Name: Andrew Brown	**Lab Time:** T 12:00
People Worked With: Kevin Daellenbach	**Websites Used:**
Time spent on zyBooks (hrs):2	**Time spent on lab (hrs):4**
Submission Instructions	
Turn all work in to Lab 8 on Gradescope (PDF) and Canvas (.zip file), even if it is not complete yet. If you are not finished, complete the assignment outside of lab and re-submit to Lab 8 on Gradescope and Canvas. All labs are typically due at the same time on Monday every week, but check Canvas if in doubt.	

Learning objectives:

- Solve a simple optimization problem
- More practice with string editing
- More practice with reading and writing to files

New MATLAB commands

These are highlighted in **bold** in the instructions below.

- `myVariable = struct;` – Say you're going to make a struct
- `myVariable.myField = (can be anything)` – create a field named `myField` on the struct and assign it whatever you have on the right. This is almost exactly the same as
  - `myField = (can be anything)`
  - except that it makes the variable in `myVariable`
- `rand( 1 )` – return a random number between 0 and 1

Lab Problems

Files to download to your Lab8 Folder

- *Download the function files and data file from lecture*
- *GetCandy, AddCandy, PrintCandy, CalculateCandyCosts, data file CandyData.txt*
- *Also download Lab8CheckPick.m and MyPickCandy.m*

Getting Started

Make sure you can read and print the candy data out

- Try this on the command line
  - *Make sure you're in the directory with the GetCandy.m script and data file*
  - `candy = GetCandy('CandyData.txt')`
  - `PrintCandy(candy)`

Problem 1

Add a candy "goodness" row to the CandyData.txt file. This is a number between 1 and 10 that says how much you like that candy. Also add in a few more of your favorite candy types.
Alter GetCandy and AddCandy so that you can read and store the new data.
For this problem, as in the last lab, check your solution by calling it from the command line. We'll call it from a script in problem 3.

Deliverables:

1. Modified function file for GetCandy
    a. Read in a second row of numeric data (the goodness)
    b. Pass the values from the second row into AddCandy
2. Modified function file for AddCandy
    a. Should take in another variable – candyGoodness
    b. Should add in another field in the struct to store that value
3. The modified data file with the goodness row and at least one new column
4. Show calling your GetCandy file with your new data file

Step by Step Instructions:

- First check that you can read in the existing data file and it works
    - candy = GetCandy('CandyData.txt')
- Edit the CandyData.txt by adding a candy type of your own
- Check that it still works
- Add in a row to the CandyData.txt with goodness values
- Check that it still works
    - Note: It will work because the original script is just going to ignore those extra values
- First modify GetCandy so that it extracts the second row of data as well as the first
    - Put a breakpoint after the csvread in GetCandy. Note that this is a 2xn array – it's read the data, you just need to get it out
    - Modify the for loop to loop over the number of columns in candyCosts
        - Note: length works because (at least for right now) the number of columns is the bigger of the two dimensions. But size is the right command here
    - Change the call to AddCandy to get out the kth element of the first row
- Now modify AddCandy so that it takes in another parameter (candy goodness)
    - Pretty much just copy and paste what was done for candyCost, then change the name(s)
- Go back and modify GetCandy so that it passes the kth element of the second row (the goodness) as well as the cost

Self-check: *Use your own values here, but it should look like:*

```
>> candy = GetCandyModified( 'CandyDataModified.txt' )

candy =

  struct with fields:

        CandyName: {'Rock candy'  'Gum'  'Lollipop'  'Taffy'  'Smarties'}
        CandyCost: [35 5 10 3 18]
      CandyAmount: [0 0 0 0 0]
    CandyGoodness: [6 1 1 7 8]
    numCandyTypes: 5

>>

----------------- This my CandyDataModified.txt ------------
Rock candy, Gum, Lollipop, Taffy, Smarties
35, 5, 10, 3, 18
6, 1, 1, 7, 8
```

Grading Criteria:

[10 pts] [Added at least one column to candy data file]
[10pts] [Added row of goodness to candy data file]
[10pts] [Called GetCandy with modified file]
[20 pts] [AddCandy adds in another field (goodness)]
[20 pts] [GetCandy gets both data rows from the file]
[30 pts] [GetCandy sets the new field]

Function scripts here:

```
function [ candyList ] = AddCandy( candyList, strCandyName, candyCost,
candyGoodness)
%AddCandy Add a candy type to the candy struct
%   Make sure to add in one element to each array
%   This is one of the few times when the input and the output
%   should be the same because you're adding to the struct

%% LAB TIP: You will end up adding another input parameter called
candyGoodness to this function.
%   To make this work you will also need to update what happens in the if
%   else statements below

% First check to see if we have a list yet in the struct
%  If so, just add it in, otherwise need to make the field
%     isfield asks if there is a field called CandyCost in the struct

%% This branch will execute if the structure candyList is defined.
%   Values will be added to each field using the array [] operator.
if isfield( candyList, 'CandyCost')
```

```
    candyList.CandyName = [ candyList.CandyName, strCandyName ];
    candyList.CandyCost = [ candyList.CandyCost, candyCost ];
    candyList.CandyGoodness = [candyList.CandyGoodness, candyGoodness];
    candyList.CandyAmount = [ candyList.CandyAmount, 0 ]; % none, yet
    %% LAB TIP: Add another line here that will add a candyGoodness value to
an
    % existing field called CandyGoodness using the [] operator.

%% This branch will execute if the structure candyList is not yet defined.
%   New fields will be created and edited.
else
    candyList.CandyName = cell(1,1);    % Create a field named CandyName
    candyList.CandyName{1} = strCandyName; % Add a name to the field named
CandyName
    candyList.CandyCost = candyCost; % Create a field named CandyName
    candyList.CandyAmount = 0;
    candyList.CandyGoodness = candyGoodness;
    %% LAB TIP: Be sure to create the CandyGoodness field and assign a value
to it here.
end

% Just count the amount of names so that you can save the number of candies
in the structure.
candyList.numCandyTypes = length( candyList.CandyName );

end
```

```
function [ candy ] = GetCandy( filename )
%GetCandy Read in and set up candy from file
%   Get headers, costs, goodness

% Going to store our candy in a struct
%  We will be adding one array for candy name (note that this is a cell
%  array), amount, and cost.
%       LAB TIP: Later you will modify the code to work with another array
of
%       goodness values (a numeric rating from 1-10).
candy = struct;

%% Read in the headers and the amount of candies from a file
%   file has candy name as headers; the second row is the costs
%   csvread(name, rowstart, columnstart)
fid = fopen(filename, 'r' );
% Get all the data as a cell array
strCandies = textscan(fid, '%s', 'Delimiter',',');
strCandies = strCandies{1};
fclose(fid);

%% Try adding a break point here so that you can see how the data is stored.
%   LAB TIP: Doing this will help you locate the candyGoodness values.
candyData = csvread(filename, 1, 0 );
```

```
% Use this function to add in each candy type - this function makes sure
% that all of the parts of the struct are set up correctly
for k = 1:length( candyData )
    % LAB TIP: Notice that we are just indexing data from the first row
right now.
    % Once AddCandy() gets modified it will need to take a new parameter
from
    % the second row in candyData.
    candy = AddCandy( candy, strCandies{k}, candyData(1,k), candyData(2,k));
end

end
```

Data file here:

```
Rock candy, Gum, Lollipop, Taffy, Chocolate bar, Nerds, Butterscotch
32, 15, 10, 3, 25, 8, 5
3, 6, 5, 7, 10, 9, 9
```

Command window output

```
candy=GetCandy('CandyData.txt')

candy =

  struct with fields:

        CandyName: {1×7 cell}
        CandyCost: [32 15 10 3 25 8 5]
      CandyAmount: [0 0 0 0 0 0 0]
    CandyGoodness: [3 6 5 7 10 9 9]
    numCandyTypes: 7
```

Problem 2

Be greedy – prefer to buy candies that you like over ones that you don't, particularly ones that are cheap. But you don't want all of the same kind…. Here we're just going to write a function that implements the "preferred cheap" part.

Deliverables:

5. Calculate a "rank" score for each candy based on its price and goodness
    a. Briefly describe what you tried to do
    b. Print out the candy and the ranks to the command line
6. Create a function that randomly picks a candy based on this rank score
7. Test that the randomly-pick function is working by first modifying and then running Lab8CheckPick.m with your function
    a. Command-window output from that function

Step by Step Instructions:

2a) Make the rank score

- Create your own ranking values. They can be anything you think of as reasonable, just so long as the rank goes up if you like the candy better (same cost) and the rank also goes up if you like the candy the same, but it's cheaper.
    - This should be a calculated value – don't just manually set rank values
    - Store them in the candy structure (the script already does that)
    - You can write a function to set the rank values if you want, but you don't have to.
- To hand in:
    - Write an English description of what you were trying to do with your rank score in the section "Reasoning behind rank score"
    - Copy the script with the equation into the Answer script section
    - Print your struct out on the command line and copy the output to the

2a) Make the function that picks the candy randomly based on rank score

- MyPickCandy is a function that (currently) takes in the rank score (as an array) and returns a number from 1 to length(array). Right now, it returns each number with equal likelihood. Modify this so that it returns higher-ranked candy items more often.
    - This is something called histogram sampling; it's really just a fancy version of rolling a die or flipping a coin to randomly pick an item. Refer to lecture notes, but the brief description is: suppose you had three candies with rank values of 1 and 2 and 4. Then you should pick the first candy $1/7^{th}$ of the time, the second candy 2/7ths of the time, and the last candy 4/7ths of the time.
    - **How** to do this:
        - Sum up all of your rank values – call it sumRank

        - Generate a uniformly distributed number between zero and sumRank
        - See which bin you lie in
            - i.e., for the example above your bins would be
                - 0 0+1 0+1+2 0+1+2+4
                - which is: 0 to1, 1 to 3, and 3 to 7
            - If your random number is between 0 and 1, return 1
            - If your random number is between 1 and 3, return 2
            - If your random number is between 3 and 7, return 3
                - i.e., if your random number is between a(k) and a(k+1), return k
        - Double check that you've handled the case of your random number being zero or sumRank
    - **What** to do:
        - Modify MyPickCandy.m to use the rank values instead of just picking
- Check that you implemented your function correctly using Lab8CheckPick.m
    - You should get values that are "close", but not exactly the same as, your original rank values. It's like flipping a coin – you'll get close to 50% heads, but not quite
- To hand in:
    - The modified MyPickCandy.m script
    - The command window output from running Lab8CheckPick.m

Self-check: Note that your rank scores will be different… but the CandyRank should reflect both CandyCost and CandyGoodness – in the example below my favorite candy is the last but it costs more, so the 2nd to last candy (which I also like but is cheaper) has the highest rank

```
Part a: You do NOT need to match the rank values/goodness values I've given.
They should depend on what you've set them to be.
>> candy

candy =

        CandyName: {1x5 cell}
        CandyCost: [35 5 10 3 18]
      CandyAmount: [0 3 1 1 4]
    CandyGoodness: [6 1 1 7 8]
    numCandyTypes: 5
        RankCandy: [6 7 3.5000 81.6667 15.5556]

Part b:
>> Lab8CheckPick

  6     1     1     7     8

    5.9931    1.0136    0.9757    7.0242    7.9934

>>
```

Grading Criteria:
[20 pts] [Create a rank score from goodness & cost]
[10 pts] [Function file for picking a candy]
[30 pts] [Picking candy uses rank to pick candy]
[10 pts] [Checked that it worked with Lab8CheckPick]
[30 pts] [It actually worked…]

Answer script here:

```
%% Script to check your PickCandy function
%    Change the "CHANGE HERE" bits
%
%  Key matlab concepts
%    Structs - how to save data in one "variable"
%    Optimization - how to iteratively improve on something
%    More practice with functions, program flow

clear
clc

% Read in and setup candy
% CHANGE HERE: Make sure this is your function and your data file
candy = GetCandy('CandyData.txt');

cost = candy.CandyCost;
goodness = candy.CandyGoodness;
rank = candy.CandyRank;

%% The code to do the checking
% Basically, call your function many, many times and see if it creates the
% same probability/rank values
candyPicked = zeros( size( rank ) );
for k = 1:100000
    %% CHANGE HERE: change the function MyPickCandy
    %     See instructions in MyPickCandy
    picked = MyPickCandy( rank );
    % Record which one was picked
    candyPicked( picked ) = candyPicked( picked ) + 1;
end

% Normalize and set to same range of values as rank
candyPicked = sum( rank ) .* candyPicked ./ sum( candyPicked );

%% CHECK HERE: The current version of MyPickCandy should
%   produce roughly equal numbers (i.e., something like 3.1 2.9 3.05)
% Once MyPickCandy is correct these two arrays should be
%   fairly similar (i.e., something like 3 2 4 and 3.10 1.9 4.1)
disp( candy.CandyRank );
disp( candyPicked );
```

Function scripts here:

```
function [ candy ] = GetCandy( filename )
%GetCandy Read in and set up candy from file
```

```
%   Get headers, costs, goodness

% Going to store our candy in a struct
%  We will be adding one array for candy name (note that this is a cell
%  array), amount, and cost.
%       LAB TIP: Later you will modify the code to work with another array
of
%       goodness values (a numeric rating from 1-10).
candy = struct;

%% Read in the headers and the amount of candies from a file
%   file has candy name as headers; the second row is the costs
%   csvread(name, rowstart, columnstart)
fid = fopen(filename, 'r' );
% Get all the data as a cell array
strCandies = textscan(fid, '%s', 'Delimiter',',');
strCandies = strCandies{1};
fclose(fid);

%% Try adding a break point here so that you can see how the data is stored.
%   LAB TIP: Doing this will help you locate the candyGoodness values.
candyData = csvread(filename, 1, 0 );

% Use this function to add in each candy type - this function makes sure
% that all of the parts of the struct are set up correctly
for k = 1:length( candyData )
    % LAB TIP: Notice that we are just indexing data from the first row
right now.
    % Once AddCandy() gets modified it will need to take a new parameter
from
    % the second row in candyData.
    candy = AddCandy( candy, strCandies{k}, candyData(1,k), candyData(2,k),
candyData(2,k)./candyData(1,k));
end

end



function [ candyList ] = AddCandy( candyList, strCandyName, candyCost,
candyGoodness, candyRank)
%AddCandy Add a candy type to the candy struct
%   Make sure to add in one element to each array
%   This is one of the few times when the input and the output
%   should be the same because you're adding to the struct

%% LAB TIP: You will end up adding another input parameter called
candyGoodness to this function.
%   To make this work you will also need to update what happens in the if
%   else statements below

% First check to see if we have a list yet in the struct
%  If so, just add it in, otherwise need to make the field
%     isfield asks if there is a field called CandyCost in the struct
```

```
%% This branch will execute if the structure candyList is defined.
%   Values will be added to each field using the array [] operator.
if isfield( candyList, 'CandyCost')
    candyList.CandyName = [ candyList.CandyName, strCandyName ];
    candyList.CandyCost = [ candyList.CandyCost, candyCost ];
    candyList.CandyGoodness = [candyList.CandyGoodness, candyGoodness];
    candyList.CandyAmount = [ candyList.CandyAmount, 0 ]; % none, yet
    candyList.CandyRank = [ candyList.CandyRank, candyRank ];
    %% LAB TIP: Add another line here that will add a candyGoodness value to
an
    % existing field called CandyGoodness using the [] operator.

%% This branch will execute if the structure candyList is not yet defined.
%   New fields will be created and edited.
else
    candyList.CandyName = cell(1,1);    % Create a field named CandyName
    candyList.CandyName{1} = strCandyName; % Add a name to the field named
CandyName
    candyList.CandyCost = candyCost; % Create a field named CandyName
    candyList.CandyAmount = 0;
    candyList.CandyGoodness = candyGoodness;
    candyList.CandyRank = candyList.CandyGoodness./candyList.CandyCost;
    %% LAB TIP: Be sure to create the CandyGoodness field and assign a value
to it here.
end

% Just count the amount of names so that you can save the number of candies
in the structure.
candyList.numCandyTypes = length( candyList.CandyName );

end



function  [cost] = CalculateCandyCost( candyList )
%% Add up the current cost of my candy
% candyList is a struct of the form
%   List of candy types (strings)
%   List of candy costs (numbers)
%   List of number of each candy

% Element-wise multiplication
cost = sum( candyList.CandyCost .* candyList.CandyAmount );

end
```

Reasoning behind rank score here:

I tried to make the rank increase with goodness and decrease with cost, so I made the ranking equal goodness over cost.

Command window output, struct with rank field

```
candy

candy =

  struct with fields:

        CandyName: {1×7 cell}
        CandyCost: [32 15 10 3 20 8 5]
      CandyAmount: [0 0 0 0 0 0 0]
    CandyGoodness: [3 5 4 6 10 9 9]
        CandyRank: [1×7 double]
    numCandyTypes: 7
```

Command window output, running Lab8CheckPick

```
Columns 1 through 6

    0.0938    0.3333    0.4000    2.0000    0.5000    1.1250

  Column 7

    1.8000

  Columns 1 through 6

    2.9989    3.2531         0         0         0         0

  Column 7

         0
```

Problem 3

Actually produce candy amounts that is under the target goal of $1 AND meets your preferences.

Extra credit: Don't just stop when the next candy choice would put you over the limit – try adding a few more.

Deliverables:

8. Script that implements a greedy algorithm that tends to pick candies you rank highly
9. Run it a few times and output the result each time

Step by Step Instructions:

- Start with reading the candy in and copying your RankCandy calculation from the previous problem
- Two ways to do this – do a "look ahead" to see if the next candy pick will break the bank (this is what the lecture does) and if so, don't add that candy in, just stop the loop
    - Easiest to do this with a bKeepTrying Boolean variable
        - bKeepTrying = true
        - while bKeepTrying
            - Set bKeepTrying to be false if next one would go over limit
- Second method is to just save the last valid set of amounts and stop when cost is too high
    - while CalculateCandyCost(candy) < target
        - saveBestCandy = candy;
        - Increase one of the candy amounts
    - print out saveBestCandy
- Extra credit: Start with method 1, but use a "triedNTimes" variable. Stop when you've tried (unsuccessfully) too many times (maybe 10?) to add in another candy

Self-check:
Run 1:

```
Current cost of candy: 99.00
Candy        Cost   Num    Total
Rock candy   35.00  0      0.00
Gum          5.00   0      0.00
Lollipop     10.00  0      0.00
Taffy        3.00   3      9.00
Smarties     18.00  5      90.00
```

Run 2:

```
>>
Current cost of candy: 100.00
Candy        Cost   Num    Total
Rock candy   35.00  1      35.00
```

```
Gum          5.00   1      5.00
Lollipop     10.00  0      0.00
Taffy        3.00   14     42.00
Smarties     18.00  1      18.00
```

Grading Criteria:
[10 pts] [Sets up candy by reading in from file]
[10 pts] [Sets up CandyRank]
[20 pts] [Uses function from problem 2 to pick next candy]
[20 pts] [Cost is less than (or equal to) target cost]
[20 pts] [Gets as close as possible]
[20 pts] [Prints out results from 2 runs]
[EC 20 pts] [Tries a few more times before declaring done]

Answer script here:

```
% Copy and paste your script here. Make sure your formatting looks the same
as MATLAB, with size 10 font.
```

Command window output

```
Paste output here.
```

Extra Credit: Problem 1

Keep adding and removing candies until you're close to the target or you haven't improved the answer. Essentially the second optimization from lecture, except use your PickCandy function to pick which candy to add (and remove).

Step by Step Instructions:

- Start with the second optimization script from lecture
- Replace the top part with reading in your own data and setting the rank function
- First replace the part where you pick the candy if you're under cost
    - Call PickCandy instead
- Next replace the part where you pick the candy you want to remove
    - You can actually use PickCandy again, but you need a rank score for each candy that is high if you don't like the candy or it costs a lot… exactly how you calculate this is up to you, just do something reasonable
- Print out your rank scores and your reversed rank scores and include them below

Self-check: *Again, the actual numbers you get will vary from run to run*

```
Best answer found
Current cost of candy: 101.00
Candy        Cost   Num    Total
Rock candy   35.00  0      0.00
Gum          5.00   1      5.00
Lollipop     10.00  3      30.00
Taffy        3.00   16     48.00
Smarties     18.00  1      18.00

Best answer found
Current cost of candy: 101.00
Candy        Cost   Num    Total
Rock candy   35.00  1      35.00
Gum          5.00   0      0.00
Lollipop     10.00  0      0.00
Taffy        3.00   10     30.00
Smarties     18.00  2      36.00

>> candy.RankCandy

ans =

    6.0000    7.0000    3.5000   81.6667   15.5556

>> rankInverse

rankInverse =

   79.1667   78.1667   81.6667    3.5000   69.6111

>>
```

Grading Criteria:
[20 pts] [While loop structure from lecture working with your data]
[30 pts] [Used pick candy to pick next candy (under target)]
[20 pts] [Inverse/flipped ranking]
[30 pts] [Used pick candy with flipped ranking to pick which candy to remove]

Answer script here:

```
% Copy and paste your script here. Make sure your formatting looks the same
as MATLAB, with size 10 font.
```

Print out rank and inverse rank here:

Command window output

```
Paste two copies of the run here
```

zyBooks Challenge Exercises

Do the challenge activities for the following in Week 8

1. Strings as arrays
    a. Concatenating strings
    b. Constructing strings