Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2015-11-01

Linked list

Aim

In this laboration you will learn to handle dynamic memory and pointer structures. Specifically you will see how constructors, destructors and operators may help you keeping track of pointers and allocated memory.

Reading instructions

- Recursion
- Pointers
  - Address operator &
  - Dereference (content) operator *
- Inner classes
- The five important
  - Copy constructor (deep copy to new object)
  - Assignment operator (deep copy to existing object)
  - Move constructor (quick move from dying object to new object)
  - Move assignment (quick move from dying object to existing object)
  - Destructor (deep deallocation)
- Random generation (random_device, default_random_engine, uniform_int_distribution, etc.)

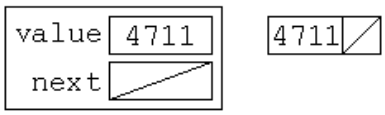
Assignment: Singly linked sorted list

You will create a straight singly linked list. It should consist of a class that represent the entire list. You need functions to handle the list correct in all situations of copying and assignment between lists, as well as functions to insert, remove and iterate items in the list. The list should of course be free of memory leaks no matter what.

Internally, as an inner class, you will have a class object representing a link in the list. A link will keep track of the value stored in this position, and the next link in the list. The chain of links will build the list.

As each link will keep track of the next link (the first link in the rest of the list) the list class need only keep track of the first link in the list. The list class should have *at least* functions for insertion, removal and printing of the list. You will notice more features are needed to test the list in a sensible way.

Figure 1 show the memory layout of a link with value 4711 and how we usually draw it in a simplified form. Drawing the pointer structure and performing operations step by step on the drawing is a good way of debugging the list.



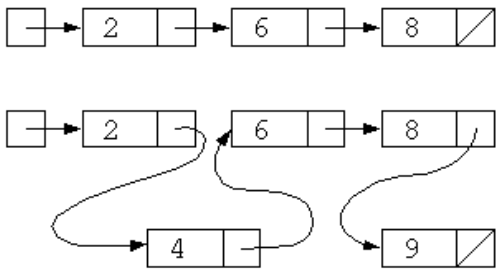
Figur 1: A link in memory, in detail and in simplified form

An important aspect of the list as a program modula and as an abstraction is the interface. The programmer that use the list should not know how the list is built internally. The link class and any functions pertaining it should thus be stashed away and inaccessible to her/him.

You should start by planning your insert and remove operations and draw how they will work on paper. It is suitable to start with an empty list, insert 5, 3, 9, 7, remove them again (in that order), and in each step think of which pointer variables that need modifications. Figure **??** show an example of a drawing after insertion of 6, 2 and 8. And then after insertion of also 9 and 4.

All functions you write should be tested. The test cases should cover normal as well as exceptional cases. Insertion first may for example need special treatment in the code, and should thus be tested more careful. More such cases are likely to exist, but may vary with solution.

During your work you should employ Test Driven Development (TDD). The idea in TDD is simple. Start by thinking of a simple test case that will bring your implementation one (only one) small step forward. Implement the test case (but not the code solving it) and watch it fail. You try the test case before implementing code to solve it to make sure the test case is correct and actually *can* fail. Once the test case is in place you implement *just enough* code to watch it succeed. Continue with a new test case. To help you we provide the start of a test program outlining the test cases you should write step by step.



Figur 2: Lista efter insättning av 6, 2, 8 och sedan efter 9, 4

All operations on your list must of course work on all special cases of a list, including empty lists. Copying and assignments from list to list should work as well, and create deep copies (changes in the

copy will not affect the original). Your test program will help you ensure this.

Note: The header file you implementation should be based on will be discussed during lecture or lesson.

REQUIREMENT: You should write the insert function *recursive* and the remove function *iterative*.