EE 205 — Final Exam

Spring 2012

Name: _______________________

login id: _____________________

May 10, 2012

This is a open book, open notes, open computer, closed neighbor exam. You will be allowed access to the peecee during the entire exam; however, you may only use a web browser to access the EE 205 course web site (no Google, no chats, no email etc.), and you may use SSH to login to wiliki to edit and compile files (no chats, no email etc.).

This exam consists of 2 problems on 12 numbered pages, including this cover sheet.
1. **Short (but complete) Answer**

   (a) (5 Points). Explain the difference between Static and Dynamic binding and explain why it is important.

   (b) (5 Points). What is a Virtual Function Table? When is it used? Show how it is used.

   (c) (5 Points). When should a parameter be passed by reference or by value?
(d) (5 Points). What is an Abstract Base Class? How can you tell if a class is abstract?

(e) (5 Points). What are “friends”? Who can be friends? When should they be used?

(f) (5 Points). At a minimum, what operations must an iterator provide?
(g) (5 Points). What considerations should you make in deciding whether to use Layering or Inheritance in building one class on top of another?

(h) (5 Points). What is a member initialization list? When MAY it be used? When MUST it be used?
2. **2D Container**

We have had so much success with our Containers this semester (Array and List based, with Iterators, and Template), that we would like to take them to the next level - 2D! For example, we can use our Container2D class for a Matrix application (build our own MatLab). Since this is a 2 hour (doable) exam, we will not provide a full implementation of the Container2D class, but just enough as a “proof of concept”. However, we will look at three different approaches to implementing the class.

(a) **Quick and Dirty** (15 Points).

Our first implementation will just build on top of our existing List based Container similar to how C linearizes 2D arrays in memory - using row major order. Consider the (partial) class declaration below. On the following page, show the code implementing the two functions listed there. For the constructor, we will assume the Container2D is of fixed size given in the arguments, and all Entries are initialized to 0. (Hint: you should not need the entire page).

```cpp
#include "container.h"
#include "entry.h"

typedef Position Row;
typedef Position Col;

class Container2D { 
public:
    Container2D(int maxr, int maxc = 10);
    Entry get( Row r, Col c);

private:
    int RowMax;
    int ColMax;
    Container M;
};
```

Below, describe the advantages and disadvantages of this approach.
Implementation One
(b) **Better** (15 Points).

With some of the problems listed above, we decide to try a different approach using our templatized Container. We will implement the `Container2D` as a Container of Containers of Entries; i.e. a Container of rows, each of which is a Container of column Entries.

```cpp
#include "Tcontainer.h" // the templatized container
#include "entry.h"

typedef Position Row;
typedef Position Col;

template <class Entry>
class Container2D { public:
    Container2D< Container<Entry> >(int maxr, int maxc = 10);
    Entry get( Row r, Col c);

    private:
    Container< Container<Entry> > M;
};
```

On the next page, show the implementations of these two member functions under the same assumptions as above.
Implementation Two
(c) **Best** (30 Points).

While it is nice to build our Container2D from our existing Container, one of the problems with the previous implementations is they are not very space efficient for large sparse Matrices (where most entries are zero; common in many matrix applications). So we decide to re-implement the Container2D from scratch as a multiply linked collection of nodes storing only the non-zero Entries. In addition to the Entry (the payload), these MNodes contain the Row and Column coordinates of the non-zero Entry and pointers to the next row and column MNodes, if any. A partial class declaration of an MNode looks like:

```c++
class MNode {
    public: // You provide this part
    private:
        Row r;
        Col c;
        Entry data;
        MNode * next_col;
        MNode * next_row;
}
```

Complete the public part of the declaration with member functions you think will be needed.
Draw the picture of the Container2D holding the following 4x5 Matrix:

\[
\begin{array}{cccccc}
9 & 0 & 0 & 8 & 0 \\
7 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
-1 & 6 & 0 & -8 & 0 \\
\end{array}
\]

(Hint: You may have an extra column heading row and an extra row heading column in your representation).
Implementation Three

Using these MNodes, a partial class declaration for the Container2D is:

```cpp
#include "entry.h"
#include "mnode.h"
typedef Position Row;
typedef Position Col;

class Container2D {
    public:
        Container2D(int maxr, int maxc);
        Entry get( Row r, Col c);

    private:
        MNode *M;
};
```
Implementation Three - Bonus

For up to 10 Bonus points (capping at 100 for the exam), show the implementation of a replace() member function that changes the Entry at some location of the Matrix, returning the old value and preserving the non-zero Entry only property of the Container2D. The prototype is:

```cpp
Entry replace(Entry x, Row r, Col c);
```