Do Now Exercise

We learned arrays last week, and hopefully you have already started working on your Project 1. To prepare you for the lecture today, please do the following exercise.

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

COMP15: Data Structures

Week 2, Summer 2019

Admin

Canvas notifications

It may be a good idea to enable some notification settings on Canvas.

> Announcement
> Discussions

(Canvas screenshot)

Git and Tufts CS GitHub

Any questions so far
about the course workflow?

T1: **man**, **ssh**, **exit**

Due by several minutes ago :)

So far, your report looks good,

except ...

So far, your reports look good,

except ... often **references** are missing.

T2: **pwd**, **cd**, **ls**

Due by 6pm on Wednesday, June 5th

(a quick demo)

P1: **Card Deck**

Due by 6pm on Sunday, June 2nd

Any questions about P1?

Discussion 1:
How would you compile your P1 source code?

$

```
$ clang++
```

```
$ clang++                                    test.cpp Card.cpp CardDeck.cpp
```

```
$ clang++                                    test.cpp Card.cpp CardDeck.cpp -o test
```

```
$ clang++ -std=c++11                    test.cpp Card.cpp CardDeck.cpp -o test
```

```
$ clang++ -std=c++11 -Wall -Wextra        test.cpp Card.cpp CardDeck.cpp -o test
```

```
$ clang++ -std=c++11 -Wall -Wextra -g     test.cpp Card.cpp CardDeck.cpp -o test
```

```
$ clang++ -std=c++11 -Wall -Wextra -g -O0 test.cpp Card.cpp CardDeck.cpp -o test
```

Discussion 2:
Where did the "**13**" stuff come from?

(By design. Case-by-case. We need a strategy for it.)

Any questions about P1?

If you finish your P1 early,
try implementing a card game, for example, *Blackjack*,
using your **Card** and **CardDeck** modules!

Note about the assignment operator of Array class

(We discussed why we need (22) in the code from the last live coding session.)

Remaining Questions

In Array class,

- The default constructor and the non-default constructor is doing something very similar.
- The copy constructor and the assignment operator is doing something very similar.
- A part of the assignment operator and the destructor is doing something very similar.

Can we logically group such operations?

=> Refactoring

In-Class Activity

How would you pronounce this?

```
bool** board;
```

A sneak peek preview (Comp 111, Operating Systems)

Memory Segments

(stack and heap)

A sneak peek preview (Comp 111, Operating Systems)

Memory Segments

(stack and heap)

(A version of the figure we drew in class.
Ref: https://en.wikipedia.org/wiki/File:Program_memory_layout.pdf
In class, our focus was on the stack and heap areas.)

Let's take a short break!

In Your Pocket

arrays

man, ssh, exit

Do Now Exercise

We learned arrays last week, and hopefully you have already started working on your Project 1. To prepare you for the lecture today, please do the following exercise.

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

Do Now Exercise

Students' answers:

(We had about 3 examples for each of "may work well" and "may not work well".)

How about this...

```
//void add(int number);

Array a;
a.add(0);
a.add(1);
a.add(2);
a.add(3);
...
...
a.add(...);
```

How about this...

```
//void add(int number);

Array a;
a.add(0);
a.add(1);
a.add(2);
a.add(3);
...
...
a.add(...);
```

At some moments,
the array will need to be expanded.

How about this...

```
//void insert(int number, int at);

Array a;
a.insert(0, 0);
a.insert(1, 0);
a.insert(2, 0);
a.insert(3, 0);
...
...
a.insert(..., 0);
```

How about this...

```
//void insert(int number, int at);

Array a;
a.insert(0, 0);
a.insert(1, 0);
a.insert(2, 0);
a.insert(3, 0);
...
...
a.insert(..., 0);
```

In this example, every time a new element is inserted, first, all of the existing elements need to be shifted by one.

Linked List

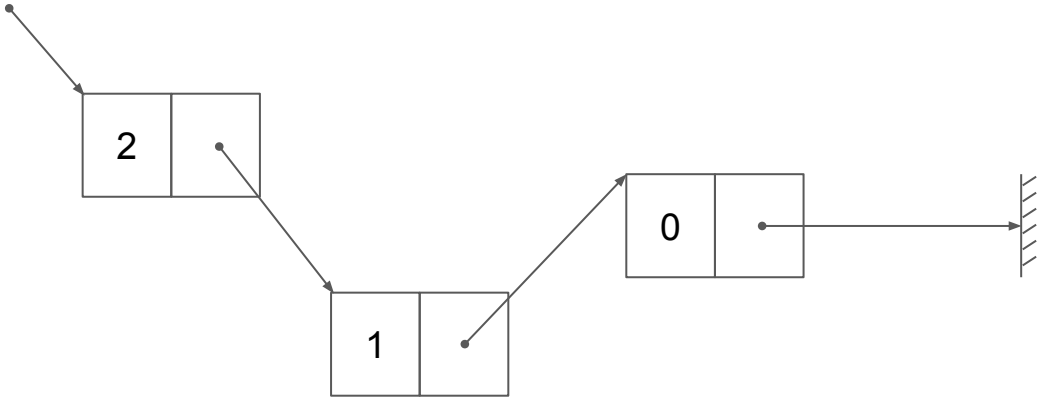
Linked structures

Live coding

Goal of Live Coding:

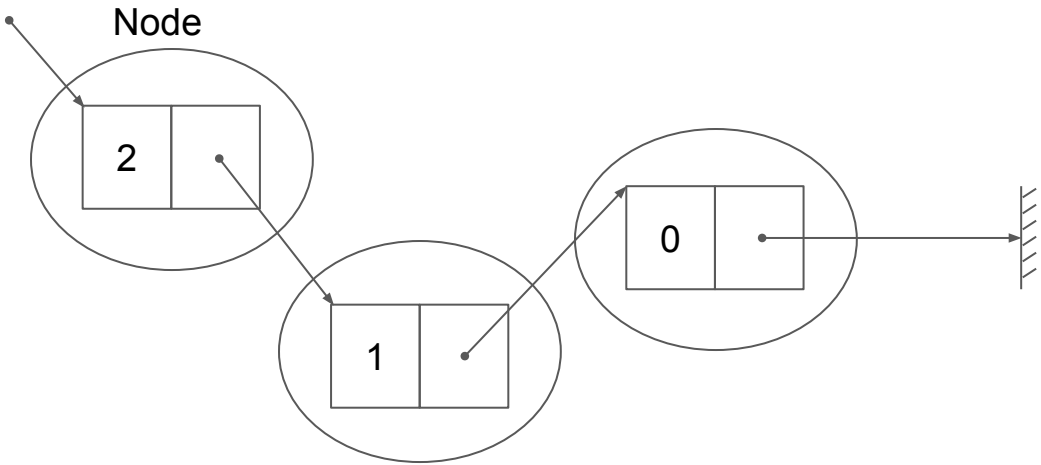
Learn a procedure of designing classes, not memorize code.

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



Node class

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



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

```
 1//Node.hpp
 2#ifndef NODE_HPP //(3)
 3#define NODE_HPP //(3)
 4
 5#include <string>
 6
 7class Node{ //(1)
 8public: //(4)
 9  Node(); //(6)
10  Node(int data); //(7)
11  //  Node(const Node& other);
12  //  Node& operator=(const Node& other);
13  //  ~Node();
14
15  int getData() const; //(14)
16  Node* getNext() const; //(11)
17  void setNext(Node* node); //(9)
18  std::string toString() const; //(12)
19
20private: //(4)
21  int data; //(5)
22  Node* next; //(5)
23}; //(2)
24
25#endif //(3)
```

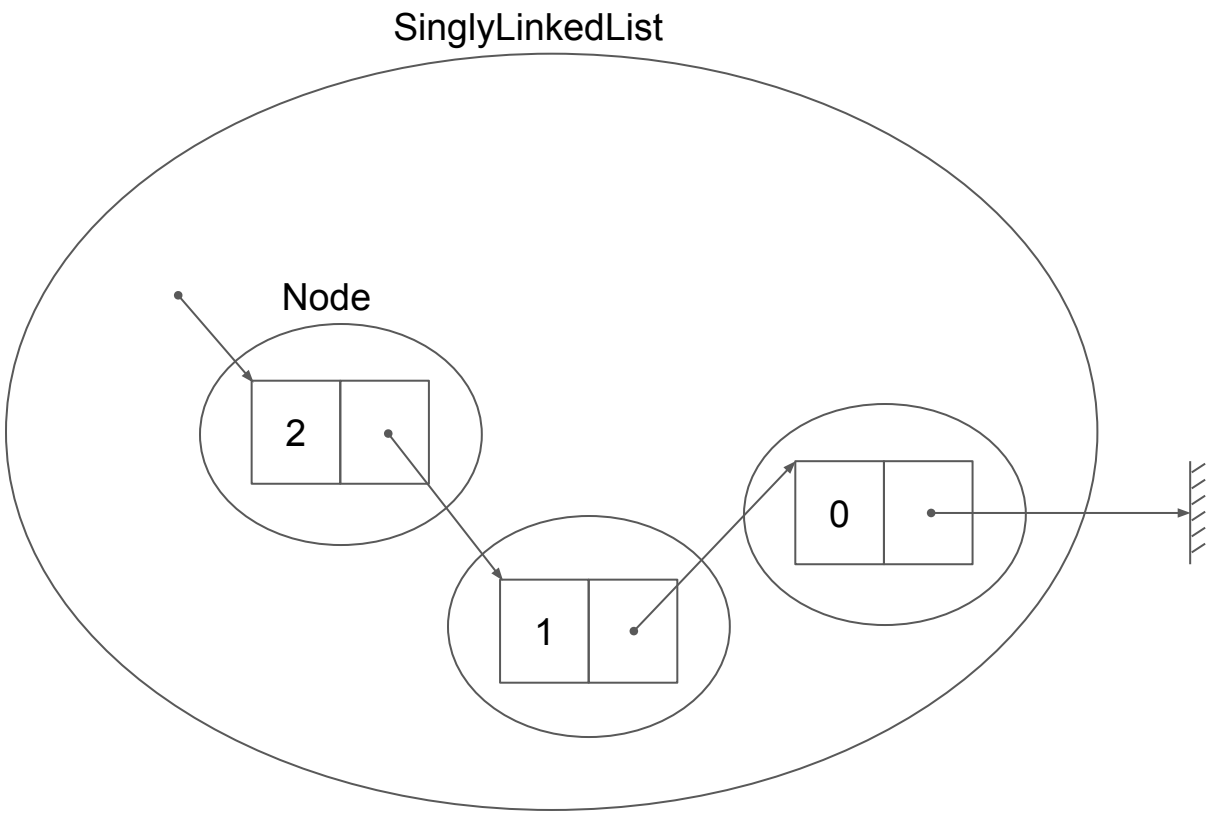
(1) class keyword
(2) semicolon
(3) header guard
(4) public, private keyword
(5) our design
(6) default constructor
(7) user defined constructor
(8, 10, 13) for tests
(9, 11, 12, 14) operations to support

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

```
1//test.cpp
2#include <iostream>
3#include "Node.hpp"
4#include "SinglyLinkedList.hpp"
5
6int main(){
7  Node* n1 = new Node(2); //(8)
8  Node* n2 = new Node(1); //(8)
9  Node* n3 = new Node(0); //(8)
10
11  n1->setNext(n2); //(10)
12  n2->setNext(n3); //(10)
13
14  std::cout << n1->toString() << std::endl; //(13)
15  std::cout << n1->getNext()->toString() << std::endl; //(13)
16  std::cout << n1->getNext()->getNext()->toString() << std::endl; //(13)
17  return 0;
18}
```

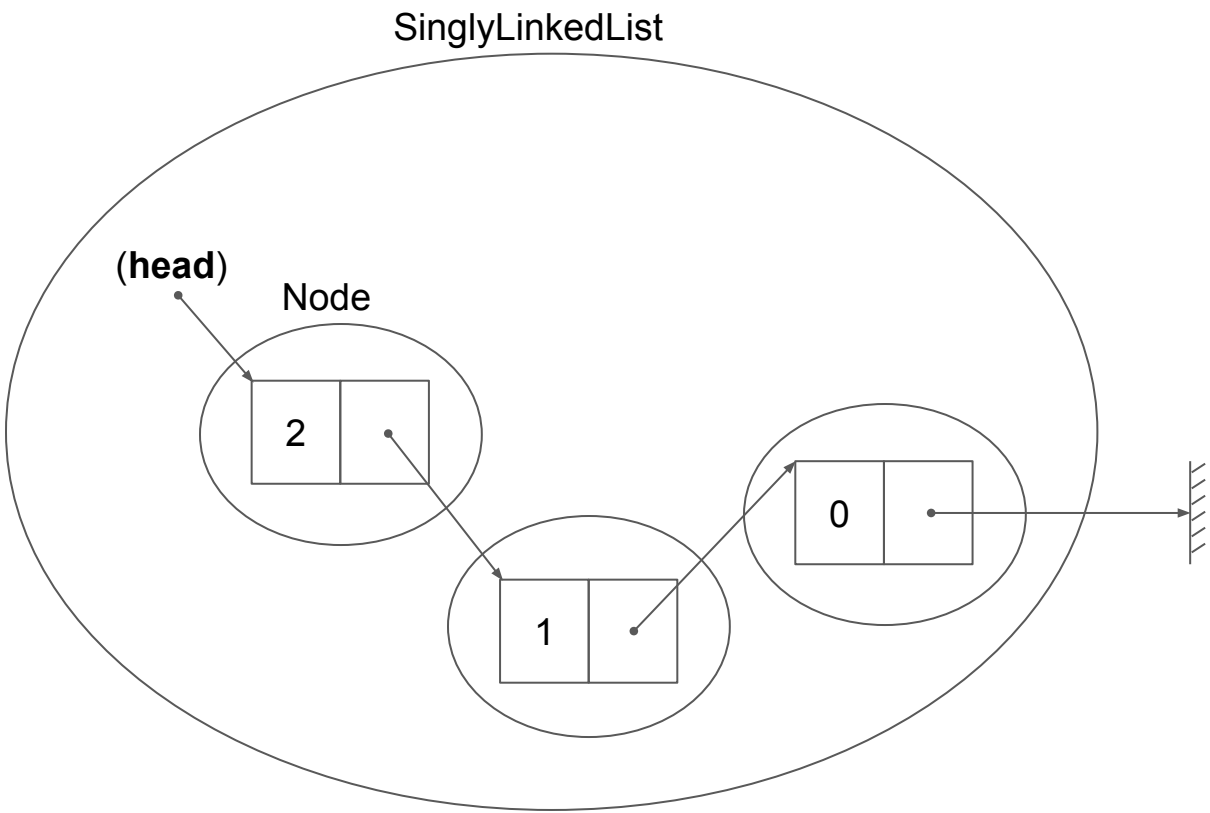
SinglyLinkedList class

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



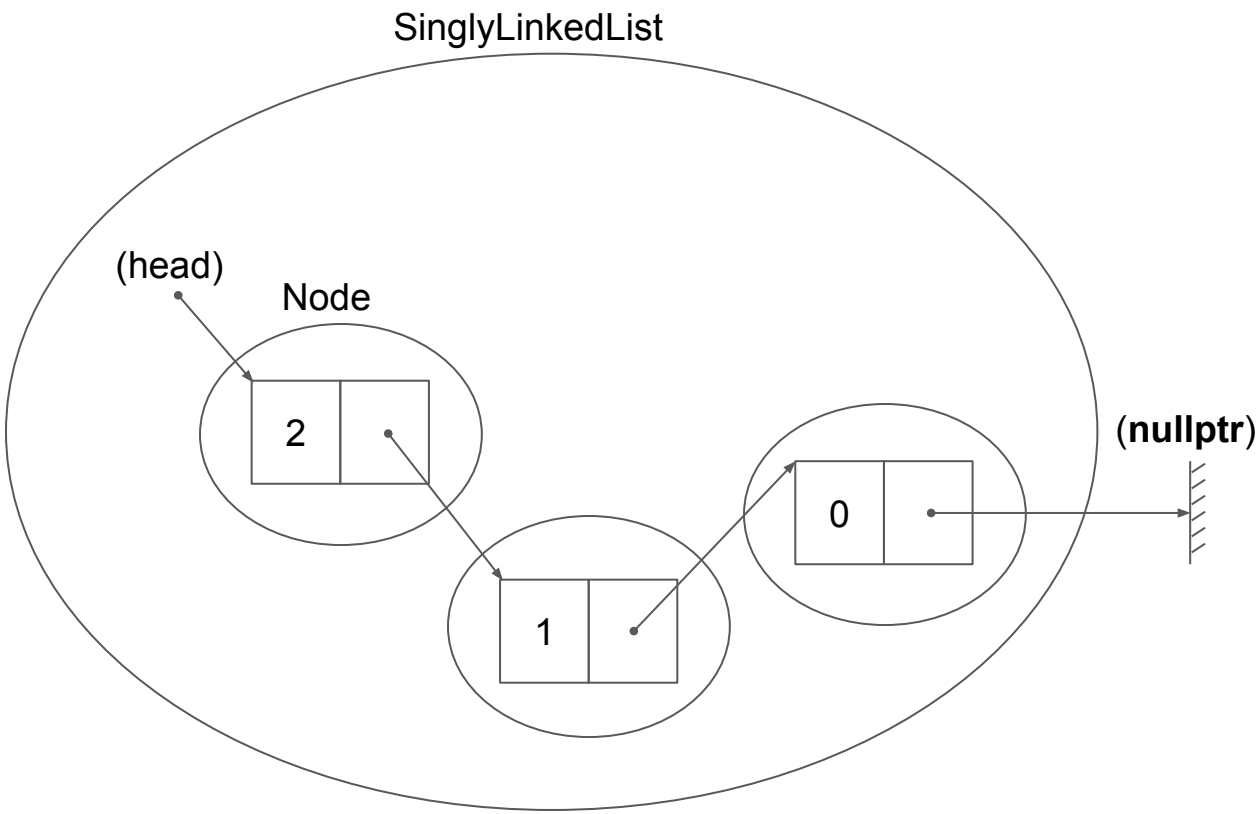
head

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



nullptr

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



Singly Linked List with **head**

(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  std::string toString() const;
16private:
17  Node* head;
18};
19
20#endif
```

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

```cpp
//test.cpp
#include <iostream>
#include "Node.hpp"
#include "SinglyLinkedList.hpp"

int main(){
  /*
  Node* n1 = new Node(2); //(8)
  Node* n2 = new Node(1); //(8)
  Node* n3 = new Node(0); //(8)

  n1->setNext(n2); //(10)
  n2->setNext(n3); //(10)

  std::cout << n1->toString() << std::endl; //(13)
  std::cout << n1->getNext()->toString() << std::endl; //(13)
  std::cout << n1->getNext()->getNext()->toString() << std::endl; //(13)
  */

  SinglyLinkedList linkedList;
  linkedList.addToBack(2);
  linkedList.addToBack(1);
  linkedList.addToBack(0);
  std::cout << linkedList.toString() << std::endl;
  return 0;
}
```

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

We have:

- designed two classes, Node and SinglyLinkedList, from the scratch.
- implemented Node class. (except for the copy constructor, assignment operator and destructor.)
- implemented some parts of SinglyLinkedList class.
  - constructor
  - addToBack()
  - addToFront() -> Lab2
  - removeFront() -> Lab2
  - toString() -> Lab2
  - destructor -> Lab2

    (not today)
    - copy constructor, assignment operator
    - removeBack()
    - insert()
    - has()
    - size(), isEmpty()
    - etc.

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

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

```
: valgrind --leak-check=full --show-leak-kinds=all ./test
```

```
==13626== LEAK SUMMARY:
==13626==    definitely lost: 16 bytes in 1 blocks
==13626==    indirectly lost: 32 bytes in 2 blocks
==13626==      possibly lost: 0 bytes in 0 blocks
==13626==    still reachable: 72,704 bytes in 1 blocks
==13626==         suppressed: 0 bytes in 0 blocks
==13626==
==13626== For counts of detected and suppressed errors, rerun with: -v
==13 ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 6 from 6)
```

Some keywords from today's lecture:

- refactoring
- a pointer pointing to a pointer pointing to XXX
- 2D array
- memory segments (stack and heap)
- Node
- Linked List (linked structure)
- Singly Linked List
- head
- nullptr
- memory management
- memory leaks
- valgrind

To the lab!