CS 315 Data Structures **Spring 2008**

Project # 1 (final version)

Due: February 26, 2008

Generating permutations and combinations

Problem statement with the output format:

The goal of this lab is to generate all the permutations of a given set of integers and to generate all the selections of k items from a set of n integers. For the permutation problem, suppose the set is {1, 2, 3}. Then the program should output the 6 permutations <1, 2, 3>, <1, 3, 2>, <2, 1, 3>, <2, 3, 1>, <3, 1, 2> and <3, 2, 1>. For the combination problem, suppose the user chooses the set 1, 2, 4, 9, 13 of 5 integers and k = 3, the program should output all the 10 subsets of the set {1, 2, 4, 9, 13} such as {1, 2, 4}, {1, 2, 9}, etc. For both problems, each list should in a separate line. For the permutation problem, each permutation should be enclosed in < and >. For the combination problem, each list should be displayed as a **sorted list**, and each set should be enclosed in an opening and closing brace. Also a comma must separate the successive elements and there should be no comma after the last element of the set. The program should report errors in the user input (as described below) and terminate.

Goals of the project:

- review linked lists: We will find that the linked list is a reasonable choice to represent each set in this and related problems.
- practice recursive programming: recursion is a natural choice for this and related problems. Although there are some well-known algorithms for this problem that are iterative, they are not easier to program than the approach we use and are more difficult to understand.
- multiple classes and interactions: We will implement a solution using three classes.

Design and implementation details:

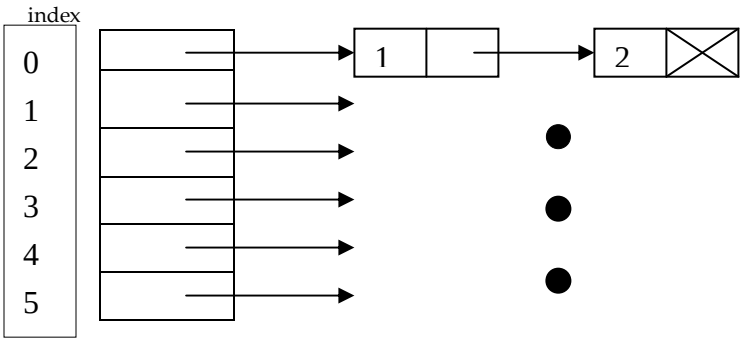
Your program should follow the details of the design and implementation as specified below. Both classes shall use essentially the same data structures.

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 we want to generate all the permutations of {1, 2, 3}. Recursive call for n = 2 will build the permutations <1, 2> and <2, 1>. We will make 3 copies of this collection. In the first copy, we insert 3 (the last item of the set) in the first place, in the second copy we insert 3 in the second place and in the last copy we insert it in the third place. The resulting collections will be (a) <3, 1, 2>, <3, 2, 1> (b) <1, 3, 2>, <2, 3, 1> and (c) <1, 2, 3>, <2, 1, 3>. By merging all these into a single permutation object, we get the result. 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 expected to be about 150 lines.

Documentation: For each function, describe the input parameters, the output produced, the relationship between the input and the output, any pre-conditions, any side-effects produced by the function. If any of these are absent, you can skip them.

Sample input/output:

```
%my_project1
for permutations, enter 1, for combinations enter 2, to quit enter 0
1
enter the set
1 4 9
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
0
%

Submission:

Your submission should include all the files that you have used to implement the project along with a makefile. The program is due *February 26, 2008* by 1 PM, the start of the lab.