# CSCA48 Exercise 1

Due: January 20 2017. 5:00pm

Let's play a game... the **banana game**! This one doesn't involve any coding at all, we're just working out manually. It should give you some practice with the general ideas of stacks and queues.

### The BANANA game

The object of the game is to transform a word (`source word`) into another word (`goal word`) using only the container provided (the container must be left empty at the end of the game). This means that we can't add or delete any letters, but we can re-arrange them by putting them into a container and taking them out again. At each stage, we have three options:

- Add the next letter of source word onto the end of goal word (we'll call this `move`)
- Put the next letter from source word into the container (we'll call this `put`)
- Remove a letter from the container and add it to the end of goal word (we'll call this `get`)

For example, if we wanted to change the word `CAT` into the word `ACT`, we could do so with a simple container that only holds one letter (we'll call this container a `Bucket`):

- `put(C)`
- `move(A)`
- `get(C)`
- `move(T)`

Try it out and make sure you understand how that works before you continue.

### Using a Queue

With our simple container, we can't do too much. For example, there's no way we could change `NICK` into `CNKI`. (Try it... doesn't work). But could we do it if we had a `Queue` as our container? Let's see:

| Operation | source word | goal word | Container |
|---|---|---|---|
| | NICK | | |
| `put(N)` | ICK | | N |
| `put(I)` | CK | | NI |
| `move(C)` | K | C | NI |
| `get(N)` | K | CN | I |
| `move(K)` | | CNK | I |
| `get(I)` | | CNKI | |

Try changing `BANANA` into `AAABNN`. Make sure you can do it before moving on.

### Stacks

What if we want to change `BANANA` into `AAANNB`? Well now we have a problem. We've put the B into our container (a Queue), which means if we want to put anything else in (the `N`s), we'll need to dequeue our `B` before we can get to the `N`s. So we're stuck (convince yourself of this before moving on).

Maybe a new ADT will help us:

In lecture, we defined a `Queue` as: `A container of objects accessed in FIFO (First-In First-Out) order`. What if we had a container that held objects in LIFO (Last-In First-Out) order? You can think of this as putting each new item on top of the previous items, so we can only access the most recently added item at any given time. So we'll call this new container a `Stack`.

Can we solve our problem using a `Stack` as our container? Let's see:

| Operation | source word | goal word | Container |
|---|---|---|---|
| | BANANA | | |
| `put(B)` | ANANA | | B |
| `move(A)` | NANA | A | B |
| `put(N)` | ANA | A | BN |
| `move(A)` | NA | AA | BN |
| `put(N)` | A | AA | BNN |
| `move(A)` | | AAA | BNN |
| `get(N)` | | AAAN | BN |
| `get(N)` | | AAANN | B |
| `get(B)` | | AAANNB | |

### Your Task

Your task this week is to play the Banana Game (who says CS students can't have any fun?). Try to create the following permutations of `BANANA` using the containers (`bucket`, `stack` and `queue`):

1. AAABBN
2. AAANNB
3. BNAAAN
4. NBNAAA
5. NNAAAB
6. NNBAAA
7. ANANAB
8. NABANA
9. NANANANABATMAN

### What to Hand in

On Markus, you must submit a file called `ex1.txt` Note the file extension, it's not a Python file. Where, for each of the above cases, you indicate which (if any) of the three containers (`Stack`, `Queue` or `Bucket`) can be used to solve the banana game, and for each container that can be used, the series of operations that will turn `BANANA` into the given permutation (note that there may be more than one solution). For any example that can't be solved using any of the containers, simply write `impossible`.

Your file should be formatted as follows: each section will start with the word number on a line by itself. The next line will be just the word `impossible` if it can't be done with any container, or the list of containers

with which it is possible. For every container with which the game can be solved, there will then be a line containing the container name, followed by a move list separated by spaces

A properly formatted, but in no way correct example submission file is below:

```
1
impossible
2
bucket stack queue
bucket put(A) move(B) get(A)
stack put(A) move(B) get(A)
queue put(A) move(B) get(A)
3
queue
queue put(A) put(B) move(C) get(A) get(B)
```