Data Structures

August 31, 2009

Deadline: September 25

Programming Assignment #2

Part A

Consider the Abstract Data Type (ADT) List

The ADT List operations:

- **createList()** // Creates an empty list.
- **destroyList()** // Destroys a list.
- **listIsEmpty()** // Determines whether the list is empty.
- **listLength()** // Returns the number of items in list.
- **listInsert(newPosition, newItem, success)** // Inserts newItem at position newPosition of a list, if 1 ≤ newPosition ≤ listLength() + 1 if newPosition ≤ listLength(), the items are shifted as follows the item at newPosition becomes the item at newPosition becomes the item at newPosition + 1, the item at newPosition + 2, and so on. Success indicates whether the insertion was successful.
- **listDelete(pos, success)** // Deletes the item at position pos of a list, if 1 ≤ pos ≤ listLength(). If pos < lengthList(), the items are shifted as follows: the item at pos+1 becomes the at pos, the item at pos+1 becomes the item at pos+1, and so // on. Success indicates whether the deletion was successful.
- **listRetrieve(pos, dataItem, success)** // sets dataItem to the item at position pos of a list, if 1 ≤ pos ≤ listLength(). The list is left unchanged by this operation. Success indicates whether the retrieval was successful.

Implement the ADT List in three different ways: a) using an array (but as efficiently as possible), b) using pointers so that the list can grow and shrink dynamically, and c) using the STL vector.

If need be you may declare the class List in a header file (.h) and include the implementation of its member functions /constructors/destructor in the same header file. A better way is to define the class *List* in a header file (.h) and the implementation of its member functions/constructors/destructor in an implementation file (.cpp).

Make sure that in case client programmers have (or want) to use another implementation of the ADT List, the client code can stay the same and only one line of the client code needs to be changed before recompilation.

Work incrementally! Exercise your implementation on some simple problem such as checking for balancedness of parentheses in a string (see Drozdek for the client code).

Part B

Think of a good problem or situation which is solved or dealt with by using an implementation of the above specified ADT List. Implement your solution.

You can work on this programming assignment in pairs. Put your name (or the two names of the team) in each source file. Also state clearly to which programming assignment your work pertains. Thirdly: mention the deadline for the assignment (for this assignment it is September 25). Fourthly: mention the date of your submission. Furthermore create a README file in which you state what you are turning in and if need be include directions on how to compile the code; possibly also sample input and output generated by your program. Submit your code and README file as a zipped file to Simon Zaaijer (email: szaaijer@liacs.nl).