CS 106X, Lecture 16
More Linked Lists

reading:

Programming Abstractions in C++, Chapters 11-12, 14.1-14.2

Plan For Today

- Implementing a Linked List
  - Pointers
  - Dynamic memory
  - Classes
  - Testing
- Announcements
- Template Classes
- Doubly-Linked Lists

Learning Goals

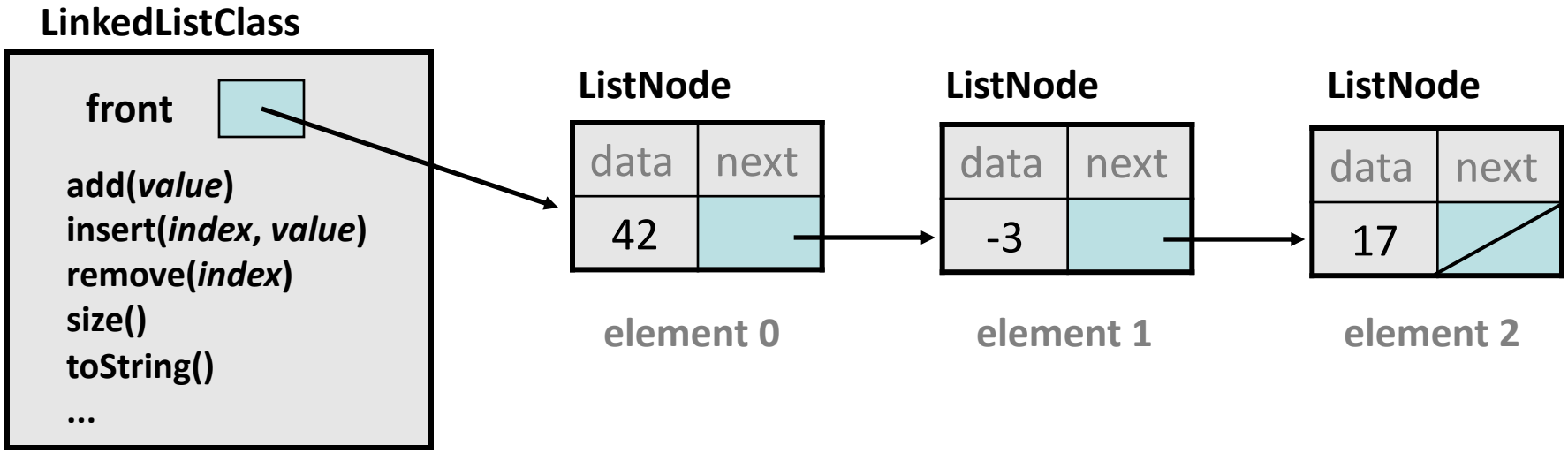
- Understand the implementation of the LinkedList ADT
- Understand how to make generic classes using templates
- Understand the benefits and drawbacks of doubly-linked lists

Plan For Today

- Implementing a Linked List
  - Pointers
  - Dynamic memory
  - Classes
  - Testing
- Announcements
- Template Classes
- Doubly-Linked Lists

LinkedListClass

- Let's write a collection class named **`LinkedListClass`**.
  - Has the similar public members to `Vector`
    - add, `clear`, `get`, `insert`, `remove`, `size`, `toString`
  - The list is internally implemented as a chain of linked nodes
    - The `LinkedListClass` keeps a pointer to its `front` node as a field
    - `nullptr` is the end of the list; a null front signifies an empty list



Destructor (12.3)

```
// ClassName.h
~ClassName();
```

```
// ClassName.cpp
ClassName::~ClassName() { ...
```

- **destructor**: Called when the object is deleted by the program.

  (when the object goes out of { } scope; opposite of a constructor)

  – Useful if your object needs to do anything important as it dies:
    - saving any temporary resources inside the object
    - freeing any dynamically allocated memory used by the object's members
    - ...

Plan For Today

- Implementing a Linked List
  - Pointers
  - Dynamic memory
  - Classes
  - Testing
- Announcements
- Template Classes
- Doubly-Linked Lists

Announcements

- Midterm **Thursday 11/1 7-9PM in 420-040**
- Midterm Review session **tomorrow Tues. 10/30 5-6:30PM** in Hewlett 102
- Assignment 5, MiniBrowser, out later today, due **Wed. 11/7 @ 11AM**

Plan For Today

- Implementing a Linked List
  - Pointers
  - Dynamic memory
  - Classes
  - Testing
- Announcements
- Template Classes
- Doubly-Linked Lists

Template function (14.1-2)

```
template<typename T>
returntype name(parameters) {
    statements;
}
```

- **Template**: A function or class that accepts a *type parameter(s)*.
  - Allows you to avoid redundancy by writing a function that can accept many types of data.
  - Templates can appear on a single function, or on an entire class

Template func example

```
template<typename T>
T max(T a, T b) {
    if (a < b) { return b; }
    else       { return a; }
}
```

– The template is *instantiated* each time you use it with a new type.
  - The compiler actually generates a new version of the code each time.
  - The type you use must have an operator < to work in the above code.

```
int i    = max(17, 4);           // T = int
double d = max(3.1, 4.6);        // T = double
string s = max(string("hi"),     // T = string
               string("bye"));
```

Template class (14.1-2)

- **Template class**: A class that accepts a type parameter(s).
  - In the header and cpp files, mark each class/function as templated.
  - Replace occurrences of the previous type `int` with T in the code.

```
// ClassName.h
template<typename T>
class ClassName {
    ...
};

// ClassName.cpp
template<typename T>
type ClassName::name(parameters) {
    ...
}
```

Template .h and .cpp

- Because of an odd quirk with C++ templates, the separation between .h header and .cpp implementation must be reduced.
  - Either write all the bodies in the .h file (suggested),
  - Or `#include` the .cpp at the end of .h file to join them together.

```
// ClassName.h
#ifndef _classname_h
#define _classname_h

template<typename T>
class ClassName {
    ...
};

#include "ClassName.cpp"
#endif   // _classname_h
```

Exercise

- Convert the **`LinkedListClass`** to use templates.
  - A client should be able to create a LinkedListClass of any type.

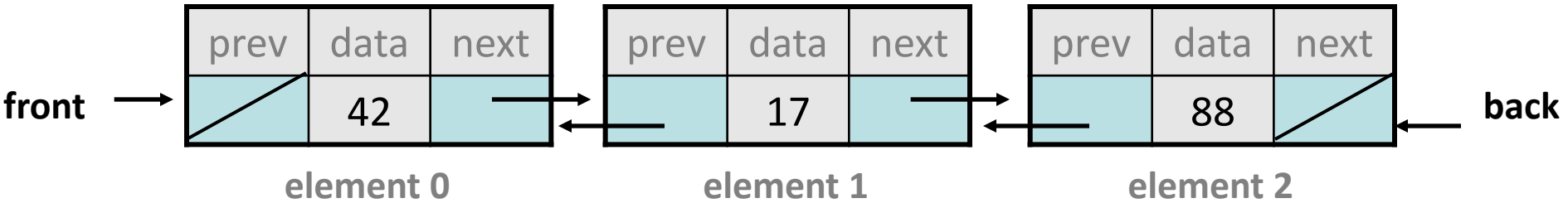
```
LinkedListClass<int> s1;
s1.add(42);
s1.add(17);

LinkedListClass<string> s2;
s2.add("hello");
s2.add("there");
...
```

Plan For Today

- Implementing a Linked List
  - Pointers
  - Dynamic memory
  - Classes
  - Testing
- Announcements
- Template Classes
- Doubly-Linked Lists

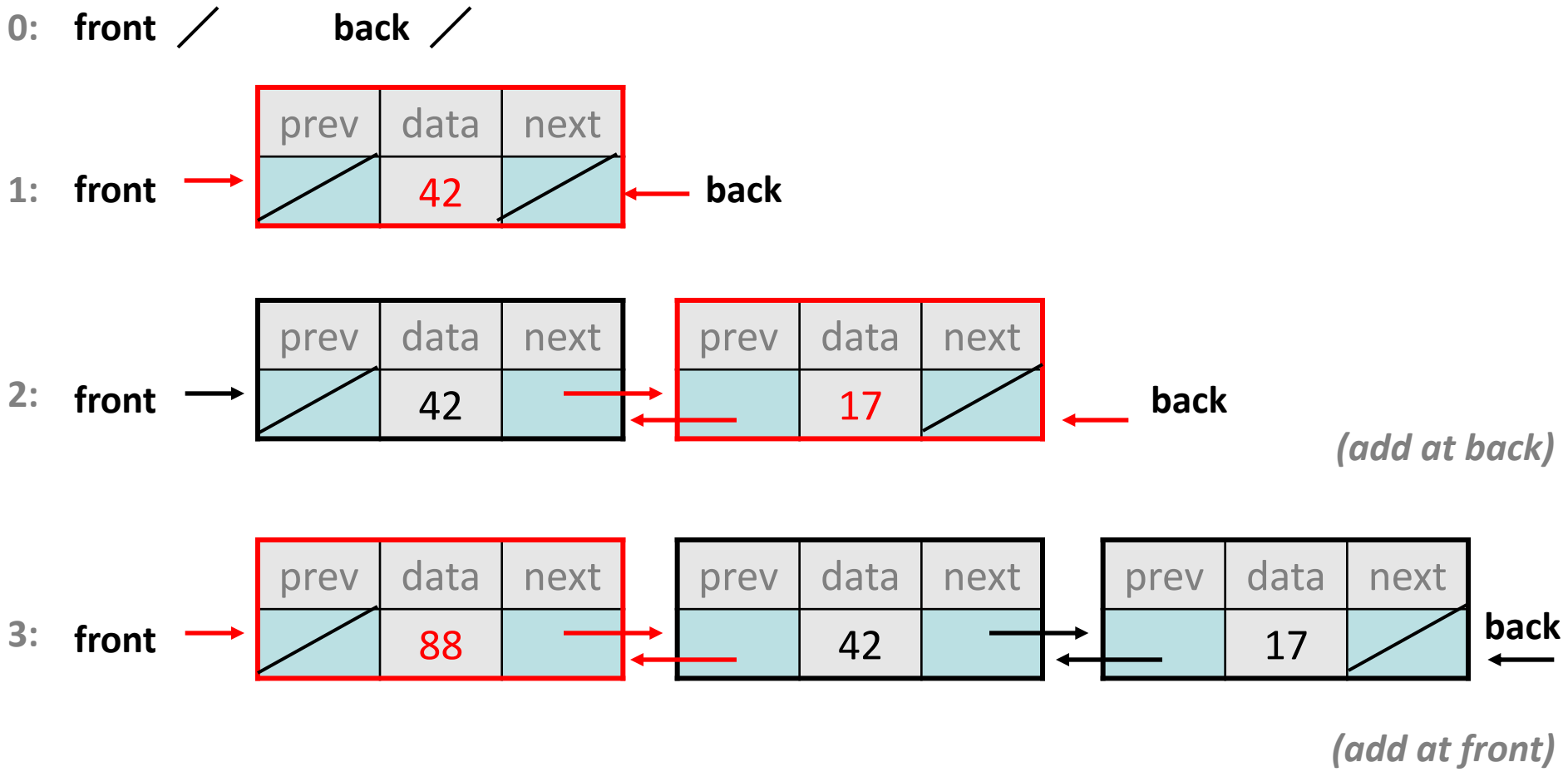
Doubly linked list

- **doubly linked list**: Each node has a pointer to next and prev node.
  - Allows walking forward and backward in list efficiently.
  - Overall list often maintains a **back** pointer to end of list.



D.L. list growth

- State of a doubly linked list of 0, 1, 2, *N* nodes:



D.L. list remove

- When removing a node, must change two pointers.
  - Might also need to change front and/or back.
  - Example: Try removing each of the three nodes below.