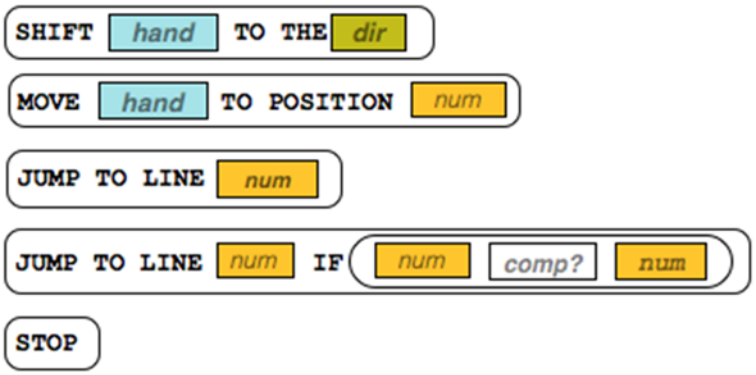
CS 100 Lab 02 - Algorithms

The “Human Machine” Language - Part 1
Submission, one per team from your lab room, due before next week’s lab.

Here are the beginnings of a more formalized low-level language you can use to create programs for a “Human Machine” to solve problems with playing cards.

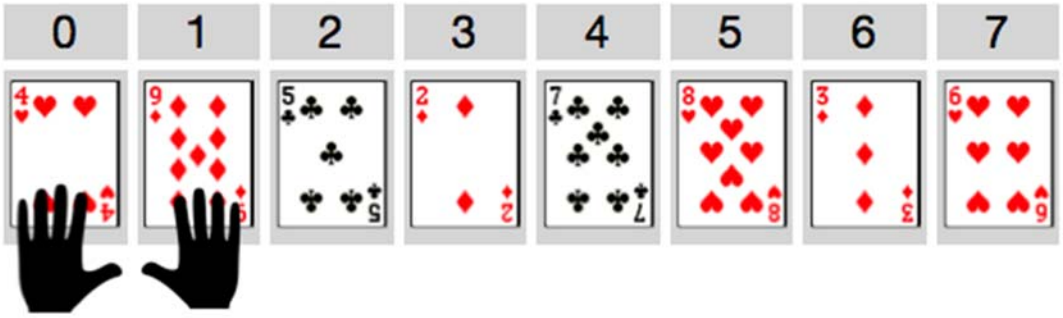


To simplify things we’ll get rid of the need to pick cards up and put them down. Instead leave cards face up and just touch them. The 5 commands you can use are shown to the right. **See the *Reference Guide* on the next page for descriptions of what these commands do.**

Some of these commands might seem unusual, but we can write programs with just these commands to control the “human machine’s” hands to touch or pick up the cards, look at their values, and move left or right down the row of cards.

Standard Card Setup

You should assume this standard initial setup. Here is a diagram for an 8-card setup:



- There will be some number of cards with random values, lined up in a row, face up.
- Positions are numbered starting at 0 and increasing for however many cards there are.
- The left and right hands start at positions 0 and 1 respectively.

Try out some example programs (not to be submitted)

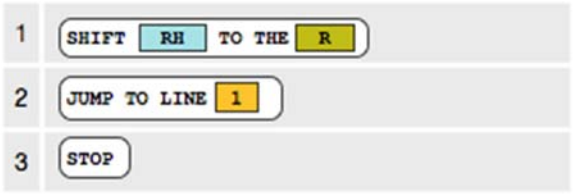
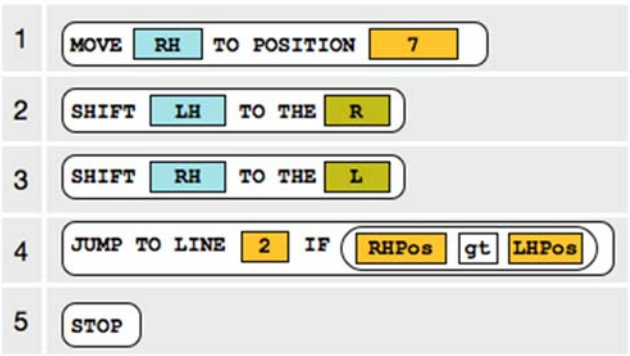
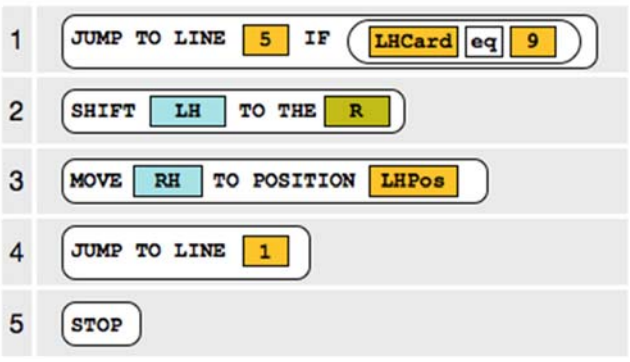
Get to know the Human Machine Language by acting out the examples on the following page with a partner. For each of the examples on the next page you should:

- Lay out a row of **8 cards** in front of you to test out the program.
- Have one partner read the instructions in sequence starting at line 1, and the other partner act out each command as the human machine.
- Use the ***code reference*** to answer your questions and verify you’re interpreting the code correctly.
- Give a brief description of what the program does, or its ending state.

NOTES:

- Some of the programs are very simple
- Some of the programs might not ever stop
- The point is simply to practice using the language and executing commands as a “Human Machine”

Example Program	What does it do?

	
	*Note: this one has a problem, can you find it?*
	
	
	*Note: there is a potential problem with this one too. But only in certain circumstances. Can you find it?*

Human Machine Code Reference Guide

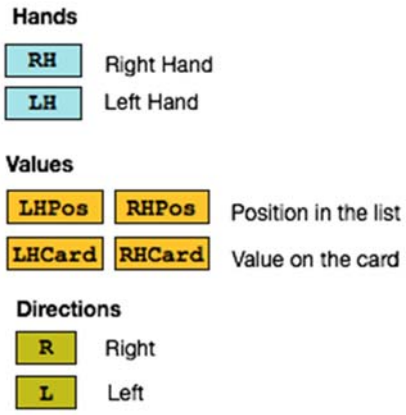
Hands, Values and Direction

There are some short-hand abbreviations for referring to the human machine, the cards, positions, and directions of movement.

Hands - The Human Machine has hands! You can refer to a specific hand abbreviated **`LH`** or **`RH`** (left hand or right hand).

Values - Each hand has two values you can refer to:
1. **`LHPos`**, **`RHPos`** - The hand's position in the list (a number)
2. **`LHCard`**, **`RHCard`** - The value of the card the hand is holding (a number)

Direction - There are two directions **`R`** and **`L`** (right and left) that hands can move along the row of cards.



Commands

Description	Examples
`SHIFT` *hand* `TO THE` *dir* Shift the given hand one position to the right or left along the row of cards.	`SHIFT LH TO THE R`
`MOVE` *hand* `TO POSITION` *num* Move a given hand to a specific position number in the row of cards.	`MOVE RH TO POSITION 4` `MOVE LH TO POSITION RHPos`
`JUMP TO LINE` *num* Jump to a specific line number in the program and continue execution from that point.	`JUMP TO LINE 1`

`JUMP TO LINE` *num* `IF` *num* *comp?* *num*
Jump to line but ONLY IF the comparison of two numbers is *true*. If the comparison is *false* then just proceed onto the next line of code.
• For numbers, you can use **integers** or any of the hand values **`RHCard, LHCard, RHPos, LHPos`**
• For comparisons you can use **`eq, ne, lt, gt`**, (equal, not equal, less than, greater than)
	`JUMP TO LINE 4 IF LHCard eq 7`
	`JUMP TO LINE 2 IF LHCard lt RHCard`
	`JUMP TO LINE 7 IF RHPos gt 9`

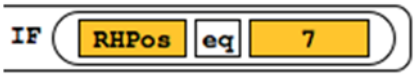
`STOP` End of program. Stop doing anything, stop executing lines of code.	*This should be the last line of code in the program, or on a line that is jumped to when you want the program to stop.*

Challenge: Find Min (for submission)

Using only the Human Machine Language design a program to find the card with the smallest value in the list of cards.

Goal: When the program stops, the left hand should be touching the card with the smallest value.

Hint: How do you know you're at one end of the list or the other?
Use the hand position values to check whether the position is 0 or the largest position in the list - you can assume that you know how big the list is ahead of time. For example, if the last position is 7, then the comparison: **`IF RHPos eq 7`** would tell you that the right hand was as the end of the list.

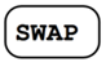


Write your program in the space provided below

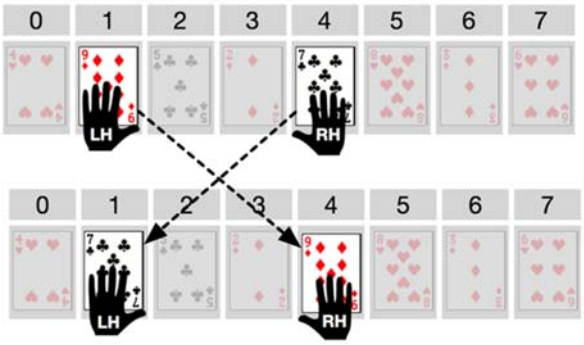
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

The “Human Machine” Language - Part 2

We’re going to add one command to the Human Machine Language called **SWAP** - see description below. All of the other commands are still available to you. So, there are 6 commands total in the language now.

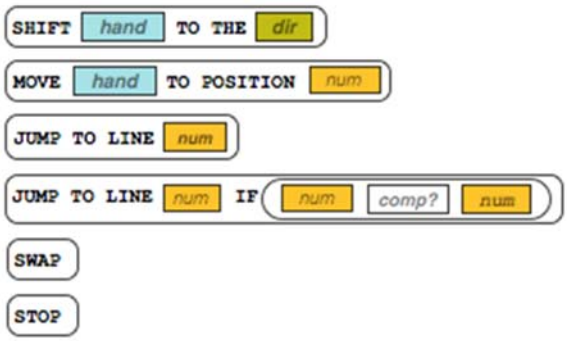


Swap the positions of the cards currently being touched by the left and right hands. After a swap the cards have changed positions but hands return to original position.



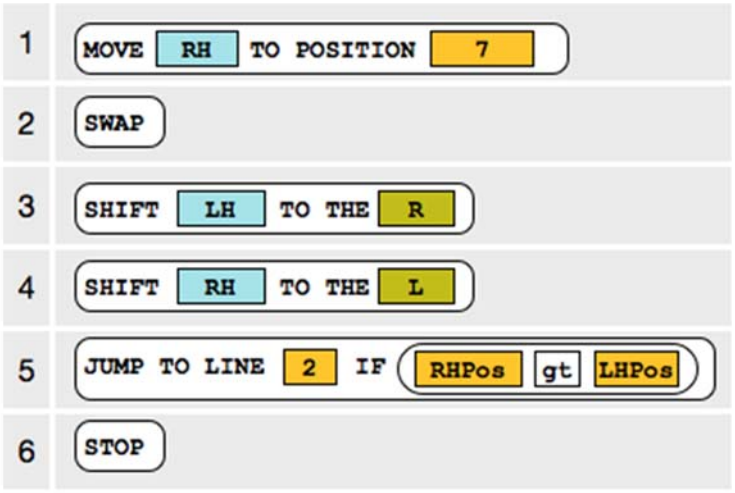
The human machine action is: pick up the cards, exchange the cards in hand, and return hands to original position in the list with the other card.

Human Machine Language Reference



Try an example with Swap

Trace the program below with a partner and describe what it does.



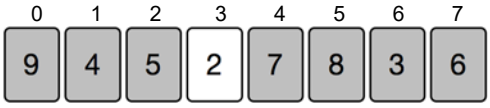
What does this program do?

Challenge: Min To Front (for submission)

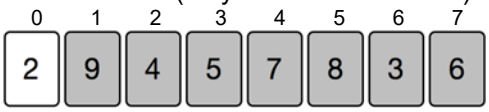
Using only the Human Machine Language design an algorithm to find the smallest card and move it to the front of the list (position 0). All of the other cards *must remain in their original relative ordering.*

END STATE: When the program stops, the smallest card should be in position 0. The ending positions of the hands do not matter, the ending positions of the other cards do not matter. *As a challenge:* try to move the min-to-front and have all other cards be in their original relative ordering.

Cards BEFORE:



Cards AFTER (may not be in this order)



1

2

3

4

5

6

7

8

9

10

(If you need more lines, just keep going).

This list of optional challenges is given in no particular order. You can make the following assumptions:

- Cards start randomly valued, and randomly ordered, and are dealt from an infinitely large deck. I.e. you could face a row of all one value, or there could be seven 2s and one 6, and so on.
- Algorithms should work in principle for any number of cards, and any values that are comparable.
- Algorithms must STOP and be in the END STATE given in the challenge description.

Challenge Description	**Example**
Search for 2 <u>or</u> a 10 - Search the list and stop when find EITHER a 2 OR a 10 (you could substitute 2 and 10 for any other two values if you like). END STATE: the left hand should be touching the first 2 or 10 encountered in the list. End state does not matter if there is no 2 or 10, but the program should stop.	BEFORE: 4 3 7 5 10 4 7 2 2 AFTER: 4 3 7 5 10 4 7 2 2 ^ **LH**
Hi-Lo - Find the min and max values in the list and move them to the first and last positions, respectively. END STATE: The card with lowest value in the list is in position 0, and the card with the highest value is in the last position (position 7 if there are 8 cards). The end state of the hands does not matter, the positions of the other cards does not matter.	BEFORE: 4 3 7 5 **10** 4 7 **1** 2 AFTER: **1** 4 3 7 5 4 7 2 **10**
Search for 2 <u>and</u> a 10 - Search the list and stop once you have found BOTH a 2 AND a 10. END STATE: the left hand should be touching a 2 and the right hand should be touching a 10. End state does not matter if there is not both 2 and a 10.	BEFORE: 4 3 7 5 10 4 7 2 2 AFTER: 4 3 7 5 10 4 7 2 2 ^ ^ **RH** **LH**
Sort - Get the cards into sorted order from least to greatest. END STATE: end state of the hands does not matter, but cards should be in ascending order, and the program should stop.	BEFORE: 4 3 7 5 10 4 7 2 2 AFTER: 2 2 3 4 4 5 7 7 10
Partition - Call the last card in the list the *pivot value*. Arrange the list so that all of the cards less than the pivot value are to the left of it and all the cards greater than the pivot are to the right. (Cards equal to the pivot can go to the left or right of it, your choice). END STATE: The pivot value is in the middle of the list somewhere with all values less than it to the left, and all values greater than it to the right. The ordering of the cards to the left and right do matter. The end state of the hands does not matter.	BEFORE: 8 7 4 2 3 7 5 1 **<u>6</u>** AFTER: 4 2 3 5 1 **<u>6</u>** 8 7 7 BEFORE: 5 4 3 5 4 3 5 4 **<u>1</u>** AFTER: **<u>1</u>** 5 4 3 5 4 3 5 4 BEFORE: 5 4 3 5 4 3 5 4 **<u>7</u>** AFTER: 5 4 3 5 4 3 5 4 **<u>7</u>**
Count - Count the number of 2s in the list and set the right hand so that its position number is equal to the number of 2s in the list. END STATE: the position number of the right hand is equal to the number of 2s in the list. If the count is higher than the possible position numbers (i.e. every value is a 2) then set the right hand to the position past the end of the list. I.e. if there are 8 positions (0-7) the right hand should end in position 8.	BEFORE: 4 **2** 7 5 10 4 7 **2 2** AFTER: 4 2 7 5 10 4 7 2 2 **RH** BEFORE: 4 3 7 5 10 4 7 1 1 AFTER: 4 3 7 5 10 4 7 1 1 RH BEFORE: 2 2 2 2 2 2 2 2 2 AFTER: 2 2 2 2 2 2 2 2 2 RH