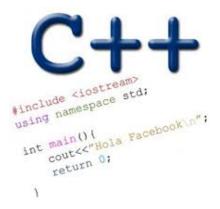
LINKED LISTS AND THE RULE OF THREE UNIT TESTING OPERATOR OVERLOADING

Problem Solving with Computers-II





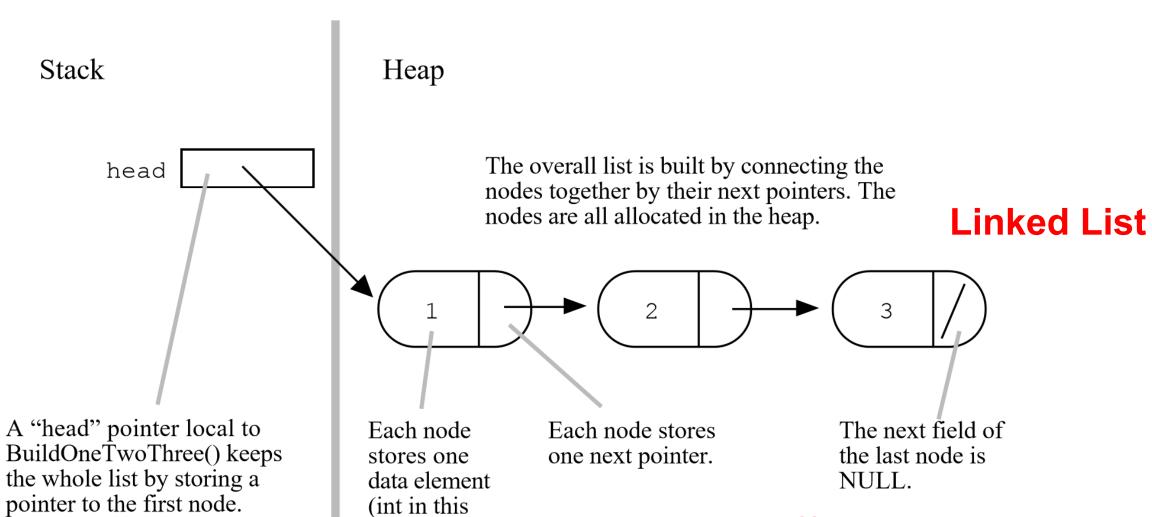
Linked Lists

The Drawing Of List {1, 2, 3}

1 2 3

example). What is the key difference between these?

Array List



Questions you must ask about any data structure:

- What operations does the data structure support?

 A linked list supports the following operations:
 - 1. Insert (a value)
 - 2. Delete (a value)
 - 3. Search (for a value)
 - 4. Min
 - 5. Max
 - 6. Print all values
- How do you implement each operation?
- How fast is each operation?

Linked-list as an Abstract Data Type (ADT)

```
class LinkedList {
public:
    LinkedList();
                                 // constructor
    ~LinkedList();
                                 // destructor
    // other methods
private:
    // definition of Node
    struct Node {
        int info;
        Node *next;
    };
    Node* head; // pointer to first node
    Node* tail;
```

Unit testing

- The goal of unit tests is to design your software robustly (usually viaTest Driven Development)
- For our purposes each public method of a class is a unit under test (UUT)
- Organizing your unit tests
 - One test class for every class under test.
 - If the class to test is Foo, the test class should be called FooTest (not TestFoo)
 - One test function for every public function of Foo. This a suite of individual test cases
- Test cases should be independent
- Test cases should be orthogonal
- For additional guidelines see: https://petroware.no/unittesting.html

Overloading Binary Comparison Operators

We would like to be able to compare two objects of the class using the following operators

```
and possibly others
void isEqual(const LinkedList & lst1, const LinkedList &lst2){
   if(Ist1 == Ist2)
       cout<<"Lists are equal"<<endl;
   else
      cout<<"Lists are not equal"<<endl;
```

RULE OF THREE

If a class defines one (or more) of the following it should probably explicitly define all three:

- 1. Destructor
- 2. Copy constructor
- 3. Copy assignment

The questions we ask are:

- 1. What is the behavior of these defaults?
- 2. What is the desired behavior?
- 3. How should we over-ride these methods?

Assume default destructor, copy constructor, copy assignment AND Correct implementation of the methods append() and vectorize()

```
void test_append_0(){
       string testname = "test 0: append [1] ";
       vector<int> v_exp = {1};
       LinkedList 11;
       11.append(1);
       vector<int> v_act = ll.vectorize();
       if(v_act!=v_exp){
              cout <<"\tFAILED "<<testname<<endl;</pre>
       }else{
              cout <<"\tPASSED "<<testname<<endl;</pre>
           What is the expected behavior of this code?
           A. Compiler error
           B. Memory leak
           C. Code is correct and the test passes
           D. None of the above
```

Behavior of default copy constructor

Assume that your implementation of LinkedList uses the overloaded destructor, default: copy constructor, copy assignment

```
I1:1->2->5->null
void default_copy_constructor(LinkedList& l1){
    // Use the copy constructor to create a
    // copy of l1
```

```
}
* What is the default behavior?

* Is the default behavior the outcome we desire ?

* How do we change it?
```

Behavior of default copy assignment

Assume that your implementation of LinkedList uses the override destructor, copy constructor, default copy assignment

```
destructor, copy constructor, default copy assignment
[1:1->2->5-> null

void default_assignment_1(LinkedList& 11){
   LinkedList 12;
   12 = 11;
}
* What is the default behavior?
```

Behavior of default copy assignment

Assume that your implementation of LinkedList uses the overloaded destructor, default: copy constructor, copy assignment

```
I1:1->2->5-> null
void test_default_assignment_2(LinkedList& 11){
    // Use the copy assignment
    LinkedList 12;
    12.append(10);
    12.append(20);
    12 = 11;
}
* What is the default behavior?
```

Next time

- Linked Lists contd.
- GDB