Lab 1 Directions and Problems

1. The instructor will present a brief overview of this week's new concepts and Maple features, and provide commentary on the lab required activities. (20-25 minutes)

   Notes
   • A verification sheet will be passed out for each group of lab partners.

2. With your lab partner(s), work on part 1, below. (60 minutes)
   • Form a group. Group members should introduce themselves to each other if they haven't already met.
   • All of the partners should log onto a computer.
   • Work read and attempt the problems individually, but stop frequently to compare results and help each other out. Eventually, the graders will be looking for a single collection of answers from the group that everyone in the group can explain. If you are in a room with computer projectors, then please prepare a single computer which can be used to present the group's results to the grader by projection. In most rooms, it's difficult for the grader to stare over your shoulder at results on a screen. Furthermore, the other members of the group also have to stare over a shoulder to see something that they may be asked to talk about.
   • Write down the selected answers on the verification sheet. When you are finished with part 1, have a staff member come over to sign the verification sheet for you. Be prepared to show your work to the staff member, and to explain how you got your answers. This is also the opportunity to clear up any questions or uncertainties you may have at this point.

Problems -- Part 1

Problem 1.1

In the download files provided for this lab is a copy of the project grade data file named KnexSmall.csv from a ENG 101 class from 2007. Read this file in as in cs121 using the Tools-> Assistants -> Import Data menu item. Use "delimited" format, and then select "anything" as the data type.

Assign the result to the variable KnexSmall using the assign to a name item of the right-click
(control-click on Macintosh) pop up menu of the worksheet. Then assign the variable `KnexSmallListList` the result of converting `KnexSmall` into "listlist" format. Finally, give this data a short name by assigning `KnexSmallListList` as the value of `T`.

You should see something like this:

```
13 x 4 Matrix
Data Type: anything
Storage: rectangular
Order: Fortran_order
```

assign to a name

```
13 x 4 Matrix
Data Type: anything
Storage: rectangular
Order: Fortran_order
```

```
KnexSmallListList := convert(KnexSmall, listlist)
[["Height", "Front", "Rear", "Corner"], [2, 3.19, 4.05, 3.11], [2, 3.7, 4, 3.91], [2, 4.12, 3.69, 4.23], [2, 1.07, 1.04, 1.03], [4.5, 4.87, 2.76, 8.52], [4.5, 4.77, 5.76, 5.29], [4.5, 10.06, 9.5, 10.18], [4.5, 12.7, 13.7, 12.5], [4.5, 9.96, 11, 12.9], [2, 3.23, 3.39, 3.54], [2, 3.32, 3.19, 3.36], [2, 3.87, 3.42, 3.79]]
```

```
T := KnexSmallListList :
```

Next, in a code edit region, enter this "tablePrint" routine, which is a modification of the one in Chapter 15.

```
tablePrint1 := proc(LL, numRows, numCols)
    local i, r, c;
    # first row is headers
    for i from 1 to numCols do
        printf("%10s", LL[1,i]);
    end do;
    printf("\n"); # end the line
    for r from 2 to numRows do
        for c from 1 to numCols do
            printf("%10.2f", LL[r,c]);
        end do;
        printf("\n"); # end the line
    end do;
end proc;
```
proc \( LL, num\text{Rows}, num\text{Cols} \)

\[
\text{local } i, r, c, \\
\text{for } i \text{ to } num\text{Cols} \text{ do } printf(”%10s", LL[1, i]) \text{ end do; } \\
printf(”\n”);
\]

\[
\text{for } r \text{ from 2 to } num\text{Rows} \text{ do} \\
\text{ for } c \text{ to } num\text{Cols} \text{ do } printf(”%10.2f”, LL[r, c]) \text{ end do; } printf(”\n”) \\
\text{end do}
\]

end proc

Test this procedure on \( T \), which you will try to print as a table with \( nops(T) \) rows and \( nops(T[1]) \) columns. These two numbers should be the number of items in the list (13), and the number of items in the first element of the list (4). Assuming that you have \text{tablePrint1} and \( T \) set up properly you should see something like this:

\[
n := nops(T)
\]

\[
m := nops(T[1])
\]

\text{tablePrint1}(T, n, m)

<table>
<thead>
<tr>
<th>Height</th>
<th>Front</th>
<th>Rear</th>
<th>Corner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>3.19</td>
<td>4.05</td>
<td>3.11</td>
</tr>
<tr>
<td>2.00</td>
<td>3.70</td>
<td>4.00</td>
<td>3.91</td>
</tr>
<tr>
<td>2.00</td>
<td>4.12</td>
<td>3.69</td>
<td>4.23</td>
</tr>
<tr>
<td>2.00</td>
<td>1.07</td>
<td>1.04</td>
<td>1.03</td>
</tr>
<tr>
<td>4.50</td>
<td>4.87</td>
<td>2.76</td>
<td>8.52</td>
</tr>
<tr>
<td>4.50</td>
<td>4.77</td>
<td>5.76</td>
<td>5.29</td>
</tr>
<tr>
<td>4.50</td>
<td>10.06</td>
<td>9.50</td>
<td>10.18</td>
</tr>
<tr>
<td>4.50</td>
<td>12.70</td>
<td>13.70</td>
<td>12.50</td>
</tr>
<tr>
<td>4.50</td>
<td>9.96</td>
<td>11.00</td>
<td>12.90</td>
</tr>
<tr>
<td>2.00</td>
<td>3.23</td>
<td>3.39</td>
<td>3.54</td>
</tr>
<tr>
<td>2.00</td>
<td>3.32</td>
<td>3.19</td>
<td>3.36</td>
</tr>
<tr>
<td>2.00</td>
<td>3.87</td>
<td>3.42</td>
<td>3.79</td>
</tr>
</tbody>
</table>

Now, create a new function \text{tablePrint2} that prints out the row number as the first item in each row, like this:

<table>
<thead>
<tr>
<th>Student no.</th>
<th>Height</th>
<th>Front</th>
<th>Rear</th>
<th>Corner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.00</td>
<td>3.19</td>
<td>4.05</td>
<td>3.11</td>
</tr>
<tr>
<td>2</td>
<td>2.00</td>
<td>3.70</td>
<td>4.00</td>
<td>3.91</td>
</tr>
<tr>
<td>3</td>
<td>2.00</td>
<td>4.12</td>
<td>3.69</td>
<td>4.23</td>
</tr>
<tr>
<td>4</td>
<td>2.00</td>
<td>1.07</td>
<td>1.04</td>
<td>1.03</td>
</tr>
<tr>
<td>5</td>
<td>4.50</td>
<td>4.87</td>
<td>2.76</td>
<td>8.52</td>
</tr>
<tr>
<td>6</td>
<td>4.50</td>
<td>4.77</td>
<td>5.76</td>
<td>5.29</td>
</tr>
<tr>
<td>7</td>
<td>4.50</td>
<td>10.06</td>
<td>9.50</td>
<td>10.18</td>
</tr>
<tr>
<td>8</td>
<td>4.50</td>
<td>12.70</td>
<td>13.70</td>
<td>12.50</td>
</tr>
<tr>
<td>9</td>
<td>4.50</td>
<td>9.96</td>
<td>11.00</td>
<td>12.90</td>
</tr>
<tr>
<td>10</td>
<td>2.00</td>
<td>3.23</td>
<td>3.39</td>
<td>3.54</td>
</tr>
<tr>
<td>11</td>
<td>2.00</td>
<td>3.32</td>
<td>3.19</td>
<td>3.36</td>
</tr>
</tbody>
</table>
In order to do this, you will need two more printf statements. One of them would be

```c
printf("%10s","Student no.");
```

The other would be

```c
printf("%10d", r-1); //r is off by one since the first row is headers.
```

The way to create `tablePrint2` is to copy the description of `tablePrint1`, and then to edit it to insert the two additional printf statements. The hard part of this is to decide where to insert them to get the right effect.

After you have defined `tablePrint2`, you should see this:

```
<table>
<thead>
<tr>
<th>Student no.</th>
<th>Height</th>
<th>Front</th>
<th>Rear</th>
<th>Corner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.00</td>
<td>3.19</td>
<td>4.05</td>
<td>3.11</td>
</tr>
<tr>
<td>2</td>
<td>2.00</td>
<td>3.70</td>
<td>4.00</td>
<td>3.91</td>
</tr>
<tr>
<td>3</td>
<td>2.00</td>
<td>4.12</td>
<td>3.69</td>
<td>4.23</td>
</tr>
<tr>
<td>4</td>
<td>2.00</td>
<td>1.07</td>
<td>1.04</td>
<td>1.03</td>
</tr>
<tr>
<td>5</td>
<td>4.50</td>
<td>4.87</td>
<td>2.76</td>
<td>8.52</td>
</tr>
<tr>
<td>6</td>
<td>4.50</td>
<td>4.77</td>
<td>5.76</td>
<td>5.29</td>
</tr>
<tr>
<td>7</td>
<td>4.50</td>
<td>10.06</td>
<td>9.50</td>
<td>10.18</td>
</tr>
<tr>
<td>8</td>
<td>4.50</td>
<td>12.70</td>
<td>13.70</td>
<td>12.50</td>
</tr>
<tr>
<td>9</td>
<td>4.50</td>
<td>9.96</td>
<td>11.00</td>
<td>12.90</td>
</tr>
<tr>
<td>10</td>
<td>2.00</td>
<td>3.23</td>
<td>3.39</td>
<td>3.54</td>
</tr>
<tr>
<td>11</td>
<td>2.00</td>
<td>3.32</td>
<td>3.19</td>
<td>3.36</td>
</tr>
<tr>
<td>12</td>
<td>2.00</td>
<td>3.87</td>
<td>3.42</td>
<td>3.79</td>
</tr>
</tbody>
</table>
```

assuming that $T$, $n$, and $m$ still have the values you previously assigned them in the session.

### Problem 1.2

Recall that this data was from a project where a "car" (actually a toy car built out of Knex) was rolled down an inclined and crashed into a wall. The impact forces for the car were measured. The car crash project's grade was computed in the following way:

a) Each car was rolled down either a 2" or 4.5" incline. Along the way, the maximum deceleration (g-force) was measured from three directions, from the front ($gf$), from the rear ($gr$), and from a corner ($gc$). The lower the deceleration number, the better.

b) For each car, the average of the three forces was used as the $r$-score ("r" for "raw") for each car.
\[ r_{gf, gr, gc} = \frac{(gf + gr + gc)}{3} \]  
(Note: this is ordinary math "textbook-ese" for the function \( r \), not Maple syntax.)

c) The mean (average) of the \( r \) scores for all the cars that used the 2 foot incline was computed. Call this \( r_2 \).

d) Similarly, the mean (average) of all the \( r \) scores involving the 4.5 foot incline were computed. Call this \( r_4 \).

e) The standard deviation of all the scores involving the 2 foot incline was computed. Call this \( s_2 \). The standard deviation is another statistical measure that is a measure of about the amount of spread around the mean. When the standard deviation of a collection of data is small, it means that most scores were close to the average. When it is large, it means that many scores were not that close, either high or low.

f) Similarly, the standard deviation of all the scores involving the 4.5 foot incline was computed. Call this \( s_4 \).

g) The derived score \( d \) for a car was based on the following formula:

\[ d = \begin{cases} 
85 + 10*\left( r_2 - r \right)/s_2 & \text{for cars used in the 2 foot drop} \\
85 + 10* \left( r_4 - r \right)/s_4 & \text{for cars used in the 4 foot drop} 
\end{cases} \]

We can unify the two cases by defining a function

\[ dScore(r, m, s) = 85 + 10 \cdot \frac{(m - r)}{s} \]

and say that

\[ d = d_{\text{score}}(r, r_2, s_2) \]  for cars used in the 2 foot drop

\[ = d_{\text{score}}(r, r_4, s_4) \]  for cars used in the 4 foot drop

Note that if a car's \( r \)-score was below the average (meaning less deceleration and therefore a softer/better crash), then \( r_2 - r \) (or \( r_4 - r \), if it were rolled down the other incline) would be positive, so \( d \) will be above 85.

h) The final grade \( f \) for a car was

\[ \begin{align*}
100 & \quad \text{for } d \geq 100 \ ("\text{no A++s}"") \\
d & \quad \text{for } 70 \leq d \leq 100 \\
70 & \quad \text{for } d \leq 70 \ ("\text{mercy rule}" )
\end{align*} \]

We want to compute the final grade for each student project.
We want to compute the \( r \)-score, \( d \), and the final grade for each car. While we can compute
the r scores from the information already recorded in the lists, in order to compute \( d \) and the final grades we first have to calculate means and standard deviations for the 2 and 4.5 foot drops.

First write a short script that will:

1. Define (using arrow notation) the function \( r \) defined in b above.
2. Initializes a table \( rValue \).
3. Run a for loop that computes r's value applied to row \( i \) of the data list, and stores it in \( rValue[i] \).
4. Calculates the average of all the \( r \) values for the 2 foot drop. Assigns this value to the variable \( r2avg \). In order to do this, you can initialize two variables \( r2 \) and \( twoCount \) to zero. These variables will be used in a for loop that goes from 2 to the number of rows in the table. The role of \( r2 \) is to accumulate (gather) the sum of all the \( rValues[i] \) where the \( i \)-th row contains data for a 2 foot drop. \( twoCounts \) role is to accumulate (gather) the count of the number of 2 foot drops. After the loop is over, \( r2avg \) can be assigned the value \( \text{evalf}(r2/twoCount) \).
5. Calculates the average of all the \( r \) values for the 4 foot drop in a similar way. Assigns this value to the variable \( r4avg \).

When you are running this script, you should print out a nice message of your average-finding results, for example:

Two foot average r value is: 3.250000.
4.5 foot average r value is: 8.964667.

We next have to compute the "mean" and "standard deviation" for the 2 foot and 4.5 foot data. We could write a script to do this but we will rely on Maple's built in functions for computing them.

To compute the mean of a list of values in Maple, we can use the built in function Statistics [mean]:

\[
\text{Statistics \{\text{Mean}\}}([[1, 3, 7]])
\]

\[3.666666667\]  \hspace{1cm} (1.2.2.1)

\[
\text{Statistics \{\text{StandardDeviation}\}}([[1, 3, 7]])
\]

\[3.055050463\]  \hspace{1cm} (1.2.2.2)

Thus, we should write a script to produce two tables, consisting of just the \( rValues \) for 2 foot drops, and \( rValues2 \), and \( rValues4 \).

Assign \( r2 \) and \( d2 \) the mean and standard deviation of the 2 foot data, and \( r4 \) and \( d4 \) the same for the 4.5 foot data. You should see the following for the results of the separate lists and the statistical calculations:

\[
[3.450000000, 3.869999999, 4.013333333, 1.046666667, 3.386666667, 3.290000000, 3.693333333]
\]

\[
[5.383333333, 5.273333333, 9.913333333, 12.96666667, 11.28666667]
\]
2 foot mean and standard deviation are: 3.250000, 1.006527.  
4.5 foot mean and standard deviation are: 8.964667, 3.491402.

**Problem 1.3**

Now initialize dScores as a table, and write a for loop that computes the d-value for each group. Convert the table into a list at the end to see if things have been computed reasonably. Call this list *dScoreList*. This result should be something like this:

\[ 83.01296905, 78.84020406, 77.41616521, 106.8904576, 95.25757996, 95.57263966, 82.28284880, 73.53755538, 78.34937621, 83.64219551, 84.60259381, 80.59541473 \]

**Problem 1.4**

Define a function *g* using a `proc... end` with an *if* statement in it, that takes as an input a score and returns a result that is the final mark for the project according to the rule defined in h) in Problem 1.2. With this function developed and tested (devise your own tests). Write a short script that uses a loop to fill a table *grade*, where *grade[i]* has the grade of the group corresponding to row *i* of the original data. Convert the table into a list and assign it to *glist*. You should get results that look roughly like this:

\[ 83.01296905, 78.84020406, 77.41616521, 100, 95.25757996, 95.57263966, 82.28284880, 73.53755538, 78.34937621, 83.64219551, 84.60259381, 80.59541473 \]

**Part 2 -- Problems**

**Problem 2.1**

Having computed the numbers in Part 1, we now take additional steps to analyze them.

Create a procedure `tablePrint3` that prints out the data, the r scores, dscores, and final mark. In order to get everything to fit on a line, you may have to reduce some of the columns to be 8 characters wide instead of 10.

When you are finished, you should see something like this:
Problem 2.2

What has gone on before was a test of our computational procedures on a small amount of data.

Now we will use the programming you built on the real data, which is in KnexLarge.csv.

In a new document (use File -> New -> Document), read in KnexLarge.csv. Convert it to a list of lists and call it KnexLargeListList.

Then, do $T := \text{KnexLargeListList}$;

Combine all of your work for Problems 1.2-1.5 into a single code edit region. Assign $T$ the value KnexLargeListList and execute the code region. If you have done things properly, it should execute without a hitch and print out the grades for the entire class using $\text{tablePrint3}$. The results from that should start off looking like this:

<table>
<thead>
<tr>
<th>Student No.</th>
<th>Height</th>
<th>Front</th>
<th>Rear</th>
<th>Corner</th>
<th>rValue</th>
<th>dScore</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.00</td>
<td>3.19</td>
<td>4.05</td>
<td>3.11</td>
<td>3.45</td>
<td>83.01</td>
<td>93.76</td>
</tr>
<tr>
<td>2</td>
<td>2.00</td>
<td>3.70</td>
<td>4.00</td>
<td>3.91</td>
<td>3.87</td>
<td>78.84</td>
<td>92.74</td>
</tr>
<tr>
<td>3</td>
<td>2.00</td>
<td>4.12</td>
<td>3.69</td>
<td>4.23</td>
<td>4.01</td>
<td>77.42</td>
<td>92.74</td>
</tr>
<tr>
<td>4</td>
<td>1.07</td>
<td>1.04</td>
<td>1.03</td>
<td>1.05</td>
<td>106.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4.50</td>
<td>4.87</td>
<td>2.76</td>
<td>8.52</td>
<td>5.38</td>
<td>95.26</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4.50</td>
<td>4.77</td>
<td>5.76</td>
<td>5.29</td>
<td>5.27</td>
<td>95.57</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4.50</td>
<td>10.06</td>
<td>9.50</td>
<td>10.18</td>
<td>9.91</td>
<td>82.28</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4.50</td>
<td>12.70</td>
<td>13.70</td>
<td>12.50</td>
<td>12.97</td>
<td>73.54</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4.50</td>
<td>9.96</td>
<td>11.00</td>
<td>12.90</td>
<td>11.29</td>
<td>78.35</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2.00</td>
<td>3.23</td>
<td>3.39</td>
<td>3.54</td>
<td>3.39</td>
<td>83.64</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2.00</td>
<td>3.32</td>
<td>3.19</td>
<td>3.36</td>
<td>3.29</td>
<td>84.60</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2.00</td>
<td>3.87</td>
<td>3.42</td>
<td>3.79</td>
<td>3.69</td>
<td>80.60</td>
<td></td>
</tr>
</tbody>
</table>
Problem 2.3

Now that we have calculated grades for the class, we can do some analysis. Using the KnexLarge data, answer the following questions:

a) What is the mean grade for the class?
b) What is the median grade for the class? (Hint: the Statistics package has a median function, too. Look it up in the on-line help.
c) What is the standard deviation of the grade for the class?

d) Do you think that there's a significant difference between the means and standard deviations for the 2 foot drop and the 4.5 foot drop? (You should design and implement the technique that will separate the grades for the two kinds of drops in order to do this.)

Problem 2.4

As a final analysis on the grades, we will plot rank versus score, where the 2 foot scores are plotted in green, and the 4 foot scores in red. We will draw a horizontal line at the mean value for the class. The result should be something like this:
Do this in several steps.

**Step 1:** Create a new list of lists, `glistlist`, whose ith element is a list of two numbers of the form `[height, grade[i+1]]`.

**Step 2:** Sort `glistlist` using the second element of each item as determining value. In order to do this you should build a function `sortpred(La, Lb)` which returns true if `La[2] > Lb[2]`, false otherwise. See the sorting Example 16.5.2. Call the sorted version of `glistlist`, `gsorted`.

**Step 3:** Write a short script to separate the results of `gsorted` into two tables, one containing values of the form `[rank, grade]` for the 2 foot items, and the other table containing values of the same form for the 4.5 foot items.

**Step 4:** Create three plot structures, one a red point plot, one a green point plot, and one with a plot of just the mean line.

**Step 5:** Use `plots[display]` to display all three together.

After looking at the graph, what do you find unusual or noteworthy about the distribution of grades? Did one group appear to do better than the other? Does the distribution of grades work the way you think it should? If not, is there a different "grade formula" that would reflect a better distribution of grades? Why would it be better?
3. Hand in the signed verification sheet. You don't get credit for the lab without doing this. The lab staff will give you permission to leave the lab after you do this. You can get partial credit for the lab if a portion of your work is verified.

**Notes**

Final grades for the course will be curved if necessary, so don't fret excessively if you don't finish but it looks like others are in the same shape. However, you should try to learn the material you don't complete in lab so that you can pass the quiz and be ready for the next lab. Computer work at this introductory level introduces a lot of ideas and concepts that appear pervasively in subsequent work. The plus side is that you'll probably see next time more of what you worked on this time, so you'll have another chance to practice and improve. The down side is that you can't ignore tough details and hope that little will depend on mastering them in the future.