LINKED LISTS (CONTD)
RULE OF THREE
MEMORY ERRORS
OPERATOR OVERLOADING

Problem Solving with Computers-II

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

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

Memory Errors

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

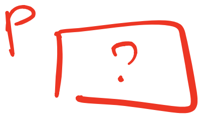
e.g.

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

p is removed from the stack but the object it points to is never free'd

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

```
Node *p = 0;
cout << p->data << endl;
// dereferencing a null pointer
```



RULE OF THREE

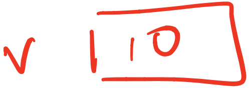
If a class overload one (or more) of the following methods, it should overload all three methods:

1. Destructor
2. Copy constructor
3. Copy assignment

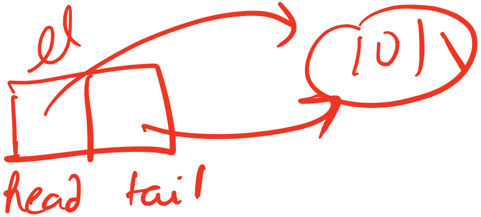
The questions we ask are:

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

Behavior of default destructor



```
void test_append_0(){
    string testname= “Append 10 to empty list”;
     vector<int> v = {10};
     LinkedList ll;
     ll.append(10);
     TESTEQ(ll, v, testname);
}
```



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

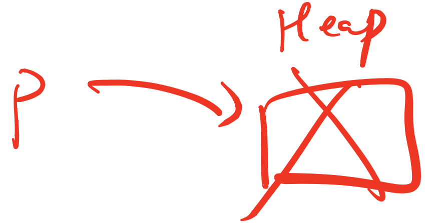
Why do we need to write a destructor for LinkedList?

A. To free LinkedList objects

[✓ B.] To free Nodes in a LinkedList

C. Both A and B

D. None of the above



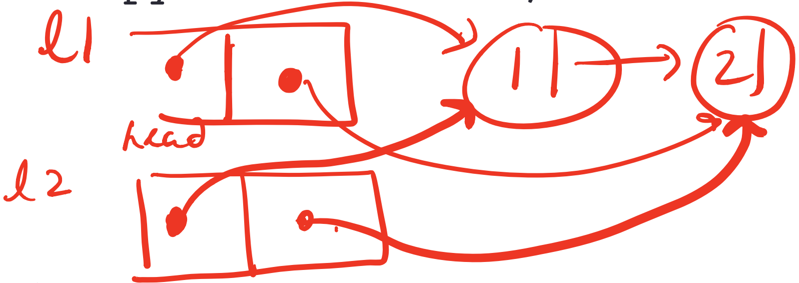
```
int * p = new int;

delete p;
delete p;  →  Segfault!
```

Deleting heap memory that has already been freed result in a segfault

Behavior of default copy constructor

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

other



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

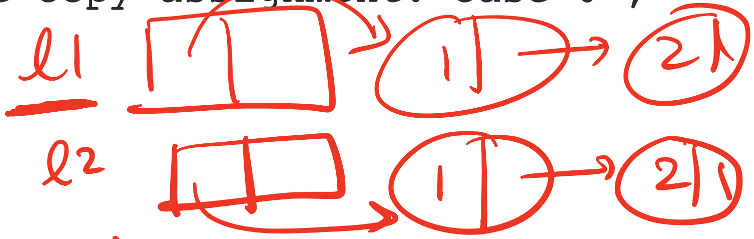
E. None of the above

Behavior of default copy assignment

```
void test_copy_assignment_0(){
  string testname = "test copy assignment: case 0";
   LinkedList l1;
   l1.append(1);
   l1.append(2);
   LinkedList l2;
   l2 = l1;
   TESTEQ(l1, l2,);
}
```

l2.operator=(l1)

copy-assignment function



Assume:

destructor: overloaded ✓

copy constructor: overloaded ✓

copy assignment: default

This line calls the copy assignment function of l2

What is the output?

A. Compiler error

B. Memory leak

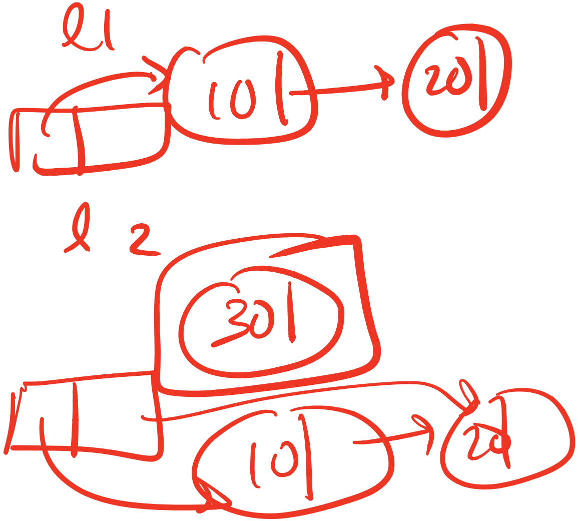
[✓ C. Segmentation fault]

D. Test fails

E. None of the above

Write another test case for the copy assignment

```
void test_copy_assignment_2(){
    Linked List l1 ;
     l1. append ( 10);
     l1. append (20);
     LinkedList l2;
       l2. append (30)
      l2 = l1 ;

}
```



If we used the same code as the copy constructor in our implementation of the copy assignment we would have a memory leak.

Overloading Binary Comparison Operators

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

==

!=

and possibly others

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 TESTEQ(const LinkedList & lst1, const LinkedList &lst2, string test){
    cout<<test<<endl;
    if(lst1.isEqual(lst2))
        cout<<“ PASSED”<<endl;
    else
        cout<<“  FAILED”<<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