CISC 220 – 010 Homework #6 Due Monday November 20, 2000

For the next two weeks' homework, we will begin working with various versions of searchable tables, and with and with several access methods for the tables. The access methods we will implement will be Key-index, hashing, and binary search tree – each as a separate class. In each implementation, we'll have the same application to test the class – with some high-level functions (client functions) find and display, store, and update --- in order to see how the access method is used, and to debug our classes. You can use the inventory example data, and the Part class as specific example, but make sure your part class has a cast operator that returns a key (part number). As usual, use a step-by-step approach to develop your codes.

Part A) A key-index class. For a first step, define and implement a key-index class as follows.

```
template <class K>
class KeyRec{
   private:
      K theKey;
      int index;
   friend class KeyIndex;
};
template <class K>
class KeyIndex {
   private:
      KeyRec* keyIndexArray;
      int maxSize;
      int currentSize;
   public:
      KeyIndex(int maxS );
      ~KeyIndex();
      int insertKey( const K& theKey, int index);
      int deleteKey( const K& theKey );
      int keyPosition( const K& theKey );
};
```

This is basically just an array of key-index pairs, but kept in order by key. The function `insertKey` must insert the new key in the correct location in the array. Function `deleteKey` should remove the key-index pair and close up the "hole" (use binary search to find the key to be deleted). And function `keyPosition` should lookup the key using the binary search method, and return the corresponding index – into the table (see part B). Make a simple driver program to test that the

class methods work, then make a script file (HW-6-A.scr) in which you cat keyIndex.h, your driver code, and show a sample run.

Part B) Now Implement the Table ADT - using the following sample class definition for starters. Add appropriate predicate methods to complete the class.

```
template <class T>
class Table{
   private:
      T* dataArray;
      int currentSize;
      int maxSize;
   public:
      Table(int size = 100);
      ~Table();
      int insert( int pos, const T& thisOne);
      int retrieve( int pos, T& thisOne);
      int remove( int pos, const T& thisOne );
};
```

In this case, think of class T as a record – e.g., Student, Employee, or Part. You can do a quick test of your Table class by using the Part class. A simple driver that includes the necessary classes, instantiates a table, and does a few inserts, retrieves, and removes will be sufficient to debug syntax errors. Once this class compiles without error, make a script file (HW-6-B.scr) cat table.h, and compile and run your test driver.

Part C) An indexed table has a both a table for the data and a keyIndex for lookup. A starting definition for this class might look like:

```
template <class T, class K, int Size>
class IndexedTable{
   private:
      Table theTable(Size);
      KeyIndex theIndex(Size);
   public:
      int insert( const T& thisOne );
      int remove( const K& theKey, T& thisOne );
      int find( const K& theKey, T& thisOne );
      bool full();
      bool empty();
      int getCurrentSize();
};
```

Note that we are using the "has-a" relationship – and indexed table has an index and it has a table. It should be fairly clear what each of these functions does. For example, the insert function must call the Table insert function, and then the KeyIndex insert function (and, of course, also check for error conditions). The find

function will use a KeyIndex function to get the position, and then use the Table retrieve function to get the record. Once your class is written write a driver that reads the inventory file, builds an indexed table of Parts, and does some simple operations to test the class. Once the class works make a script file (`HW-6-C.scr`) in which you cat IndexTable.h, your driver code, compile and run.

Part D) Discuss the running time of the IndexedTable functions: insert, remove, and find. Discuss the performance of this kind of table for 1) a database where the vast majority of operations are table lookups; 2) a database where there a many lookups, some insertions, but very few removals; and 3) a database in which removals are just as frequent as insertions. Give an example of actual databases for these three situations.

What to hand in: Hand in the three script files generated above, and your written answers to Part D, stapled together as one set. (Don't forget to print your name on the front page!)