EECE2560 Fundamentals of Engineering Algorithms
Department of Electrical and Computer Engineering

Style and Documentation Guidelines

All code you submit must meet the requirements listed below. The functions and declarations at the bottom give examples of how your code should look.

- Each file should begin with a comment that explains what the file contains. The top of each file should include your name and the problem set number.
- Your code and documentation should be no more than 80 columns wide. Text should also not usually be wrapped around at a width of less than 80 columns. In other words, use the full 80 characters. Do not put function parameters or long expressions on separate lines unless you have to.
- Each function must be documented as shown below. At the top of the function (or in the prototype), precisely describe what the function does, what the input parameters are, and what assumptions the function makes, if any. These comments should be placed immediately after the function name or prototype. You do not need to document code in libraries that I have given you.
- Within the code, put comments that describe how the function works.
- Put spaces on either side of arithmetic $(*, /, +, -)$, logical $(\&\&, ||, !)$, relational $(<, <=, >, >=, ==, !=)$ and assignment operators $(= +=, -=, *=, /=)$, and the $<<$ and $>>$ operators.
- Put brackets { } on lines by themselves, and indent them as shown below.
- Avoid using global variables, except for certain constants. Global identifiers should be declared starting in the leftmost column. All other identifiers, including local variables, should be indented at least 3 columns.
- All #include statements should be at the top of each file. Global variables should also be placed at the top of each file.
- Comments on lines by themselves should be indented the same amount as the lines they refer to, and should have a blank line before them. Multi-line comments usually should have an empty space after them. For instance, instead of:

```
   x = 0;

// list (no shift if index == size+1)
   for (int pos = size; pos >= index; --pos)
      items[translate(pos+1)] = items[translate(pos)];
// next code
```

use

```
x = 0;

// list (no shift if index == size+1)
for (int pos = size; pos >= index; --pos)
   items[translate(pos+1)] = items[translate(pos)];
// next code
```

- if statements, for-loops, while-loops and do-while loops should be preceded by a blank line and followed by a blank line (or a line with a bracket).
- Put spaces before, but not after, the arguments of for-loops, and put a space after the "if," "for," and while keywords. For instance, instead of

  ```
  for(int pos = size;pos>=index ;--pos)
  ```

  use

  ```
  for (int pos = size; pos >= index; --pos);
  ```

- Final brackets } that are more than about 10 lines from the initial bracket should be commented, as shown below.
- Comments within the class declarations (at the end of the lines) should be lined up as much as possible.
- Do not put extra spaces between parentheses in expressions:

  For example, instead of

  ```
  while ( (x <= 0) || (y > N) ),
  ```

  use

  ```
  while ((x <= 0) || (y > N)).
  ```

- The commas separating the list of parameters in function calls and prototypes should be followed by a space, as in the following:

  ```
  void swap(int &x, int &y)
  ```

```
// Homework 1                                                    Ningfang Mi
//                                                               ningfang@ece.neu.edu
//
// Main program file for homework 1.  Contains declarations for Node,
// linkedListInsert, insert, and find.
//

#include "ListException.h"
#include <iostream>
using namespace std;

typedef desired-item-type ItemType;


struct Node
// The basic Node type used in the linked list
{
   ItemType item;
   Node *next;
}; // end struct

void List::linkedListInsert(Node *headPtr, ItemType newItem)
// Inserts a new node containing item newItem into the list pointed
// to by headPtr.
{
   int x,y;
   float f = 1.2;

   if ((headPtr == NULL) || (newItem < headPtr->item))
   {
      // base case: insert newItem at beginning
      // of the linked list to which headPtr points

      Node *newPtr = new Node;

      if (newPtr == NULL)
      {
         cout << "Error" << endl;
         cout << "in function" << endl;
      }
      else
      {
         newPtr->item = newItem;
         newPtr->next = headPtr;
         headPtr = newPtr;
      } // end if
   }
   else
```

```
      linkedListInsert(headPtr->next, newItem);
} // end linkedListInsert


void List::insert(int index, ListItemType newItem, bool& success)
\\ Inserts an item into the list at position index.  If insertion is
\\ successful, newItem is at position index in the list, and other items
\\ are renumbered accordingly, and success is true; otherwise success is
\\ false.  Note: Insertion will not be successful if index < 1 or index >
\\ getLength()+1.
{
   success = bool( (index >= 1) && (index <= size+1) && (size < MAX_LIST) );

   if (success)
   {
      // make room for new item by shifting all items at
      // positions >= index toward the end of the
      // list (no shift if index == size+1)

      for (int pos = size; pos >= index; --pos)
         items[translate(pos+1)] = items[translate(pos)];

      // insert new item

      items[translate(index)] = newItem;
      ++size; // increase the size of the list by one

   } // end if
} // end insert


ListNode * List::find(int index) const
// Locates a specified node in a linked list.  Index is the number of the
// desired node.  Returns a pointer to the desired node. If index < 1 or
// index > the number of nodes in the list, returns NULL.
{
   if ((index < 1) || (index > getLength()))
      return NULL;

   else // count from the beginning of the list
   {
      ListNode *cur = head;
      for (int skip = 1; skip < index; ++skip)
         cur = cur->next;
      return cur;
   } // end if
} // end find
```

```
// Header file TableH.h for the ADT table.
// Hash table implementation.
// Assumption: A table contains at most one item with a
// given search key at any time.


#include "ChainNode.h"
#include "HashTableException.h"

typedef KeyedItem TableItemType;

class HashTable
{
public:
// constructors and destructor:

   HashTable();
   HashTable(const HashTable& table);
   ~HashTable();

// table operations:

   virtual bool tableIsEmpty() const;
   virtual int tableGetLength() const;
   virtual void tableInsert(const TableItemType& newItem)
         throw (HashTableException);
   virtual bool tableDelete(KeyType searchKey);
   virtual bool tableRetrieve(KeyType searchKey,
                              TableItemType& tableItem) const;

protected:
   int hashIndex(KeyType searchKey); // hash function

private:
   enum {HASH_TABLE_SIZE = 101};     // size of hash table
   typedef ChainNode * HashTableType[HASH_TABLE_SIZE];

   HashTableType table;              // hash table
   int Size;                         // size of ADT table
}; // end HashTable class

// End of header file.
```