Programming (FEB22012[X])

5. Exercise

Deadline for submission: 2014-10-12 23:59 CET

Introduction

In this assignment you will implement linked lists in Matlab and solve a classical problem using your implementation of a circular single-linked list. To achieve this, we need to apply the object-oriented extensions of Matlab.

Creating classes in Matlab is achieved similarly as writing function files (if you use the GUI, you can select "class" for adding them from the menu). Classes are accessed using handles that are similar to Java object references. For using classes without methods, we can define them to contain simply the fields (properties), e.g. for class `ExampleClass` with a single field `exampleField`:

```
classdef ExampleClass < handle
  properties
    exampleField;
  end
end
```

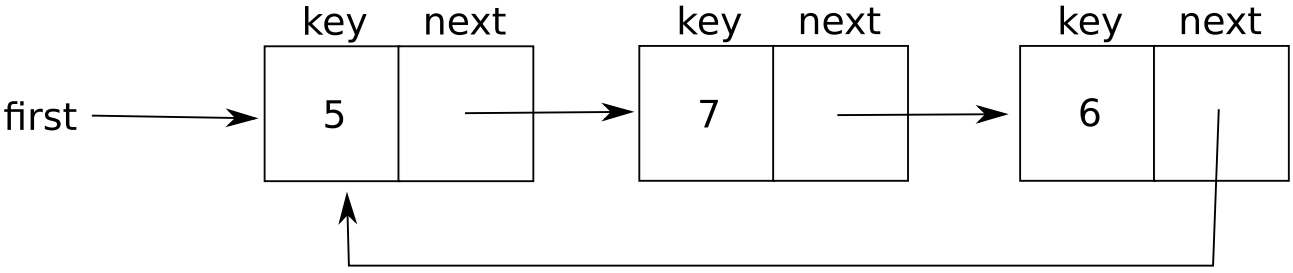
Multiple field variables can be defined by separating them with commas. Now we could instantiate this class and set the field value with:

```
>> t = ExampleClass();
>> t.exampleField = 3;
```

Note that in this exercise we will not actually program according to the object oriented paradigm, but we will just use the OO-extensions of Matlab to enable dynamic memory allocation and passing of parameters by reference. All the linked list operations will be implemented as separate methods.

Exercise, part 1: implementing circular linked list

In this part, you will implement linked lists in Matlab using objects. Start by examining the linked list code in the lecture notes, typing it down, and modifying it so that you implement a *circular linked list* (see Figure below). Note that in a circular linked list when the "first" node is deleted, the "first" reference of the LinkedList object has to be updated as well. Implement also an additional function for computing the length of the list. Remember to document and check the pre-conditions. Make unit tests for all implemented linked list functions (better yet, write the tests before proceeding with the implementation).



After changing the implementation, make sure that you have the following methods for operating with lists:

1. `L = initLinkedList(value)`, that initializes a new circular linked list with a single node

2. `insertIntoBeginning(L, value)`, that inserts value into the beginning of the list

3. `deleteNodeAfter(L, prevNode)`, that deleted the node coming after `prevNode`

4. `listLength(L)`, that gives the number of elements in the linked list `L`

Hint: you should keep track of the first node in your list, otherwise you do not know where the list starts. Make sure all your methods work and that you know how to iterate through the list nodes in a circular manner.

Exercise, part 2: Joseph's problem with linked lists

You are now going to solve the so-called "Joseph's problem" using a circular linked list. Although there exists a analytical solution for the problem, we are going to provide a numerical solution for an adapted version of the original problem.

Let us consider a single bachelorette girl and n candidates interested in her. The candidates are standing in a circle, and one of them is designated as the "first" one. Our girl will assess the candidates but is a bit picky so she will not accept anyone until there are only k candidates left, in which case she will marry them all (this exercise does not model a monogomous society). For each one she assesses, the candidates will be sent away. Then she skips the next k persons (going clockwise), and again assess the following candidate (who is sent away). This continues until k people are left. For example, for $n = 5$ and $k = 1$ first index 1 is sent away, index 2 skipped, index 3 sent away, 4 skipped, 5 sent away (at which point 2 and 4 are left), 2 skipped and 4 sent away. At this point there is $k = 1$ person left, so the algorithm can terminate (result = [2] is chosen).

Create a function `chosenIndicesForMarriage(n, k)` which returns the k chosen indices to stand in when n candidates are standing in a circle and the bachelorette starts assessing and sending the candidates away with a skipping schedule k. You have to do this using the circular linked lists you implemented in part 1. Create a main script that prints out the chosen indices indices for $n \in \{6, \ldots, 20\}$, $k \in \{1, \ldots, 5\}$.