LINKED LISTS (CONTD)
RULE OF THREE
OPERATOR OVERLOADING

Problem Solving with Computers-II

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

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

GitHub

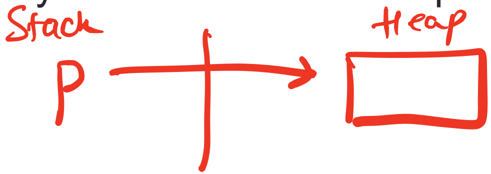
Why do we need to override the big four for linked list?

Memory Errors

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

```
void  foo () {
        int *p = new int;

}
```



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

① Dereferencing a null pointer

② " memory that was deallocated

③ Out of bound array access

④ Double free error

```
int *p = 0;
cout<< *p; //Def. segfault

p = new int;
→ delete p;
cout<< *p;
delete p; //double fre
```

RULE OF THREE

If a class overload one (or more) of the following methods, it should overload all three methods:

1. Destructor
2. Copy constructor
3. Copy assignment

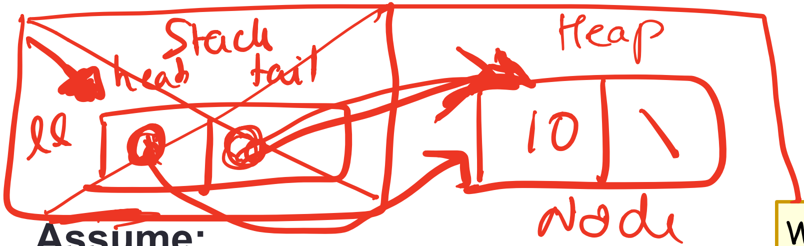
The questions we ask are:

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();
}
```

(Stack)

```
LinkedList :: ~LinkedList () {
    //
}
```



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

Why do we need to write a destructor for LinkedList?

A. To free LinkedList objects

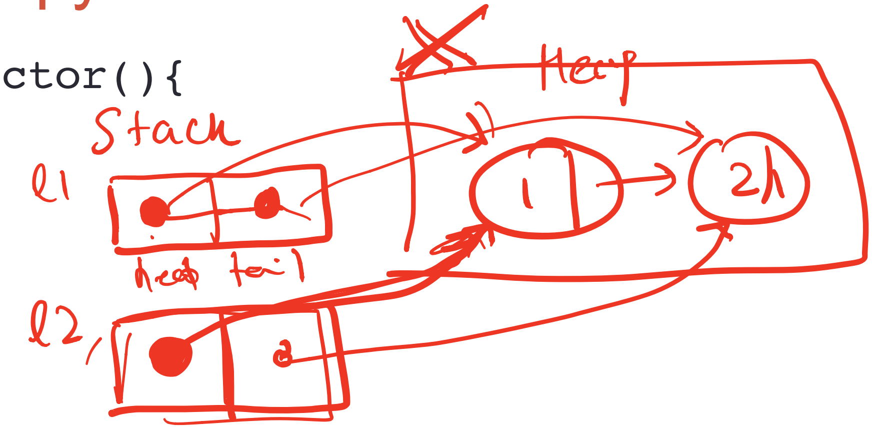
[✓ B.] To free Nodes in a LinkedList (that are generally created on the heap)

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);
   l1.print();
   l2.print();
}
```



Shallow copy

After the function returns l1's destructor is called which deletes all the nodes on the heap.

Assume:

destructor: overloaded

copy constructor: default

copy assignment: default

What is the output?

A. Compiler error

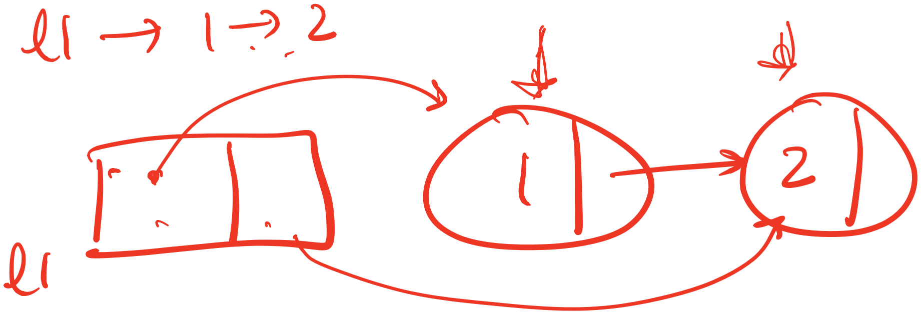
B. Memory leak

[✓ C. Segmentation fault]

D. Test fails

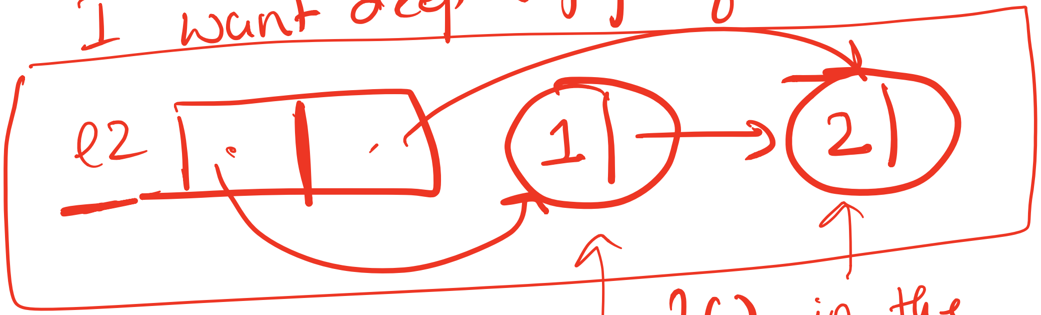
E. None of the above

l1 → 1 → 2



LinkedList l2 {l1};

I want deep copy of l1



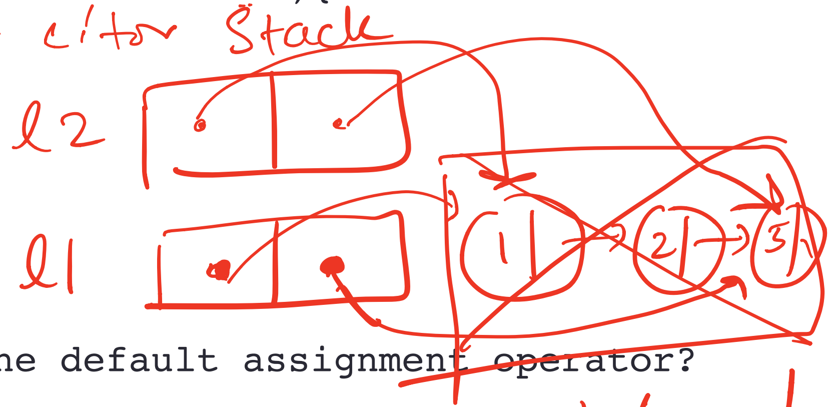
Reuse the append() in the copy c'tor!

Behavior of default copy assignment

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

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

// default c'tor



* What is the behavior of the default assignment operator?

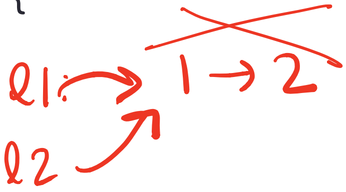
Assume:

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

Segfault waiting to happen!

Behavior of default copy assignment

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



l1 & l2's destructors are called. [l1's destructor deletes all the nodes
l2's destruct segfaults]

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**

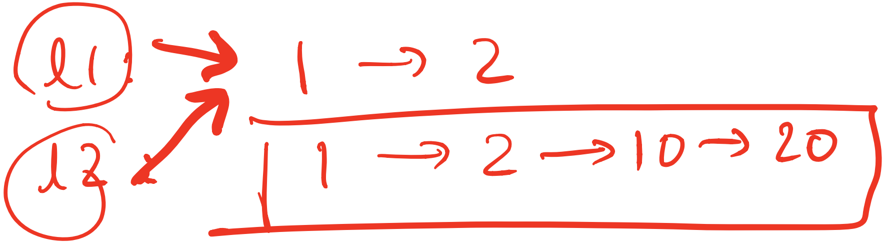
* **Default copy constructor**

* **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()
}
```

l2 already has some nodes.



Memory leak

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 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(lst1 == lst2)
            cout<<“Lists are equal”<<endl;
     else
           cout<<“Lists are not equal”<<endl;

}
```

Overloading Binary Arithmetic Operators

We would like to be able to add two points as follows

```
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

- Recursion + PA01