CS 202 – Fundamental Structures of Computer Science
Section 1, Spring 2003
Homework 2

Instructions:

- You will use C++ to implement the programming exercises. We suggest using Visual C++ 6.0.
- You will make a demo of the programs that you write and submit. You will be asked questions about your programs.
- The class definitions that you will use must include the provided class definitions in the figures of this assignment (same identifiers must be used). You can enhance these classes in the way you want.
- More information about submission rules and testing will be provided later.

---

1. Exercise 5.1
2. Exercise 5.19
3. Exercise 5.20 – Write a program that will implement extendible hashing (Figure 1). The skeleton of basic classes that you will use is given in Figure 2. You can enhance these classes, but the given data members and member functions must be there and must be used.
4. Write a program that will implement heap data structure using a binary tree (not an array as we have seen in the class). The skeleton of basic classes that you will use is given in Figure 3. You can enhance these classes, but the given data members and member functions must be there and must be used.

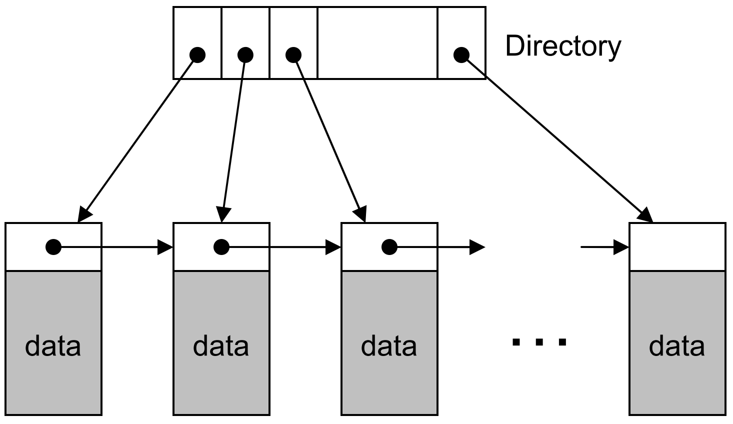

Figure 1

```
class Directory
{
        // it is upto you how to define and implement this class
        friend class ExHashTable;
};

class diskBlock
{
        unsigned int  array[16]; // a block can hold 16 positive integers (M=16)
        diskBlock     *next;     // points to the next block in the chain
        // … any more private data members you want to include
        friend class ExHashTable;
};

class ExHashTable
{
        diskBlock *hash(unsigned int x); // return the pointer to the diskBlock
                                  //  which may contain x
        int insert(unsigned int x);  // insert integer x into the table
                                   // - return success or failure
        int delete(unsigned int x); // delete item x from the table
                                    // – return success or failure
        int find(unsigned int x);    // return the item x if found,
                                     // otherwise an error condition
        buildTable(char filename[]); // read integers from a file
                                        // and insert them into the table;
                                        // each line of the file stores one integer.
        prinTable();     // print the hash table.
        deleteTable();  // free all diskBlocks and initialize the directory.
        //…. Any more public member functions you want to implement
   private:
        Directory    dir;         // directory that is used to reach the blocks.
        diskBlock   *head;     // head of chain of disk blocks
        // … any more private data members you want to include

        // … any more private member functions you want to implement.
};
```

Figure 2

```
class Element
{
        int key;                // key for item (use this in heap operations)
        char data[12];          // some data about item – just leave it there –
                                // you don't have to use this data member,
                                // but you have to use this class and its structure.
        friend class heapNode;
        friend class Heap;
};

class heapNode
{
        Element    item;        // item that is stored in the heap node.
        heapNode *left;         // points to the left child
        heapNode *right;        // points to the right child
        heapNode *parent;       // points to the parent node
        friend class Heap;
};

class Heap
{
        int insert(int  x);      // insert a new item, which has x as key, into the heap
        int deleteMin();         // delete the minimum keyed item from heap
        void buildHeap(char filename[]);    // build a heap from integers
                                            // stored in a file. Each line of the file
                                            // stores exactly one integer.
        printHeap();                        // print the content of the heap
  private:
        int size;               // current size of the heap – initially zero.
        heapNode *root;         // points to the root node of the heap
        heapNode *last;         // points to the last node of the heap
};
```

Figure 3