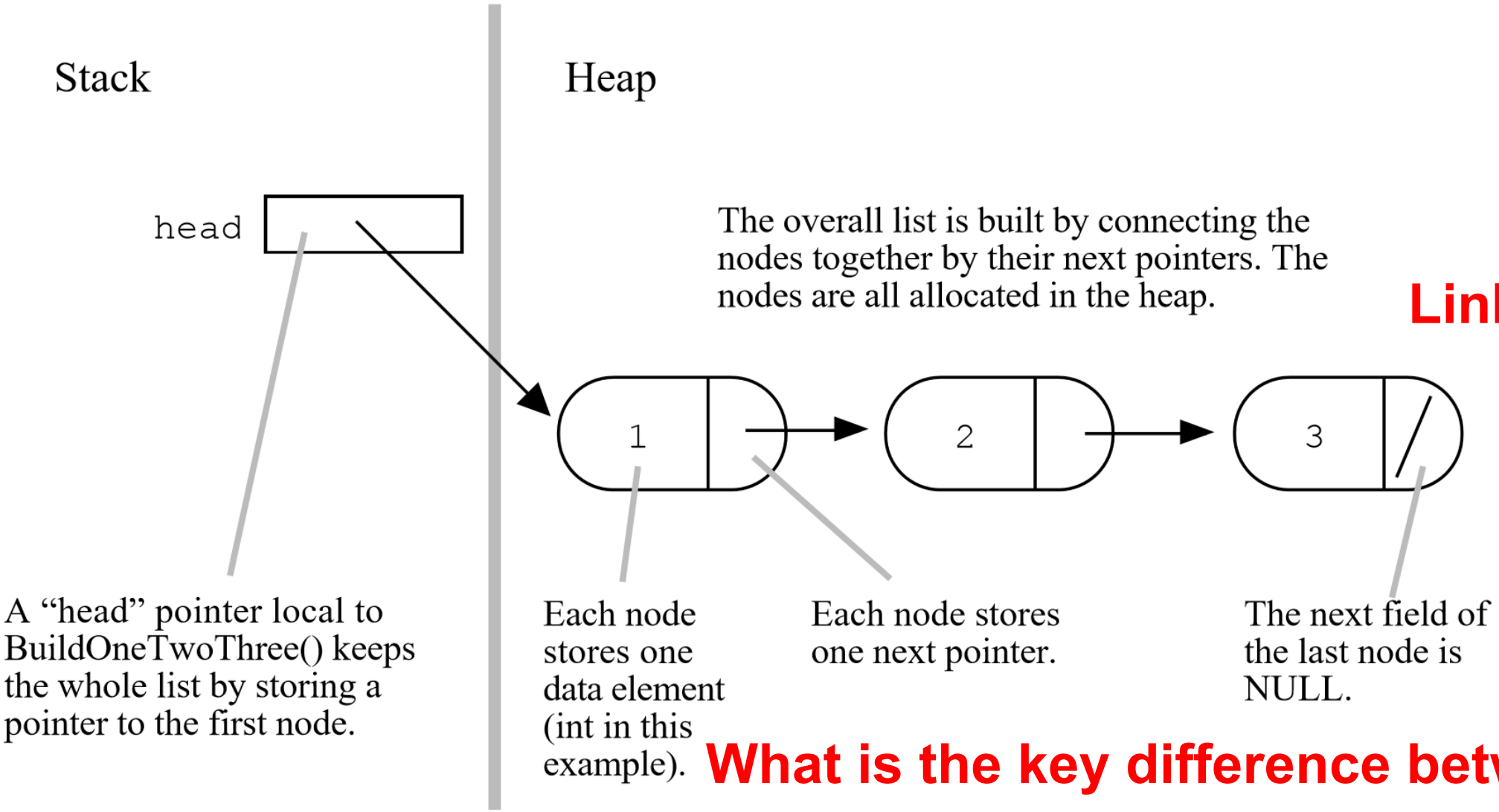
LINKED LISTS AND THE RULE OF THREE

Problem Solving with Computers-II

Linked Lists

Array List

The Drawing Of List {1, 2, 3}



Linked List

What is the key difference between these?

Questions of interest 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;
};
```

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?

Behavior of default destructor

```
void test_append_0(){
     vector<int> v_exp = {1};
     LinkedList ll;
     ll.append(1);
     vector<int> v_act = ll.vectorize();
     TESTEQ(v_exp, v_act, “test 0");
}
```

Assume:

destructor: default

copy constructor: default

copy assignment: default

What is the output?

A. Compiler error

B. Memory leak

C. Segmentation fault

D. Test fails

E. None of the above

Why do we need to write a destructor for LinkedList?

A. To free LinkedList objects
B. To free Nodes in a LinkedList
C. Both A and B
D. None of the above

Behavior of default copy constructor

```
void test_copy_constructor(){
   LinkedList l1;
   l1.append(1);
   l1.append(2);
   LinkedList l2(l1);
   TESTEQ(l1, l2, "test copy constructor”);
}
```

Assume:

destructor: overloaded

copy constructor: default

copy assignment: default

What is the output?

A. Compiler error

B. Memory leak

C. Segmentation fault

D. Test fails

E. None of the above

Behavior of default copy assignment

```
void test_copy_assignment(){
   LinkedList l1;
   l1.append(1);
   l1.append(2);
   LinkedList l2;
   l2 = l1;
   TESTEQ(l1, l2, "test copy assignment”);
}
```

Assume:

destructor: overloaded

copy constructor: overloaded

copy assignment: default

What is the output?

A. Compiler error

B. Memory leak

C. Segmentation fault

D. Test fails

E. None of the above

Write another test case for the copy assignment

```
void test_copy_assignment_2(){
// Write another test case for the copy assignment operator




}
```

Behavior of default copy assignment

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

l1 : 1 -> 2- > 5 -> null

```
void test_default_assignment_2(LinkedList& l1){
    // Use the copy assignment
    LinkedList l2;
    l2.append(10);
    l2.append(20);
    l2 = l1;
}
* What is the default behavior?
```

Next time

- Linked Lists counted
- Operator overloading
- Unit testing
- GDB