Do Now Exercise

Last week, we implemented some operations performed on linked lists. To prepare you for the lecture today, please do the following exercise.

Write a use case in which you think linked lists (do or don't) work well.

COMP15: Data Structures

Week 3, Summer 2019

Admin

T3: **touch, mkdir, cp**

Due by 6pm on Wednesday, June 12

(a quick demo)

P2: **Event Scheduler**

Project Due by 6pm on Sunday, June 16

Project planning is important.

What **milestones**
should be included in project plans?

Students' answers:

The development process is often **iterative**.

Update your **project plan** whenever needed!

Do you have any questions about the refinement phase?

As a programmer,

if your program passed **9 / 10** tests,
should you be satisfied with the result?

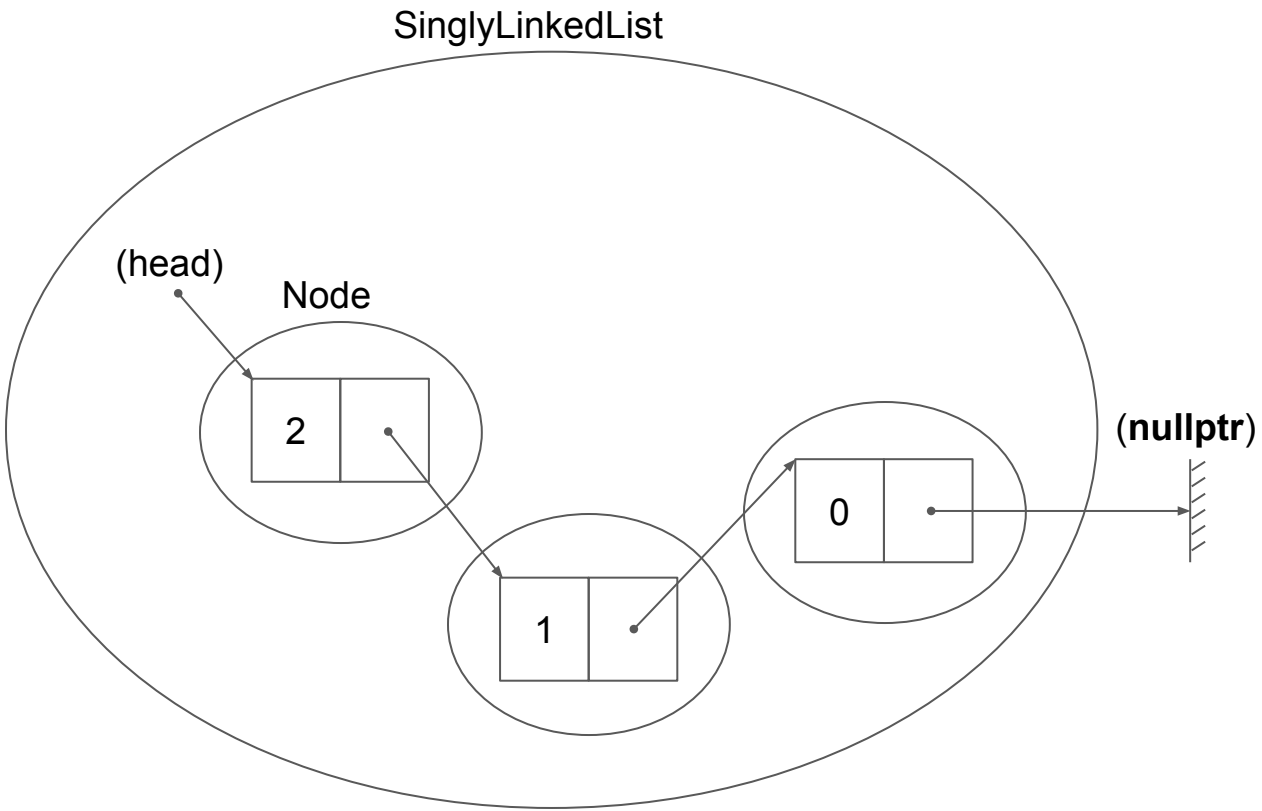
As a programmer,

if your program passed **9 / 10** tests,
should you be satisfied with the result?

Feedbacks on **P1** from Matt

Linked List

(Notes from the live coding. Please do NOT assume the code is complete.)



Singly Linked List with **head**

Memory Management

"With great power comes great responsibility" from Spider-Man comic

https://en.wikipedia.org/wiki/With_great_power_comes_great_responsibility

Memory leaks

valgrind

Memory errors

(Notes from the live coding. Please do NOT assume the code is complete.)

```
 1#include <iostream>
 2
 3int main(){
 4  int* c = new int(3);
 5
 6  std::cout << (*c) << std::endl;
 7
 8  delete c;
 9
10  int* d = c;
11
12  std::cout << (*d) << std::endl;
13  return 0;
14}
```

core.* and **vgcore.*** files

Let's implement **removeBack()**

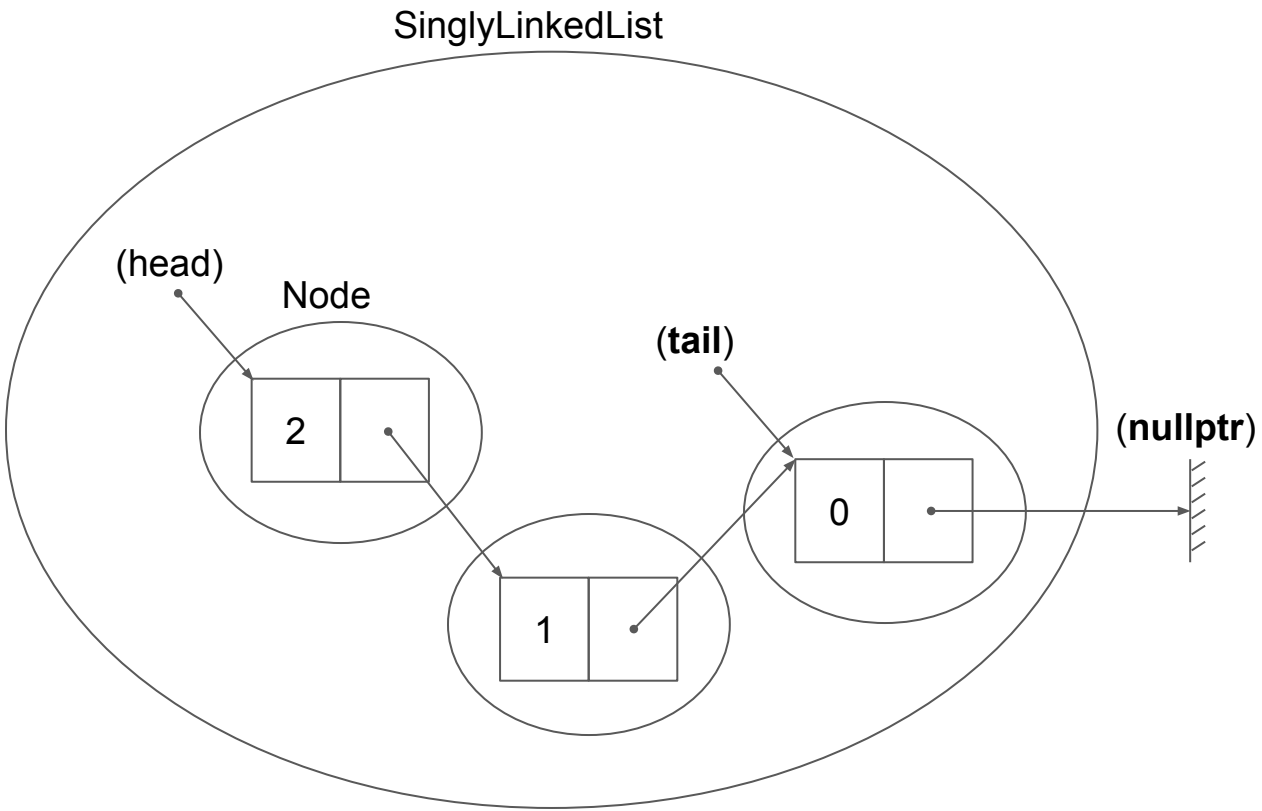
(Notes from the live coding. Please do NOT assume the code is complete.)

```
1//SinglyLinkedList.hpp
2#ifndef SINGLYLINKEDLIST_HPP
3#define SINGLYLINKEDLIST_HPP
4
5#include "Node.hpp"
6
7class SinglyLinkedList{
8public:
9  SinglyLinkedList();
10  //  SinglyLinkedList(const SinglyLinkedList& other);
11  //  SinglyLinkedList& operator=(const SinglyLinkedList& other);
12  ~SinglyLinkedList();
13
14  void addToBack(int data);
15  void removeBack(); //(1)
16  std::string toString() const;
17
18private:
19  Node* head;
20};
21
22#endif
```

tail

Singly Linked List with **head** and **tail**

(Notes from the live coding. Please do NOT assume the code is complete.)



Discussion 1:

If a SinlgyLinkedList has the head and tail,

how will the **addToBack()** be changed?

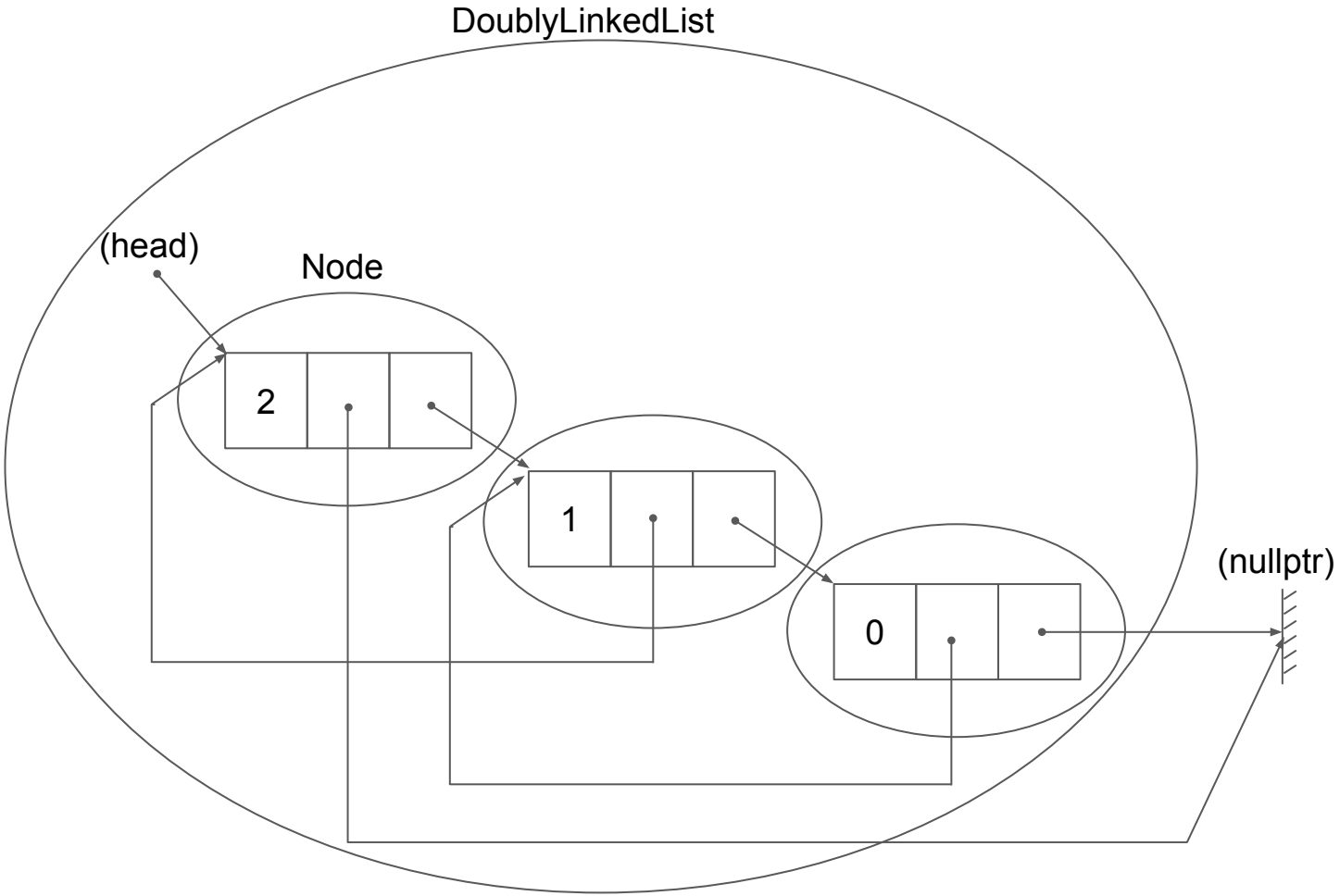
Discussion 2:

If a SinlgyLinkedList has the head and tail,

how will the **removeBack()** be changed?

DoublyLinkedList class

(Notes from the live coding. Please do NOT assume the code is complete.)



Doubly Linked List with **head**

Doubly Linked List with **head** and **tail**

Discussion 3:

If a DoublyLinkedList has the head and tail,

how will the **removeBack**() be changed,

compared to the case where the DoublyLinkedList only has the head?

Given the following scenario, what should the "number" be?

```
//to retrieve i-th (0-based) data in this linked list
//int at(int ith);



LinkedList linkedList;

int number = linkedList.at(0);
```

Given the following case, what should the "number" be?

```
//to retrieve i-th (0-based) data in this linked list
//int at(int ith);



LinkedList linkedList; //This linked list is empty.

int number = linkedList.at(0);
```

Exceptions

```
#include <stdexcept>

std::runtime_error
```

(and more...)

```
LinkedList linkedList; //currently empty

try{

  int number = linkedList.at(0);

}catch(std::runtime_error& e){
  std::string message = e.what();
  std::cout << message << std::endl;
}
```

```
int LinkedList::at(int ith){
  if(/* the given ith is not appropriate */){

    throw std::runtime_error("MESSAGE");

  }else{
    return /* the i-th data */;
  }
}
```

Deep copy and Shallow copy

(Instead of defining them, we discuss them using demos.)

(Notes from the live coding. Please do NOT assume the code is complete.)

```cpp
#include <iostream>

int main(){
  int a = 3;
  std::cout << " a: " << a << std::endl;
  std::cout << "&a: "<< &a << std::endl;

  int* b = &a;
  std::cout << "*b: " << (*b) << std::endl;
  std::cout << " b: " << b << std::endl;

  int c[3];
  c[0] = 0;
  c[1] = 1;
  c[2] = 2;

  int* d = new int[3];
  d[0] = 0;
  d[1] = 1;
  d[2] = 2;

  int* e = d;

  int* f = new int[3];
  f[0] = d[0];
  f[1] = d[1];
  f[2] = d[2];
  return 0;
}
```

In Your Pocket

arrays

linked lists

man ssh exit pwd
cd ls valgrind

Do Now Exercise

Last week, we implemented some operations performed on linked lists. To prepare you for the lecture today, please do the following exercise.

Write a use case in which you think linked lists (do or don't) work well.

Do Now Exercise

Students' answers:

Complexity Analysis

We have been using
an ambiguous term "work well".

What does the "work well" mean really?

For example...

For example...

(We discussed the at() method in Array and LinkedList class.)

(Slide from Week 1)

Because we do care about

performances of software program.

(This is based on Tomoki's view. You will be able to find various answers on the Internet.)

Computing running time

Time (or Speed)

Time Complexity Analysis

Big-O notation

Asymptotic Analysis

In-Class Activity

In terms of the size of data (n)

O(1)

constant time

O(n)

linear time

Some keywords from today's lecture:

- memory errors
- tail pointer
- Doubly Linked List
- exceptions
- runtime error exception (and more in the lab)
- "throw" keyword
- "try" and "catch" keywords and blocks
- deep copy v.s. shallow copy
- (time) complexity analysis
- Big-O notation
- asymptotic analysis
- O(1), constant time
- O(n), linear time

To the lab!