THE RULE OF THREE (CONT.) LINKED LISTS WITH CLASSES

Problem Solving with Computers-II

Last lecture:

- Modified the Complex class member variables to be pointers
- Wrote user-defined versions of the
  - Constructor
  - Destructor
  - Copy-constructor

Assume:

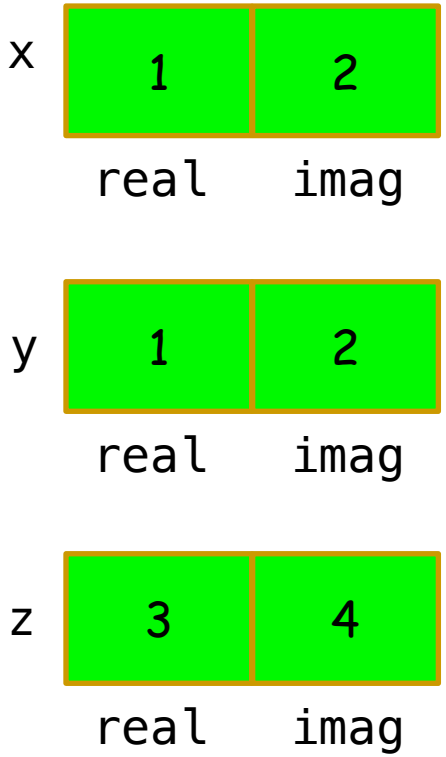
* **User-defined destructor**
* **User-defined copy constructor**
* **Default copy assignment**

```
class Complex
{
private:
    double *real;
    double *imag;
public:
    Complex(double re = 0, double im = 0);
    Complex(const Complex& other);
     ~Complex();
    double getMagnitude() const;
    double getReal() const;
    double getImaginary() const;
    void print() const;
    void conjugate();
    void setReal(double r);
    void setImag(double r);
    Complex operator+(const Complex& y);
};
```

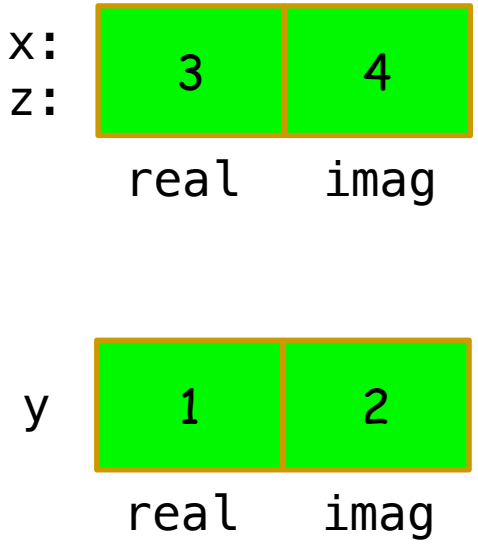
```
void bar(){
    Complex x(1, 2);
    Complex y(x);
    Complex z(3, 4);
    x = z;
}
```

Assume execution has reached the last line of bar(), which diagram correctly depicts the objects x, y, z (and the values stored in them)

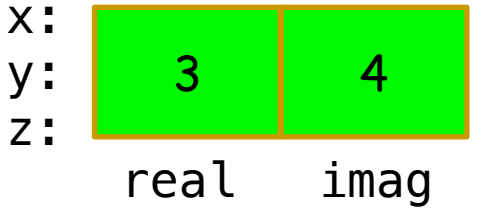
A



B



C



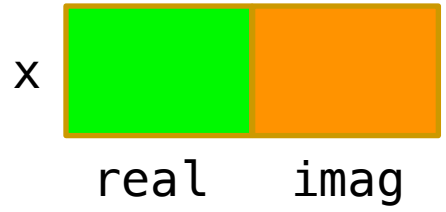
D: None

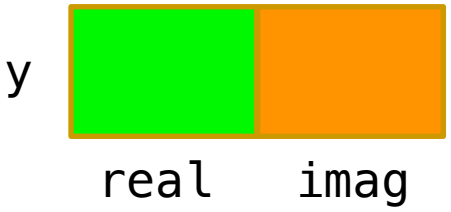
```
void bar(){
    Complex x(1, 2);
    Complex y(x);
    Complex z(3, 4);
    x = z;
}
```

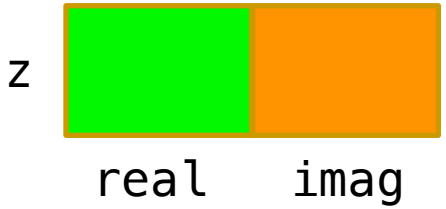
Which pointers have the same value?

Pointers with the same value are depicted with the same color

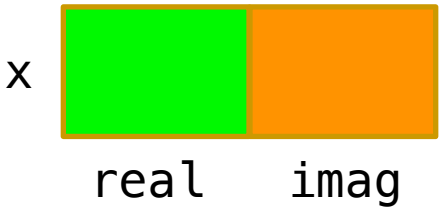
A

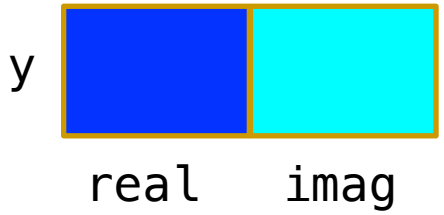


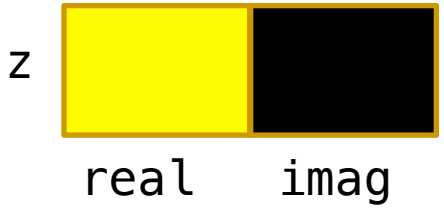




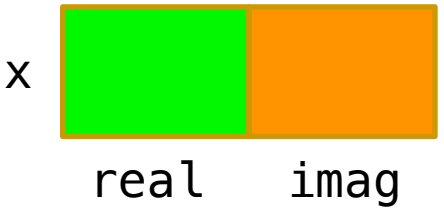
B

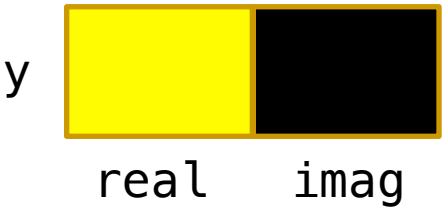


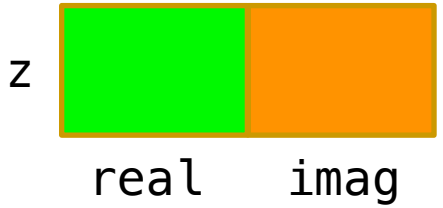




C







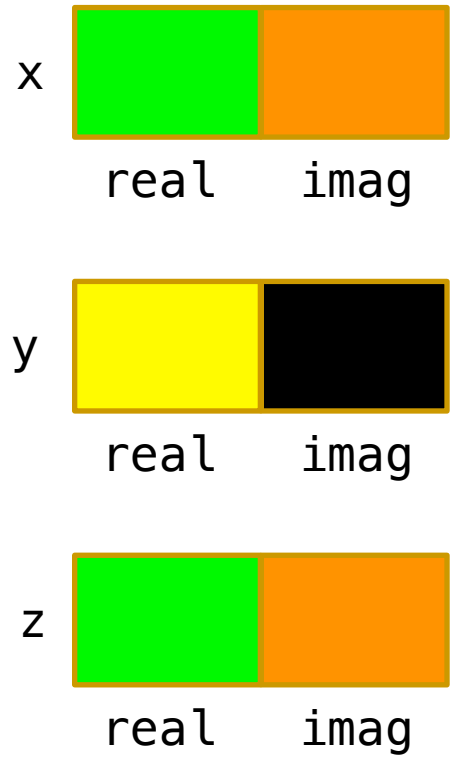
```
void bar(){
    Complex x(1, 2);
    Complex y(x);
    Complex z(3, 4);
    x = z;
}
```

Will calling bar() result in a seg-fault?

A. Yes

B. No

C. I don't know!



Assume:

* **User-defined destructor**
* **User-defined copy constructor**
* **Default copy assignment**

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

We answered the following questions for the Complex class:

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

Questions to ask about any data structure:

- **What operations does the data structure support?**

  A linked list supports the following operations:

  1. push_front (add a value to the head)
  2. append/push_back (add a value to the tail)
  3. delete (a value)
  4. search (for a value)
  5. min
  6. max
  7. print all values

- **How do you implement each operation?**
- **How fast is each operation?**

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;
};
```

Memory Errors

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

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

```
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: user-defined

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:

* **User-defined 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:

* **User-defined 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()
}
```

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:

* **User-defined destructor**
* **User-defined 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