Card Search and Sort

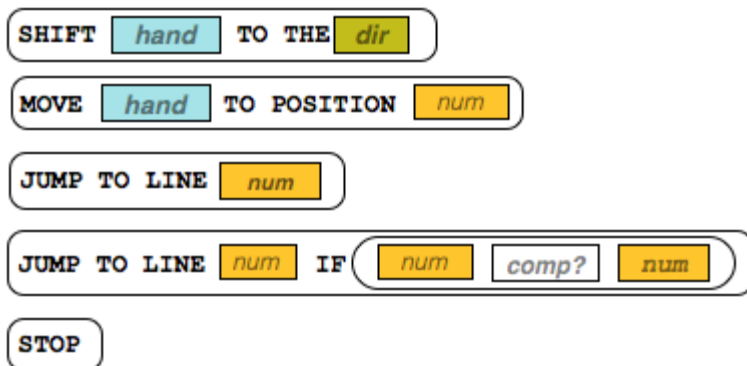## Your Tasks (Mark these off as you go)

- ☐ Get familiar with the machine language commands
- ☐ Practice with the Machine Language Commands
- ☐ Write an algorithm to find a card in an unordered list
- ☐ Write an algorithm to bring the min card to the front
- ☐ Write an algorithm to sort the cards from low to high
- ☐ Write an algorithm to find a card in an ordered list
- ☐ Receive credit for this lab guide

☐ Get familiar with the machine language commands

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

The 5 commands you can use are shown to the right. **See the _Reference Guide_ provided 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.

SHIFT _hand_ TO THE _dir_

MOVE _hand_ TO POSITION _num_

JUMP TO LINE _num_

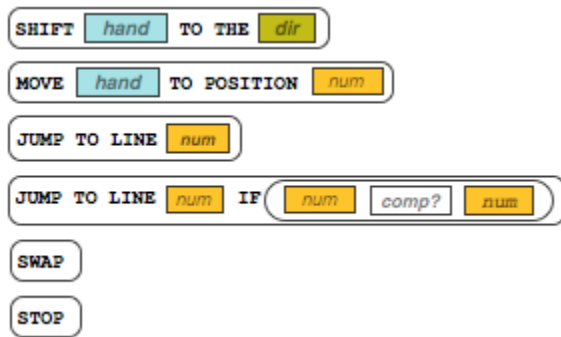JUMP TO LINE _num_ IF (_num_ comp? _num_)

STOP

In addition to the above commands, we have an additional command called **SWAP** - see description below.

SWAP

The SWAP command swaps the positions of the cards currently being touched by the left and right hands. After a swap the cards have changed positions but <u>hands return to original position</u>.
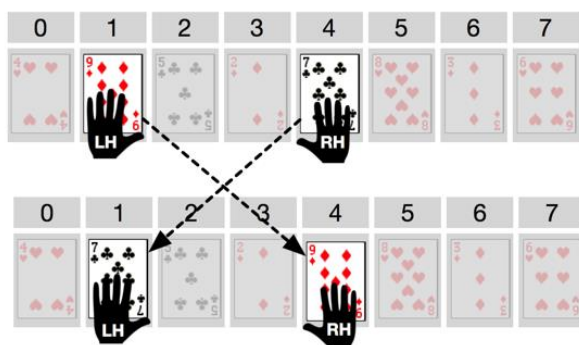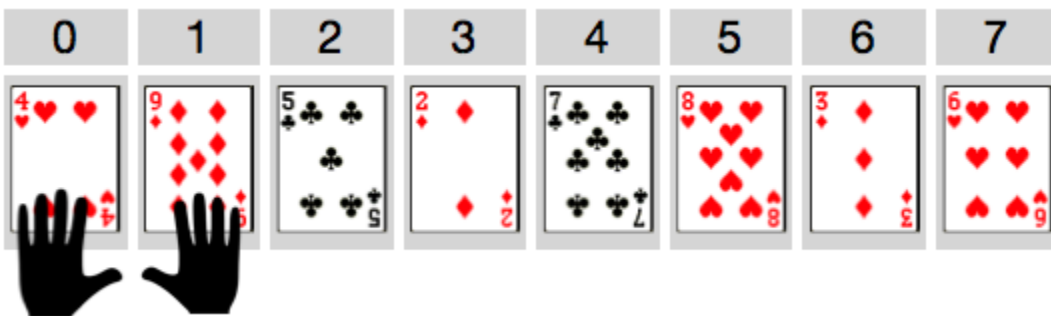

There are 6 commands total in the language available for you to use. These are shown below.

```
SHIFT  [hand]  TO THE  [dir]

MOVE  [hand]  TO POSITION  [num]

JUMP TO LINE  [num]

JUMP TO LINE  [num]  IF ( [num]  [comp?]  [num] )

SWAP

STOP
```

☐ Practice with machine language commands

For this activity, you should assume this standard initial setup. Here is a diagram for an 8-card setup:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

- 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.

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

- Lay out a row of **8 cards** randomly 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.

Example Program	What does it do?
1 SHIFT [RH] TO THE [R] 2 JUMP TO LINE [1] 3 STOP	*Note: this one has a problem, can you find it?*
1 SHIFT [RH] TO THE [R] 2 JUMP TO LINE [1] IF ([RHPos] [ne] [7]) 3 STOP	

1 MOVE RH TO POSITION 7 2 SHIFT LH TO THE R 3 SHIFT RH TO THE L 4 JUMP TO LINE 2 IF (RHPos gt LHPos) 5 STOP	
1 MOVE RH TO POSITION 7 2 SWAP 3 SHIFT LH TO THE R 4 SHIFT RH TO THE L 5 JUMP TO LINE 2 IF (RHPos gt LHPos) 6 STOP	

☐ **Write an algorithm to locate a card in an unordered list**

Using only the Human Machine Language design a program to find the card with a specified value. When the program stops, the left hand should be touching the specified card. Your program must stop if one of the following conditions is met,

- The left hand is touching the specified card
- The card is not found

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 at the end of the list.

Once you have figured out an algorithm that works. Take a picture of your algorithm and paste it in the box below. Or you can type the commands instead.

☐ Write an algorithm to bring the min card to the front

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:

0	1	2	3	4	5	6	7
9	4	5	2	7	8	3	6

Cards AFTER (may not be in this order)

0	1	2	3	4	5	6	7
2	9	4	5	7	8	3	6

Once you have figured out an algorithm that works. Take a picture of your algorithm and paste it in the box below. Or you can type the commands instead.
Navigate to the machine language simulator below to test out your code.
https://timberlinecs.github.io/HML/?swap=

☐ Write an algorithm to sort the cards from low to high

Using only the Human Machine Language design an algorithm to sort the cards from low to high.

END STATE: When the program stops, the cards should be sorted from smallest to highest

Once you have figured out an algorithm that works. Take a picture of your algorithm and paste it in the box below. Or you can type the commands instead.
Navigate to the machine language simulator below to test out your code.
https://timberlinecs.github.io/HML/?swap=

☐ Write an algorithm to locate a card in an ordered list

Previously you wrote an algorithm to locate a card in an unordered list.   To do this required that you apply a linear search – that is, you started at the beginning of the list and compared each card to the card you were looking for until you found it. If you got to the end of the list, then you concluded that the card was not in the list.

What are situations where a linear search would be impractical?

Now that the list is ordered, how might you locate the item in the list using a different, more efficient algorithm?

Just like before, 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 at the end of the list.

If we know the list has a length of 7, then half the list has a length of 3 (if we apply integer division).   This allows us to move the hand to a specific location in the list. For example,

MOVE RH TO POSITION 3

Once you have figured out an algorithm that works. Take a picture of your algorithm and paste it in the box below. Or you can type the commands instead.

☐ Receive Credit for this lab guide

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code challenges will also be graded and will count towards your final lab grade.