THE RULE OF THREE
LINKED LISTS (CONT.)

Problem Solving with Computers-II

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

GitHub

As the designer of a C++ class we need to

- the user of the class (determines the public functions needed)
- testing (each public function to make sure it is correct)
- manage any dynamic memory associated with objects of the class

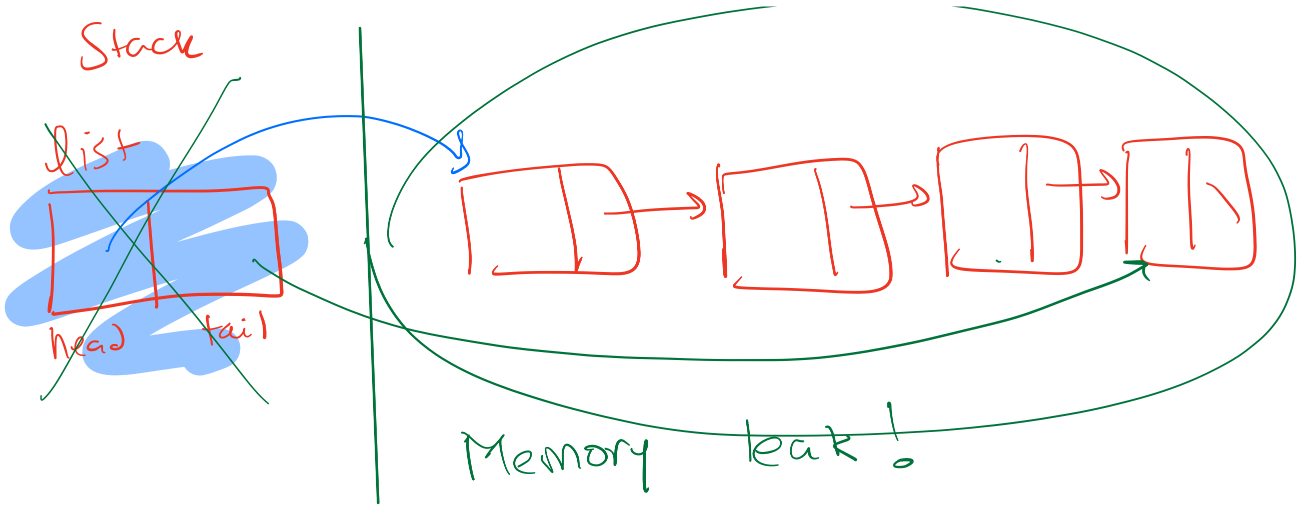
Rule of Three

Linked List Abstract Data Type (ADT)

```
class LinkedList {
public:
    LinkedList();
    ~LinkedList();
    // other public methods



private:
    struct Node {
        int info;
        Node* next;
    };
    Node* head;
    Node* tail;
};
```


Stack
list
head
tail
Memory leak!

Memory Errors

- Memory Leak: Program does not free memory allocated on the heap.

program crashes

- Segmentation Fault: Code tries to access an invalid memory location

(1) access memory location that doesn't exist

(2) 〃 〃 〃 that code doesn't have permission for.

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?

```
void test_append_0(){
      LinkedList ll;
      ll.append(10);
       ll.print();
}
```

Assume:

* **Default destructor**

* **Default copy constructor**

* **Default copy assignment**

What is the result of running the above code?

A. Compiler error

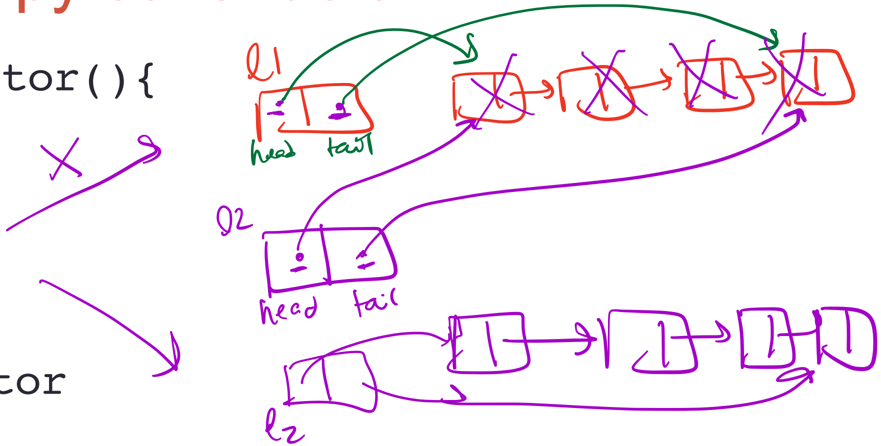
B. Memory leak

C. Segmentation fault

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};
    // calls the copy c'tor
    l1.print();
    l2.print();
}
```



Assume:

destructor: overloaded

copy constructor: default

What is the output?

A. Compiler error

B. Memory leak

[✓ C. Segmentation fault]

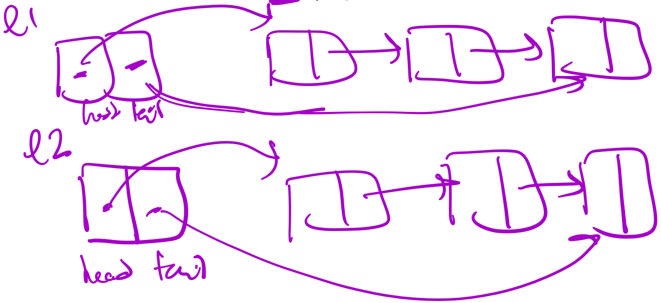
D. All of the above

E. None of the above

Behavior of default copy assignment

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

```
void default_assignment_1(LinkedList& l1){
    LinkedList l2;
    l2 = l1;
}
```



```
* What is the behavior of the default assignment operator?
```

Assume:

- **Overloaded destructor**
- **Default copy constructor**
- **Default copy assignment**

Behavior of default copy assignment

```
void test_default_assignment_2(){
    LinkedList l1, l2;
    l1.append(1);
    l1.append(2)
    l2 = l1;
    l2.print()
}
```

What is the result of running the above code?

A. Prints 1 , 2

B. Segmentation fault

C. Memory leak

D. A &B

E. A, B and C

Assume:

*** Overloaded destructor**

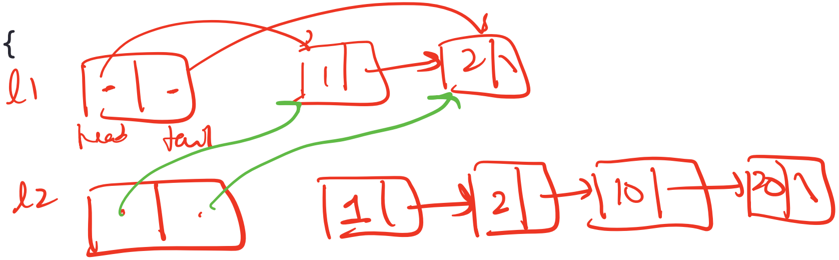
*** Default copy constructor**

*** Default copy assignment**

Behavior of default copy assignment

```
void test_default_assignment_3(){
    LinkedList l1;
    l1.append(1);
    l1.append(2)
    LinkedList l2{l1};
    l2.append(10);
    l2.append(20);
    l2 = l1;
    l2.print()
}
```

using defa



What is the result of running the above code?

A. Prints 1 , 2

B. Segmentation fault

C. Memory leak

D. A &B

[✓ E. A, B and C]

Assume:

* **Overloaded destructor**

* **Overloaded copy constructor**

* **Default copy assignment**

Overloading Operators

Overload relational operators for LinkedLists

==

!=

and possibly others

```
void test_equal(const LinkedList & lst1, const LinkedList &lst2){
    if (lst1 == lst2)
          cout<<“Lists are equal”<<endl;
    else
         cout<<“Lists are not equal”<<endl;

}
```

Overloading Arithmetic Operators

Define your own addition operator for linked lists:

```
LinkedList l1, l2;

//append nodes to l1 and l2;

LinkedList l3 = l1 + l2 ;
```

Overloading input/output stream

Wouldn’t it be convenient if we could do this:

```
LinkedList list;
cout<<list; //prints all the elements of list
```

Next time

- Binary Search Trees