CS100M **November 30, 2000**
Makeup Solutions **7:30 PM – 9:30 PM**

_____ _____
(Print last name, first name, middle initial/name) (Student ID)

Statement of integrity: I did not, and will not, break the rules of academic integrity on this exam:

_____
(Signature)

Circle Your Section:

	Tuesday			Wednesday	
	PH 403	PH 407	UH 111	HO 306	UH 111
1:25	1 Nagarajan		2 Fan	6 Rohde	
2:30		3 Nagarajan	5 Fan		7 Fernandes
3:35		4 Fernandes		8 Rohde	

Instructions:

- Read all instructions *carefully,* and read each problem *completely* before starting it!
- This test is closed book – no calculators, reference sheets, or any other material allowed.
- Conciseness, clarity, and style all count. Show all work to receive partial credit, especially box diagrams.
- Carefully comment each loop and major variable.
- If *you* use **`break`** or **`System.exit`** to exit any control structure (except **`switch`**), you will lose points!
- You may **not** alter, add, or remove any code that surrounds the blanks and boxes.
- Only **one** statement, expression, modifier, type, or comment per blank!
- Use the backs of pages if you need more space or scrap. You may request additional sheets from a proctor.
- If you supply multiple answers, we will grade only **one**.

Core Points:

1. _____ (15 points) _____

2. _____ (35 points) _____

3. _____ (20 points) _____

4. _____ (30 points) _____

Total: _____/(100 points) _____

MATLAB Reminders

[] (square brackets), e.g,

```
>> v = [7  12  5]
v =
      7  12  5
```

[] (empty array), e.g.,

```
>> v = []
v =
      []
```

() (parentheses), e.g,

```
>> v([1 2])
ans =
      7  12
```

{} (braces), e.g.,

```
>> w(2) = {'hi'}
w =
      []    'hi'
```

. (dot), e.g.,

```
>> z.x = 1
z =
      x: 1
```

: (colon), e.g,

```
>> 1:3
ans =
      1  2  3
```

relations (**<**, **>**, **<=**, **>=**, **==**, **~=**),
boolean values (**0**, **1**), and
logical operators (**|**, **&**), e.g,

```
>> 1==0 | 1==1
ans =
      1
```

abs (absolute value), e.g.,

```
>> abs([1 -2])
ans =
      1  2
```

any (return **1** if any element in a vector is non-zero), e.g.,

```
>> any([0,0,1,0])
ans =
      1
```

char (character), e.g.,

```
>> char(107)
ans =
      k
```

end (for ranges), e.g.,

```
>> x = 1:4; x(2:end)
ans =
      2  3  4
```

isreal, e.g.,

```
>> ~isreal(i)
ans =
      1
```

linspace (linear space), e.g.,

```
>> linspace(0,2,3)
ans =
      0  1   2
```

mod (modulus), e.g.,

```
>> mod(4,2)
ans =
      0
```

sum (summation), e.g.,

```
>> sum(1:2)
ans =
      3
```

ones (fill array with value 1),

```
>> ones(1,4)
ans =
      1  1  1  1
```

Problem 1 [15 points] *Java: conditions, Boolean values*

Suppose that Java lost its *and* (**&**, **&&**) and *or* (**|**, **||**) operators. You need to write methods that replace them. Complete class **AndOr** which contains methods **and** and **or** that:

- take two input Boolean values **v1** and **v2**
- return the Boolean value that represents **v1 && v2**, and **v1 || v2**, respectively

You may assume that **v1** and **v2** are always Boolean.

```
public class AndOr {
   public static void main(String[] args) {
      // Fill in blanks for output:

         System.out.println( and(true,false) ); // the output is: false

         System.out.println( or(true,false) );  // the output is: true
   }

   // Return the result of $v1$ && $v2$ without using $&&$:
      public static boolean and(boolean v1, boolean v2) {
         if (v1==true)
            if (v2==true)
               return true;
            else
               return false;
         else
            return false;
      }

   // Return the result of $v1$ || $v2$ without using $||$:
      public static boolean or(boolean v1, boolean v2) {
         if (v1==true)
            return true;
         else if (v2==true)
            return true;
         else
            return false;
      }

} // Class AndOr
```

Problem 2 [35 points] *Java: OOP,* **`this`**, **`static`**, *methods, encapsulation,* **`toString`**

Complete the following code in class **`Time`** by filling in the blanks and boxes.
`Time` represents a measurement of time in hours, minutes, and seconds. This class has the following members:

- instance variables **`hrs`**, **`min`**, **`sec`** that store hours, minutes, and seconds, respectively
- class constants **`SEC_PER_HRS`** and **`SEC_PER_MIN`**
- constructor **`Time`** that sets all instance variables
- method **`sec2time`** that takes as input the number of seconds and returns a **`Time`** with **`hrs`**, **`min`**, and **`sec`** set according to the input
- method **`time2sec`** that returns the number of seconds according to the **`hrs`**, **`min`**, and **`sec`** of the current **`Time`** (this instruction was unclear -- fixed)
- method **`add`** that adds a **`Time`** to the current **`Time`** and returns the result as a new **`Time`**
- method **`sub`** that subtracts a **`Time`** from the current **`Time`** and returns the result as a new **`Time`**
- method **`toString`** that returns a String containing a description of the current **`Time`**. See the output for **`main`**!

Hints: You do not need to perform type checking in the constructor or the methods. In **`add`** and **`sub`**, find the seconds of each **`Time`** with **`time2sec`**, perform the necessary addition/subtraction, and return a new **`Time`** using **`sec2time`**. The output for method **`main`** is:

```
1hrs:55min:2sec+1hrs:5min:59sec=3hrs:1min:1sec
1hrs:55min:2sec-1hrs:5min:59sec=0hrs:49min:3sec
```

```
public class TimeCalc {
   public static void main(String [] args) {

   // Create two objects of class Time:
      Time t1 = new Time(1,55,2);
      Time t2 = new Time(1,5,59);

   // Add $t1$ and $t2$ and report result:
      System.out.println( t1+"+"+t2+"="+t1.add(t2) );

   // Subtract $t2$ from $t1$ and report result:
      System.out.println( t1+"-"+t2+"="+t1.sub(t2) );

   }
} // class TimeCalc

class Time {
   private int hrs;                // number of hours
   private int min;                // number of minutes
   private int sec;                // number of seconds
   final static int SEC_PER_HRS=3600;  // number of seconds per hour
   final static int SEC_PER_MIN=60;    // number of seconds per minute

   public Time(int hrs, int min, int sec) {
      this.hrs=hrs ;
      this.min=min ;
      this.sec=sec ;
   }

   // Return the current Time as a String:
      public String toString() {
          return (hrs+"hrs:"+min+"min:"+sec+"sec") ;
      }
```

```
   // Convert number of seconds $s$ into hours, minutes, and seconds (as integers)
   // and return a new Time with instance variables set to these values:
      private Time sec2time(int s) {
         int hrs = Math.floor(s/SEC_PER_HRS);
         int min = Math.floor((s-SEC_PER_HRS*hrs)/SEC_PER_MIN);
         int sec = s-SEC_PER_HRS*hrs-SEC_PER_MIN*min;
         return new Time(hrs,min,sec);
      }

   // Determine total number of seconds for current Time:
      private int time2sec() {
         return SEC_PER_HRS*hrs + SEC_PER_MIN*min + sec ;
      }

   // Add current Time to input Time $t$ and return the sum as a new Time:
      public Time add(Time t) {
         return sec2time( time2sec() + t.time2sec() );
      }

   // Subtract current Time from input Time $t$ and return the result as a new Time:
      public Time sub(Time t) {
         return sec2time( time2sec() + t.time2sec() );
      }

} // class Time
```

Problem 3 [20 points] *MATLAB: arrays, strings, loops*

Goal: Write a MATLAB function **`stars(size)`** that prints an ASCII (text) image of a triangle using stars (**`*`**).

Example session:

```
>> stars(5)
*****
 *  *
  * *
   **
    *
```

Approaches: There are different ways to design **`stars`**:

- Think of the triangle as a plot with the origin at the top, left position under the command line. Represent the diagonal as a line with coordinates (x, y), where you use the equation of a line $y = mx$:
  - x is the position to the right of the origin
  - y is the position below the origin.
  - m is the slope, which equals 1 in this case.
- Note that the column position (x) is equivalent to the row position (y) along the diagonal. This fact will help you design a loop to print each row.
- Use MATLAB's built-in array functions to create an array of stars and print them out.

Hints: We have provided a function **`chars(n,s)`** that returns a string composed of **`n`** characters **`c`**. Assume that **`chars`** is a subfunction of **`stars`**. You are not required to use **`chars`**. If you wish to design your own subfunction(s), write it/them on another sheet and submit with this problem.

```
function s = chars(n,c)
% Return 1-d String-array that contains n characters:
%   N: number of characters
%   C: character
%   S: string composed of N characters C
%   Example:
%       >>DISP([CHARSCHARS(3,'*')])
%       ***
s = char(ones(1,n)*double(c));
```

```
function stars(size)
%STARS(SIZE) prints a triangle of stars (*)
% Variables:
  % size: base and height of triangle
  % pos: position of row and col
  % no need for an x and y because x=y for the diagonal!

% Draw rows:
  for pos=1:size

  % Draw shape:
    % First line: fill base with stars:
      if pos==1
         disp([chars(size,'*')])
    % Middle portion: note that row=col for diagonal:
      elseif pos~=size
         disp([chars(pos-1,' '),'*',chars(size-pos-1,' '),'*'])
    % Last portion: print only 1 star:
      else
         disp([chars(size-1,' '),'*'])
      end
  end
```

Problem 4 [30 points] *MATLAB: processing input, random numbers, functions, recursion*

Problem: Complete the blanks and boxes for function **`numberGuess`** to write a number guessing game that prompts the user to guess a random number within a supplied range **`low`** to **`high`**, including the end points.

Details: The user calls **`numberGuess`** from the command window. The function must prompt the user for guesses until the user picks the correct answer. Each time the user guesses incorrectly, the program must report that the guess is either `too high` or `too low`. The user may halt guessing by entering an out-of-bounds number. Assume that the user always enters integers for **`low`**, **`high`**, and all guesses. (So, you do not need to perform type checking!) After the user halts guessing or guesses correctly, **`numberGuess`** either ridicules or congratulates the user. In both cases, the number of valid guesses is reported for the current game. The function will also prompt the user to choose whether or not to repeat the game.

```
function numberGuess(low,high)
% NUMBERGUESS: number guessing game
% Computer picks a random number and user guesses it.
% Welcome and startup:
  clc;
  disp('Welcome to Number Guess!');

% Obtain a random number $target$ for the user to guess:
  target = ceil(rand*(high-low)) + low;

% Obtain initial input, assuming user enters an integer:
  guess = input('Enter guess: ');

% Initialize $count$:
  count = 1;

% Process next guesses:
  while guess ~= target & guess <= high & guess >= low
    if guess < target
       disp([num2str(guess),' is too low!']);
    elseif guess > target
       disp([num2str(guess),' is too high!']);
    end
    guess = input('Enter your next guess: ');
    count = count+1;
  end

% Report results:
  if guess==target
     disp('Congratulations!');
     disp(['You took ',num2str(count),' guesses.']);
  elseif guess > high | guess < low
     disp('Halting!');
     disp(['The correct guess was ',num2str(target)]);
  count = count-1; % remove last guess
  end

% Repeat the game if user chooses to do so:
  repeat = input('Repeat? [y/n] ','s');
  if repeat=='y'
     feval('ng',low,high);
  %   return; % not needed because no return value to function
  end
```