CS 211 : Thurs 01/30 (lecture 08)

Prof. Hummel
(he/him)

- **<u>Topics</u>: unit testing, code coverage, google test**

Notes:

- *Lecture slides available on Canvas*
- ***Project 03** extended through tonight (Tues 1/30)*
- ***HW 04** due Thursday night – complete before working on project 04*
- ***Project 04** due Sunday night – NO late period, all submissions due by Sunday night.*

Northwestern University

Looking forward…

February 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

www.a-printable-calendar.com

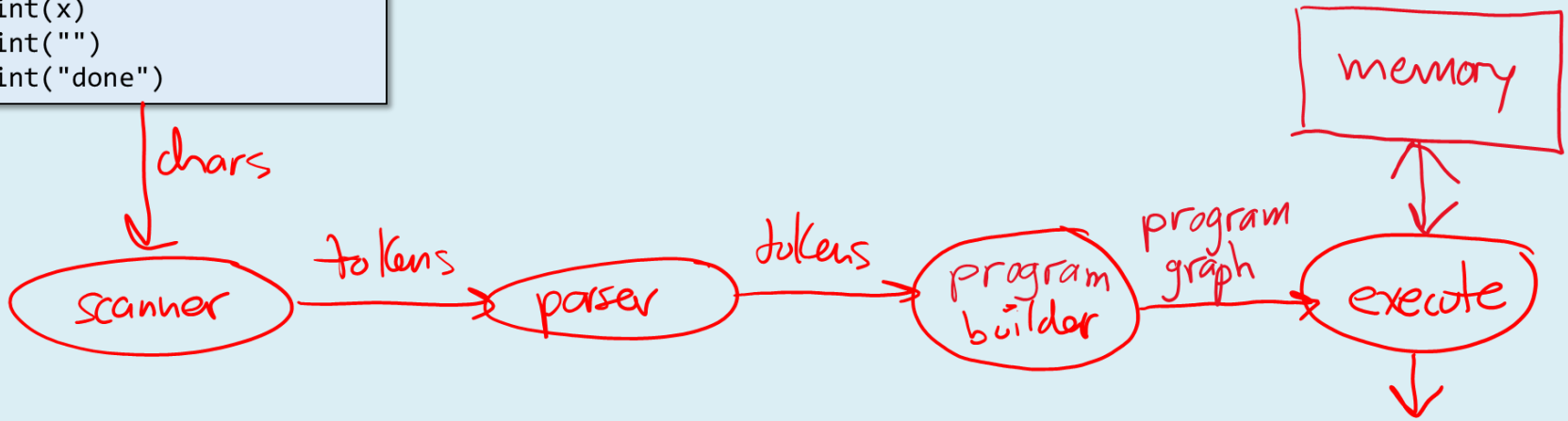
March 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

www.a-printable-calendar.com

nuPython

```
print("starting")
print("")
x = input("enter int> ")
x = int(x)
x = x + 1
print(x)
print("")
print("done")
```

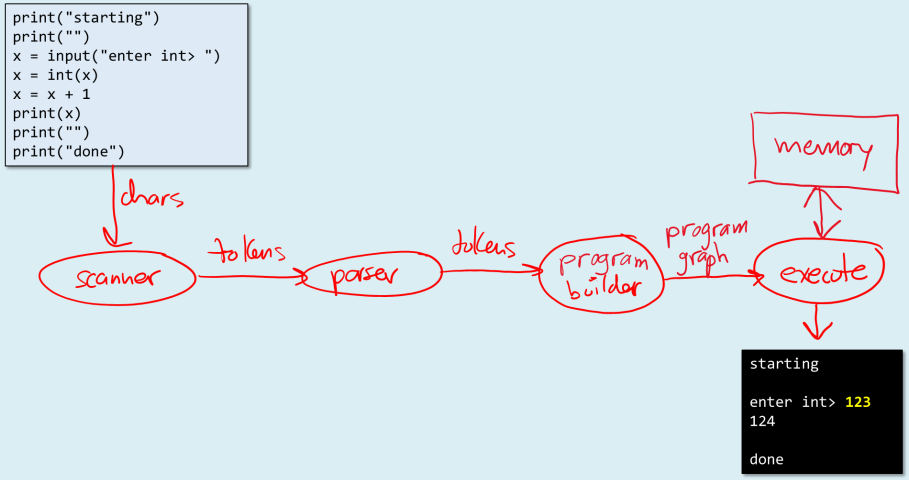


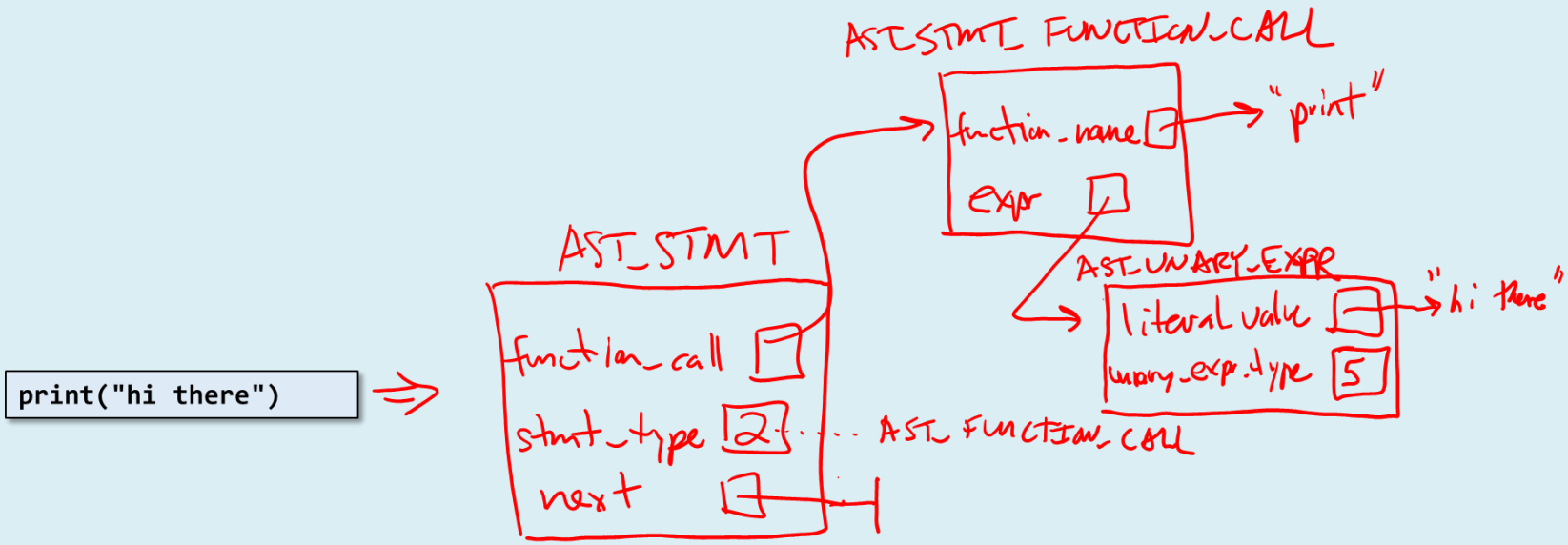

```
starting

enter int> 123
124

done
```

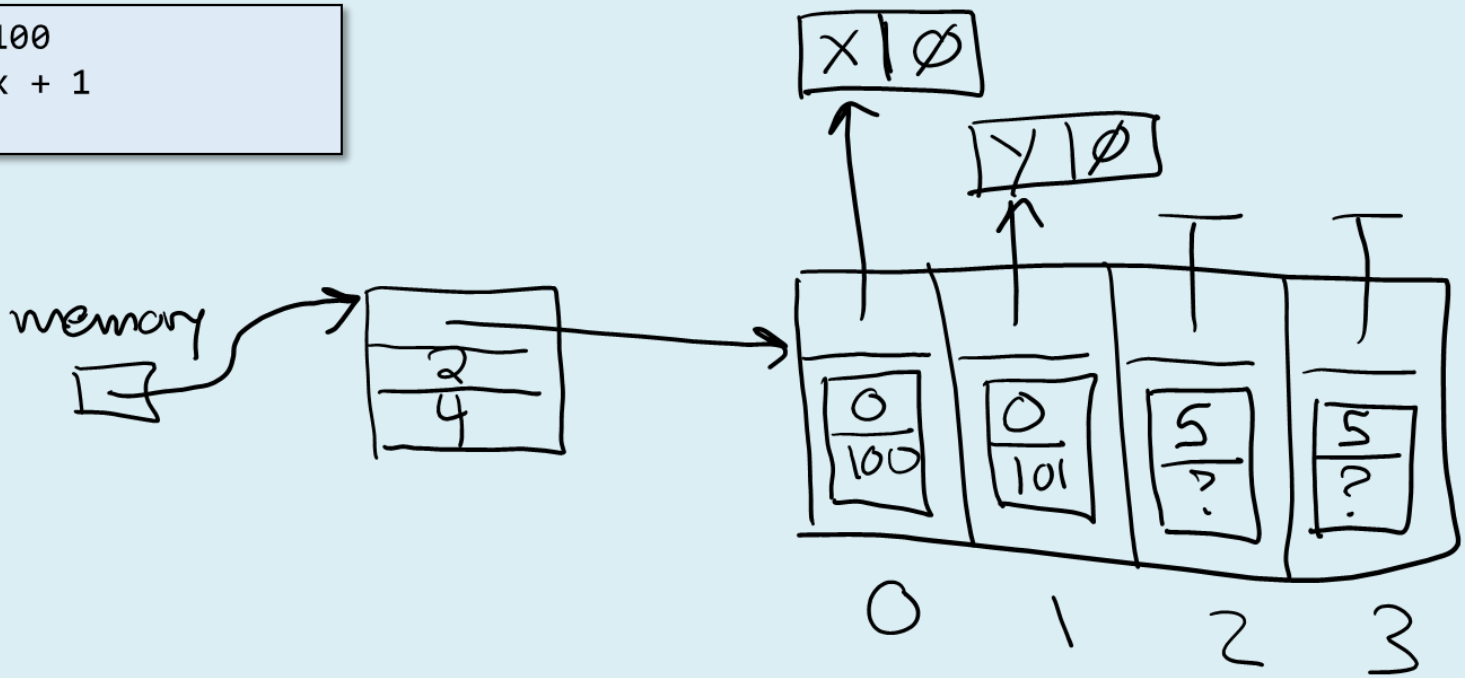
Program graph



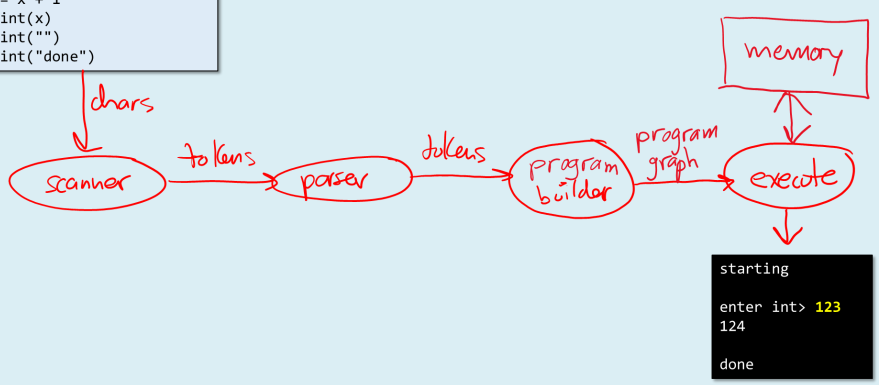


RAM module (project 04)

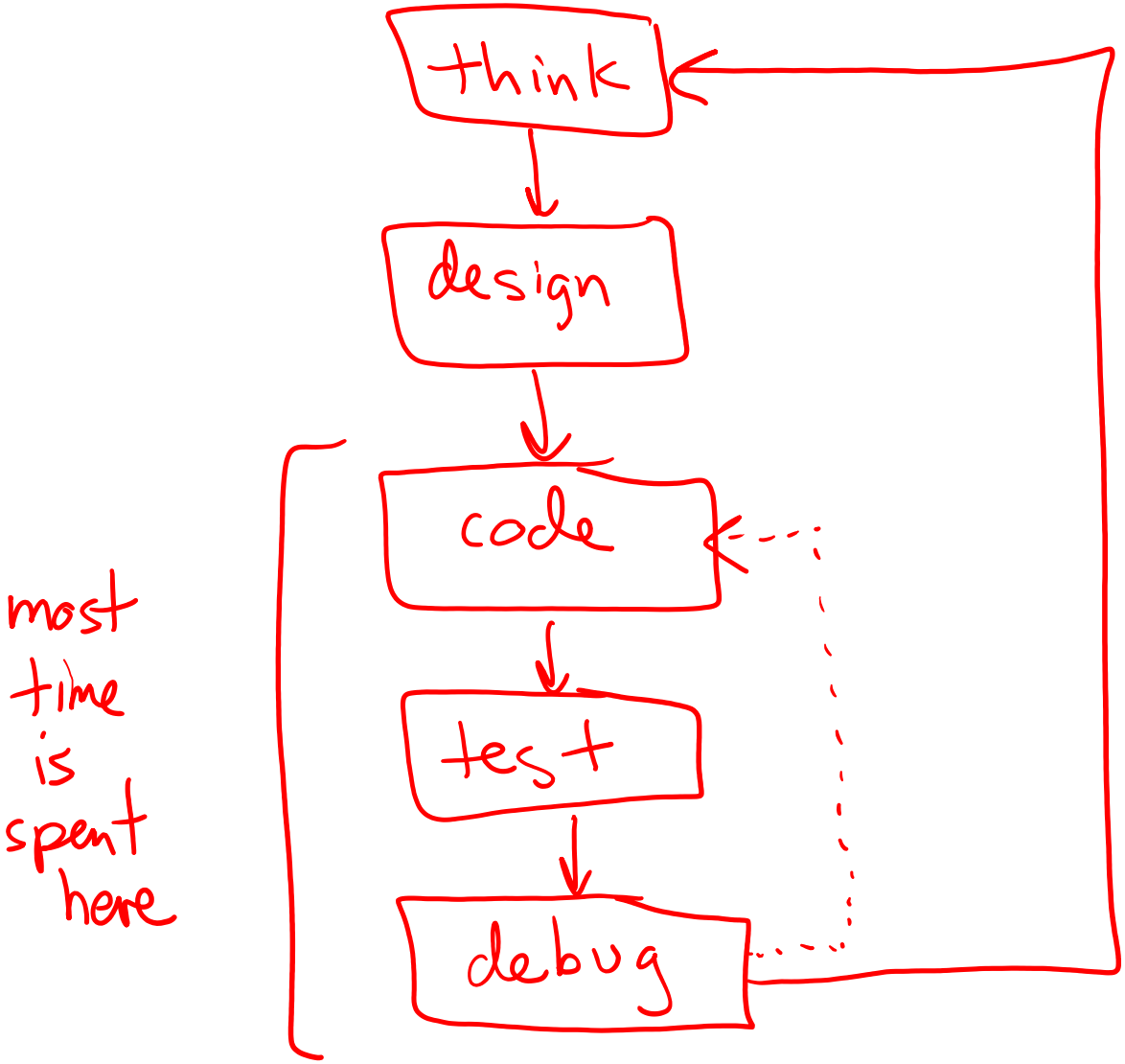
```
x = 100
y = x + 1
```




```
print("starting")
print("")
x = input("enter int> ")
x = int(x)
x = x + 1
print(x)
print("")
print("done")
```



Software development



Cyclomatic complexity

Cyclomatic complexity

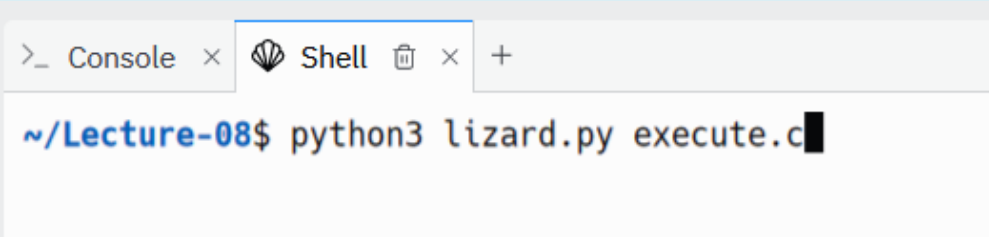
Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. It was developed by Thomas J. McCabe, Sr. in 1976. Wikipedia

What is good cyclomatic complexity?

Any function with a cyclomatic complexity below 10 can be considered simple and testable while a cyclomatic complexity greater than 20 indicates an overly complex function, so an acceptance threshold for functions should be defined between 10 and 20, depending on the application domain. Jun 23, 2021

CCN

- **Replit? Download "execute.c" from Project 03, open project "Lecture 8", upload "execute.c", switch to Shell:**



- **Login to the EECS computers, cd to release directory for Project 03:**

```
hummel@moore$
hummel@moore$ python3 /home/cs211/w2024/tools/lizard.py execute.c
```

- **iClicker: what is your largest CCN reported?**

Testing

- **Testing is hard**
- **You have to create scenarios that execute *all* possible paths through the code**
- **You have to repeat it over and over again…**

Unit testing

- **Industry standard approach**
- **Idea:**
  - *Break software into "units"*
  - *Lots of tests (think **thousands**)*
  - *Automated by a testing framework*
    - CATCH, Google Test, JUnit

```
Test01()
{ ... }
```

```
Test02()
{ ... }
```

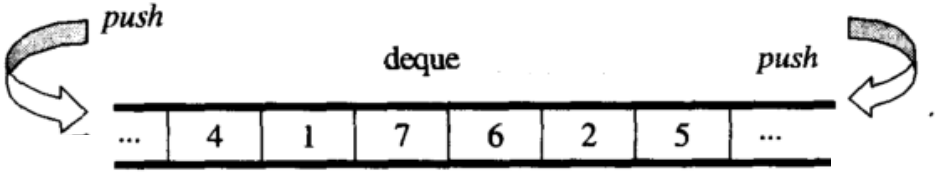
```
Test03()
{ ... }
```

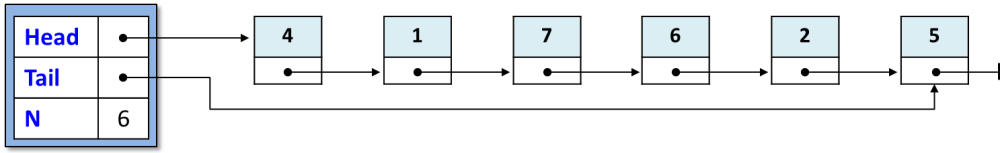
```
Test04()
{ ... }
```

⋮

Example

- **Deque ("deck")**
  - *From C++ library*
  - *An **abstract data type** that allows insert @ front and back*
  - *Implemented using a **linked-list** data structure*





Google Test ("gtest")

- **Google test** is an industry standard unit testing framework
- Using in Project 04

```
#include <stdio.h>
#include <stdlib.h>

#include "gtest/gtest.h"

int main()
{
  ::testing::InitGoogleTest();

  //
  // run all the tests, returns 0 if all pass
  //
  int result = RUN_ALL_TESTS();

  return result;
}
```

```
TEST(deque, initialization)
{
  struct IntDeque *dq = intdeque_create();
  ASSERT_TRUE(dq != NULL);
  ASSERT_TRUE(intdeque_size(dq) == 0);
}

TEST(deque, add_to_front)
{
  struct IntDeque *dq = intdeque_create();
  ASSERT_TRUE(dq != NULL);
  ASSERT_TRUE(intdeque_size(dq) == 0);

  intdeque_push_front(dq, 123);

  ASSERT_TRUE(intdeque_size(dq) == 1);

  int value;
  ASSERT_TRUE(intdeque_get(dq, 0, &value));
  ASSERT_TRUE(value == 123);
}
```

Working with Google test

- Login to replit.com
- Open team…
- Open project "Lecture 8"

```
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from deque
[ RUN      ] deque.initialization
[       OK ] deque.initialization (0 ms)
[ RUN      ] deque.add_to_front
[       OK ] deque.add_to_front (0 ms)
[----------] 2 tests from deque (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

```
struct IntDeque* intdeque_create(void);

void intdeque_destroy(struct IntDeque* dq);

void intdeque_push_front(struct IntDeque* dq, int value);

void intdeque_push_back(struct IntDeque* dq, int value);

int  intdeque_size(struct IntDeque* dq);

bool  intdeque_get(struct IntDeque* dq, int position, int* value);

void intdeque_print(struct IntDeque* dq);
```

Add another unit test...

- Let's test intdeque_push_back( )

```
TEST(deque, add_to_back)
{
   struct IntDeque* dq = intdeque_create();
   ASSERT_TRUE(dq != NULL);
   ASSERT_TRUE(intdeque_size(dq) == 0);

   intdeque_push_back(dq, 123);

   ASSERT_TRUE(intdeque_size(dq) == 1);

   int value;
   ASSERT_TRUE(intdeque_get(dq, 0, &value) == true);
   ASSERT_TRUE(value == 123);
}
```

How good are my tests?

- Is there a way to measure test quality?
- Yes!
- Code coverage
- The % of code “covered” (executed) by the tests
- The goal?

Code coverage with gcov

1. Build and run program with coverage options (if these don't work, try "--coverage")
2. Run “gcov” to collect coverage information
3. Open .gcov file(s) to view results

```
gcc -std=c11 -g –Wall –fprofile-arcs –ftest-coverage
 main.c  intdeque.c  ...

./a.out

gcov a-intdeque.c

<< open intdeque.c.gcov in editor >>
```

```
$ gcov a-intdeque.c
File 'intdeque.c'
Lines executed:44.23% of 52
Creating 'intdeque.c.gcov'
```

% of code covered

intdeque.c.gcov

```
        -:   20:
        1:   21:  dq->head = NULL;
        1:   22:  dq->tail = NULL;
        1:   23:  dq->N = 0;
        -:   24:
        1:   25:  return dq;
        -:   26:}
        -:   27:
    #####:   28:void intdeque_destroy(struct IntDeque *dq) {
    #####:   29:  struct IntNode *cur = dq->head;
        -:   30:
        -:   31:  //
        -:   32:  // free the nodes in the list:
        -:   33:  //
    #####:   34:  while (cur != NULL) {
    #####:   35:    struct IntNode *next = cur;
        -:   36:
    #####:   37:    free(cur);
        -:   38:
    #####:   39:    cur = next;
        -:   40:  }
        -:   41:
    #####:   42:  free(dq); // now free head structure
    #####:   43:}
```

Lines marked with "#####" were not executed…

More testing…

- **Test push_back and push_front together…**

```
TEST(deque, add_to_back)
{
   struct IntDeque* dq = intdeque_create();
   ASSERT_TRUE(dq != NULL);
   ASSERT_TRUE(intdeque_size(dq) == 0);

   intdeque_push_back(dq, 123);
   .
   .
   .

   intdeque_push_back(dq, 456);
   .
   .
   .
}
```

Status?

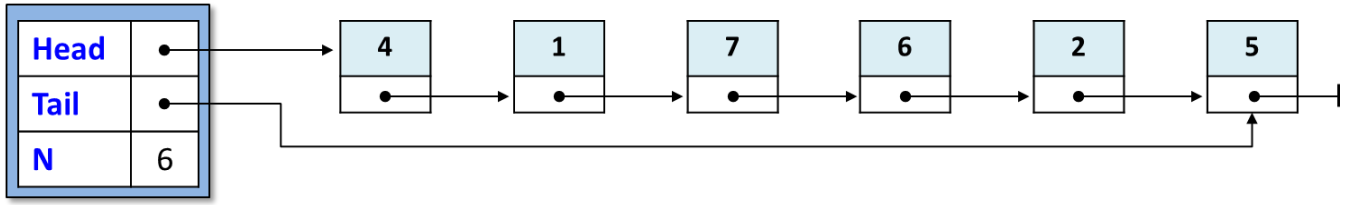
- **Fix the error...**

- **Code coverage?**

- **Are we good?**
- **Not so much…**
  - *Think edge cases…*

Deletion

- **Let's add the function intdeque_delete( )**
- **When you delete from a linked-list, use two pointers**
  - *Position prev and cur around the deletion point…*




```
struct IntDeque {
  struct IntNode* head; // 1st element
  struct IntNode* tail; // last element
  int             N;    // # of values
};
```

```
struct IntNode {
  int             value;
  struct IntNode* next;
};
```

What should I be working on?

Project 03? *Due tonight…*

HW 04 *due Thursday night…*

Project 04 *due Sunday night…*