Задачи по курсу программирования на Java

Как сдавать задачи

1. сообщите адрес электронной почты, чтобы я пригласил вас в dropbox и дал доступ для общей папки
2. создайте пакет, соответствующий вашей фамилии com.petrov.ivan и в него положите классы:

3. исправляйте ошибки в программах, пока я не поставил + в электронной таблице
4. зачтенные задачи удаляйте
5. после исправления моих комментариев, удаляйте комментарии, чтобы я мог отличить проверенную задачу от непроверенной
6. если я забыл про какую-то задачу, просьба напомнить

Uturn

Написать код, который переворачивает массив: {1,2,3,7} -> {7,3,2,1}

AverageAndMax

Написать код, который ищет среднее значение и максимум в массиве: {2,2,4,4} -> 3.0, 4

Biquadratic

Написать программу, которая решает биквадратное уравнение вида a*x*x*x*x+b*x*x+c=0

BubbleSort

Самостоятельно реализовать пузырьковую сортировку массива.

Primes

Простое число -- это такое, которое не делится ни на одно из других простых чисел. Напишите алгоритм для поиска простых чисел. Перебираем все числа от 1 до N и пытаемся делить на уже найденные числа (их храним в ArrayList). Если ни на кого не делится, то значит простое и его можно добавить в массив.

TreeMapSort

Реализовать сортировку массива с помощью TreeMap<Integer,Integer>. В TreeMap будут лежать пары из двух интов -- ключа и значения. Ключ -- число, которое встретилось в массиве, а значение -- количество раз, которое оно встретилось. Сначала собираем статистику, а потом, с помощью цикла по TreeMap выводим числа нужное количество раз. После работы программы, в массиве должны быть отсортированные данные.

MergeSort

Самостоятельно реализовать сортировку слиянием: входной массив делится на две половины. Каждая сортируется с помощью слияния. Далее две половины сливаются в одну с помощью двух индексов.

LongAdd

Есть две строки a = “1231231231231423” и b = “8888888880000059556665455”. Реализовать алгоритм, который будет складывать соответствующие числа. Результат также должен быть представлен в виде строки. Подсказка: посмотрите как вы делаете такое сложение на бумажке и заставьте компьютер делать то же самое. Алгоритм должен работать для чисел, которые не водят в Double и Long и не использовать BigDecimal.

LongestGrowing

Напишите код, который ишет в массиве int arr[]={1,2,3,2,1,3,1,2,6,7,8,9,10,11,0,-1} самую длинную непрерывную возрастающую подпоследовательность: {1,2,6,7,8,9,10,11}.

LongMult

Аналогично задаче 0.0.4, использовать то, что длинное число может быть представлено в виде: 123456789 = 789 + 1000*456 + 123*1000*1000

Spelling

Дан словарь voc = {“арбуз”, “абакан”, “абориген”, “лето”, …. } и строка с ошибками: txt = “Орбуз пишется совершенно не так!”. Написать программу, которая исправит все слова, которые отличаются на одну букву от словаря.

CalcBits

Написать функцию static int numOnes(int x){...}, которая вычисляет количество единиц в бинарном представлении числа x.

Amazon

Дан массив чисел a[]={1,2,3,4} и число s=7. Найти два числа из массива, которые в сумме дают s (в данном случае 7=3+4). Добиться, чтобы программа работала приемлемое время на огромных массивах (например длинной миллион).

BinarySearch

Самостоятельно реализовать бинарный поиск по значению в упорядоченном массиве.

Capitalizator

Написать программу, которая принимает строку и делает все слова заглавными:

“ЭТО настояшая пРоверка Букв” => “Это Настоящая Проверка Букв”

WordCounter

Написать программу, которая принимает строку, вытаскивае оттуда все слова и вычисляет частоту, с которой они встречаются. Например:

“Ленин всегда живой, ленин всегда со мной.”

должна дать следующую статистику:

ленин 2
всегда 2
со 1
мной 1
живой 1

упорядочить слова по частотам

RLE

1. Написать архиватор, который сжимает текст с большим количеством повторений букв. Например такой текст:

   ТТТТТТТТУУУУУУУТТТТТТТ ММММНННННООООГГООО
   ПОООООООВВВВВТТТТТООООРРООВВ

   должен быть сжат вот так:

   Т8У7Т7 1М4Н4О3Г2О3 1П1О6В5Т5О3Р2О2В2

2. Написать программу, которая разархивирует строки, созданные в предыдущей программе

RLEStream

Написать RLE архиватор/разарзиватор, который работает с файлами, не помещающимися в память.

FileComparator

Написать функцию int compare(File f1, File f2). Если файлы разной длины, то функция возвращает разницу длин. В противном случае -- считывает оба файла по байту и

возвращает разницу первых не совпавших байтов. Если файлы одинаковые -- функция должна вернуть 0.

RecursiveFileSearch

Написать функцию File[] getAllFiles(String path){...}, которая выдает массив, содержащий все файлы, находящиеся в пути path (включая подпапки). Использовать класс File и вызов функции из самой себя.

DuplicateFileFinder

Опираясь на задачу FileComparator и используя TreeSet написать функцию, которая ищет файлы-дубликаты в массиве File[]files. Результаты вывести на консоль:

[c:/a.txt, c:/a.bak] are duplicates
[c:/x/a.dat, c:/x/a2.dat,c:/x/a3.dat] are duplicates
…

HighCompress

Сжать/разжать строку написанную на языке из 53 букв, реализовать архиватор/разархиватор, который будет максимально плотно укладывать данные.

StackCalc

Нужно написать интерпретатор который исполняет программу стэкового для калькулятора, например такую:

PUSH 6
PUSH 5
PUSH 4
ADD
PRINT

в данной программе мы кладем в стэк 6, потом 5, потом сверху кладем 4. В результате там оказываются 3 числа 6,5,4. На верху стэка 4, т.к. его положили последним. Стэк можно себе представить в виде глубокой коробки, стоящей на полу. Вы можете класть только на самый верх этой коробки и брать элементы только с самого верха. Стэк уже реализован в классе Stack<Double> в java. В результате выполнения ADD программа

должна вытащить (stack.push()) два самых верхних значения из стэка и положить в замен них сумму 9. PRINT выдает значение стэка на экран (System.out.println(stack.toString()), в данном случае два числа: 6 и 9. Реализовать помимо этих следующие комманды:

POP -- вытащить самое верхнее число и написать его в консоль

MUL, DIV, SUB -- умножение, деление, сложение

DEFINE a 5 -- определяет константу a, равную 5, которая будет храниться в Map<String,Double> в калькуляторе. Далее константы можно использовать в операторе PUSH.

Калькулятор написать в первом приближении без многих классов с одной функцией main, внутри которой бесконечный цикл, читающий с консоли. Каждая строка, прочитанная с консоли, разделяется по пробелу с помощью String.split(“ ”). Нулевой элемент полученного массива -- имя комманды, первый (если есть) -- её аргумент. Далее делается перебор возможных комманд в блоках if-else и для каждой комманды исполняется свой код.

Во втором приближении внедрить в калькулятор ООП: для каждой команды сделать свой класс с методом exec(Map<String,Double> vars, String params[], Stack<Double> stack), где vars -- переменные, объявленные с помощью комманды DEFINE, params -- строка, разрубленная по пробелам, а stack -- стэк с которым мы работаем.

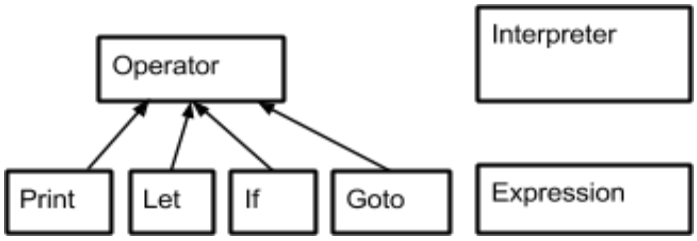
HashMap

На собеседованиях модно спрашивать о том, как работает HashMap. Предлагается сделать собственную реализацию Map с помощью хэш таблицы. По ключу, добавляемому в операторе put(Key k, Value v) вычисляется хэш код (.hashCode()). Хэш таблица представляет из себя массив объектов Object[] размером, равным 2^n, где n постепенно растет по мере увеличения числа ключей. Хэш код берется по модулю 2^n (оператор %), что дает адрес в массиве. Если в массиве в этой ячейке null, то туда кладется пара k,v (для этого создать вспомогательный класс Pair{..}), если там уже есть некоторый элемент, то вместо него кладем ArrayList<Pair> с двумя элементами -- старым и новым Pair. Если там уже ArrayList -- добавляем элемент в хвост ArrayList. Если в массиве или в ArrayList уже есть объект с равным ключом (k0.equals(k) == true), то старый объект затирается. Соответственно в операторе get у нашего HashMap смотрим на содержание массива. Если там Pair -- сравниваем ключи по equals и возвращаем pair.v, если они равны. Если там ArrayList, то перебором находим нужный ключ и возвращаем соответствующий value. Во время работы с map следим за длинной самого длинного ArrayList. Если переваливает за некоторое критическое значение -- удваиваем размер массива и перераспределяем значения из старого.

BASIC

Используя приёмы из задачи про стэковый калькулятор написать интерпретатор языка BASIC. Программа вычисления корней квадратного уравнения на BASIC выглядит примерно так:

```
10 LET a = 2
20 LET b = 1
30 LET c = 5
40 LET d = b*b-4*a*c
50 IF d>0 THEN GOTO 100
60 IF d=0 THEN GOTO 200
70 PRINT “NO ROOTS”
80 GOTO 300
100 LET x1 = (-b + SQRT(d))/2/a
110 LET x2 = (-b + SQRT(d))/2/a
120 PRINT “x1=”;
130 PRINT x1
140 PRINT “x2=”;
150 PRINT x2
160 GOTO 300
200 PRINT “x=”; -b/2/a
300 PRINT “PROGRAM FINISHED”
```



Во время парсинга программы идем по всем строкам входного потока. Каждую строку программы распиливаем по пробелам. Первый кусок -- номер строки. Его используем в качестве ключа в TreeMap<Integer,Operator> code. Второй кусок -- название оператора. Нам нужно реализовать простейшие: LET, GOTO, IF, PRINT. Метод Operator.execute(...) получает в качестве параметров экземпляр объекта Interpreter. При создании экземпляра оператора, в него передается оставшаяся часть строки (например для строки 200 это ”x=”; -b/2/a.

Объект Interpreter -- главный, в нем есть поле code, о котором сказано выше, там хранится распарсенная программа. TreeMap используется для того, чтобы можно было легко искать операторы по номеру строки (code.get(lineNumber)) и находить следующий элемент (можно с помощью TreeMap.iterator() и Iterator.hasNext()). Кроме того, в нем есть поле curLineNumber и методы Interpreter.nextLine() (его вызывают все операторы, кроме Goto в конце своей работы) и Interpreter.gotoLine(num) (он нужен для Goto). Распарсивание программы осуществляется в Interpreter.parse(InputStream is).

Кроме того, в Interpreter ещё есть Map<String, Double> vars. Её можно получить с помошью Interpreter.getVars(). Оператор Let складывает переменные в эту мэпу и берет их оттуда, чтобы вычислить выражение.

То, что стоит справа от знака равенства в операторе LET -- это выражение. Это комбинация констант, переменных и арифметических знаков. Вычислить выражение проще всего следующим образом: заменить все переменные на их значения с помощью String.replaceAll, а дальше использоровать движок js:

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByName("js");
Object result = engine.eval("3+4");
```

(можно написать вычисление выражений самостоятельно, но это дополнительная большая задача). Отдельно нужно обратить на функции, которые могут быть в выражении, например SQRT. Если в выражении есть подстрока SQRT, то можно заменить его на Math.sqrt, который есть в движке js. Логику работы с выражениями нужно поместить в класс Expression.

Оператор PRINT должен разделить свои аргументы по подстроке “;” и вывести их на экран. Если это выражение (то есть не строка, когда нет двойных кавычек), то использовать класс Expression. Для вычисления выражения внутри оператора IF также можно использовать js.

Не требуется написать идеальный интерпретатор языка BASIC, нужно сделать такой, в котором в большинстве случаев работают описанные выше операторы.

3.0 Консольная гостевая книга с использованием JDBC

Написать консольное приложение, которое позволяет добавлять, удалять и просматривать записи в гостевой книге. Достаточно сделать одну таблицу posts, в которой будут лежать сами записи. Подключиться к базе MySQL, найти в интернете код, который это делает.

Docx2HTML

1. docx файл это zip архив, найти в интернете код для открытия zip архивов в java
2. Основной текст документа хранится в word/document.xml, найти в интернете код для распарсивания xml с помощью SAX, применить его для того, чтобы вытащить текст из блоков <w:t>
3. Написать код, который генерирует из каждого <w:p> параграф HTML (<p>).

4. поддержать жирный шрифт (исследовать, играясь с word, какой тэг влияет на жирность)
5. Для особо упорных: поддержать таблицы
6. Избавиться от всех временных файлов в процессе работы программы.

LJ2fb2

распознавание текста

huffman coding

сжатие изображения

Google 0.0

*альфа-бета перебор, крестики нолики 3*3*

doom