CSE 326
Autumn 2005
Assignment 3
Due 10/19/05

For all algorithm and data structure design problems please provide elegant pseudocode and an adequate explanation of your methods. If is often helpful to include small examples demonstrating the method. Put your name at the top of each sheet of paper that you turn in.

1. Design a cursor implementation of a circular doubly linked list of integers. A node in a circular doubly linked list has three fields previous, next, and data. The nodes are arranged in the circular doubly linked list so that by following the next pointers from a node you eventually return to that node, and by following the previous pointers from a node you eventually return to that node. Remember that the cursor implementation only uses arrays and array indices to implement pointers. The dot notation is not allowed.
   (a) Give an initialization of your cursor data structure where all nodes are found on a free list.
   (b) Implement insert(p: integer, x: integer), where p is a "pointer" into a circular doubly linked list, and x is a data value. In this operation, a new node with data value x is inserted just before p in the list. Be sure to handle the case where the list is empty correctly.
   (c) Implement delete(p:integer):integer. In this operation you can assume the list is nonempty. The node pointed to by p is removed from the list and its data is returned. Don't forget to return the node pointed to by p to the free list when done.

2. On the problem about Horner's Rule from the first homework, we represented a polynomial by storing its coefficients in an array. This method of representation is not efficient if the polynomial is "sparse," meaning the number of nonzero coefficients is much less than the degree of the polynomial. We can instead represent such polynomials as a linked list in which each node represents a nonzero term, and the node exponents are in increasing order. The structure of each node is: record poly: (exp : integer, coef : integer, next : poly pointer). For example, the sparse polynomial $10 + 4x^2 + 8x^{23}$ is represented as $p \rightarrow |0|10| \rightarrow |2|4| \rightarrow |23|8|$ (where p: poly pointer). The zero polynomial is represented by the null pointer. Let $p(x) = a_1x^{c_1} + a_2x^{c_2} + \cdots + a_mx^{c_m}$ and $q(x) = b_1x^{d_1} + b_2x^{d_2} + \cdots + b_nx^{d_n}$ be two sparse polynomials represented in this way. We can multiply $p(x)$ and q(x) to yield a new polynomial $r(x)$ by $r(x) = r'(x) + r''(x)$, with $r'(x) = (a_1x^{c_1})q(x) = a_1b_1x^{c_1+d_1} + a_1b_2x^{c_1+d_2} + \cdots + a_1b_nx^{c_1+d_n}$ and $r''(x) = (a_2x^{c_2} + \cdots + a_mx^{c_m})q(x)$. Design a recursive pseudocode function [with signature `Mult(P, Q : poly pointer): poly pointer`] to multiply two sparse polynomials in this manner. The function takes pointers to the two polynomials

$p(x)$ and $q(x)$ and returns a pointer to $r(x)$. You are given a function `Add(P, Q : poly pointer): poly pointer` that you can use to add two polynomials. Assume that Add is destructive, that is, does not preserve its arguments. Your Mult function should also be destructive.

(a) Analyze the running time of Mult in terms of m and n the number of terms in $p(x)$ and $q(x)$ respectively. Assume that Add takes $O(n + m)$ steps. Because it is recursive, your time bound should first be defined by a recurrence, then the recurrence should be solved.

(b) Suppose we want to multiply two polynomials that have a different number of nonzero terms. Why does it matter (in terms of running time) which we choose as p(x) and which as q(x)?

(c) (Optional) Design and analyze a recursive sparse polynomial multiplication algorithm that uses divide and conquer and runs much faster than the one described above. (There are no additional points for optional parts. However, we will keep track of who does optional part to help in borderline grading cases.)