CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

November 8, 2017

Outline

Outline

A ListNode Class

- To support a Linked List container class, `LList`, a class `ListNode` is used for the individual nodes.
- A ListNode has two attributes: `item` and `link`.
- Public access is allowed for these attributes–the only class using the ListNode class is the `LList` class.

A ListNode Class

In Python:

- Data in a node can be of any type–a linked list can be heterogeneous.
- All values are references; the link attribute need not be declared to be a pointer.
- A link with the `None` value is used to indicate the end of a list.

A ListNode Class

In C++:

- A `typedef` statement allows the type of data to be specified at compile time. (The Linked List will still be homogeneous, but at least a different version of the class for another type can be compiled for another program by simply changing the `typedef` statement.
- The `item` attribute must be declared to be a particular type (for now).
- The `link` attribute must be declared to be a pointer to another `ListNode`.
- A link with the `NULL` value (a C++ pointer set to 0) is used to indicate the end of a list.

A ListNode Class

In C++:

- Later, we will see that the C++ Standard Template Library allows homogeneous lists for different data types (say, a list for the `int` type and a list for the `Rational` type) to be written at the same time, using a template class.
- Homogeneous lists for different data types can then be declared and used in the same program.
- The linked lists of these types will still be homogeneous. A list of integers can coexist with a list of doubles, but there can be no list containing integers and doubles mixed together.

Header File: ListNode.h

```
typedef int ItemType;
class ListNode {
   friend class LList;
public:
   ListNode(ItemType item, ListNode* link=NULL);
private:
   ItemType item_;
   ListNode *link_;
};
```

Header File: LList.h

```
class LList {
public:
   LList();
   LList(const LList& source);
   ~LList();
   LList& operator=(const LList& source);
   int size()  return size_;
   void append(ItemType x);
   void insert(size_t i, ItemType x);
   ItemType pop(int i=-1);
   ItemType& operator[](size_t position);
```

Header File: LList.h

```
private:
   void copy(const LList &source);
   void dealloc();
   ListNode* _find(size_t position);
   ItemType _delete(size_t position);
   ListNode *head_;
   int size_;
};
```

_FIND METHOD

```
ListNode* LList::_find(size_t position)
{
   ListNode *node = head_;
   size_t i;
   for (i=0; i<position; i++) {
      node = node->link_;
   }
   return node;
}
```

_DELETE METHOD

```
ItemType LList::_delete(size_t position)
{    ListNode *node, *dnode;
   ItemType item;
   if (position == 0) {
      dnode = head_;
      head_ = head_->link_;
      item = dnode->item_;
      delete dnode;
   }
```

_DELETE METHOD

```
    else {
       node = _find(position - 1);
       if (node != NULL) {
          dnode = node->link_;
          node->link_ = dnode->link_;
          item = dnode->item_;
          delete dnode;
       }
    }
    size_ -= 1;
    return item;
}
```

INSERT METHOD

```
void LList::insert(size_t i, ItemType x)
{
   ListNode *node;
   if (i == 0) {
      head_ = new ListNode(x, head_);
   }
   else {
      node = _find(i - 1);
      node->link_ = new ListNode(x, node->link_);
   }
   size_ += 1;
}
```

Destructor

```
LList::~LList() {
   dealloc();
}
void LList::dealloc()
{
   ListNode *node, *dnode;

   node = head_;
   while (node) {
      dnode = node;
      node = node->link_;
      delete dnode;
   }
}
```

Breaking Links

The integrity of a linked structure depends on the correct maintenance of all the links, since these are required to access the information in the structure. In our linked list class, if the `ListNode`'s `link_` attribute is set incorrectly, the resulting list will not be valid:

- If the link is incorrectly set to `NULL`, the list will be shortened, losing all data after that node. A memory leak will also occur, since there is no way to access the nodes to deallocate them.
- If the link is set to a node further along on the list, all nodes in between will be stranded: their data will be lost and their memory will not be deallocated.
- If the link is incorrectly to a node earlier in the list, then a circular structure results. traversing the list becomes an infinite loop.

Breaking Links

```
// this code is incorrect
void LList::insert(size_t i, ItemType x)
{
   ListNode *node;
   if (i == 0) {
      head_ = new ListNode(x, head_);
   }
   else {
      node = _find(i - 1);
      node->link_ = new ListNode(x); // incorrect
   }
   size_ += 1;
}
```