

CMSC 15200- Intro to Computer Science 2
Summer 2009
Homework #4 (8/8/2009)
Due Date: 8/12/2009 (by class time — 1:30 pm)

Exercise 1 - Ordering Data <<10 points>>

Write a call-by-reference function called `reorder` that takes as arguments references to 3 integers, `a`, `b` and `c` (in that order) and reorders them so that $a \leq b$ and $b \leq c$. In homework 2, you wrote a program that did this for an arbitrary number of numbers, but now you need to use a call-by-reference function. Use this function declaration:

```
void reorder(int & a, int & b, int & c)
```

Your output should look something like this:

```
$ ./prob1
Number #1: 3
Number #2: 2
Number #3: 8
Original values: (3,2,8)
Final values:   (2,3,8)
```

Exercise 2 - Iterative repeated squaring <<15 points>>

You should write an iterative version of the function `int pow(int a, int x)` from lab2. There is a very easy way to compute a^x iteratively:

```
int pow(int a, int x)
{
    int f = 1;
    for (int i=0; i<x; i++)
        f*=a;
    return f;
}
```

However, this takes x multiplications. You should write a program that takes the same number of multiplications as the recursive program from lab 2 exercise 2. The idea is the same:

- $5^8 = ((5^2)^2)^2$ (3 multiplications instead of 8)
 - $5^9 = 5 ((5^2)^2)^2$ (4 multiplications instead of 9)
 - $5^{10} = 5^2 ((5^2)^2)^2$ (4 multiplications instead of 10; 5^2 is computed only once)
- etc.

First write the function if x is a power of 2 (10 points if you only do this).

Now what about x not a power of 2? Note that you are multiplying together 5^t for t a power of 2. Moreover, the powers of 2 you are multiplying together add up to x : in the second example, you are multiplying together 5^1 and 5^8 , and $9=1+8$. In the second example, $10=8+2$. In fact, the powers of 2 you are adding to get x correspond to the digits of x that are 1 in binary.

Hint 1: You will have to compute $a^{(2^k)}$, where 2^k means 2^k for all k such that $2^k \leq x$. You can either compute these first, or compute them as you go along.

Hint 2: The $(k+1)$ st digit of x in binary (corresponding to 2^k) is $(x/2^k)\%2$, where in the division we round down.

For example,

10 is 0101 in binary, and
 $(10/1)\%2 = 10\%2 = 0$ (the first digit)
 $(10/2)\%2 = 5\%2 = 1$ (the second digit)
 $(10/4)\%2 = 2\%2 = 0$ (the third digit)
 $(10/8)\%2 = 1\%2 = 1$ (the fourth digit)

So to compute a^{10} we would multiply together $a^{(2^k)}$ for all the k such that $(10/2^k)\%2$ is 1.

Exercise 3 - Doubly-Linked List <<30 points>>

You will implement a doubly linked list (as described in class).

The structure and function declarations are the following (dlist.h in the homework files):

```
struct ListNode {
    int data;
    ListNode *next;
    ListNode *prev;
};

struct DList {
    ListNode *head;
};

void createList(DList &l);
ListNode* first(DList &l);
void insertHead(DList &l, int data);
void insertAfter(ListNode* node, int data);
void printData(DList &l);
bool find(DList &l, int data);
void deleteHead(DList &l);
void deleteAfter(ListNode* node);
void deleteData(DList &l, int data);
void deleteList(DList &l);

// New functions
/* Inserts a new node with provided data before the specified node */
void insertBefore(DList &l, ListNode* node, int data);
/* Deletes the specified node */
void deleteNode(DList &l, ListNode* node);
```

Don't reinvent the wheel. Reuse as much code as possible from the list implementation seen in class! In fact, you should only tweak the existing functions to make sure that the "prev" pointer always has a valid value, and then implement the new "insertBefore" and "deleteNode" functions.

Note that, although the book describes how to implement a doubly-linked list, you *cannot* take code directly from the book. This exercise also evaluates your ability to read code (both from the book and the provided list implementation) and adapt it to your own needs.

To test your list implementation, a main_double.cpp is provided in the homework files. Running this program with a correct doubly linked list implementation should yield the following:

5 4 3 2 1 5 4 3 2 1

```

5 1 4 3 2 1 5 4 3 2 1
1
0
1 4 3 2 1 5 4 3 2 1
1 3 2 1 5 4 3 2 1
1 2 1 5 4 3 2 1
42 1 2 1 5 4 3 2 1
42 37 1 2 1 5 4 3 2 1
37 1 2 1 5 4 3 2 1
37 1 2 1 4 3 2 1
37 2 4 3 2
List is empty!

```

Note: Debugging programs that use data structures is no easy matter, and you are likely to encounter run-time errors due to dangling pointers. One good strategy is to draw the data structure on paper, and for each operation see how the "next" and "prev" pointers are affected. Don't forget to consider special cases (e.g. "What happens if I try to delete the *first* node?") Also, if your program crashes unexpectedly (the hallmark of a dangling pointer), the Eclipse debugger can come in handy to pinpoint the dangling pointer (the debugger will pause execution at the exact line that caused the crash).

Note 2: In your code, make sure you explicitly point out (with comments) what code you had to add/modify to turn the list into a doubly linked list.

The point weights for this exercise are the following: <<10 points>> for correctly updating the "prev" pointer, <<10 points>> for implementing insertBefore, <<10 points>> for implementing deleteNode.