr.length; i++){

          result = Math.min(arr[i], result);

     }

     return result;

}
```

Question 5, Expansion 2 – Index of max Element Array

- Given an integer array arr, find the **index** of maximum value between the element of the given array. If there is tie return the first appearance.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - indexOfMaxElement({1,2,3})
    - 2
  - indexOfMaxElement({-51,82,-20})
    - 1
  - indexOfMaxElement({3,13,5,1})
    - 1
  - indexOfMaxElement({1})
    - 0
  - indexOfMaxElement({1,2,2,1,2,3,0,-23})
    - 5
  - indexOfMaxElement({-1,-2,-3})
    - 0

Question 5, Expansion 2 – Solution

```java
public static int indexOfMaxElement(int [] arr){

     int index = -1;

     int result = Integer.MIN_VALUE;

     for (int i = 0; i < arr.length; i++){

          if (arr[i] > result){

               result = arr[i];

               index = i;

          }

     }

     return index;

}
```

Question 6 – Is in Ascending Order

- Given an integer array arr, find if the array is ordered from lowest value to highest value.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - isInAscendingOrder({1,2,3})
    - true
  - isInAscendingOrder({-51,82,-20})
    - false
  - isInAscendingOrder({3,13,5,1})
    - false
  - isInAscendingOrder({1})
    - true
  - isInAscendingOrder({1,2,2,1,2,3,0,-23})
    - false
  - isInAscendingOrder({-1,-2,-3})
    - false
  - isInAscendingOrder({1,2,2})
    - true

Question 6 – Solution

```
public static boolean isInAscendingOrder(int [] arr){

      if (arr.length <= 1) {

            return true;

      }

      for (int i = 0; i < arr.length - 1; i++){

            if (arr[i] > arr[i + 1}){

                  return false;

            }

      }

      return true;

}
```

Recitation 4

Call By Value

ARE YOU COMING TO BED?
I CAN'T. THIS IS IMPORTANT.
WHAT?
Someone claims Java passes objects by reference
JUC

Arrays vs. int variables

```
int a = 7;

int b = a;

a = 8;

System.out.println(b);

// Prints 7
```

```
int[] a = {7};

int[] b = a;

a[0] = 8;

System.out.println(b[0]);

// Prints 8
```

Call By Value

- In Java, **all arguments are passed by value**, which means a copy of the value is passed to the method.
- For **primitive types** (e.g., int, double, char), the value itself is passed. Changes to the parameter inside the method do not affect the original variable outside the method.
- For **reference types** (e.g. arrays), the **reference** (memory address) is passed by value. While the reference itself is a copy, it still points to the same object in memory. Therefore:
  - If you modify the object that the reference points to, the changes will be reflected outside the method.

Call By Value with Primitive Types

```
public static void main(String[] args) {
    int i = 111;
    int j = 222;
    swap(i, j);
    System.out.println("i : " +  i + ", j : " + j);
    // i : 111 , j : 222
}

public static void swap(int i, int j) {
    int temp = i;
    i = j;
    j = temp;
}
```

Call By Value with Reference Types

```
public static void main(String[] args) {
    int[] nums = {111, 222};
    swap(nums, 0, 1);
    System.out.println("i : " +  nums[0] + ", j : " + nums[1]);
    // i : 222 , j : 111
}

public static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

Question 7 – Map into cubed Value

- Given an integer array arr, return an array of all values of the given array but raised to the power of 3.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - mapIntoCubed({1,2,3})
    - {1,8,27}
  - mapIntoCubed({-5,8,-20})
    - {-125,512, -8000}
  - mapIntoCubed({3,13,2,1})
    - {27, 2197, 8, 1}
  - mapIntoCubed({1})
    - {1}
  - mapIntoCubed({1,2,2,1,2,3,0})
    - {1,8,8,1,8,27,0}
  - mapIntoCubed({-1,-2,-3})
    - {-1,-8,-27}

Question 7 – Solution

```
public static int [] mapIntoCubed(int [] arr){

     int [] res = new int [arr.length];

     for (int i = 0; i < arr.length; i++){

          res[i] = arr[i] * arr[i] * arr[i];

     }

     return res;

}
```

Question 8 – Filter All words with length bigger than 3,

- Given a string array arr, return an array of all the strings whose length is equal or bigger than 3.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - filterStringsBigger3({"hello", "test", "hi"})
    - {"hello", "test"}
  - filterStringsBigger3({"hi", "cat", "apple", "no"})
    - {"cat", "apple"}
  - filterStringsBigger3({"dog", "sun", "sky"})
    - {"dog", "sun", "sky"}
  - filterStringsBigger3({"me", "to", "on"})
    - {}

Question 8 – Solution

```
public static String [] filterStringsBigger3(String [] arr){

      int count = 0;

      for (int i = 0; i < arr.length; i++){

            if (arr[i].length() >= 3) {

                  count++;

            }

      }

      int index = 0;

      String [] res = new String [count];

      for (int i = 0; i < arr.length; i++){

            if (arr[i].length() >= 3) {

                  count++;

            }

      }

      return res;

}
```

Question 8 – Alternative Solution

```
public static String [] filterStringsBigger3(String [] arr){
    String [] res = new String [countBiggerThan3(arr)];
    int index = 0;
    for (int i = 0; i < arr.length; i++){
        if (arr[i].length() >= 3){
            res[index] = arr[i];
            index++;
        }
    }
    return res;
}
```

```
public static int countBiggerThan3(String [] arr){
    int count = 0;
    for (int i = 0; i < arr.length; i++){
        if (arr[i].length() >= 3) {
            count++;
        }
    return count;
}
```

Question 8, Expansion 1 – Filter All words bigger length n,

- Given a string array arr, and an non negative integer n, return an array of all the strings whose length is equal or bigger than n.
- Note: you may assume that the arr.length >= 1.
- Examples:
  - filterStringsBiggerN({“hello”, “test”, “hi”}, 3)
    - {“hello”, “test”}
  - filterStringsBiggerN({"hi", "cat", "apple", "no"}, 4)
    - { “apple”}
  - filterStringsBiggerN({"hi", "cat", "apple", "no"}, 2)
    - {"hi", "cat", "apple", "no"}
  - filterStringsBiggerN({"dog", "sun", "sky"}, 3)
    - {"dog", "sun", "sky"}
  - filterStringsBiggerN({“new", "two", "onto”, “a”}, 0)
    - {“new", "two", "onto”, “a”}
  - filterStringsBiggerN({“new", "two", "onto”, “a”}, 10)
    - {}

Question 8, Expasion 1 – Solution

```
public static String [] filterStringsBiggerN(String [] arr, int n){

      int count = 0;

      for (int i = 0; i < arr.length; i++){

            if (arr[i].length() >= n) {

                  count++;

            }

      }

      int index = 0;

      String [] res = new String [count];

      for (int i = 0; i < arr.length; i++){

            if (arr[i].length() >= n) {

                  count++;

            }

      }

      return res;

}
```

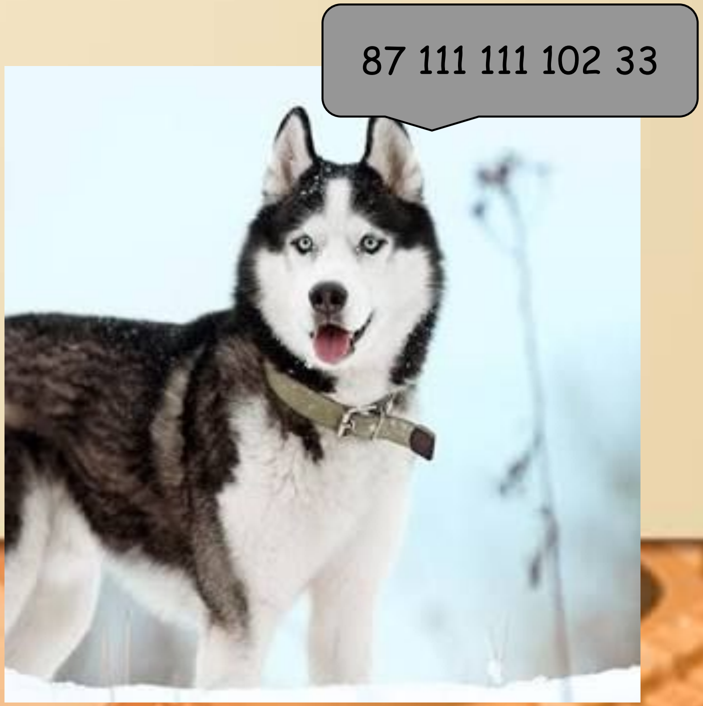
Question 8, Expasion 1 – Alternative Solution

```
public static String [] filterStringsBiggerThan(String [] arr, int n){
    String [] res = new String [countBiggerThan(arr, n)];
    int index = 0;
    for (int i = 0; i < arr.length; i++){
        if (arr[i].length() >= n){
            res[index] = arr[i];
            index++;
        }
    }
    return res;
}
```

```
public static int countBiggerThan(String [] arr, int n){
    int count = 0;
    for (int i = 0; i < arr.length; i++){
        if (arr[i].length() >= n) {
            count++;
        }
    return count;
}
```

Recitation 4

ASCII



Question 9 - stringToInt

- The following question was given at midterm 2019
- The function stringToInt receives a string that contains only digit characters ('0', '1', '2',…, '9'), and returns the integer value that the string represents. Example if the string "70352", the function returns the integer 70352. You may assume the range of the string's numerical values are in [-2^31, 2^31-1].
- Note: the function is very similar to Integer.parseInt() function, which you are not allowed to use in your code.
- Example:
  - stringToInt("72498") => 72498;
  - stringToInt("3892") => 3892;
  - stringToInt("0") => 0;

Implementation Tip: Use ASCII

Question 9 - Solution

```java
public static int stringToInt(String str){
     int sum = 0;
     for (int i = 0; i < str.length(); i++) {
          int digit = str.charAt(i) - '0';
          sum += digit * Math.pow(10, ((str.length() - 1) - i));
     }
     return sum;
}
```

Alternatively:

```java
public static int stringToInt(String str){
     int res = 0;
     int digit;
     for (int i = 0; i < str.length(); i++) {
          digit = str.charAt(i) - '0’;
          res = res * 10 + digit;
     }
     return res;
}
```

Question 9, Expansion 1 - stringToIntSign

- The function stringToIntSign receives a string that contains only digit characters ('0' , '1' , '2' ,…, '9'), except for the first character which can be either a digit or a sign ('+' , '-'), and returns the integer value that the string represents. Example if the string "70352", the function returns the integer 70352. You may assume the range of the string's numerical values are in [-2^31, 2^31-1].
- Note: the function is very similar to Integer.parseInt() function, which you are not allowed to use in your code.
- Example:
  - stringToIntSign("-234") => -234;
  - stringToIntSign("+234") => 234;
  - stringToIntSign("234") => 234;

Implementation Tip: Use ASCII

Question 9 Expansion 1 - Solution

```
public static int stringToIntSign(String str){
     int sign = str.charAt(0) == '-' ? -1 : 1;
     String noSignStr = str;
     if (str.charAt(0) == '+' || str.charAt(0) == '-') {
          noSignStr = str.substring(1);
     }
     return stringToInt(noSignStr) * sign;
}
```

Question 9, Expansion 2 - stringToDouble

- The following question was given at midterm 2022
- In the following question, you need to write the function 'stringToDouble' which receives a real number represented as a String and returns the same value as a double.
- The number will contain at least one digit. The number might have a sign in the first character ('-' or '+' only). Furthermore, the String might have the decimal point (represented by '.'), which will appear at most once.
- Note: The function 'stringToDouble' is a version of the known function 'Double.parseDouble(String str)'. You are not allowed to use that function or the function 'Integer.ParseInt(String str)'. You may use only the functions given in the helper pages and operators taught in the course so far. (You may assume everything will be calculated accurately).
- Note: You must use ASCII in order to convert digits into numbers

```java
public static void main(String [] args){

      System.out.println(stringToDouble("345"));            //345.0

      System.out.println(stringToDouble("+234.6") + 1.3);         //235.9

      System.out.println(stringToDouble("-234.126")));            //-234.126

      System.out.println(stringToDouble("-.788"));          //-0.788

      System.out.println(stringToDouble("345.88"));         //345.88

      System.out.println(stringToDouble("765."));           //765.0

}
```

Question 6, Expansion 2 - Solution

```
public static double stringToDouble(String str) {

      String numStrNoSign = str;

      int sign = str.charAt(0) == '-' ? -1 : 1;        // sign handle

      if (str.charAt(0) == '+' || str.charAt(0) == '-') {

            numStrNoSign = str.substring(1);      // removing sign

      }

      int dotIndex = numStrNoSign.indexOf('.');        // finding dot

      if (dotIndex == -1) {

                  // removing cases without dot

                  return sign * (stringToInt(numStrNoSign) + 0.0);

      }

      double sum = 0;

      // continue in the next slide
```

Question 6, Expansion 2 – Solution (Continue)

```
      // continued from the previous slide
      // sum before dot
      for (int i = 0; i < dotIndex; i++) {
            int digit = numStrNoSign.charAt(i) - '0';
                  sum += digit * Math.pow(10, ((dotIndex - 1) - i));
      }
      // sum after dot
      for (int i = dotIndex + 1; i < numStrNoSign.length(); i++) {
            int digit = numStrNoSign.charAt(i) - '0';
                  sum += digit * Math.pow(10, (dotIndex - i));
      }
      return sum * sign;
}
```

Question 6, Expansion 2 – Alternative Solution

```
public static double stringToDouble(String str) {
     int dotIndex = str.indexOf('.');           // finding dot
     if (dotIndex == -1) {
               // removing cases without dot
               return stringToIntSign(str);
     }
     if (dotIndex == str.length() – 1){
           // removing cases where dot is the last index
          return stringToIntSign(str.substring(0, str.length() – 1));
     }
     String wholeStr = str.substring(0, dotIndex);
     String decimalStr = str.substing(dotIndex + 1, str.length());
     // continue in the next slide
```

Question 6, Expansion 2 – Alternative Solution (Continue)

```
      // continued from the previous slide

      int whole = stringToIntSign(wholeStr);

      int decimalInt = stringToIntSign(decimalStr);

      double decimal = Math.pow(10, -decimalStr.length()) * decimalInt;

      if (whole < 0){

            return whole – decimal;

      }

      return whole + decimal;

}
```