COSC 121: Computer Programming II

Dr. Bowen Hui

University of British Columbia Okanagan

Admin

- Lab 5:
  - Week 6: Start this week
  - Week 7 (next week): reading week
  - Week 8: Due this week at beginning of your lab
- A2:
  - Due this Saturday morning
  - Will return before Midterm
- Linked list:
  - Tested on midterm
  - Practice using class and textbook exercises

Quick Review

- Linked lists basic operations
  - add, delete, insert
  - traverse, toString
  - Keeping track of current and previous nodes
- Manipulations in:
  - Visual representation
  - Java code

Exercise: insert( )

```
// inserts a new node with given name and ID
// assumes list is sorted by IDs from largest to smallest
// adds node in the sorted order based on node IDs
public void insert( String name, int id ) { … }
```

Write the pseudocode first, then fill in code details. Test your code with these cases:

- Inserting a new node to an empty list
- Inserting a new node to the beginning of the list
- Inserting a new node to the end of the list
- Inserting a new node to the middle of the list

Sample Solution (cont.)

```
// inserts a new node with given name and ID
// assumes list is sorted by IDs from largest to smallest
// adds node in the sorted order based on node IDs
public void insert( String name, int id )
{
  // make new elem with input name and id
  // case 1. if student_list is null
  // otherwise, student_list has at least one node
  // case 2. traverse list to find insertion spot
  //    a. elem id is bigger than id of first node
  //    b. elem id is bigger than id of some node in list
  //    c. elem id is smaller than all nodes in list
}
```

Sample Solution (cont.)

```
// inserts a new node with given name and ID
// assumes list is sorted by IDs from largest to smallest
// adds node in the sorted order based on node IDs
public void insert( String name, int id )
{
  // make new elem with input name and id
  SNode elem = new SNode( name, id );

  // case 1. if student_list is null

  // case 2. traverse list to find insertion spot
}
```

Sample Solution (cont.)

```
// inserts a new node with given name and ID
// assumes list is sorted by IDs from largest to smallest
// adds node in the sorted order based on node IDs
public void insert( String name, int id )
{
  // make new elem with input name and id

  // case 1. if student_list is null
  //    set student_list to elem
  if( student_list == null )
  {
     student_list = elem;
  }
  else
  {
    // case 2. traverse list to find insertion spot
  }
}
```

Sample Solution (cont.)

```
// inserts a new node with given name and ID
// assumes list is sorted by IDs from largest to smallest
// adds node in the sorted order based on node IDs
public void insert( String name, int id )
{
  // make new elem with input name and id
  // case 1. if student_list is null
  // case 2. traverse list to find insertion spot
  //    a. elem id is bigger than id of first node
  //    b. elem id is bigger than id of some node in list
  //    c. elem id is smaller than all nodes in list
}
```

Sample Solution (cont.)

```
public void insert( String name, int id )
{
  // make new elem with input name and id
  // case 1. if student_list is null
  // case 2. traverse list to find insertion spot
  //    a. elem id is bigger than id of first node
  SNode curr = student_list;
  if( id > curr.getId() )
  {
    student_list = elem;
    elem.next = curr;
  }
  //    b. elem id is bigger than id of some node in list
  //    c. elem id is smaller than all nodes in list
}
```

Sample Solution (cont.)

```
public void insert( String name, int id )
{
  // case 2. traverse list to find insertion spot
  //    a. elem id is bigger than id of first node
  SNode curr = student_list;
  if( id > curr.getId() )
  {
    student_list = elem;
    elem.next = curr;
  }
  //    b. elem id is bigger than id of some node in list
  SNode prev = null;
  while( curr.next != null )
  {
    prev = curr;
    curr = curr.next;
    if( id > curr.getId() )
    {
      prev.next = elem;
      elem.next = curr;
      break;
    }
  }
  //    c. elem id is smaller than all nodes in list
}
```

Sample Solution (cont.)

```
public void insert( String name, int id )
{
  // case 2. traverse list to find insertion spot
  //    a. elem id is bigger than id of first node
  SNode curr = student_list;
  if( id > curr.getId() )
  {
    student_list = elem;
    elem.next = curr;
  }
  //    b. elem id is bigger than id of some node in list
  SNode prev = null;
  while( curr.next != null )
  {
    …
  }
  //    c. elem id is smaller than all nodes in list
  if( curr.getId() > id )
    curr.next = elem;
}
```

Practical Example

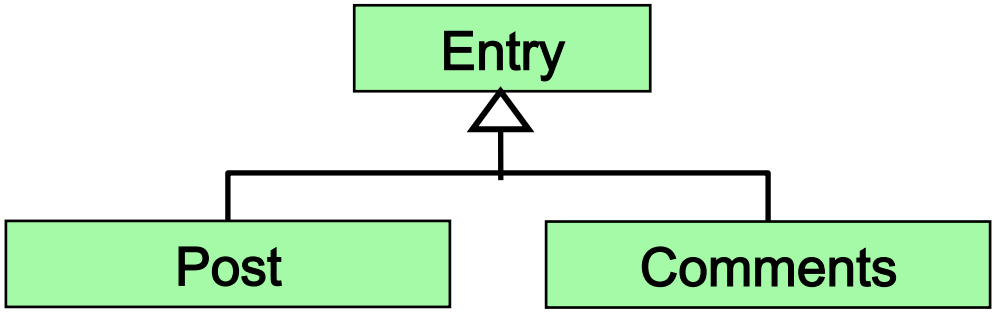
- Think about a blog
- What is it consist of?
  - Posts (by the same author)
  - Comments (by various readers)
- How many posts are there in a blog?
- How many comments are there in a post?
- What data structure should be used to model a blog?

A Blog as a Linked List

- A blog is a list of `Post` nodes
- Each `Post` node has some content including:
  - Author's name
  - Text to be posted
  - A list of zero or more `Comment` nodes
- How to visually represent a blog?

Modeling Nodes

- What do these have in common?
  - Posts (by the same author)
  - Comments (by various readers)
- Classes to model:



Entry class

```
public abstract class Entry
{
   protected String author;
   protected String text;
   public abstract String getAuthor();
   public abstract String getText();
   public abstract void String setAuthor( String name);
   public abstract void setText( String content );
   public String toString() { return text + "By: " + author; }
}
```

- How would you define `Comment` and `Post` classes?
  - Comments can be anonymous
  - Posts must have authors and associated comments

`Comment` class

- Create constructor
  - Initialize attributes
- Overload constructor with:
  `author = "anonymous";`
- Define abstract methods

`Post` class

- Declare list of `Comment` nodes
- Create constructor
  - Initialize attributes
- Define abstract methods
- Define methods to operate on comments:
  - Add a `Comment` node
  - For simplicity: don't worry about delete/insert
- Override `toString()` to include printing list of comments

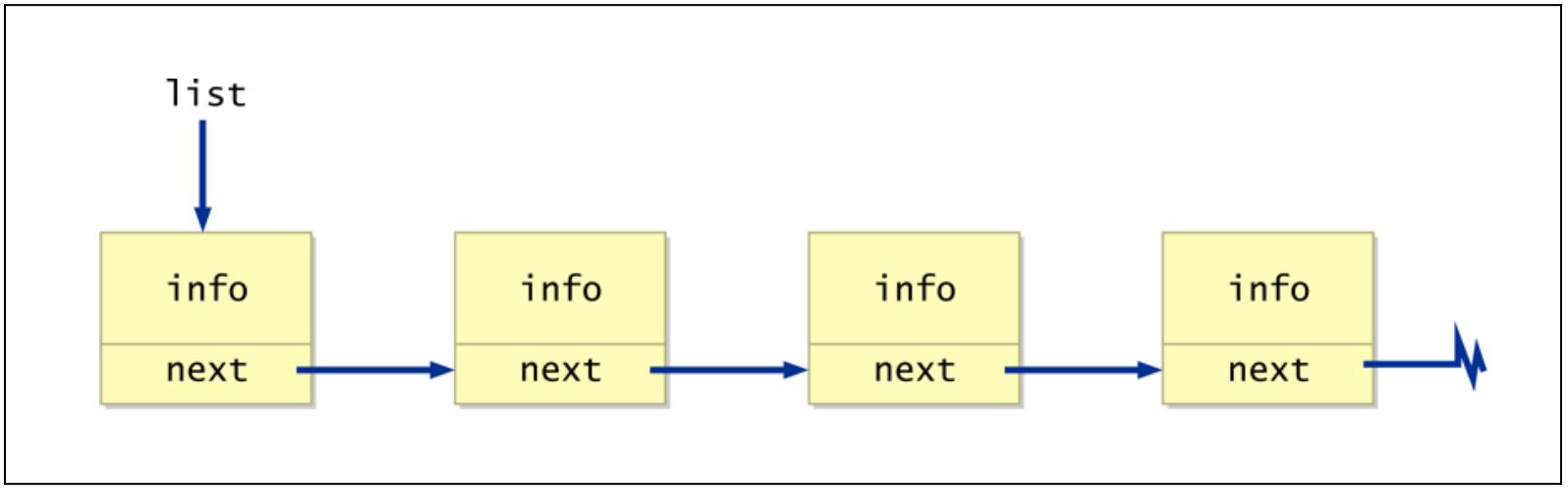
Where to Create and Manage a Blog?

Where to Create and Manage a Blog?

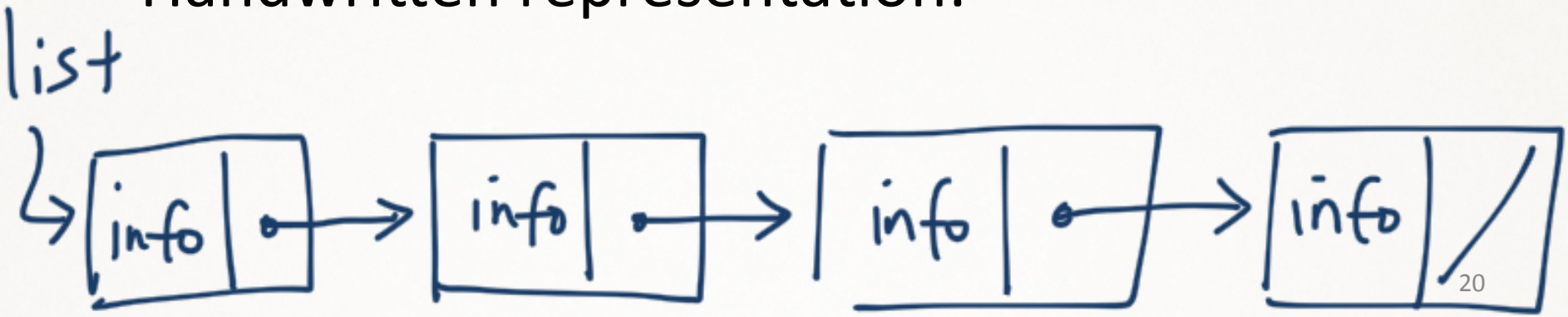
- A `Blog` class
- Define methods to operate on the blog:
  - Create a new Blog
  - Add a `Post` to it
  - Delete a `Post` (and its `Comment` nodes) from it
  - Print a `Blog`
- Where do we create a new `Blog`?
- Note: Real-world blogs have users and permissions

So far: Singly Linked List

- Textbook representation:

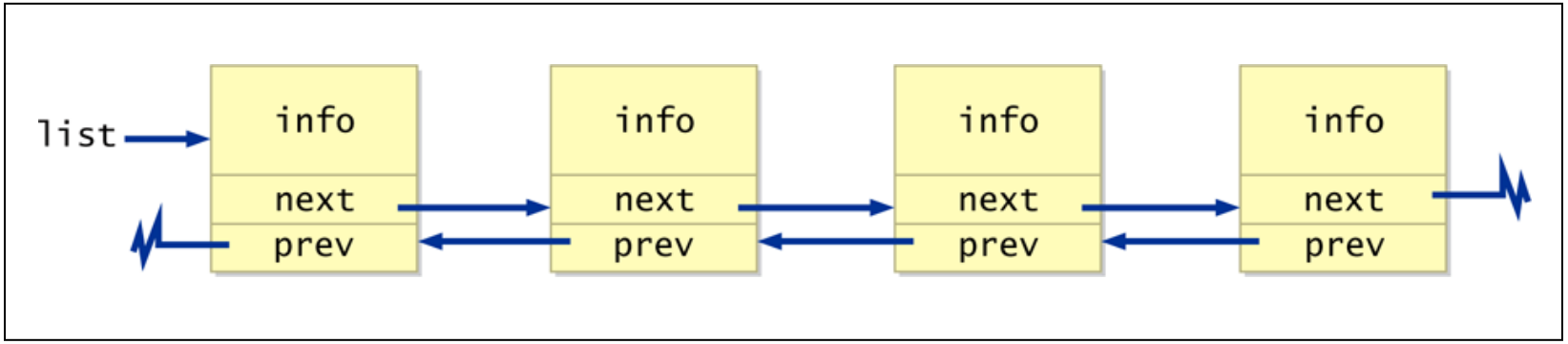


- Handwritten representation:



Other Dynamic Representations

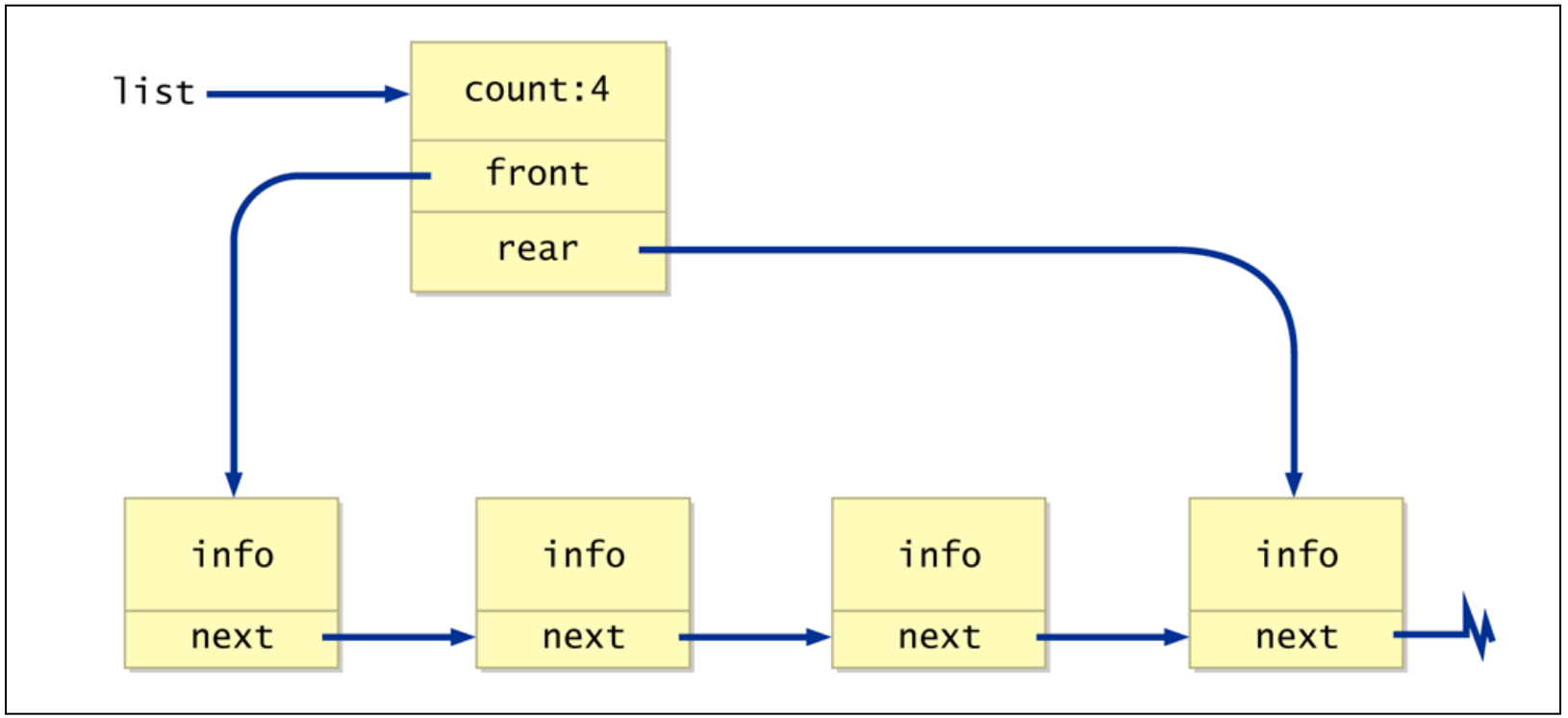
- Bidirectional linked list is called a doubly linked list
- With next and previous references:



- When might this be more useful than a singly linked list?

Other Dynamic Representations

- Another approach is to use a separate header node, with a count and references to both the front and rear of the list:



Summary of Linked Lists

- Basic operations:
  - add, delete, insert
  - toString
  - traverse
- Singly vs. doubly linked lists
- Use of test cases:
  - Consideration of multiple scenarios to ensure method details are implemented properly

Admin

- A2 due this Saturday
- Reading week next week: no classes, no labs
- Week after reading week:
  - Review class – bring questions
  - Midterm: 20% of grade
- Missed midterm without valid medical note will receive 0