CSCI 2170 **Lecture Notes (4)**

ADT as parameter and array of objects

- **Pass objects to function – by value vs. by reference**

  void Function1(Sphere s1, Sphere s2);

  - ***In most cases, objects should be passed by reference***

  void Function2(Sphere & s1, Sphere &s2);
  void Function3(const Sphere&s1, const Sphere &s2);

- **Array of objects**

  const int SIZE=10;
  Sphere manySpheres[SIZE];
  // default constructor is used to create the objects in the array.

  // data of individual objects may be modified later separately
  // each array element is an object of Sphere class
  for (int i=0; i<SIZE; i++)
  manySpheres[i].SetRadius(i*2);

  - **Pass array of objects to function**

  void Function1(Sphere mSphere[], int size);
  void Function2(const Sphere mSphere[], int size);

- **ADT list – array implementation**

```
///*****************************************
// Header file List.h for the ADT list
// Array-based implementation
//*****************************************
const int MAX_LIST =200;
typedef int ListItemType;    // can be easily changed to
                             // ADT of other types
#ifndef List_H
#define List_H

class List
{
public:
  List(); // default constructor
        // destructor is supplied by compiler

  // list operations:
  bool isEmpty() const;
  // Determines whether a list is empty.
  // Precondition: None.
  // Postcondition: Returns true if the list is empty;
  // otherwise returns false.

  int getLength() const;
  // Determines the length of a list.
  // Precondition: None.
  // Postcondition: Returns the number of items
  // that are currently in the list.

  void insert(int index, ListItemType newItem,
          bool& success);
  // Inserts an item into the list at position index.
  // Precondition: index indicates the position at which
  // the item should be inserted in the list.
  // Postcondition: 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.

  void remove(int index, bool& success);
  // Deletes an item from the list at a given position.
  // Precondition: index indicates where the deletion
  // should occur.
  // Postcondition: If 1 <= index <= getLength(),
  // the item at position index in the list is
  // deleted, other items are renumbered accordingly,
  // and success is true; otherwise success is false.

  void retrieve(int index, ListItemType& dataItem,
                bool& success) const;
  // Retrieves a list item by position.
  // Precondition: index is the number of the item to
  // be retrieved.
  // Postcondition: If 1 <= index <= getLength(),
  // dataItem is the value of the desired item and
  // success is true; otherwise success is false.


private:
  ListItemType items[MAX_LIST];
                          // array of list items
  int          size;      // number of items in list

  int translate(int index) const;
  // Converts the position of an item in a list to the
  // correct index within its array representation.
}; // end List class
// End of header file.

#endif
```

```
//*********************************************
// Implementation file List.cpp for the ADT list
// Array-based implementation
//*********************************************
#include "List.h" //header file

List::List() : size(0)
{
} // end default constructor


bool List::isEmpty() const
{
  return bool(size == 0);
} // end isEmpty


int List::getLength() const
{
  return size;
} // end getLength


void List::insert(int index, ListItemType newItem,
                  bool& success)
{
  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


void List::remove(int index, bool& success)
{
  success = bool( (index >= 1) && (index <= size) );

  if (success)
  {  // delete item by shifting all items at positions >
     // index toward the beginning of the list
     // (no shift if index == size)
     for (int fromPosition = index+1;
          fromPosition <= size; ++fromPosition)
       items[translate(fromPosition-1)] =
                    items[translate(fromPosition)];
     --size; // decrease the size of the list by one
  } // end if
} // end remove


void List::retrieve(int index, ListItemType& dataItem,
                    bool& success) const
{
  success = bool( (index >= 1) &&
                  (index <= size) );

  if (success)
     dataItem = items[translate(index)];
} // end retrieve

int List::translate(int index) const
{
  return index-1;
} // end translate
// End of implementation file.
```

```
//*********************************************
//   Client Program using ADT list
//*********************************************
#include “List.h”
#include <iostream>
using namespace std;

// user defined functions:
void  PrintInReverse(const List& aList);
void  SortList(List& aList);
```

```
int main
{
    // declare aList of “List” type
     List                aList;
     ListItemType    item;
     bool                success;

     for (int i=1; i<=MAX_LIST; i++)
     {
            cout << “Enter list item “ << i << endl;
            cin >> item;
            aList.insert(i,  item, success);
      }

      PrintInReverse(aList);
      SortList(aList);
}

void PrintInReverse(const List& aList)
{
            … <fill in>



}

void  SortList(List  &  aList)
{

            …..   < fill in > …



}
```