CS 315 Data Structures **Fall 2008**

Project # 1

Due: September 29, 2008

Generating permutations and combinations

Problem statement with the output format:

The goal of this lab is to generate all the ordered k-sequences (permutations) as well as unordered k-selections (combinations) of a given set of n (distinct) integers. The difference between the two versions is that in the latter, the different orderings of a collection are considered the same. As a simple example, suppose the input set is {1, 2, 3} and suppose k = 2. The output for the ordered selections should be the sequences: < 1, 2 >, < 2, 1 >, < 3, 2 >, < 2, 3 >, < 1, 3 > and < 3, 1 > and for the unordered selections, the output will be the three sets {1, 2}, {2, 3} and {1, 3}. For both problems, each list should be printed in a separate line. For the permutation (combination) problem, each output should be enclosed in < and > (in { and }). The combination outputs should be displayed as **sorted list**s. For both problems, a comma must separate the elements. A more precise specification of the problem follows.

Goals of the project:

- review linked lists: learn to implement a singly-linked list class with some basic support functions.
- using recursion: recursion is a natural choice for this and related problems. Although there are some well-known algorithms for this problem that are iterative, recursion is a good way to implement the main function (build).
- program with multiple classes and interactions: an effective way to implement the solution is by using three classes – a Node class, a List class and a Set class.

Design and implementation details:

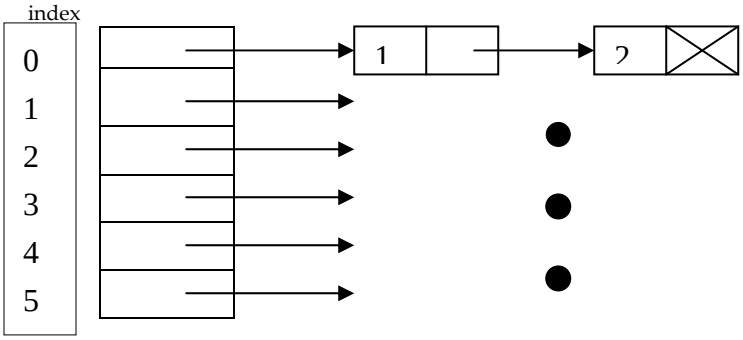
Your program should follow the details of the design and implementation as specified below. Both classes shall use essentially the same data structures.

Node class: You are to implement a node class that contains two fields – key (an integer) and next (a pointer to another node). Node class provides a constructor and some accessors and mutators such as `get_key, set_key, get_next and set_next;`

List class: A list class implements a singly-linked list of nodes. The list class has only one member – a pointer to the first node. This class should have the following functions:

- `void insert (int x)` - to insert a key x in its correct place.
- `void print()` – print the members of the list

For the combination problem, create a class called **combinations** to hold a collection of lists by storing the headers of the lists in an array (or vector). (See the figure below.



First we describe a recursive algorithm for generating the various combinations. Specifically, the principal function in this class is called build that takes as input the array a and two integers r and s and generates all the s-element subsets of the set {a[1], … , a[r]}.

To generate all the s-element subsets of the set {a[1], a[2], …, a[r]}, we proceed as follows. (Note that for convenience, we use the indices starting from 1 rather than 0. In your implementation, you can start the index at 0 or 1 as you find appropriate.) First consider the base case s = 0. In this case, the output consists of one subset which should be the empty set (denoted by { }). In the general case of s, proceed as follows: recursively generate the s element subsets of {a[1], … , a[r – 1]}. This gives all the s element subsets of {a[1], … , a[r]} that DO NOT INCLUDE a[r]. Next, we will generate all the subsets that INCLUDE a[r] as follows: recursively generate s – 1 subsets of {a[1], a[2], …, a[r – 1]}. In each of these subsets, `insert` the element a[i]. (Insert will insert a[I] as the first item of each list.) Now, we have generated all the s-element subsets. But these two collections of subsets should be merged. So you need one more procedure – to `merge` two combinations into a single combination.

If the array a[] is arranged in descending order, then `build` correctly builds the keys in ascending order in each of the selections by inserting a[i] as the first item. See the following example: Suppose n = 3 , the set A is {9, 4, 1} and m = 2. Thus we are generating all 2-element subsets the set A. Recursive call to `build(A, 2, 1)` will return the collection {9}, {4}. Inserting 1 (the last item of the array A) in front of each of these lists will result in {1, 9}, {1, 4} and you can see that these lists are in ascending order.

Since this requires you to arrange A in ascending order, you should use `insertion sorting` to sort the input set. The only procedure that is required to use recursion is build. Many others (including insertion sorting, merge, and the various list class functions such as insert, print etc.) can be done recursively. But all of these can be accomplished with equal ease iteratively and hence you are not required to use recursion.

For the permutation problem, the idea is similar and we will explain it with an example. Suppose the set is {1, 2, 3, 4} and k = 3. Recursive call with the set {1, 2, 3} and k = 3 will output the permutations <1, 2, 3>, <1, 3, 2>, <2, 1, 3>, <2, 3, 1>, <3, 1, 2>, <3, 2, 1>. Next make a recursive call with the set {1, 2, 3} and k = 2. This call outputs <1, 2>, <2, 1>, <1, 3>, <3, 1>, <2, 3>, <3, 2>. Make 4 copies of this output. In the first copy, insert 4 (the last item) in position 1, in the next copy insert it in position 2 etc. This will produce 6 x 3 = 18 sequences. Finally merge all these into a single permutation object (with 24 sequences) that becomes the output. Thus the main differences between combination generation and permutation generation are the following:

- there is no need for sorting the input list for permutation problem
- for permutation problem, we need to be able to insert the last item in different places (while for the combination problem, the insertion is always in the front).
- for permutation problem, we need to perform multiple merges while combination problem requires only one merging.

Expected size of the code is about 200 lines.

Documentation: Provide appropriate comments for each function/procedure. Specifically, for each function, state what each the input parameter is (such as A is the set from which we want to select, k is the number of items being selected etc.). Also describe the relationship between the input and the output, any pre-conditions, any side-effects produced by the function etc.

Sample input/output:

```
%project1_out
for permutations enter 1, for combinations enter 2, to quit enter 0
1
enter the set
1 4 9
enter the number of elements you want to select
3
the permutations of the set are:
<9, 1, 4>
<9, 4, 1>
<1, 9, 4>
```

<4, 9, 1>
<1, 4, 9>
<4, 1, 9>
for permutations enter 1, for combinations enter 2, to quit enter 0
2
enter the set
1 3 4 9
enter the number of elements you want to select
2
the selections of the set are
{1, 3}
{1, 4}
{3, 4}
{1, 9}
{3, 9}
{4, 9}
for permutations enter 1, for combinations enter 2, to quit enter 0
2
enter the set
1 3 4 9
enter the number of elements you want to select
5
the selections of the set are
for permutations, enter 1, for combinations enter 2, to quit enter 0
0
%

Submission:

Your submission should include all the files that you have used to implement the project along with a make file. The program is due mid-night September 19.