OPERATOR OVERLOADING LINKED LIST

Problem Solving with Computers-II

Today's goals

- Operator overloading
  - what is operator overloading?
  - why/when would we need to overload operators?
  - how to overload operators in C++ ?
- Linked List
  - Procedural implementation vs OOP style
  - Using recursion to implement linked list operations

Overloading the + operator for Complex objects

```
p = q + w;
```

Goal: We want to apply the + operator to Complex type objects

New method: add()

```
int main(){
  Complex p;
  Complex q(2, 3);
  Complex w(10, -5);
  w.conjugate();
  p = add(q, w);
  p.print();
}
```

Approach 1

```
int main(){
  Complex p;
  Complex q(2, 3);
  Complex w(10, -5);
  w.conjugate();
  p = q.add(w);
  p.print()
}
```

Approach 2

Overloading the + operator for Complex objects

```
p = add(q, w);
```

```
p = q.add(w);
```

```
p = q + w;
```

Goal: We want to apply the + operator to Complex type objects

Overloading the << operator

```
int main(){
  Complex w(10, -5);
  w.conjugate();
  w.print();
}
```

```
int main(){
  Complex w(10, -5);
  w.conjugate();
  cout << w;
}
```

```
Before overloading the << operator
```

```
After overloading the << operator
```

```
cout << w;
```

Select any equivalent C++ statement:

```
w.operator<<(cout);
```

A

```
cout.operator<<(w);
```

B

```
operator<<(cout, w);
```

C

```
operator<<(cout, w);
```

Select the function declaration that does NOT match the above call

A
```
void operator<<(ostream &out,
                const Complex &c);
```

B
```
void Complex::operator<<(ostream &out);
```

C
```
Complex operator<<(ostream &out,
                   Complex c);
```

Operator Overloading

We would like to be able to perform operations on two objects of the class using the following operators:

<<

==

!=

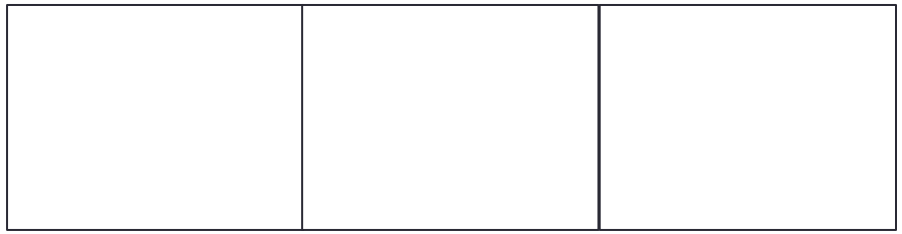
+

-

and possibly others

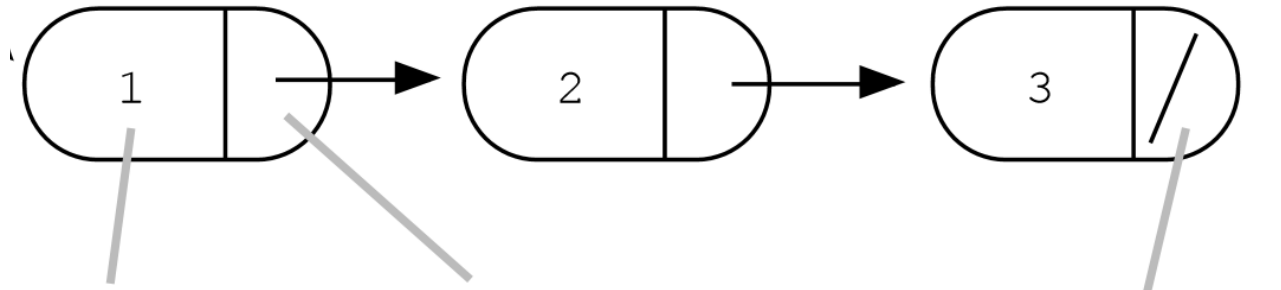
Linked list vs Array

Array



Defining the type Node

The overall list is built by connecting the nodes together by their next pointers. The nodes are all allocated in the heap.



Each node stores one data element (int in this example).

Each node stores one next pointer.

The next field of the last node is NULL.

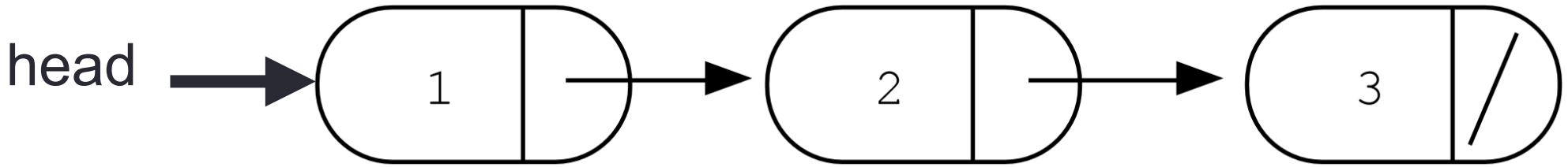
Simplest Linked List (just a head pointer)

- Create an empty list
- Add a node with data 3

```
struct Node {
        int data;
        Node* next;
};
```

Assume the following linked list exists

```
struct Node {
      int data;
      Node *next;
};
```

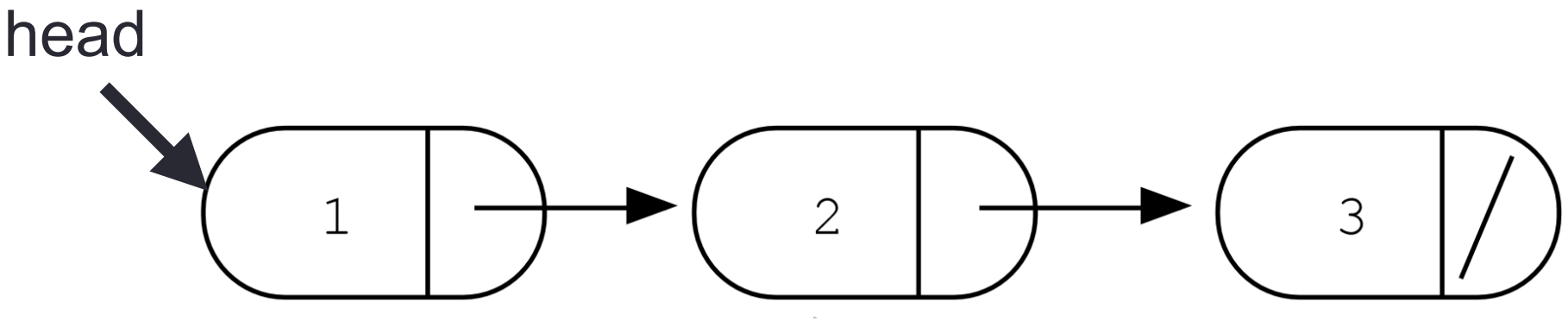


Evaluate each of the following expressions?

1. `head->data`
2. `head->next->data`
3. `head->next->next->data`
4. `head->next->next->next->data`

A. 1
B. 2
C. 3
D. nullptr
E. Run time error

Write a C++ function to add a node to the head of the list (procedural style)




```
struct Node {
      int data;
      Node *next;
};
```

Questions to ask about any ADT:

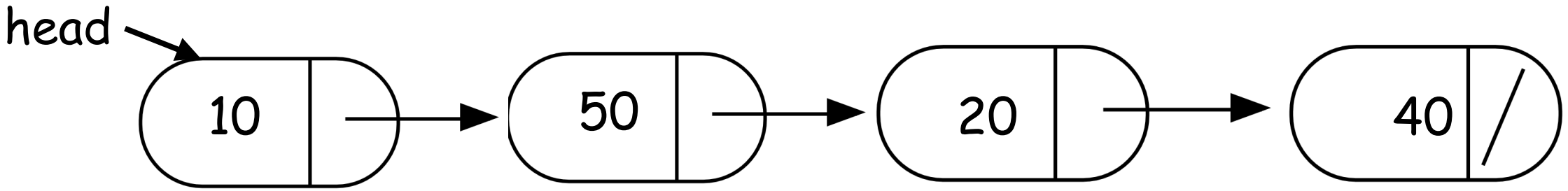
- **What operations does the ADT support?**

  The list ADT supports the following operations on a sequence:

  1. push_front (add a value to the beginning of the sequence)
  2. push_back (add a value to the end of the sequence)
  3. pop_front (delete the first value in the sequence)
  4. pop_back (delete the last value in he sequence)
  5. front() (return the first value)
  6. back() (return the last value)
  7. delete (a value)
  8. print all values
- **How do you implement each operation (data structure used)?**
- **How fast is each operation?**

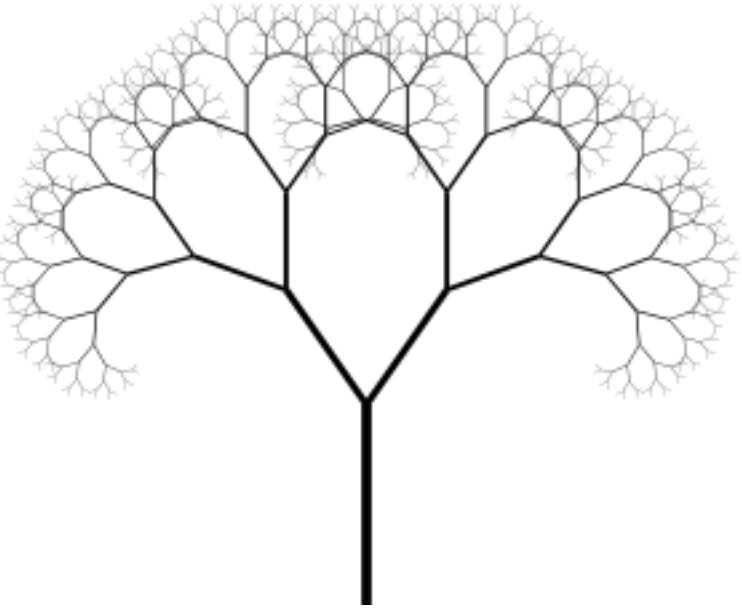
List Abstract Data Type (ADT)

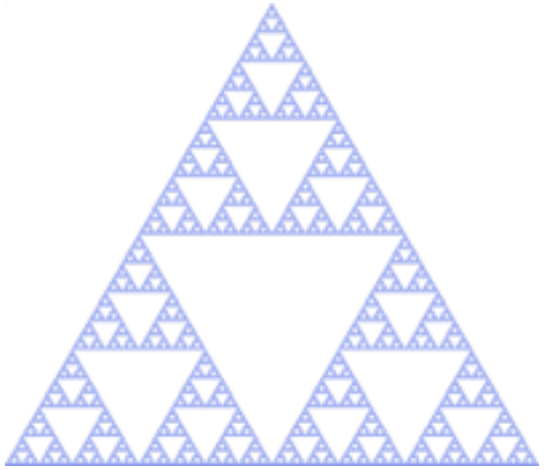
```
class IntList {
public:
    IntList();
    // other public methods


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




```
int IntList::push_front(int value){

     //add value to the beginning of the sequence
  }
```

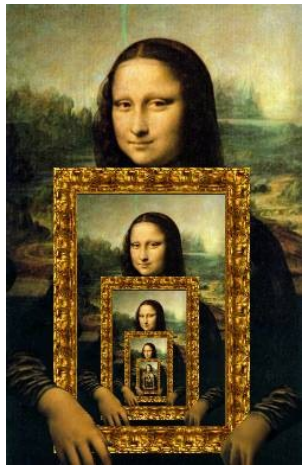
Recursion



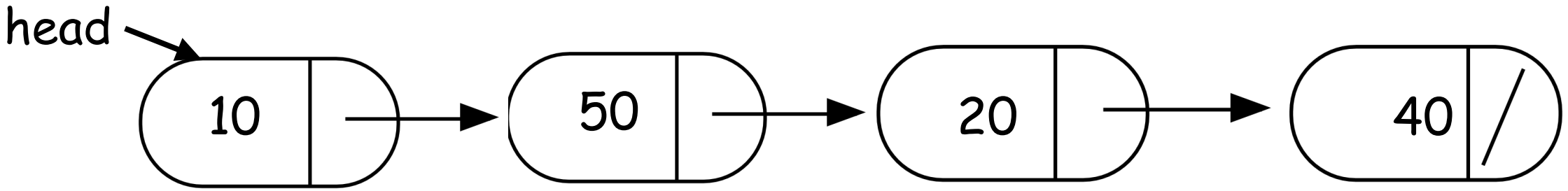


Sierpinski triangle

Zooming into a Koch's snowflake



Using recursion to implement operators involving a linked list



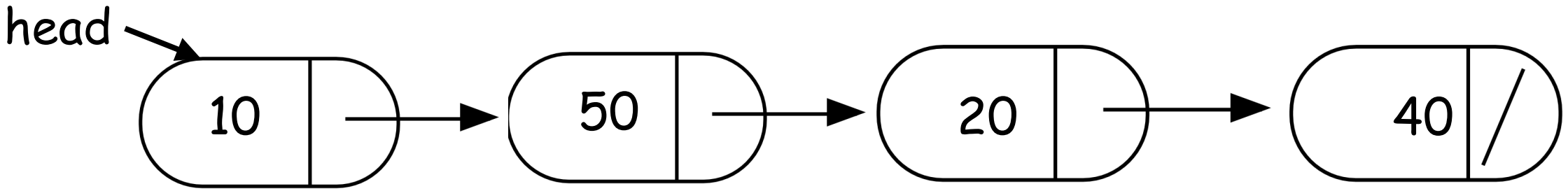

```
int IntList::sum(){

    //return the sum of the sequence
}
```

Helper functions

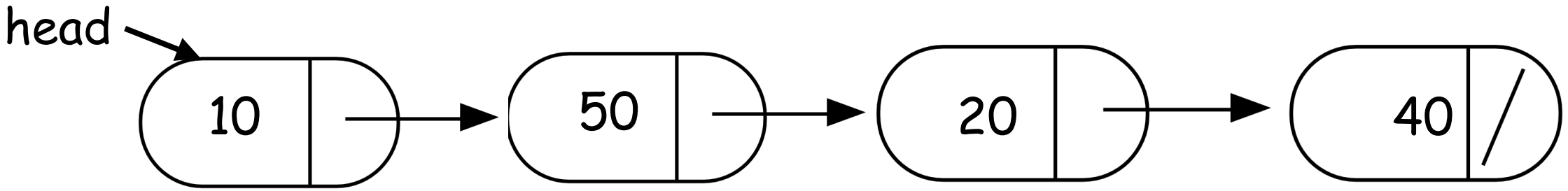
- Sometimes your functions takes an input that is not easy to recurse on
- In that case define a new function with appropriate parameters: This is your helper function
- Call the helper function to perform the recursion
- Usually the helper function is private

For example

```
Int IntList::sum(){

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

}
```




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




}
```




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









  }
```

Overloading Operators for IntList

In lab02 you will overload operators for the IntList ADT

==

!=

+ (list concatenation)

<< (overloaded stream operation to print the sequence)