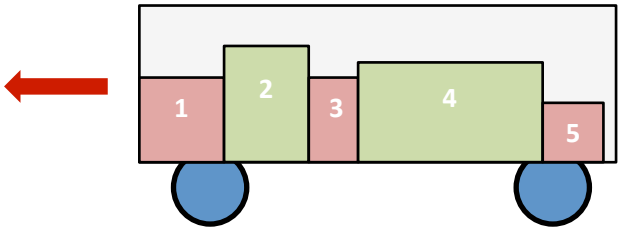
CSCI 135 Programming Exam #0
Fundamentals of Computer Science I
Fall 2013

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #0 dropbox. Please ***double check you have submitted all the required files***.

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. You will lose a substantial number of points if your code does not compile or crashes on typical inputs.

Overview. You are planning to move and are writing some software to help in your planning. You will be developing two programs `RandomBoxes.java` and `MoveBoxes.java`. The first program `RandomBoxes.java` generates random data about boxes that you need moved. The second program takes input of the form generated by `RandomBoxes.java` and simulates a simplistic moving process. It will calculate how many loads it will take with a truck of a given size and how many hours you'll spend driving. For the purposes of this problem, we assume a simplistic packing arrangement in the truck. We assume boxes take up the entire width of the truck and we never stack boxes on top of each other. So basically each box has only one dimension, namely its length. You have a single moving truck of a certain length.



Stub versions of these two programs as well as test files and `StdIn.java` can be downloaded from here: http://katie.mtech.edu/classes/csci135/moving.zip

Part 1, RandomBoxes: Your first task is to develop a program that generates random box data. Your `RandomBoxes.java` program should generate a specified number of boxes with each box appearing on a separate line. Each line of output should contain three whitespace separated fields (in this order):

1. ***Unique ID***, a string that uniquely identifies this box (e.g. "kitchen0")
2. ***Length***, positive integer length of box in feet
3. ***Value***, non-negative integer value of the box contents in dollars

Here are our two provided example data files:

```
% more kitchen5.txt
kitchen0 6 821
kitchen1 6 854
kitchen2 7 37
kitchen3 6 639
kitchen4 5 87
```

```
% more tools7.txt
tools0 6 64
tools1 10 766
tools2 6 668
tools3 2 15
tools4 1 951
tools5 5 259
tools6 3 711
```

Your `RandomBoxes` program should take four command-line arguments (in this order):

1. ***Box prefix***, a string value that is used as the prefix of all unique box IDs. A number starting at 0 and increment by 1 each time is appended to the prefix string to arrive at the unique box identifiers.
2. ***Number of boxes to generate***, a positive integer specifying how many boxes to generate.
3. ***Maximum box length***, a positive integer (denoted here `maxLength`) specifying the maximum length of a box in feet. All box lengths are reported in integer units. Each generated box has a random length drawn uniformly from the interval `[1, maxLength]`. That is, a box length is always at least `1` foot and can be up to and *including* `maxLength` feet.
4. ***Maximum value in dollars***, a non-negative integer (denoted here `maxValue`) specifying the maximum value in dollars. All box values are reported in integer units. Each generated box has a random value in `[0, maxValue]`. That is the box value can have a value between `0` and `maxValue` including the endpoints.

Here are some example runs:

```
% java RandomBoxes bedroom 10 15 200
bedroom0 8 26
bedroom1 15 15
bedroom2 7 35
bedroom3 12 26
bedroom4 4 40
bedroom5 2 22
bedroom6 12 99
bedroom7 10 36
bedroom8 1 111
bedroom9 6 191
```

```
% java RandomBoxes junk 8 3 5
junk0 1 1
junk1 2 2
junk2 2 0
junk3 1 5
junk4 3 2
junk5 3 1
junk6 1 4
junk7 2 4
```

Part 2, MoveBoxes: This program simulates a very simplistic moving process. Boxes are loaded in the exact order they appear in the input file. As soon as we find the next box would cause the total length of loaded boxes to exceed the truck length, we drive a load to the our new house. So for example, for the input file `kitchen5.txt`, we first load box `kitchen0`, then if that fits, `kitchen1`, then if it fits `kitchen2`, and so on.

An added complication is that our truck only has insurance coverage for a certain dollar value of cargo. If the next box would cause the total value of all boxes loaded in the truck to exceed the insurance limit, we send the truck on a roundtrip (without loading the box that would have put us over the limit).

`MoveBoxes.java` should take four command-line arguments (in this order):

1. ***Length of truck***, a positive integer specifying the length of the truck in feet. NOTE: You can assume no single box exceeds the length of the truck.
2. ***Insurance limit***, a positive integer specifying the maximum value you can transport in the truck. NOTE: You can assume no single box has a value exceeding the insurance limit.
3. ***Distance***, a positive floating-point value specifying the distance in miles between your old house and your new house. NOTE: this is the distance one-way, not round-trip!
4. ***Speed***, a positive floating-point value specifying your average driving speed in miles per hour.

Part 2a: Your first goal should be to read in the command-line arguments and print out the informative messages shown below. Note all floating-point values in `MoveBoxes.java`'s output should be rounded to two decimal places:

```
% java MoveBoxes 24 5000 45.5 60.0
Moving truck is 24 feet long.
Insurance limit per load is 5000 dollars.
Distance each way is 45.50 miles.
We drive 60.00 miles per hour.
```

Part 2b: After printing out the informative text, your program should then read in each line of box data from standard input. A message should be printed as each box is loaded. Here is a similar run to the previous one with differences highlighted in red:

```
% java MoveBoxes 24 5000 45.5 60.0 < kitchen5.txt
  Moving truck is 24 feet long.
  Insurance limit per load is 5000 dollars.
  Distance each way is 45.50 miles.
  We drive 60.00 miles per hour.

  Loading kitchen0
  Loading kitchen1
  Loading kitchen2
```

At any point, if the next box would cause the loaded boxes to exceed the length or insurance limit of the truck, you should send the truck on a roundtrip. So in the above output, the truck currently has 19 feet of boxes worth $1712. The next box, `kitchen3` is 6 feet long so would exceed the truck length of 24 feet. At this point, you should print a message about making a trip, the length of boxes carried on this trip, the dollar value of boxes carried on this trip, and the total elapsed driving time (all trips including this one):

```
  % java MoveBoxes 24 5000 45.5 60.0 < kitchen5.txt
  Moving truck is 24 feet long.
  Insurance limit per load is 5000 dollars.
  Distance each way is 45.50 miles.
  We drive 60.00 miles per hour.

  Loading kitchen0
  Loading kitchen1
  Loading kitchen2
  Making trip: 19 feet, 1712 dollars
  Total driving time: 1.52 hours
```

The truck is now empty again and the last two boxes can be loaded. So the full output would be:

```
  % java MoveBoxes 24 5000 45.5 60.0 < kitchen5.txt
  Moving truck is 24 feet long.
  Insurance limit per load is 5000 dollars.
  Distance each way is 45.50 miles.
  We drive 60.00 miles per hour.

  Loading kitchen0
  Loading kitchen1
  Loading kitchen2
  Making trip: 19 feet, 1712 dollars
  Total driving time: 1.52 hours
  Loading kitchen3
  Loading kitchen4
  Making trip: 11 feet, 726 dollars
  Total driving time: 3.03 hours
```

Here is another sample run, in this case we had to make several trips with a mostly unloaded truck due to a low insurance limit and the high value of our boxes:

```
% java MoveBoxes 16 1000 100.0 50.0 < tools7.txt
Moving truck is 16 feet long.
Insurance limit per load is 1000 dollars.
Distance each way is 100.00 miles.
We drive 50.00 miles per hour.

Loading tools0
Loading tools1
Making trip: 16 feet, 830 dollars
Total driving time: 4.00 hours
Loading tools2
Loading tools3
Making trip: 8 feet, 683 dollars
Total driving time: 8.00 hours
Loading tools4
Making trip: 1 feet, 951 dollars
Total driving time: 12.00 hours
Loading tools5
Loading tools6
Making trip: 8 feet, 970 dollars
Total driving time: 16.00 hours
```

If we were to buy higher insurance coverage, we could get this done a lot quicker:

```
% java MoveBoxes 40 20000 100.0 50.0 < tools7.txt
Moving truck is 40 feet long.
Insurance limit per load is 20000 dollars.
Distance each way is 100.00 miles.
We drive 50.00 miles per hour.

Loading tools0
Loading tools1
Loading tools2
Loading tools3
Loading tools4
Loading tools5
Loading tools6
Making trip: 33 feet, 3434 dollars
Total driving time: 4.00 hours
```