CS 161 The Art of Programming
Workshop 7: Recursion
Nov 1, 2, 3, 2009

** Materials needed for workshop: A deck of playing cards **

Last week we were introduced to the concept of recursion: solving a problem by reducing it to smaller instances of the same kind of problem. In this workshop, we will explore recursion through a series of hands on exercises.

There is a classic recursive algorithm for sorting called "merge sort". In order to understand it, we first need to know how to merge together two already-sorted decks. Here is the algorithm for merging:

```
Merge
input: Two sorted decks of cards, call them deck A and deck B
output: A sorted deck of all of the cards, call it deck C
1. Put decks A and B on the table face up. Deck C will be created face down to the right of A and B.
2. If either deck A or deck B become empty, turn over the non-empty deck and place it on deck C. You are done, return deck C.
3. If the rank of the card on top of deck A is smaller or equal to the rank of the card on top of deck B, move it to deck C (face down). Otherwise move the top card from deck B to deck C.
4. Repeat from step (2) as long as necessary.
```

Exercise 1: Deal cards to pairs of students. Each pair should split their cards into two decks, sort each deck face up from low to high, using whatever method they like, and then merge the decks together using the Merge algorithm above. They should do this until everyone understands how Merge works.

Now that we know how to merge, we can create a recursive algorithm for Merge Sort.

```
Merge Sort
input: an unsorted deck of cards
output: a sorted deck of cards
1. If the unsorted deck of card only contains a single card, then you are done. Return the card as output.
2. Split the unsorted deck in half, as evenly as possible.
3. Recursive step: Merge Sort each half deck of cards.
4. Merge the two sorted half decks together and return the result.
```

Exercise 2: The entire class will participate in performing a run of Merge Sort. Each student will execute one invocation of the algorithm. Start with a number of cards based on the class size according to the table below. If the number of students in the workshop section is even, the workshop leader should participate to make the class size odd.

class size	number of cards
5	3
7	4
9	5
11	6
13	7
15	8

One person starts with all of the cards, shuffled. She starts performing Merge Sort. When she reaches a recursive call, she hands the half deck to another student, who performs Merge Sort recursively and then returns it. Note that some of the recursive calls of Merge Sort will involve just a single card: in that case, the "calling function" hands a single card to the "called function", who immediately hands it back.

Perform this exercise a number of times, allowing different students to be the starting player.

Exercise 3: We will now write a Python program to merge sort lists of numbers. The functions will be:

```
def Merge(deckA, deckB)
# decks A and B are sorted, returns a sorted list combining the two

def Merge_Sort(deck)
# returns a sorted list of the elements in the deck
```

Split the class into teams of 3 to 4 students. Half the teams should work on implementing Merge, and the other half on Merge_Sort (where Merge_Sort calls Merge). At the end of the time, have the teams present their solutions. Will all of the ways of implementing Merge work with all of the ways of implementing Merge_Sort?

Hints for programming:

- How can you split a list into two pieces? Think of the slice operation and the len (length) function.
- How can you remove an item from the front of a list? Think of the slice operation.
- How can you put an item at the end of a list? Think of the concatenate (+) operator.