OPERATOR OVERLOADING
GDB
RECURSION
INTRO TO PA01

Problem Solving with Computers-II

PA01: Card matching game with linked lists

Alice:

Bob:

Review PA01: Card matching game with linked lists

Correct output after running `make && ./game alice_cards.txt bob_cards.txt`:

```
Alice picked matching card c 3
Bob picked matching card s a
Alice picked matching card h 9

Alice's cards:
h 3
s 2
c a

Bob's cards:
c 2
d j
```

Note: 0=10, a=ace, k=king, q=queen, j=jack

Contents of `alice_cards.txt`:

Contents of `bob_cards.txt`:

GDB: GNU Debugger

- To use gdb, compile with the -g flag
- Setting breakpoints (b)
- Running programs that take arguments within gdb (r arguments)
- Continue execution until breakpoint is reached (c)
- Stepping into functions with step (s)
- Stepping over functions with next (n)
- Re-running a program (r)
- Examining local variables (info locals)
- Printing the value of variables with print (p)
- Quitting gdb (q)
- Debugging segfaults with backtrace (bt)

* Refer to the gdb cheat sheet: http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

Overloading Binary Comparison Operators

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

==

!=

and possibly others

Last class: overloaded == for LinkedList

Overloading input/output stream

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

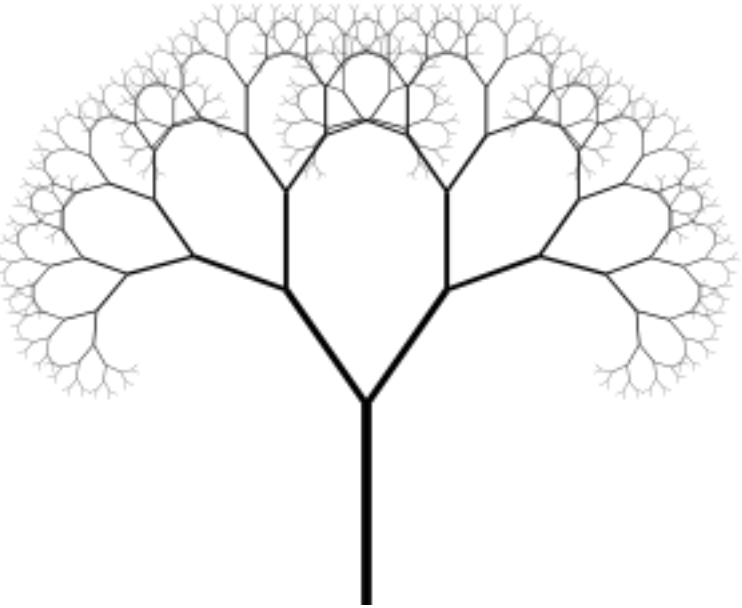
```
LinkedList list;
cout<<list; //prints all the elements of list
```

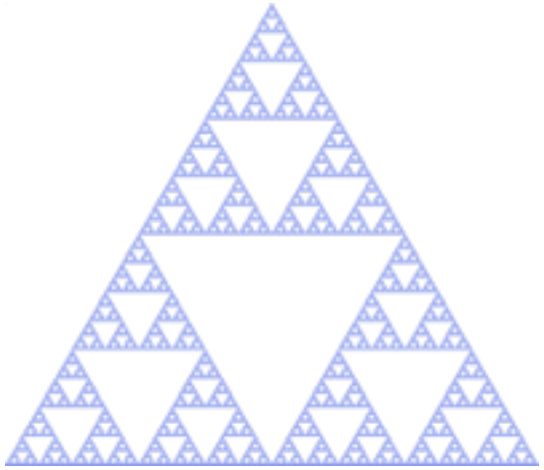
Overloading Binary Arithmetic Operators

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 ;
```

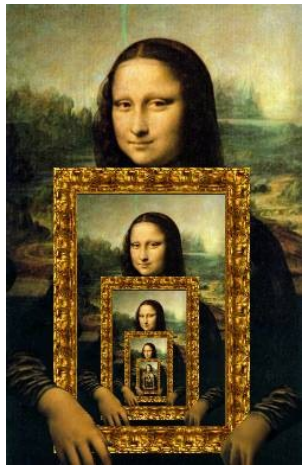
Recursion





Sierpinski triangle

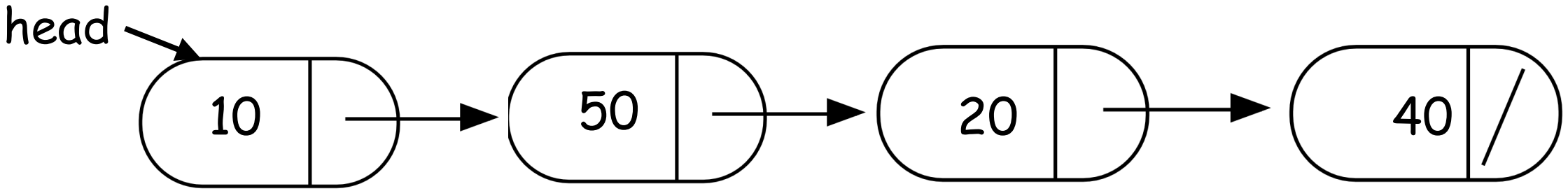
Zooming into a Koch's snowflake



Describe a linked-list recursively

Which of the following methods of LinkedList CANNOT be implemented using recursion?

A. Find the sum of all the values
B. Print all the values
C. Search for a value
D. Delete all the nodes in a linked list
E. All the above can be implemented using recursion



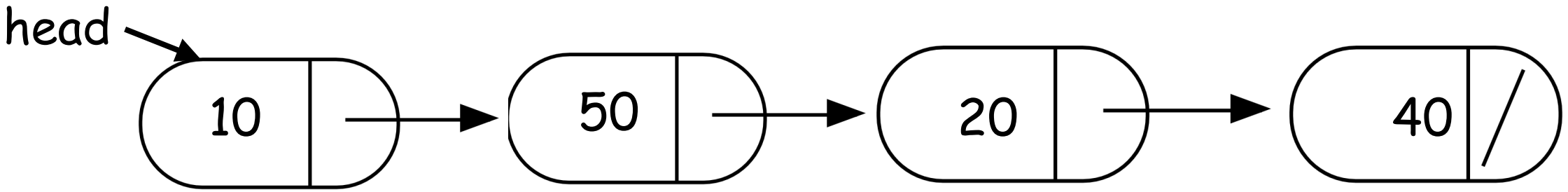

```
int IntList::sum(){

      //Return the sum of all elements in a linked list
   }
```

Helper functions

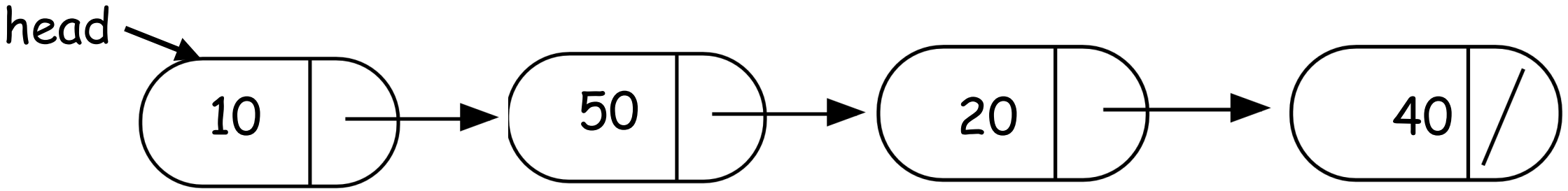
- Sometimes your functions takes an input that is not easy to recurse on
- In that case define a new function with appropriate parameters: This is your helper function
- Call the helper function to perform the recursion
- Usually the helper function is private

For example

```
Int IntList::sum(){

   return sum(head);
    //helper function that performs the recursion.

}
```




```
int IntList::sum(Node* p){




}
```



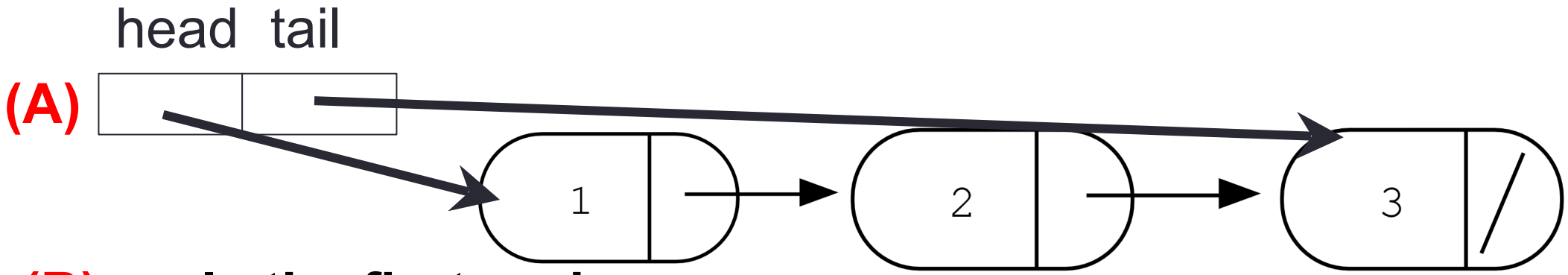

```
bool IntList::clear(Node* p){









  }
```

Concept Question

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
class Node {
    public:
        int info;
        Node *next;
};
```

Which of the following objects are deleted when the destructor of Linked-list is called?



(B): only the first node

(C): A and B

(D): All the nodes of the linked list

(E): A and D

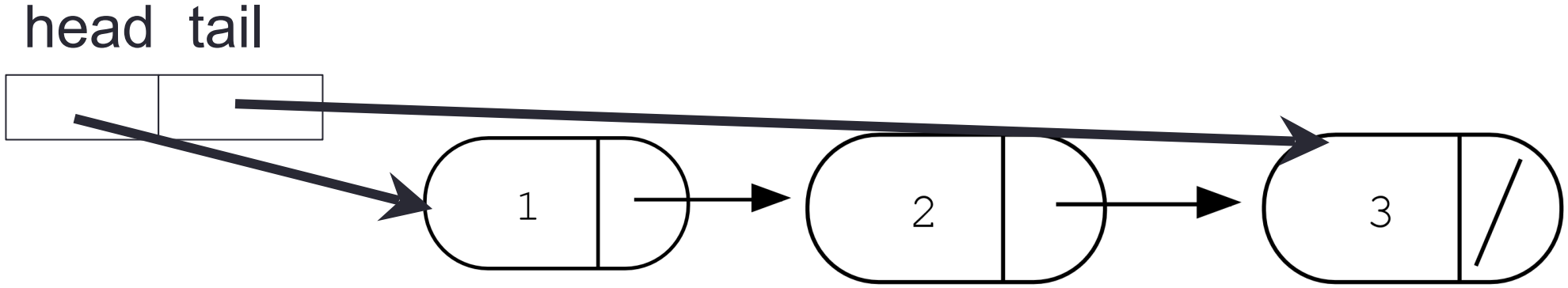
Concept question

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```

Which of the following objects are deleted when the destructor of Linked-list is called?

(A)



(B): All the nodes in the linked-list

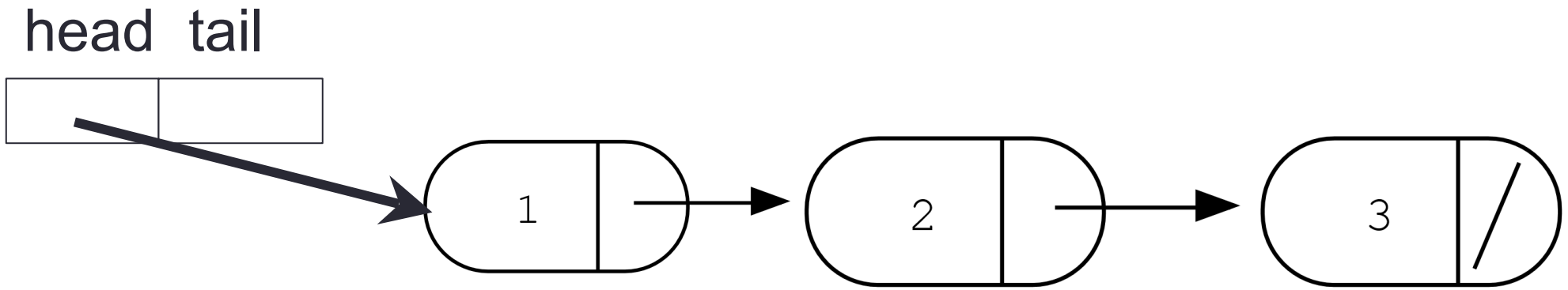
(C): A and B

(D): Program crashes with a segmentation fault

(E): None of the above

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```



Next time

- Binary Search Trees