CS100M Prelim 2 March 18, 2004

Fill in this information:

Name (clearly *print* last, first, middle): ____________________

Net ID: ____________________

CU ID: ____________________

I have followed the rules of academic integrity on this exam (sign): ____________________

Circle your lab section (just <u>one</u> location!):

We distribute CS100M graded exams in ***lab section***. Uncollected exams will be available in Carpenter B101 during consulting hours. We will post in **Announcements** when the exams are ready at Carpenter.

	12:20	1:25	2:30	3:35
Tue	Upson B7 Blue Room	Red Room Blue Room	Blue Room	Blue Room
Wed	Upson B7 Blue Room	Red Room Blue Room	Blue Room	Blue Room

Instructions:

Failure to follow any instruction may result in a point deduction on your exam:
- Turn off all cell phones, beepers, pagers, and any other devices that will interrupt the exam.
- Remove all calculators, reference sheets, or any other material. This test is closed book.
- Fill out the information at the top of this exam.
- Wait for us to announce when the exam begins.
- Skim all problems first! Read each problem completely before starting it.
- Write your solutions directly on the test using blue/black pen or pencil. Clearly indicate which problem you are solving. You may write on the back of each sheet. If you need scrap paper, ask a proctor.
- Provide only one statement, expression, value, or comment per blank!
- Do not alter, add, or remove any code that surrounds blanks and boxes.
- Do not supply multiple answers. If you do so, we will choose which one to grade.
- Follow good style! When possible, keep solutions general, avoid redundant code, use descriptive variables, use named constants, indent substructures, avoid breaking out of loops, and maintain other tenets of programming philosophy.
- Comment each control structure, major variable, and function (if used), briefly.
- You have 90 minutes. Budget about one minute per point.
- Do not work on bonus problems (if there are any) until you have thoroughly proofread all required (core-point) problems!
- Try to figure out problems yourself before raising your hand so that we can avoid disturbing people in cramped rooms.

Core Points:

```
 1.   ________  (10 points)  __________
 2.   ________  (20 points)  __________
 3.   ________  (35 points)  __________
 4.   ________  (35 points)  __________
Total: ________ /(100 points) __________
```

Bonus Points: ______

Reminders

```
&              % AND
&&             % short-circuiting AND
|              % OR
||             % short-circuiting OR
~              % NOT
1:4            % array [1 2 3 4]
1:2:5          % array [1 3 5]
v(1:end)       % all values of v from the first to the last element
abs(x)         % absolute value of x
disp           % displays array of strings
floor          % returns integer portion of a number
fprintf        % output
logical        % convert a quantity to a logical value (logical 0 or logical 1)
mod(x,y)       % returns the remainder of x divided by y
num2str        % converts a numerical value to a string
rand           % returns random value between 0 and 1, non-inclusive
strcat(s1,s2)  % concatenate strings s1 and s2
```

Problem 1 [10 points] *magenta-font items in DIS's notes and other general concepts*

Answer the following questions. Be concise and clear.

1a [1 point] Is it likely that the fate of the universe depends on your performance on this exam?

1b [1 point] What is the social impact of solving the sheet pile equation in terms of helping people?

1c [1 point] What is code vectorization in MATLAB?

1d [2 points] Although arrays and matrices are typically interchangeable in MATLAB, they are different. Please complete this sentence:

Whereas arrays represent a general collection of values, matrices represent a

____ **. (Hint: 2 words.)**

1e [2 points] Why is binary search more efficient than linear search?

1f [3 points] For binary search to work, how should your data be organized? (Hint: one word gets you 3 points!)

Problem 2 [20 points] *MATLAB arrays*

Fill in the boxes for the following MATLAB sessions. Consider each problem as a fresh MATLAB session. If you want partial credit, you *may* show your work.

2a [2 points]

```
>> a = [1:3;4:6]; a .* 2
ans =
```

2b [4 points]

```
>> a = [1:3;4:6]; a(1:end,end:-1:1)
ans =
```

Problem 2 continues on the next page.

2c [7 points]

```
>> a = [1:3;4:6]; a([2 1],[1 2]) = a([1 2],[3 2]) + a([2 1],[1 2])
a =
```

2d [7 points]

```
>> a = [1:3;4:6]; a(1,mod(a(2,:),2)~=0)
ans =
```

Problem 3 [35 points] *selection, repetition (for-loops), character graphics*

Problem: Complete the following program that prints a rectangular grid using the backslash, forward slash, and blank characters. The grid is composed of two kinds of *alternating rows*, each with a different pattern of slashes:

- ***uppyrow***: a row starting with `\` and ending with `/`, e.g., `\/\/\/\/`. This row has four ***uppies*** (`\/`).
- ***downyrow***: a row starting with `/` and ending with `\`, e.g., `/\/\/\/\`. This row has four ***downies*** (`/\`).

The user supplies a non-negative legal **`size`** parameter to function **`drawGrid`**, which determines the amount of rows in the grid and amount of pairs of characters in each row. As an added twist, *about* 10% of the time the program will randomly print a blank space instead of either slash (or both) in an uppy or downy.

Specifications: The two types of rows must alternate, starting with an uppyrow. You must use the code that we have given, which is a collection of functions **`drawGrid`**, **`drawRow`**, **`drawPair`**, and **`chooseChar`**. Note that MATLAB (and many other languages) require **`'\\'`** to represent a single backslash.

Example session:

```
>> drawGrid(4)
\/\ \/\/
/\/\/\/\
\ \/\/\/
 \/\/\/
```

```
% Draw entire grid:
function drawGrid(size)
   FS = '/';                        % forward slash char
   BS = '\\';                       % backslash char

   UPPY  = strcat( ___ , ___ );     % \/

   DOWNY = strcat( ___ , ___ );     % /\

   % draw alternating rows of uppies and downies (uppyrow, downyrow, uppyrow, ...):

     for count = ______________________

        % alternate drawing of odd and even rows:

          if ___________________________________

             ___________________________________

          else

             ___________________________________

          end
     end

% Draw just one row:
function drawRow(size,pair)

   for count = ____________________________

      _____________________________________

   end
   fprintf('\n');
```

Problem 3 continues on the next page.

```
% Draw a pair:
function drawPair(pair)
   ___________ = chooseChar(pair(1)); % choose between blank or first char in pair

   ___________ = chooseChar(pair(2)); % choose between blank or second char in pair

   fprintf(pair);

% Randomly choose between a character and a blank (blank: about 10% of the time) and
% return the chosen character:
function char = chooseChar(slash)
```

Problem 4 [35 points] *nested control structures, searching*

Background: Recall that in the binary search algorithm for discrete data you can use two “fingers” that start at the left and right side of a given search interval. The algorithm moves the “fingers” until the target is found or the fingers cross. For continuous data, there is a related approach for linear search.

Problem: On the next page, you will complete the code for function **`doubleLHSRHS`** that tries to find one root of subfunction **`f`**. Refer to the figure below, which shows an initial search interval [**`L`**, **`R`**] for **`f`**. The key idea is that both **`L`** and **`R`** “move” towards each other by **`inc`** (both directions). Each time *before* **`L`** and **`R`** are updated, **`inc`** is checked to ensure that **`L`** and **`R`** do not cross. If the current **`inc`** would cause **`L`** and **`R`** to cross, **`inc`** is reduced. **`L`** and **`R`** are updated *after* checking **`inc`**.

Detailed version of algorithm (what would have been problem “Problem 4a”):

- Check if **`decrs`** does not exceed **`MAXDECRS`** and that neither **`f(L)`** nor **`f(R)`** (using the current values of **`L`** and **`R`**) produce values within a tolerance **`eps`**. If so, do the following:
  - Check **`inc`** and reduce it, if necessary, by doing the following steps:
    * Check if the current value of **`inc`** would cause the next values of **`R`** and **`L`** to cross.
    * If so, divide **`inc`** by **`10`**, increment **`decrs`**, and repeat. (Do not modify **`L`** and **`R`** when checking **`inc`**.)
  - Using the current value of **`inc`** (which may have been reduced), update the current values of **`L`** and **`R`**.
  - Repeat the check of **`MAXDECRS`** and **`f`**.
- Report either **`root`** (answer found!) or **`-1`** (algorithm failed!).

Note that this algorithm will work well if the root is near the middle of the initial interval. Otherwise, the algorithm might perform very poorly and possibly even “hang” if **`inc`** becomes too small. For up to 7 bonus points, explain the weaknesses of the given algorithm and list some improvements on the back of the next page.

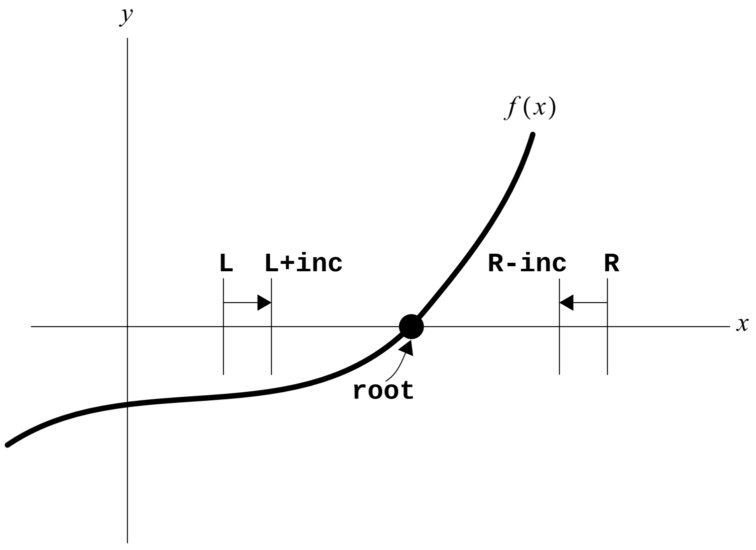
Additional Specifications:
- All inputs to **`doubleLHSRHS`** are legal.
- Do not check for a maximum number of iterations, change of sign, or “hitting” the *y*-axis, as you did in the Assignment 3.
- For full credit, do not use **`break`**. You should also avoid using additional variables.

Example session (using $f(x) = x^2 - 4$):

```
>> doubleLHSRHS(0, 4, 0.0001, 1, 100 )
ans =

     2
```

Depiction of LHSRHS with search in two directions:



```
function root = doubleLHSRHS(L,R,eps,inc,MAXDECRS)
% Returns the ROOT of F by incrementing L and decrementing R
% If MAXDECRS is exceeded during search, return an out-of-bounds value

  root  = __________ ;   % default value of ROOT before search

  decrs = __________ ;   % number of times INC has been reduced so far

% Search for root:
  while ________________________________________________________________________

     % Check INC and reduce it if necessary:
       while ________________________________________________________________



       end
     L = __________________ ; % update L
     R = __________________ ; % update R
  end

% Check if root is found; assign appropriate return value:




function y = f(x)
% code not shown for this function (could be many different functions)
% Y is the value of the function F at X. For example, y = x.^2-4
```