CS112: Data Structures

Instructor: Prof. Louis Steinberg

- **office: Hill 401**
- **email: lou@cs.rutgers.edu**
- **phone: 848-445-7289**

CS112: Data Structures

- **Today's topics:**
  - **Linked lists**

Class Web Site

http://sakai.rutgers.edu

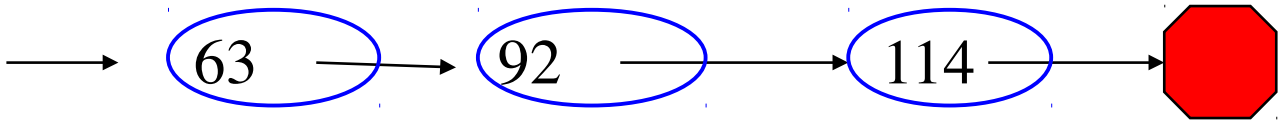
- **Syllabus**
- **Textbook**
- **Prerequisites**
- **Exam Schedule**
- **Grading**
- **etc.**

Our first data structure

- **Linked Lists**
  - **Like an array, a linked list stores an ordered list of data items, all of the same type**
  - **But has different time and space costs**

Linked Lists

- **Sometimes you care about what comes after what, but not about what is 1st, …, 32nd, ….**
- **Can be more efficient to store "what comes next"**



- **Each entry in the list is contained in a node**

Nodes

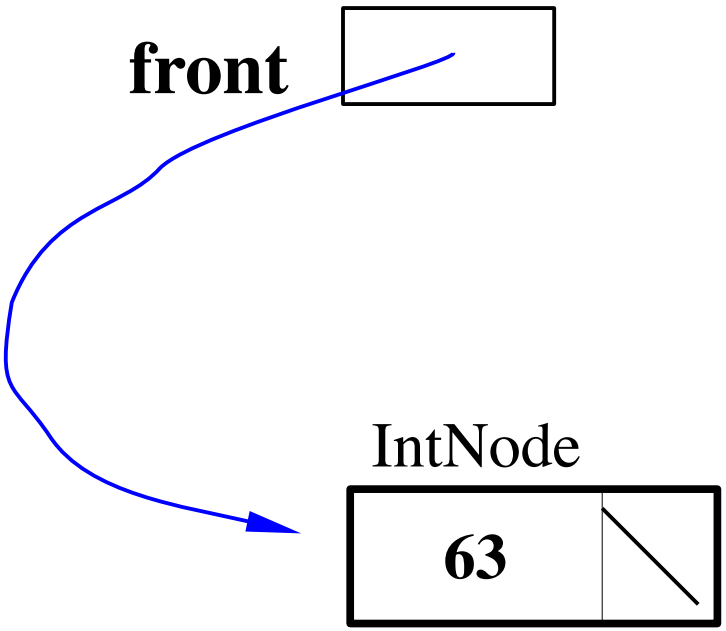
- **A node is an object that has**
  - **a field for data**
  - **a field to refer to the next node in the linked list**

```
public class IntNode{
   int data;
   IntNode next;
   public IntNode(int data, IntNode next) {
      this.data = data;
      this.next = next;
   }
}
```

a *reference to* an IntNode object

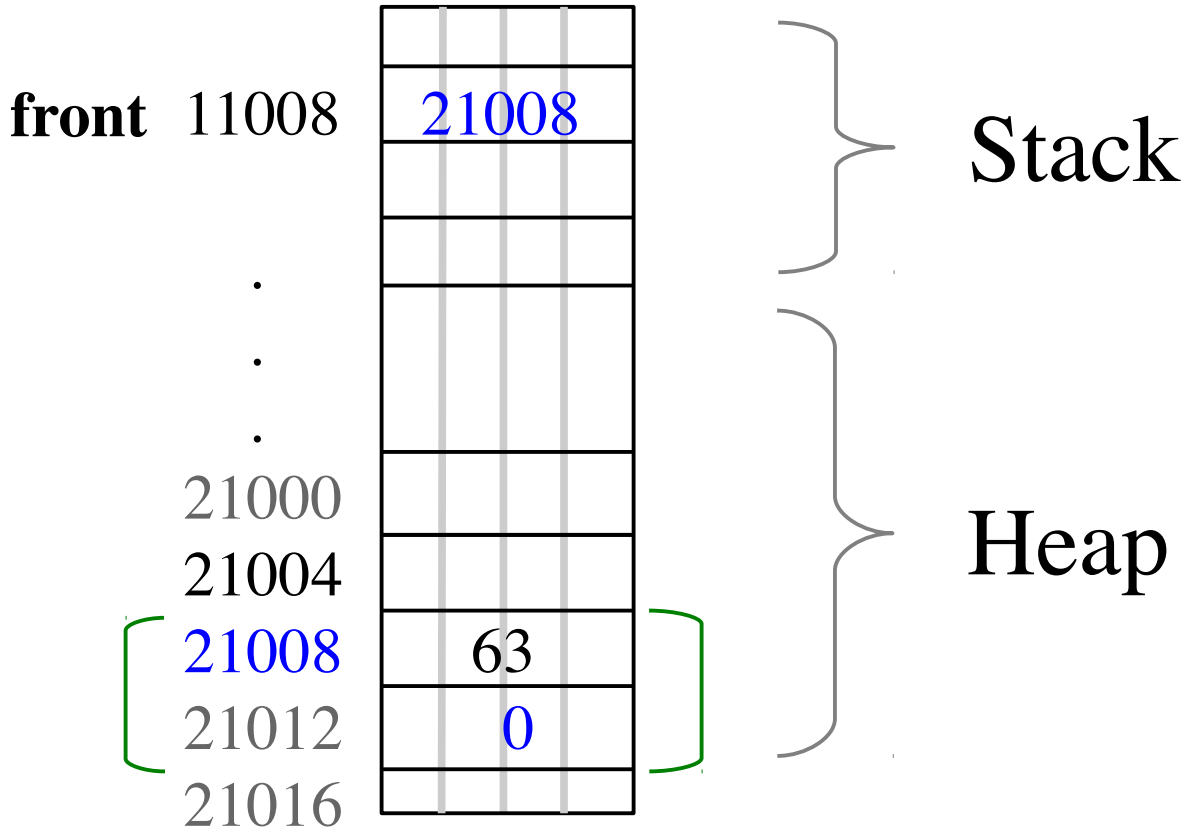
Creating a One Element List

IntNode front = new IntNode(63, null);



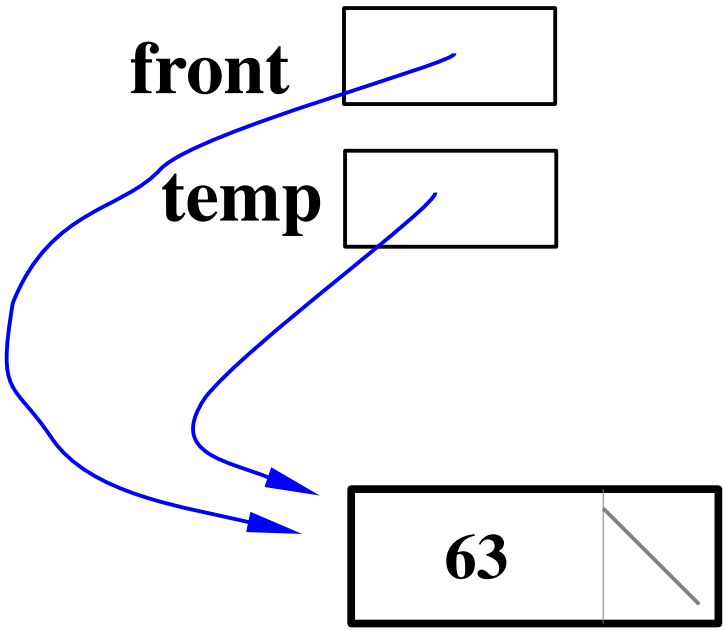
Creating a One Element List

IntNode front = new IntNode(63, null);



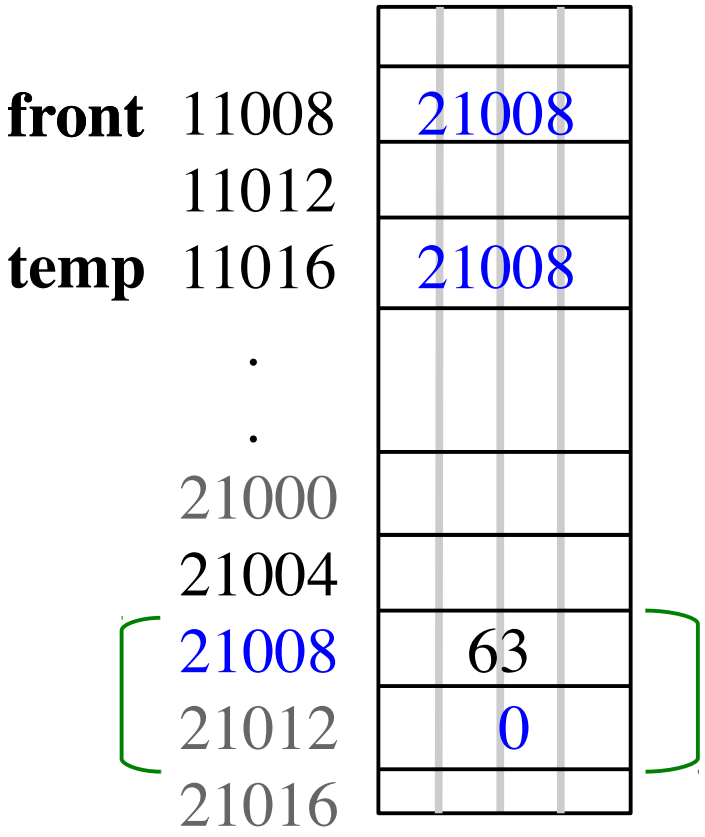
Assignment Copies the Reference

IntNode temp = front;

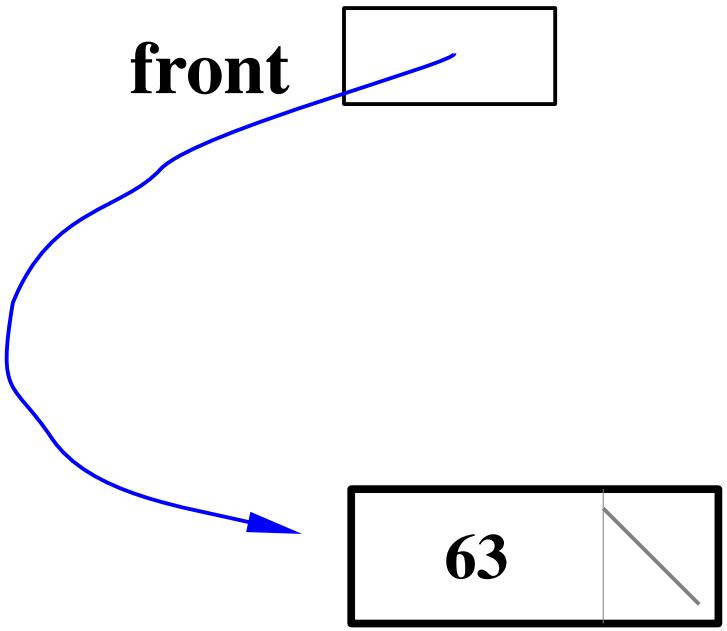


Assignment Copies a Reference

IntNode temp = front;

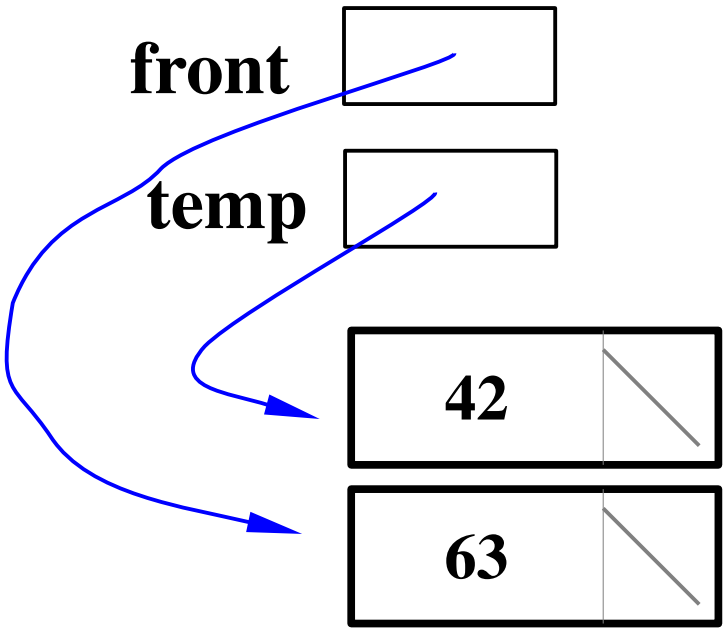


Adding a Second Element



Adding a Second Element

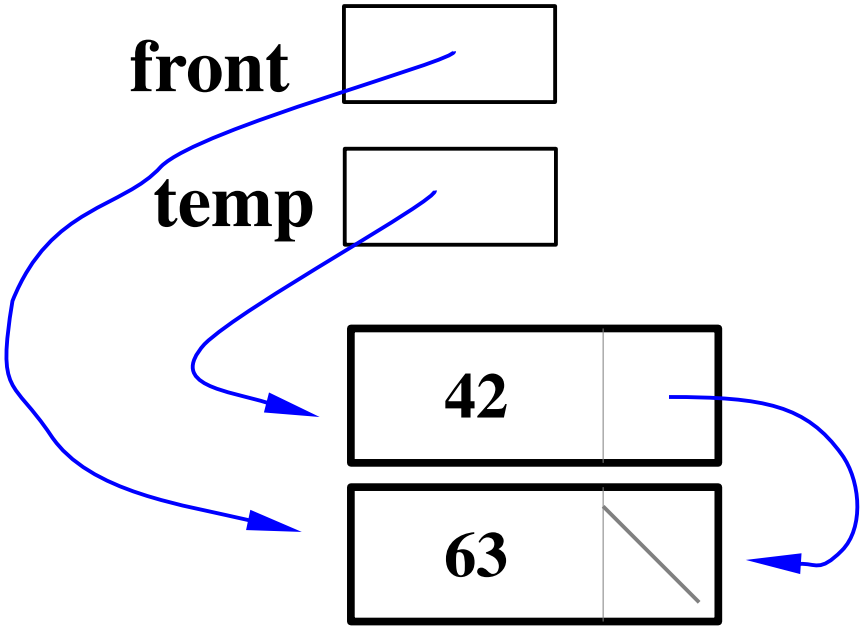
IntNode temp = new IntNode(42, null);



Adding a Second Element

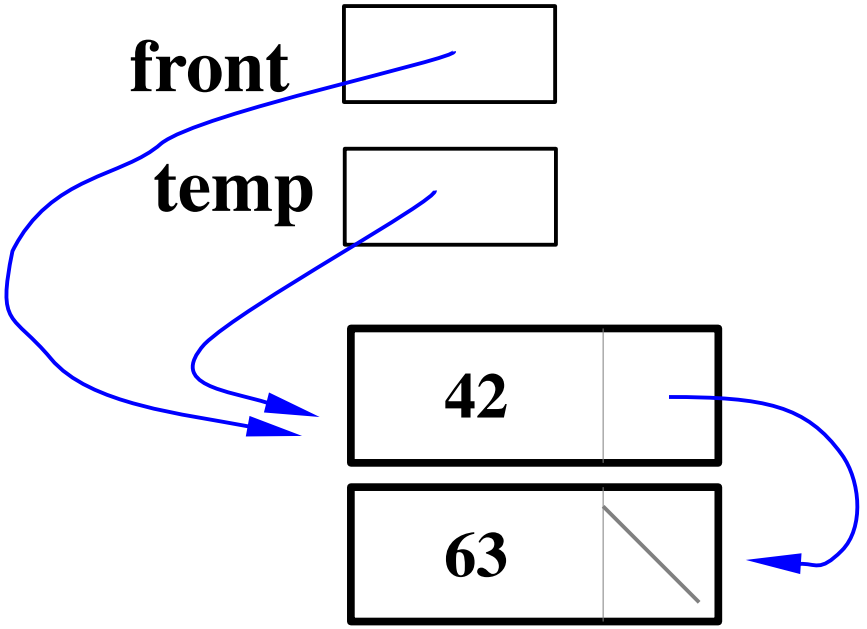
IntNode temp = new IntNode(42, null);

temp.next = front;

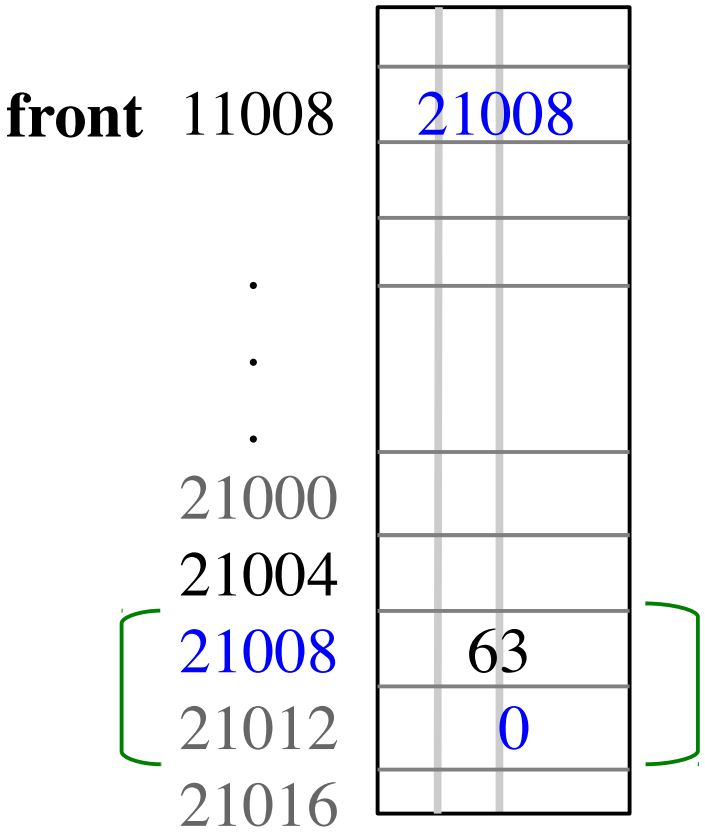


Adding a Second Element

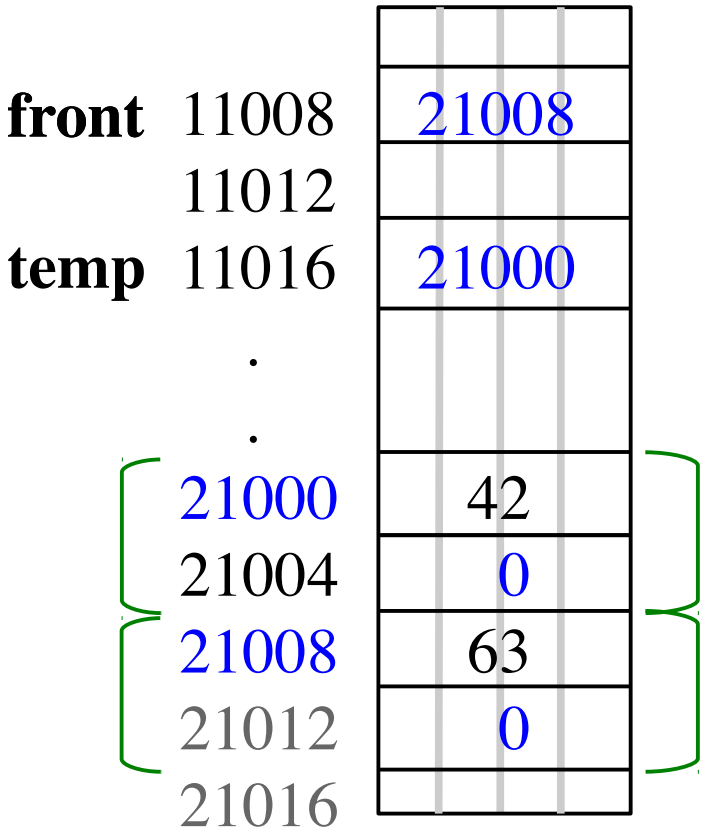
```
IntNode temp = new IntNode(42, null);
temp.next = front;
front = temp;
```



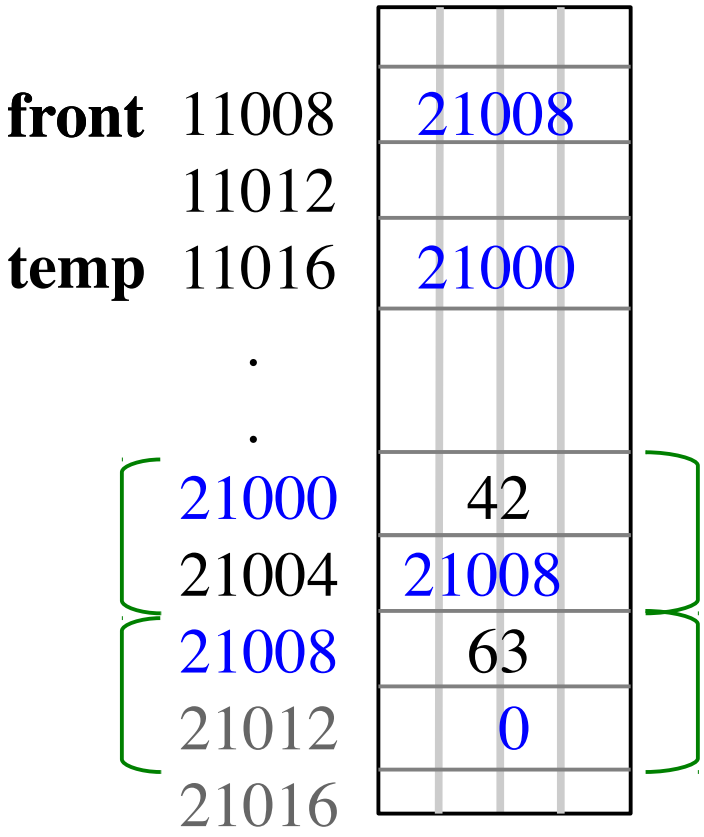
Adding a Second Element



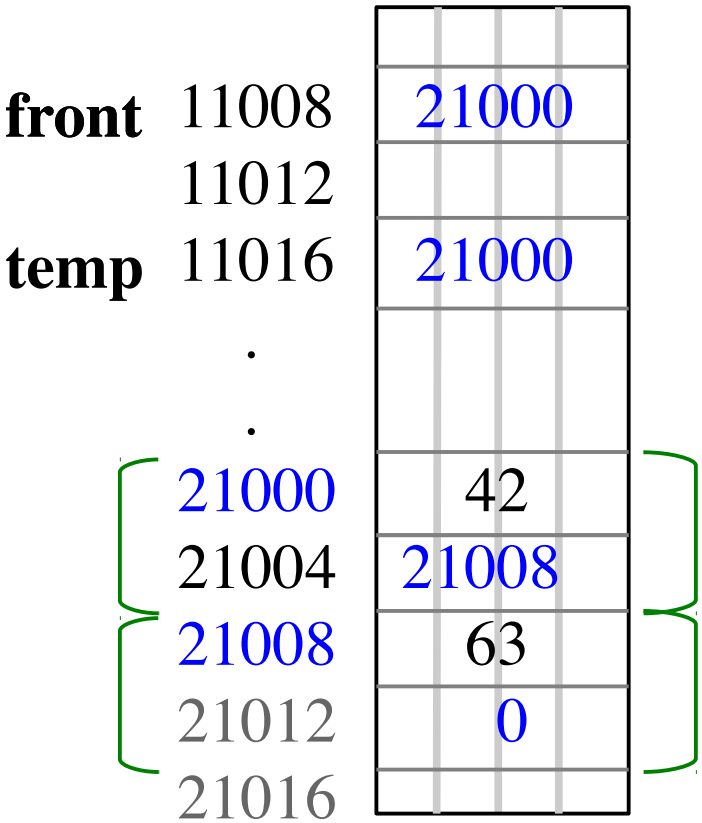
temp = new IntNode(42, null);



temp . next = front;



front = temp;


front
temp
11008
11012
11016
21000
21004
21008
21012
21016
21000
21000
42
21008
63
0

Adding an element in one line

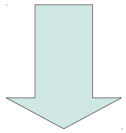
```
IntNode temp = new IntNode(20,  null);
temp.next = front;
front = temp;
```

can be replaced by

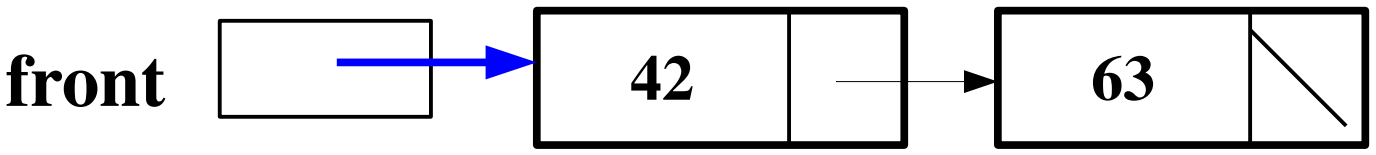
```
front = new IntNode(20, front);
```

Accessing a Two-Element List

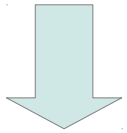
Accessing a Two-Element List



front . next . data

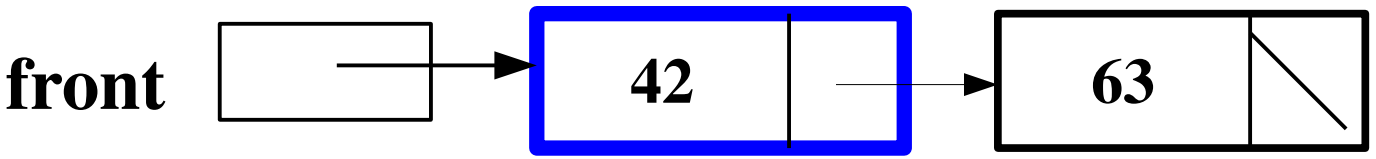


Accessing a Two-Element List

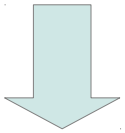


front . next . data

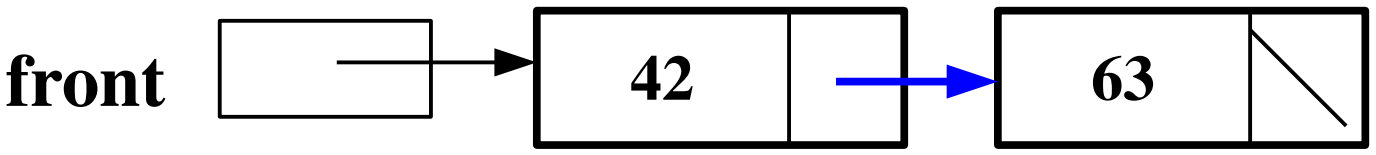
dot (.) means "dereference": follow the reference to object it points to



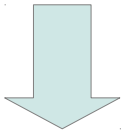
Accessing a Two-Element List



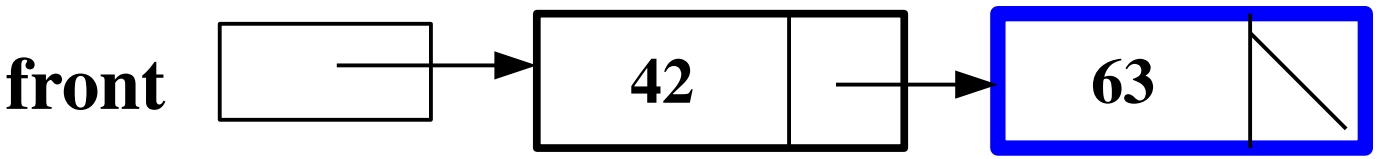
front . next . data



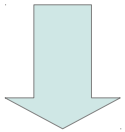
Accessing a Two-Element List



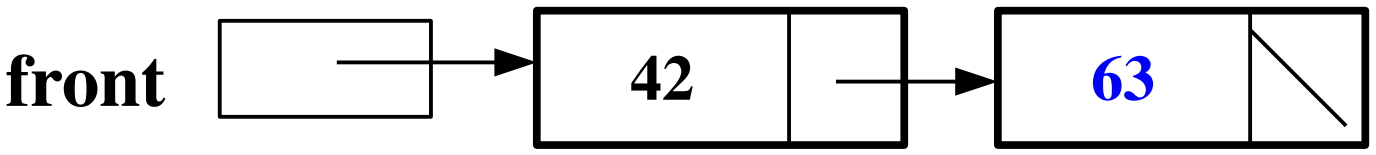
front . next . data



Accessing a Two-Element List



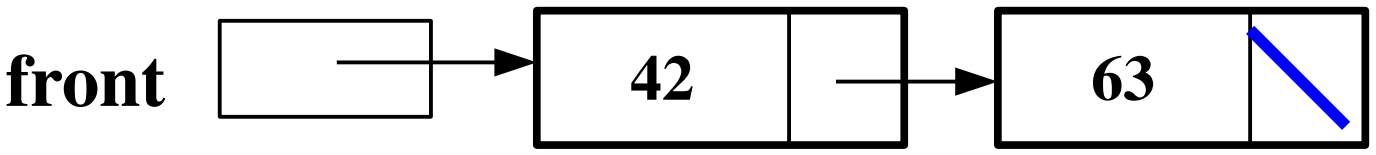
front . next . data



Accessing a Two-Element List

front . next . next

Value is null

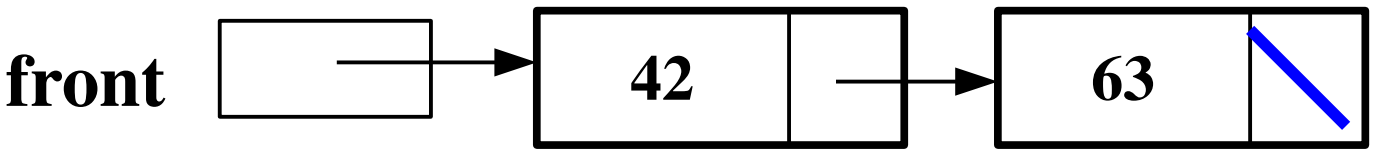


Accessing a Two-Element List

front . next . next . data

attempt to dereference null:

NullPointerException



Traverse a List

- **Array:**

```
for (int j = 0;
      j < a. length;
      j++){
   System.out.print(a[j]);
}
```

- **Linked List**

```
for (IntNode ptr = front;
      ptr != null;
      ptr = ptr . next){
   System.out.print(ptr.data);
}
```

printlist as a Method

```
public class IntNode{
   …
   public static printList(IntNode front){
      for (IntNode ptr = front;
            ptr != null;
            ptr = ptr . next){
         System.out.println(ptr . data);
      }
…
```

addAtFront as a Method

```
public static void addAtFront(int data, IntNode front){
// DOES NOT WORK
    front = new IntNode(data, front);
}
public static void main(String [ ] args){
   IntNode front = null;
   addAtFront(6, front);
   printList(front);   // prints nothing
```

addAtFront as a Method

```
// WORKS
public static IntNode addAtFront(int data, IntNode front){
    front = new IntNode(data, front);
    return front;
}
public static void main(String [ ] args){
   IntNode front = null;
   addAtFront(6, front);
   printList(front);   // prints 6
```

Reference Parameters

- **See ParamTest.java**

More Methods

- **IntNode deleteFront(IntNode front)**
- **boolean search(IntNode front, int target)**
- **boolean addAfter(IntNode front,**
  int target,
  int item) // false if target
  // not in list
- **IntNode delete (IntNode front, int target)**

Reading

- **Read DSOI**
  - **Sections 1.3, 1.4**
  - **Section 4.5, 4.6, 4.7**