Chapter 3

The easy stuff
Lists

- If you only need to store a few things, the simplest and easiest approach might be to put them in a list.
- Only if you need to search or some other more intensive operation on the data would a more complex structure be needed.
Lists

- A list is a finite, ordered sequence of data items known as **elements**
- Each element has a position
- It’s empty when it contains no elements
- The number of current elements is its length
- The first element is the head
- The last element is the tail
Basic Operations

• Must be able to insert and delete anywhere along the list
• Should be able to gain access to an elements value
• Must be able to create and clear the list
• Convenient to gain access to the next element from the current one
List ADT

- In C++ we will use the notion of an abstract class for the List
- We will increase the flexibility by making it a template
- The abstract class does not specify how the operations are implemented
- Given that we want to support the concept of a sequence, the key decision embodied in the ADT is the notion of position
Details

• Assuming that we all have implemented a linked list
• There are 4 steps for insertion
  – Create an empty node
  – Initialize the data member to the value to be stored
  – Fix the pointers for next and prev
  – Make sure any head and tail pointers are still valid
Deletion

- The node to be deleted is targeted
- A temporary pointer to that node is created
- The node is redirected around
- The temporary pointer is used to delete the node
Circular Lists

• Normally not a good idea
• But there are times when one might be useful
  – When you have multiple processes using a resource
• The end of the list simply points back to the head
• There is a tail pointer and that’s it.
• Data can be inserted either at the front or at the end
Skip Lists

- Linked lists have the drawback that each node must be searched when looking for data.
- Ordering the data can solve this to some degree, but you still must sequentially scan the list.
- Skip lists solve this problem of sequential search.
Searching

• When searching a skiplist the search begins at the first node at the highest level
• If the element we are looking for is not found, then we drop down a level and try again
• This is continued until the element is found, or the list is exhausted.
Insertion

• Searching seems straightforward, but what about insertion
• How do you decide how many levels to make the new node?
• We abandon the notion of having nodes with a certain number of levels in certain positions
• We keep the idea of nodes having a certain number of levels
More on Insertion

• But still, how do we choose the levels?
• If we assume that the maximum number of levels for a node is 4, then
  – For 15 elements, we need:
    • 8 – 1 pointer nodes
    • 4 – 2 pointer nodes
    • 2 – 3 pointer nodes
    • 1 – 4 pointer nodes
Still insertion

- So when we create a new node, we randomly choose the number of levels the new node will have.
- When the list is created we initialize an array of powers that will hold the cut off for each level.
- The array is filled using $2^j + 1$ for $j=0:maxlevel-1$. 
Space considerations

• Not all nodes need to have an array for the pointers containing all of the levels
• So we have a pointer to an array of pointers that will be the array of levels.
Self-Organizing Lists

• The motivation for skiplists was to speed the search process.
• A different idea, actually reorganizes the list through use
• The main idea is when an element is searched for and found, its position in the list is modified to make it easier to find next time
Ordering Heuristics

• There are many ways to re-order the list, here are 4
  1. Move-to-front: put the element at the beginning of the list
  2. Transpose: swap the element with its preceding neighbor
  3. Count: order the list by the number of times an element has been accessed
  4. Ordering: use some natural criteria for the ordering
Sparse Tables

• Many times, we need to store information about stuff, and a table is the natural way to store it
• For example, all the grades for all students in a university
• The problem here is that not all students take all courses and a lot of the table is empty.
What do you do?

- You can use, two one dimensional arrays of linked lists.
- One of the arrays is the classes and they point off to a linked list of students taking that course.
- The other array is the students in the university which points off to the classes taken by the student.
Ok…what about the linked list?

• The nodes in the linked list contain five data members
  – The student number
  – The class number
  – Grade
  – A pointer to the next student taking this course
  – A pointer to the next course this student is taking