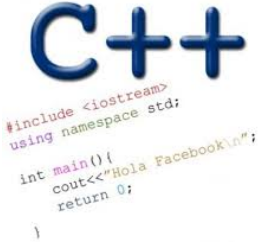
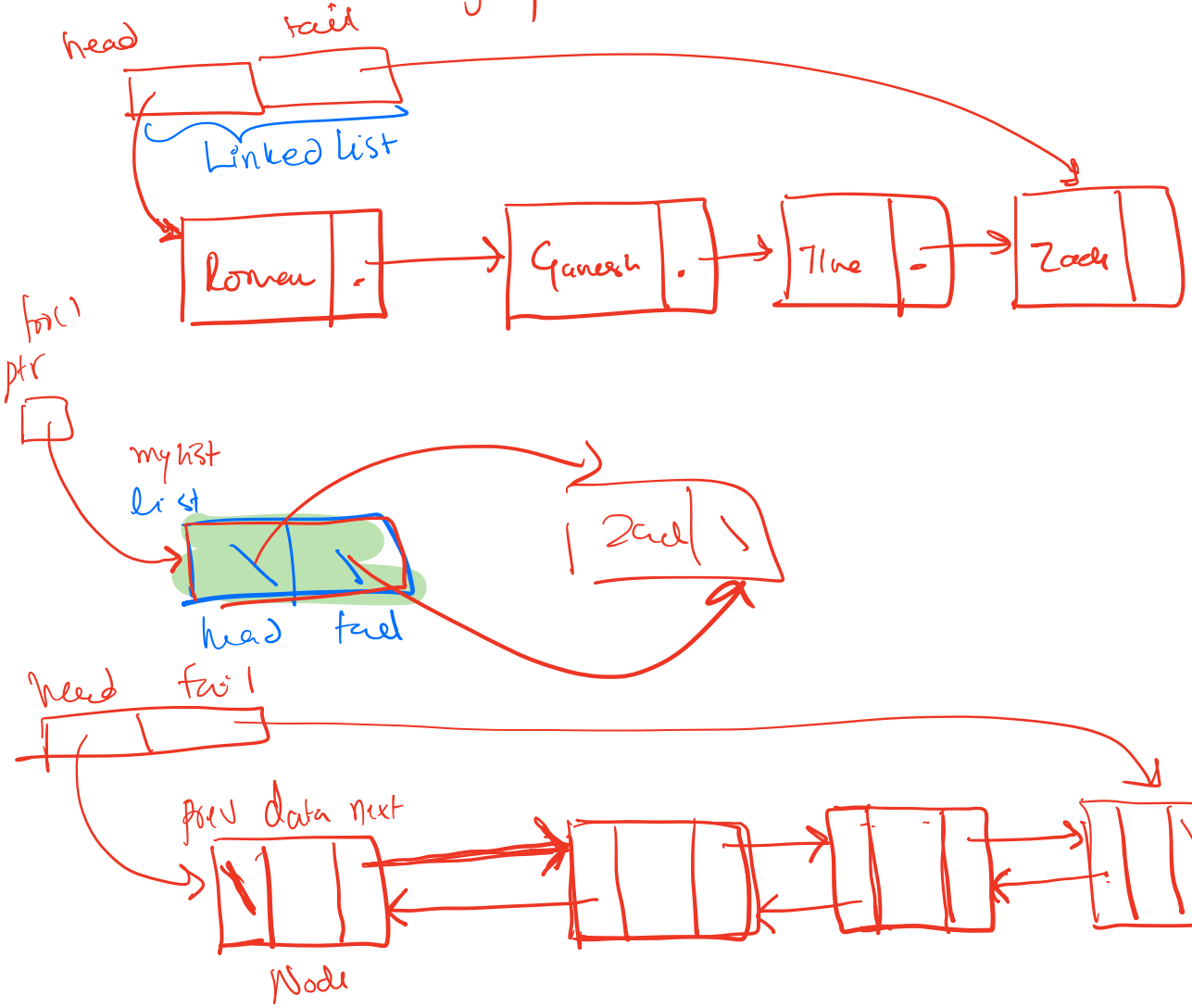
LINKED LISTS (CONTD)
INTRO TO OBJECT ORIENTED PROGRAMMING

Problem Solving with Computers-II



Singly linked list



✓ insertToFront
printList
clearList

```
struct Node {
    Node* prev;
    string data;
    Node* next;
};
```

Doubly-linked list.

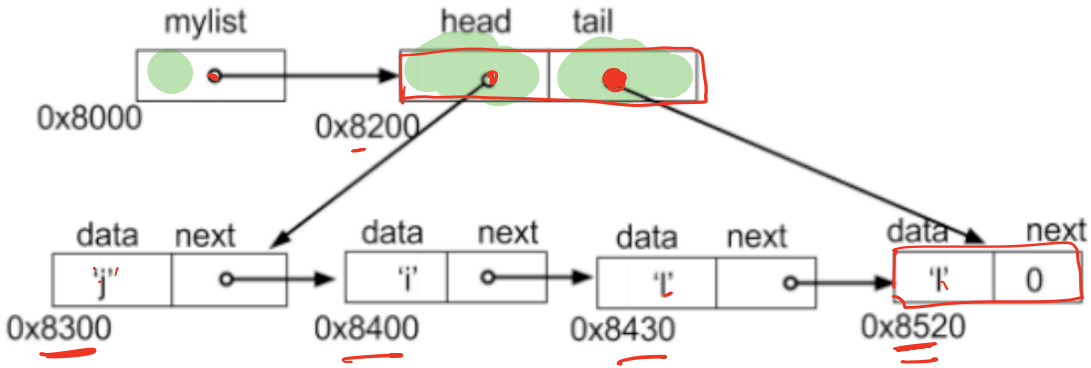
LinkedList datatype

- Define the type LinkedList
- Create an empty list
- Add a node to the list with data “Tina”

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

Accessing nodes in a linked list

(Linked list *) (Linked list)



Cout<< &mylist; 0x8000

a. cout<<mylist;	0x8200
b. cout<<mylist->tail;	0x8520
c. cout<<mylist->tail->data;	'l'
d. cout<<mylist->head->next;	0x8400
e. cout<<mylist->head->next->next	0x8430

A. 0x8000

B. 0x8200

C. 0x8300

D. Something else

Procedural Programming

- Break down a problem into sub tasks (functions)
- Algorithm to bake a cake

    Preheat the oven to 350F

    Get the ingredients: 2 eggs, 1 cup flour, 1 cup milk

    Mix ingredients in a bowl

    Pour the mixture in a pan

    Place in the over for 30 minutes

Object Oriented Programming: A cake baking example

- Solution to a problem is a system of interacting **objects**
- An object has attributes and behavior
- What are the objects in this example?

1. Preheat the oven to 350F
2. Get the ingredients: 2 eggs, 1cup flour, 1 cup milk
3. Mix ingredients in a bowl
4. Pour the mixture in a pan
5. Place in the over for 30 minutes

Objects have attributes and behavior: A cake baking example

Object	Attributes	Behaviors
Oven	Size Temperature Number of racks	Turn on Turn off Set temperature
Bowl	Capacity Current amount	Pour into Pout out
Egg	Size	Crack Separate(white from yolk)

A class: pattern for describing similar objects

A generic pattern that is used to describe objects that have similar attributes and behaviors

e.g. a bowl and a pan may be described by the same class

```
class Dish{
     void pourIn( double amount);
     void pourOut(double amount);
     double capacity;
     double currentAmount;
};
```

Objects vs classes

```
class Dish{
     void pourIn( double amount);
     void pourOut(double amount);
     double capacity;
     double currentAmount;
};
 //Creating objects of this class
```

Concept: Classes describe objects

- Every object belongs to (is an instance of) a class
- An object may have fields, or variables
  - The class describes those fields
- An object may have methods
  - The class describes those methods
- A class is like a template, or cookie cutter

Concept: Classes are like Abstract Data Types

- An Abstract Data Type (ADT) bundles together:
  - some data, representing an object or "thing"
  - the operations on that data
- The operations defined by the ADT are the *only* operations permitted on its data
- ADT = classes + information hiding

```
class Dish{
public:
     void pourIn( double amount);
     void pourOut(double amount);
private:
     double capacity;
     double currentAmount;
};
```

Approximate Terminology

- instance = object
- field = instance variable
- method = function
- sending a message to an object = calling a function

Some advice on designing classes

- Always, *always* strive for a narrow interface
- Follow the principle of information hiding:
  - the caller should know as little as possible about how the method does its job
  - the method should know little or nothing about where or why it is being called
- Make as much as possible private
- Your class is responsible for it's own data; don't allow other classes to easily modify it!

What we have spoken about so far?

- Class = Data + Member Functions.
- Abstract Data Type = Class + information hiding
- How to activate member functions.
- But you still need to learn how to write the bodies of a class's methods.

Next time

- Object Oriented Programming (contd): Big Four