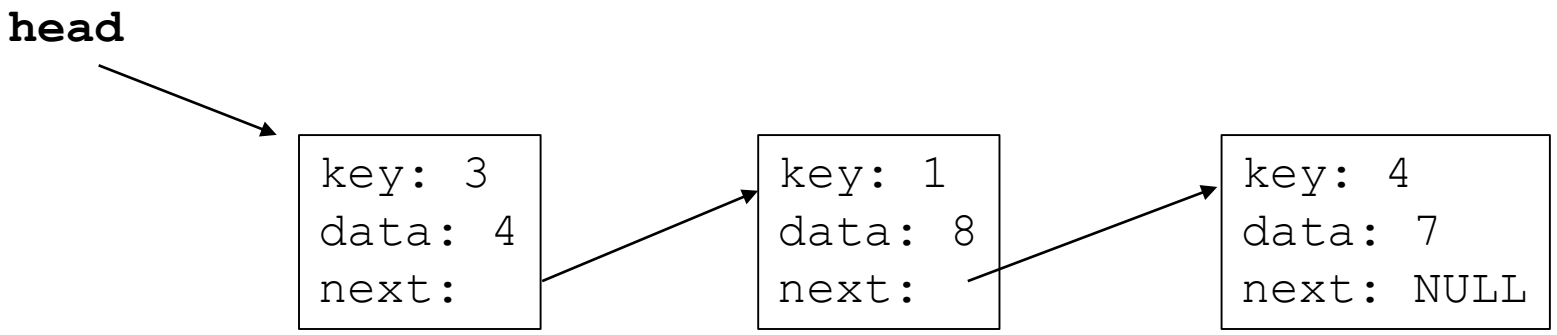
Linked List: Incremental Design and Test

Find(key) – loop thru list from head, look for match to key

Insert(key, data) – lookup key: if found, replace data; otherwise, create new node w/ key and data and push it onto the front of list.

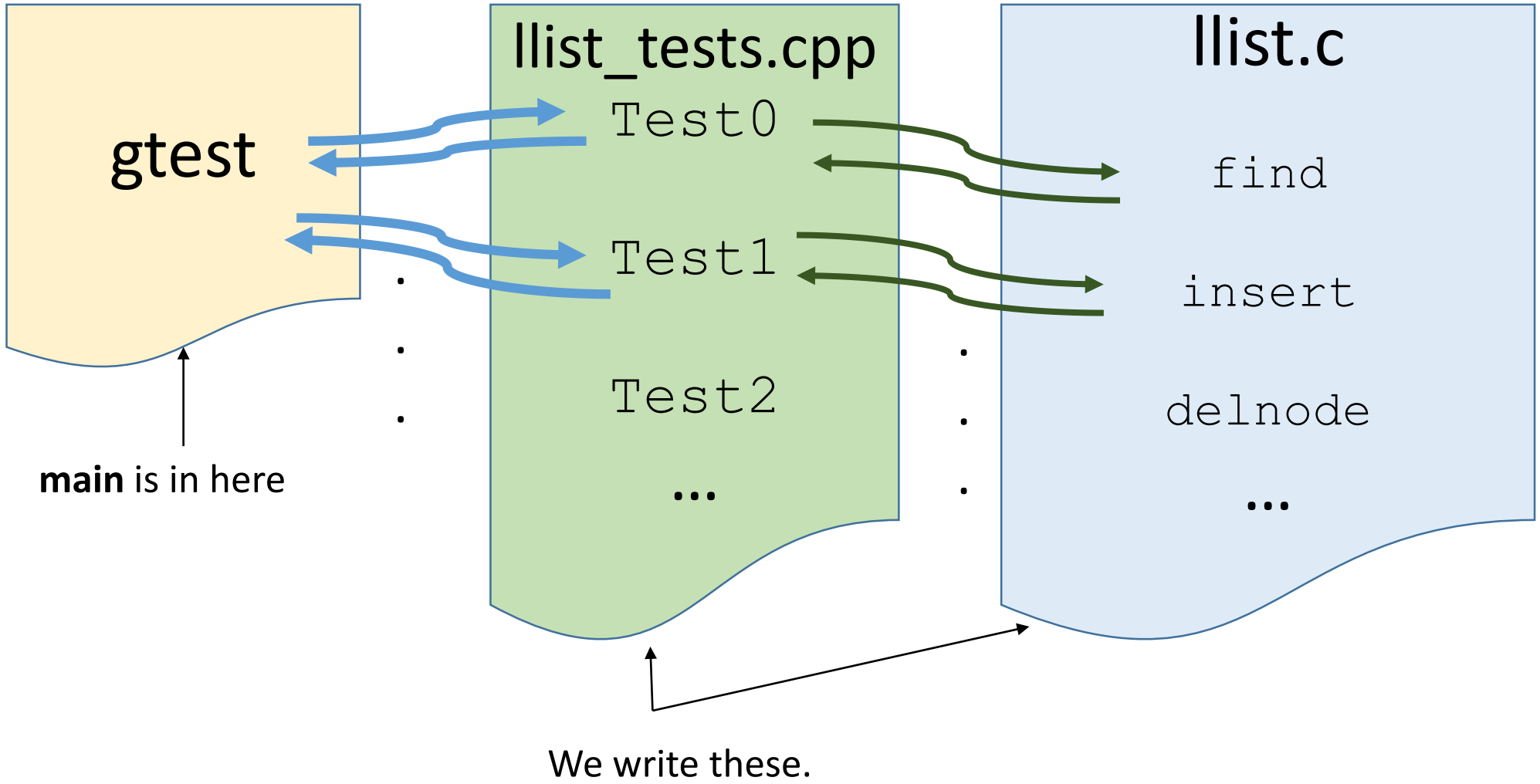
Delete Node(key) – lookup key: if found, splice out the node.



llist.h

```
typedef struct ll_node {
  int key;
  int data;
  struct ll_node * next;
} ll_node;

// global var holding pointer to head of list
ll_node * head;

// If key is in list return a ptr to node.  Otherwise return NULL.
ll_node * find(int key);

// If key is in list find and modify or if key is not in list allocate and insert
// a new ll_node into list. Return  ptr to that node. Return NULL if allocation fails.
ll_node * insert(int key, int data);

// Delete node with given key.  Return 0 if key is in list  and return -1 if not.
int delnode(int key);
```

Review: Unit Testing w/ Google Test Framework (gtest)



Review: Unit Testing Method

- Incremental Design and Test
  - Write Test before writing Code that satisfies it
  - Test gives expectation for output
  - Run gtest: should fail Test
  - Write Code to make Test succeed
  - Write new Test2 that fails
  - Write Code to make Test2 succeed
  - … repeat

```
/////////////////
// Access tests
/////////////////
// 1
TEST(AccessTest, FindKey_EmptyList)
{
  head = NULL;

  // Test when list is empty.
  EXPECT_EQ(NULL, find(0));
  EXPECT_EQ(NULL, find(1));
  EXPECT_EQ(NULL, find(2));

}
```

```
Running main() from gtest_main.cc
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from AccessTest
[ RUN      ] AccessTest.FindKey_EmptyList
[       OK ] AccessTest.FindKey_EmptyList (0 ms)
[----------] 1 test from AccessTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (1 ms total)
[  PASSED  ] 1 test.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/class
list/lltest$
```

```
-\---  llist_tests.cpp    Bot L38     (C++/l Abbrev)
#include "llist.h"

#include <stdlib.h>   // For malloc and free
#include <stdio.h>    // For printf

ll_node * find(int key) {
}

ll_node * insert(int key, int data) {
}

// delete node with given key.  Return 0 if key is in list  and return -1 if not
int delnode (int key) {
}
```

Passed w/out writing any code!
Write another test…

```
////////////////////
// Insertion tests
////////////////////
// 2
TEST(InsertTest, InsertInto_EmptyLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);

  EXPECT_EQ(n, head);
  EXPECT_EQ(n->data, 5);
  EXPECT_EQ(n->key, 2);
  // cast NULL pointer as ptr to int to avoid type error
  EXPECT_EQ(NULL, (int *) n->next);

  free(n);
}
```

`insert` returns NULL

`n == NULL`

`n->data` is `(*n).data`

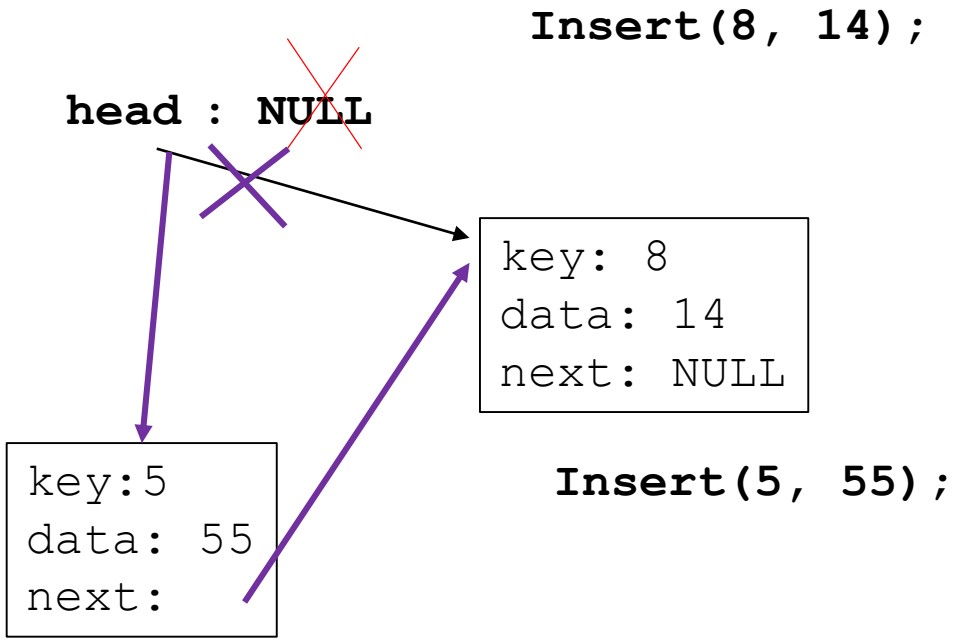
dereferencing a NULL pointer causes seg fault

linda@Sassafras: /mnt/c/Users/Linda Wills/Documents/classes/2035/lecture notes/heap-and-llist/linked-list/lltest

```
Running main() from gtest_main.cc
[==========] Running 2 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 1 test from AccessTest
[ RUN      ] AccessTest.FindKey_EmptyList
[       OK ] AccessTest.FindKey_EmptyList (0 ms)
[----------] 1 test from AccessTest (1 ms total)

[----------] 1 test from InsertTest
[ RUN      ] InsertTest.InsertInto_EmptyLL
Makefile:19: recipe for target 'test' failed
make: *** [test] Segmentation fault (core dumped)
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/classes/2
list/lltest$
```

Why?!!

Write code to allocate new node and insert it into an empty list… test passed!




```
ll_node * insert(int key, int data) {
  ll_node * this_node;
  this_node = (ll_node *) malloc(sizeof(ll_node));

  if (!this_node) return NULL; // no room in heap, allocation fails

  this_node->data = data;
  this_node->key = key;
  this_node->next = head;
  head = this_node;
  return this_node;
}
```

Typical Push on Head of List pattern

linda@Sassafras: /mnt/c/Users/Linda Wills/Documents/classes/2035/lecture notes/heap-and-llist/linked-list/lltest

```
Running main() from gtest_main.cc
[==========] Running 2 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 1 test from AccessTest
[ RUN      ] AccessTest.FindKey_EmptyList
[       OK ] AccessTest.FindKey_EmptyList (0 ms)
[----------] 1 test from AccessTest (0 ms total)

[----------] 1 test from InsertTest
[ RUN      ] InsertTest.InsertInto_EmptyLL
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms)
[----------] 1 test from InsertTest (1 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 2 test cases ran. (2 ms total)
[  PASSED  ] 2 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/classes/2035/l
```

Write test to **find** node inserted into an empty list …

```
TEST(AccessTest, FindKey_1EltLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = find(2);
  ll_node * p = find(4);

  // Test when list is empty.
  EXPECT_EQ(NULL, (int *) p);
  EXPECT_EQ(n, m);
  EXPECT_NE(n, p);
  EXPECT_NE(m, p);

  free(n);

}

//////////////////
// Insertion tests
//////////////////
// 2
TEST(InsertTest, InsertInto_EmptyLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);

  EXPECT_EQ(n, head);
  EXPECT_EQ(n->data, 5);
  EXPECT_EQ(n->key, 2);
  // cast NULL pointer as ptr to int
  EXPECT_EQ(NULL, (int *) n->next);

  free(n);
}
```

linda@Sassafras: /mnt/c/Users/Linda Wills/Documents/classes/2035/lecture notes/heap-and-llist/linked-list/lltest

```
[----------] 2 tests from AccessTest (1 ms total)

[----------] 1 test from InsertTest
[ RUN      ] InsertTest.InsertInto_EmptyLL
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms)
[----------] 1 test from InsertTest (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 2 test cases ran. (3 ms total)
[  PASSED  ] 2 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] AccessTest.FindKey_1EltLL

 1 FAILED TEST
Makefile:19: recipe for target 'test' failed
make: *** [test] Error 1
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/classes
```

Write code to search list for key…
test passed!

Typical Associative Search pattern

```
ll_node * find(int key) {
 ll_node * this_node = head;

 while(this_node){
    if (this_node->key == key) return this_node;
    this_node = this_node->next;
 }
 return NULL;  // not found
}
```

linda@Sassafras: /mnt/c/Users/Linda Wills/Documents/classes/2035/lecture notes/heap-and-llist/linked-list/lltest

```
[----------] Global test environment set-up.
[----------] 2 tests from AccessTest
[ RUN      ] AccessTest.FindKey_EmptyList
[       OK ] AccessTest.FindKey_EmptyList (0 ms)
[ RUN      ] AccessTest.FindKey_1EltLL
[       OK ] AccessTest.FindKey_1EltLL (0 ms)
[----------] 2 tests from AccessTest (1 ms total)

[----------] 1 test from InsertTest
[ RUN      ] InsertTest.InsertInto_EmptyLL
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms)
[----------] 1 test from InsertTest (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 2 test cases ran. (3 ms total)
[  PASSED  ] 3 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/classes/
list/lltest$
```

Write tests to **insert** node when

1. node w/ same key is already in list and
2. when not

```
TEST(InsertTest, InsertAsReplace_1EltLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(2, 7);

  EXPECT_EQ(n, head);
  EXPECT_EQ(n, m);
  EXPECT_EQ(n->key, 2);
  EXPECT_EQ(n->data, 7);

  free(n);

}

// 4
TEST(InsertTest, InsertNew_1EltLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);

  EXPECT_NE(n, head);
  EXPECT_NE(n, m);
  EXPECT_EQ(m, head);
  EXPECT_EQ(m->key, 3);
  EXPECT_EQ(m->data, 7);
  EXPECT_EQ(m->next, n);
  EXPECT_EQ(n->key, 2);
  EXPECT_EQ(n->data, 5);
  // cast NULL pointer as ptr to int to avoid type error
  EXPECT_EQ(NULL, (int *) n->next);

  free(m);
  free(n);
}
```

1. Failed: Test to **insert** node when node w/ same key is already in list (need to replace data)
2. Passed: when key is not already in list

```
[ RUN      ] InsertTest.InsertAsReplace_1EltLL
llist_tests.cpp:90: Failure
      Expected: n
      Which is: 0x15560b0
To be equal to: head
      Which is: 0x1556110
llist_tests.cpp:91: Failure
      Expected: n
      Which is: 0x15560b0
To be equal to: m
      Which is: 0x1556110
llist_tests.cpp:93: Failure
      Expected: n->data
      Which is: 5
To be equal to: 7
[  FAILED  ] InsertTest.InsertAsReplace_1EltLL (2 ms)
[ RUN      ] InsertTest.InsertNew_1EltLL
[       OK ] InsertTest.InsertNew_1EltLL (0 ms)
[----------] 3 tests from InsertTest (2 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 2 test cases ran. (4 ms total)
[  PASSED  ] 4 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] InsertTest.InsertAsReplace_1EltLL

 1 FAILED TEST
Makefile:19: recipe for target 'test' failed
make: *** [test] Error 1
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/classes
```

Add code to replace data when key exists

```
// IF KEY IS IN LIST FIND AND MODIFY
ll_node * insert(int key, int data){
    ll_node * this_node;
    // NEW STUFF:
    if ((this_node = find(key))){ //key
        this_node->data = data;
        return this_node;
    }

    // key is not in list
    this_node = (ll_node *) malloc(sizeo
    if (!this_node) return NULL; // no r

    this_node->data = data;
    this_node->key = key;
    this_node->next = head;
    head = this_node;

    return this_node;
}
```

```
[==========] Running 5 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from AccessTest
[ RUN      ] AccessTest.FindKey_EmptyList
[       OK ] AccessTest.FindKey_EmptyList (0 ms)
[ RUN      ] AccessTest.FindKey_1EltLL
[       OK ] AccessTest.FindKey_1EltLL (0 ms)
[----------] 2 tests from AccessTest (0 ms total)

[----------] 3 tests from InsertTest
[ RUN      ] InsertTest.InsertInto_EmptyLL
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_1EltLL
[       OK ] InsertTest.InsertAsReplace_1EltLL (0 m
[ RUN      ] InsertTest.InsertNew_1EltLL
[       OK ] InsertTest.InsertNew_1EltLL (0 ms)
[----------] 3 tests from InsertTest (1 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 2 test cases ran. (7 ms t
[  PASSED  ] 5 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/
```

Write tests to **insert** node when

1. node w/ same key is already in list and
2. when not

```
TEST(InsertTest, InsertAsReplace_1EltLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(2, 7);

  EXPECT_EQ(n, head);
  EXPECT_EQ(n, m);
  EXPECT_EQ(n->key, 2);
  EXPECT_EQ(n->data, 7);

  free(n);

}

// 4
TEST(InsertTest, InsertNew_1EltLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);

  EXPECT_NE(n, head);
  EXPECT_NE(n, m);
  EXPECT_EQ(m, head);
  EXPECT_EQ(m->key, 3);
  EXPECT_EQ(m->data, 7);
  EXPECT_EQ(m->next, n);
  EXPECT_EQ(n->key, 2);
  EXPECT_EQ(n->data, 5);
  // cast NULL pointer as ptr to int to avoid type error
  EXPECT_EQ(NULL, (int *) n->next);

  free(m);
  free(n);
}
```

```
./llist_tests
Running main() from gtest_main.cc
[==========] Running 7 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from AccessTest
[ RUN      ] AccessTest.FindKey_EmptyList
[       OK ] AccessTest.FindKey_EmptyList (0 ms)
[ RUN      ] AccessTest.FindKey_1EltLL
[       OK ] AccessTest.FindKey_1EltLL (0 ms)
[----------] 2 tests from AccessTest (6 ms total

[----------] 5 tests from InsertTest
[ RUN      ] InsertTest.InsertInto_EmptyLL
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms
[ RUN      ] InsertTest.InsertAsReplace_1EltLL
[       OK ] InsertTest.InsertAsReplace_1EltLL (
[ RUN      ] InsertTest.InsertNew_1EltLL
[       OK ] InsertTest.InsertNew_1EltLL (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_MidElt
[       OK ] InsertTest.InsertAsReplace_MidElt (
[ RUN      ] InsertTest.InsertAsReplace_TailElt
[       OK ] InsertTest.InsertAsReplace_TailElt
[----------] 5 tests from InsertTest (1 ms total

[----------] Global test environment tear-down
[==========] 7 tests from 2 test cases ran. (9 m
[  PASSED  ] 7 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documen
```

```
// 5
TEST(InsertTest, InsertAsReplace_MidElt)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);
  ll_node * p = insert(4, 8);
  ll_node * h = insert(3, 9);

  EXPECT_EQ(h, m);
  EXPECT_EQ(h->key, 3);
  EXPECT_EQ(h->data, 9);

  free(n);
  free(m);
  free(p);
  // why don't we free(h), too?

}
```

Test insert as replacement with key already present in middle and at end of list…

```
// 5
TEST(InsertTest, InsertAsReplace_TailElt)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);
  ll_node * p = insert(4, 8);
  ll_node * h = insert(4, 9);

  EXPECT_EQ(h, p);
  EXPECT_EQ(h->key, 4);
  EXPECT_EQ(h->data, 9);

  free(n);
  free(m);
  free(p);
}
```

These pass w/ no code change.

Test **find** on list of >1 node also passes w/out code change

```
[ RUN      ] AccessTest.FindKey_LongerLL
[       OK ] AccessTest.FindKey_LongerLL (0 ms)
[----------] 3 tests from AccessTest (1 ms total

[----------] 5 tests from InsertTest
[ RUN      ] InsertTest.InsertInto_EmptyLL
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms
[ RUN      ] InsertTest.InsertAsReplace_1EltLL
[       OK ] InsertTest.InsertAsReplace_1EltLL (
[ RUN      ] InsertTest.InsertNew_1EltLL
[       OK ] InsertTest.InsertNew_1EltLL (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_MidElt
[       OK ] InsertTest.InsertAsReplace_MidElt (
[ RUN      ] InsertTest.InsertAsReplace_TailElt
[       OK ] InsertTest.InsertAsReplace_TailElt
[----------] 5 tests from InsertTest (1 ms total

[----------] Global test environment tear-down
[==========] 8 tests from 2 test cases ran. (4 m
[  PASSED  ] 8 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documen
```

```
}

// 5

TEST(AccessTest, FindKey_LongerLL)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);
  ll_node * p = insert(4, 8);

  EXPECT_EQ(n, find(2));
  EXPECT_EQ(m, find(3));
  EXPECT_EQ(p, find(4));
  EXPECT_NE(n, m);
  EXPECT_NE(n, p);
  EXPECT_NE(m, p);
  EXPECT_EQ(NULL, (int *) find(5));

  free(p);
  free(m);
  free(n);

}

////////////////////
// Insertion tests
////////////////////
```

Deleting a node

Edge cases:

- List is empty
- Key not found
- Key found on head node
- Key found on a middle node
- Key found on the last node

Delete when list is empty

```
// 6
TEST(DeleteTest, Delete_w_EmptyLL)
{
  head = NULL;
  int result = delnode(2);

  EXPECT_EQ(result, -1);
  EXPECT_EQ(NULL, head);

}
```

```
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_1EltLL
[       OK ] InsertTest.InsertAsReplace_1EltLL (0
[ RUN      ] InsertTest.InsertNew_1EltLL
[       OK ] InsertTest.InsertNew_1EltLL (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_MidElt
[       OK ] InsertTest.InsertAsReplace_MidElt (0
[ RUN      ] InsertTest.InsertAsReplace_TailElt
[       OK ] InsertTest.InsertAsReplace_TailElt (
[----------] 5 tests from InsertTest (1 ms total)

[----------] 1 test from DeleteTest
[ RUN      ] DeleteTest.Delete_w_EmptyLL
llist_tests.cpp:191: Failure
      Expected: result
      Which is: 0
To be equal to: -1
[  FAILED  ] DeleteTest.Delete_w_EmptyLL (1 ms)
[----------] 1 test from DeleteTest (1 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 3 test cases ran. (4 ms
[  PASSED  ] 8 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] DeleteTest.Delete_w_EmptyLL

 1 FAILED TEST
Makefile:19: recipe for target 'test' failed
make: *** [test] Error 1
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents
```

Write code to make **delete** work w/ empty list

```
// delete node with given key.  Return 0 if key is in list  and return -1 if not
int delnode (int key) {
  ll_node * this_node = head;

  if (!this_node) return -1;

  return -1;
}
```

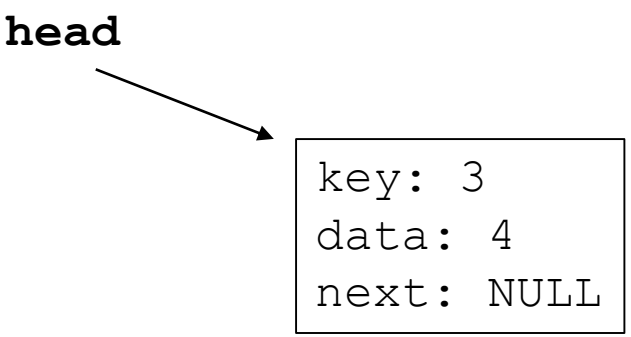
```
[----------] 1 test from DeleteTest
[ RUN      ] DeleteTest.Delete_w_EmptyLL
[       OK ] DeleteTest.Delete_w_EmptyLL (0 ms)
[----------] 1 test from DeleteTest (1 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 3 test cases ran. (3 ms
[  PASSED  ] 9 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents
```

Write tests to **`delete`** node when list has 1 elt and

1. key found (head node)
2. key not found

```
// 7
TEST(DeleteTest, Delete_1EltLL_Match)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  int result = delnode(2);

  EXPECT_EQ(result, 0);
  EXPECT_EQ(NULL, find(2));
  EXPECT_EQ(NULL, head);
  EXPECT_NE(head, n);
  //free(n); // already freed in delnode
}

// 7
TEST(DeleteTest, Delete_1EltLL_NoMatch)
{
  head = NULL;
  ll_node * n = insert(3, 5);
  int result = delnode(2);

  EXPECT_EQ(result, -1);
  EXPECT_EQ(n, find(3));

  free(n);
}
```

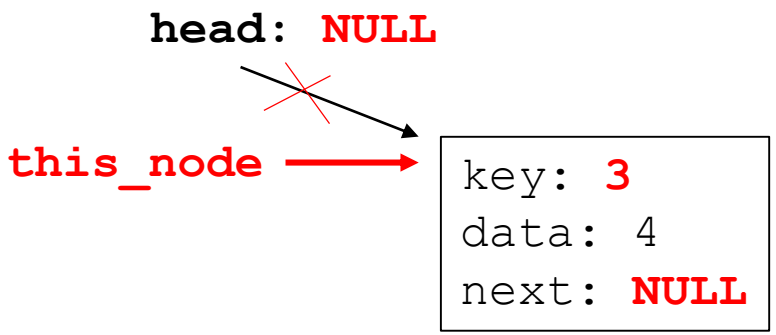
1. Failed: Test to **delete** node when list has 1 elt and key found (at head)
2. Passed: Test when 1elt and key not found




```
[ RUN      ] DeleteTest.Delete_1EltLL_Match
llist_tests.cpp:203: Failure
      Expected: result
      Which is: -1
To be equal to: 0
llist_tests.cpp:204: Failure
      Expected: __null
      Which is: NULL
To be equal to: find(2)
      Which is: 0x115b0b0
llist_tests.cpp:205: Failure
      Expected: __null
      Which is: NULL
To be equal to: head
      Which is: 0x115b0b0
llist_tests.cpp:206: Failure
Expected: (head) != (n), actual: 0x115b0b0 vs 0x
[  FAILED  ] DeleteTest.Delete_1EltLL_Match (2 m
[ RUN      ] DeleteTest.Delete_1EltLL_NoMatch
[       OK ] DeleteTest.Delete_1EltLL_NoMatch (0
[----------] 3 tests from DeleteTest (2 ms total

[----------] Global test environment tear-down
[==========] 11 tests from 3 test cases ran. (7
[  PASSED  ] 10 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] DeleteTest.Delete_1EltLL_Match

 1 FAILED TEST
Makefile:19: recipe for target 'test' failed
make: *** [test] Error 1
```

Write code to make test pass

```
result = delnode(3);
```




```
// delete node with given key.  Return 0 if key is in list  and return -1 if not
int delnode (int key) {
  ll_node * this_node = head;

  if (!this_node) return -1;

  // new stuff
  if (this_node->key == key) {
      head = this_node->next;
      free(this_node);
      return 0;
  }

  return -1;
}
```

```
[----------] 3 tests from DeleteTest
[ RUN      ] DeleteTest.Delete_w_EmptyLL
[       OK ] DeleteTest.Delete_w_EmptyLL (0 ms)
[ RUN      ] DeleteTest.Delete_1EltLL_Match
[       OK ] DeleteTest.Delete_1EltLL_Match (0 ms)
[ RUN      ] DeleteTest.Delete_1EltLL_NoMatch
[       OK ] DeleteTest.Delete_1EltLL_NoMatch (0 ms)
[----------] 3 tests from DeleteTest (1 ms total)

[----------] Global test environment tear-down
[==========] 11 tests from 3 test cases ran. (7 ms total)
[  PASSED  ] 11 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/classes/
```

Write test to **`delete`** node when list has >1 elt and

1. key found at head
2. key found in middle/tail
3. key not found

```
TEST(DeleteTest, Delete_MultiEltLL_NoMatch)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);
  ll_node * p = insert(4, 8);
  int result = delnode(1);

  EXPECT_EQ(result, -1);
  EXPECT_EQ(head, p);
  EXPECT_EQ(n, find(2));
  EXPECT_EQ(m, find(3));
  EXPECT_EQ(p, find(4));

  free(n);
  free(m);
  free(p);
}
```

```
TEST(DeleteTest, Delete_MultiEltLL_Head)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);
  ll_node * p = insert(4, 8);
  int result = delnode(4);

  EXPECT_EQ(result, 0);
  EXPECT_EQ(head, m);
  EXPECT_EQ(n, find(2));
  EXPECT_EQ(m, find(3));
  EXPECT_EQ(NULL, find(4));
  EXPECT_NE(head, p);

  free(n);  free(m);
  //free(p); // already freed in delnode
}

// 8
TEST(DeleteTest, Delete_MultiEltLL_NonHead)
{
  head = NULL;
  ll_node * n = insert(2, 5);
  ll_node * m = insert(3, 7);
  ll_node * p = insert(4, 8);
  int result = delnode(3);

  EXPECT_EQ(result, 0);
  EXPECT_EQ(head, p);
  EXPECT_EQ(n, find(2));
  EXPECT_EQ(NULL, find(3));
  EXPECT_EQ(p, find(4));
  EXPECT_NE(head, m);

  free(n); free(p);  //free(m); // already f
```

Test to **delete** node when list has >1 elt

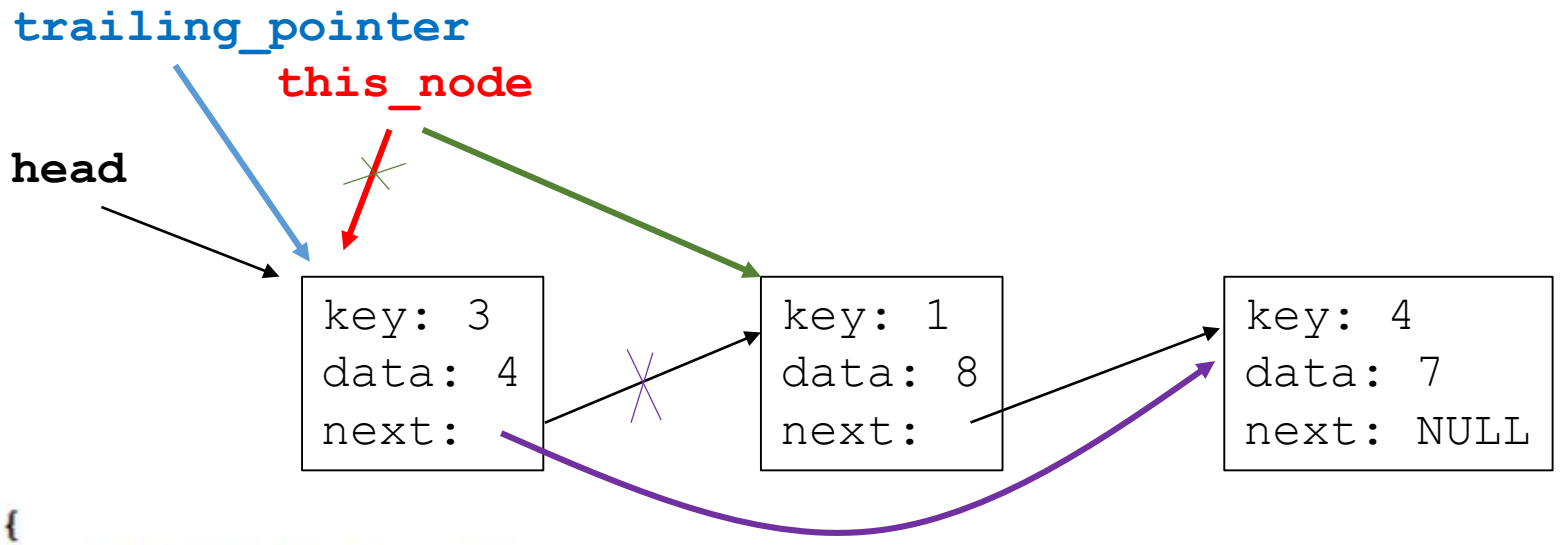
1. Passed: key found at head
2. Failed: key found in middle/tail
3. Passed: key not found

```
[       OK ] DeleteTest.Delete_w_EmptyLL (0 ms)
[ RUN      ] DeleteTest.Delete_1EltLL_Match
[       OK ] DeleteTest.Delete_1EltLL_Match (0 ms)
[ RUN      ] DeleteTest.Delete_1EltLL_NoMatch
[       OK ] DeleteTest.Delete_1EltLL_NoMatch (0 ms)
[ RUN      ] DeleteTest.Delete_MultiEltLL_Head
[       OK ] DeleteTest.Delete_MultiEltLL_Head (0 ms)
[ RUN      ] DeleteTest.Delete_MultiEltLL_NonHead
llist_tests.cpp:252: Failure
      Expected: result
      Which is: -1
To be equal to: 0
llist_tests.cpp:255: Failure
      Expected: __null
      Which is: NULL
To be equal to: find(3)
      Which is: 0x832120
[  FAILED  ] DeleteTest.Delete_MultiEltLL_NonHead (0 ms)
[ RUN      ] DeleteTest.Delete_MultiEltLL_NoMatch
[       OK ] DeleteTest.Delete_MultiEltLL_NoMatch (0 ms)
[----------] 6 tests from DeleteTest (2 ms total)

[----------] Global test environment tear-down
[==========] 14 tests from 3 test cases ran. (11 ms total)
[  PASSED  ] 13 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] DeleteTest.Delete_MultiEltLL_NonHead

 1 FAILED TEST
Makefile:19: recipe for target 'test' failed
make: *** [test] Error 1
```

3-Node List Example

Add code to **`delete`** node when list has >1 elt and key found in middle/tail - e.g., **`delete(1)`**

```
// delete node with given key.  Return 0 if key
int delnode (int key) {
  ll_node * this_node = head;
  ll_node * trailing_pointer;

  if (!this_node) return -1;

  if (this_node->key == key) {
      head = this_node->next;
      free(this_node);
      return 0;
  }
  trailing_pointer = this_node;
  this_node = this_node->next;
  while (this_node){
      if (this_node->key == key){
          trailing_pointer->next = this_node->next;
          free(this_node);
          return 0;
      }
      trailing_pointer = this_node;
      this_node = this_node->next;
  }
  return -1;
}
```

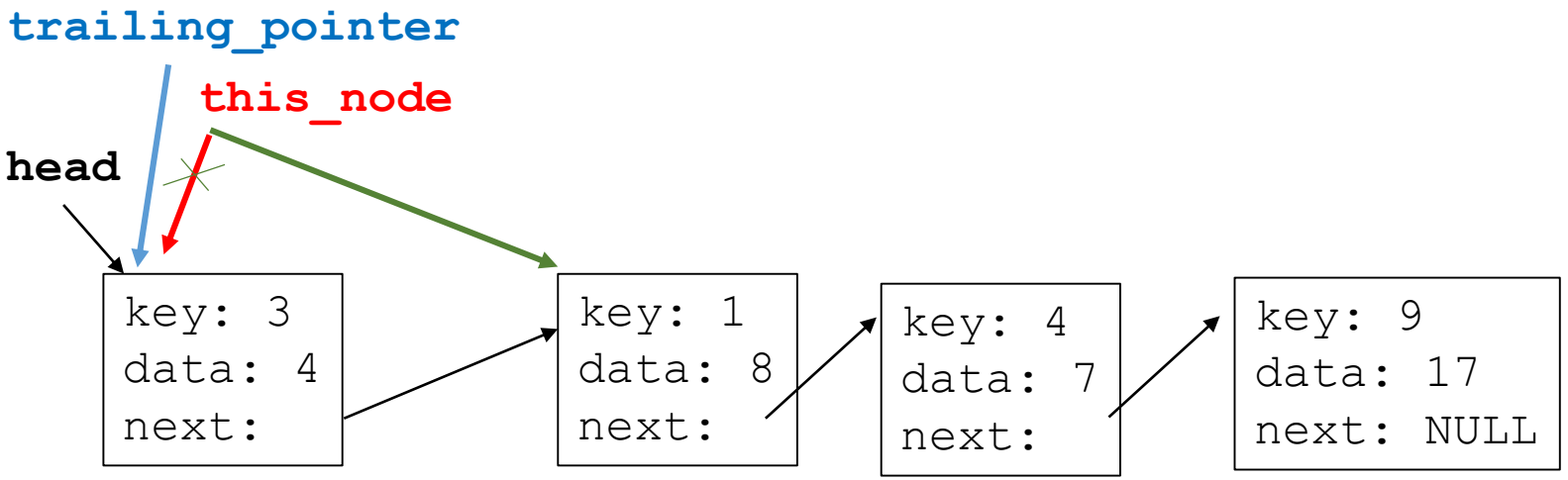


Typical Splice out pattern

Different Example

Add code to **`delete`** node when list has >1 elt and key found in middle/tail - e.g., **`delete(4)`**

```
// delete node with given key.  Return 0 if key is
int delnode (int key) {
  ll_node * this_node = head;
  ll_node * trailing_pointer;

  if (!this_node) return -1;

  if (this_node->key == key) {
      head = this_node->next;
      free(this_node);
      return 0;
  }
  trailing_pointer = this_node;
  this_node = this_node->next;
  while (this_node){
      if (this_node->key == key){
          trailing_pointer->next = this_node->next;
          free(this_node);
          return 0;
      }
      trailing_pointer = this_node;
      this_node = this_node->next;
  }
  return -1;
}
```

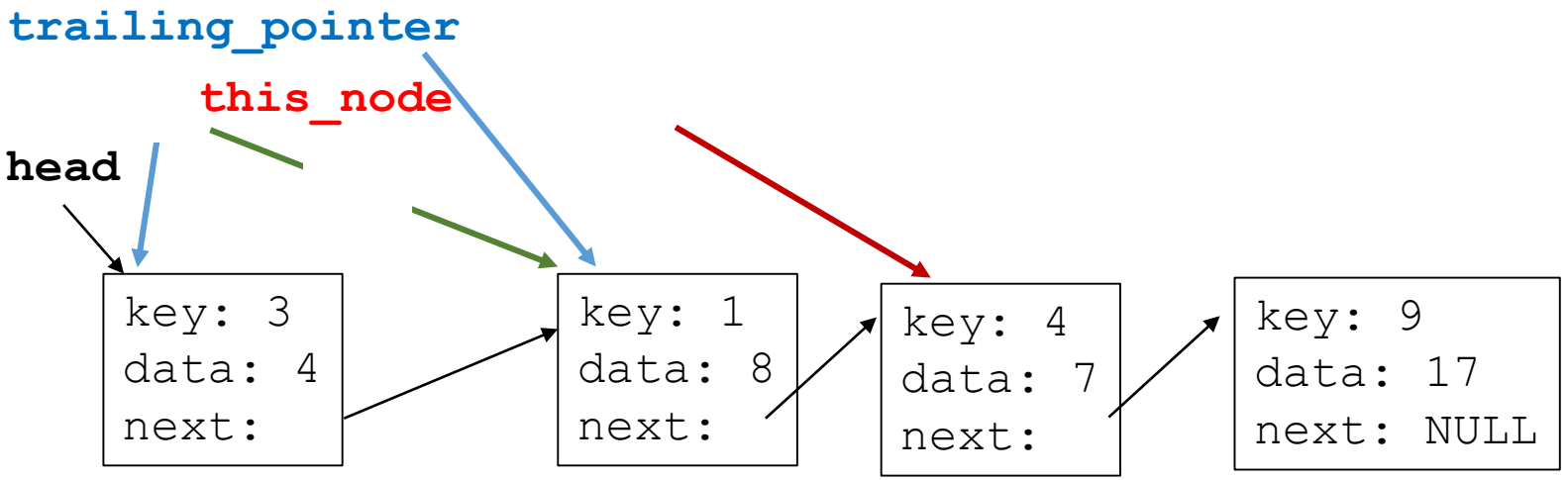


Typical Splice out pattern

Different Example

Add code to **delete** node when list has >1 elt and key found in middle/tail - e.g., **delete(4)**

```
// delete node with given key.  Return 0 if key is
int delnode (int key) {
  ll_node * this_node = head;
  ll_node * trailing_pointer;

  if (!this_node) return -1;

  if (this_node->key == key) {
      head = this_node->next;
      free(this_node);
      return 0;
  }
  trailing_pointer = this_node;
  this_node = this_node->next;
  while (this_node){
      if (this_node->key == key){
          trailing_pointer->next = this_node->next;
          free(this_node);
          return 0;
      }
      trailing_pointer = this_node;
      this_node = this_node->next;
  }
  return -1;
}
```

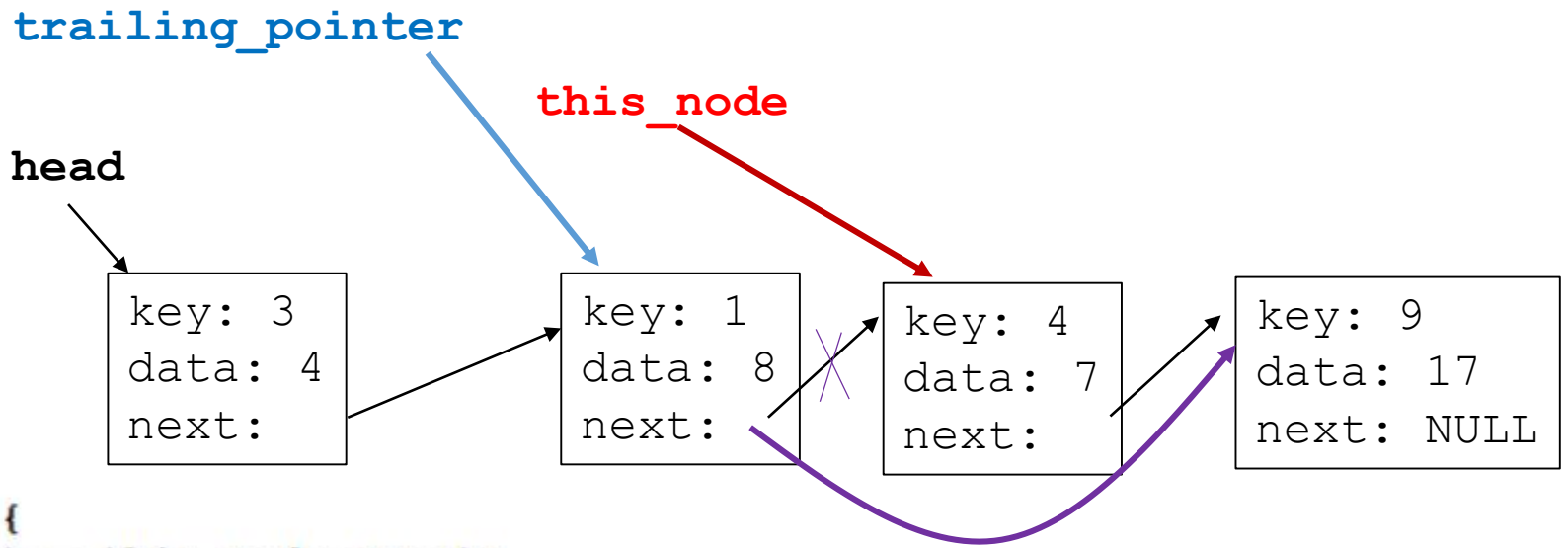


Typical Splice out pattern

Different Example

Add code to **`delete`** node when list has >1 elt and key found in middle/tail - e.g., **`delete(4)`**

```
// delete node with given key.  Return 0 if key is
int delnode (int key) {
  ll_node * this_node = head;
  ll_node * trailing_pointer;

  if (!this_node) return -1;

  if (this_node->key == key) {
      head = this_node->next;
      free(this_node);
      return 0;
  }
  trailing_pointer = this_node;
  this_node = this_node->next;
  while (this_node){
      if (this_node->key == key){
          trailing_pointer->next = this_node->next;
          free(this_node);
          return 0;
      }
      trailing_pointer = this_node;
      this_node = this_node->next;
  }
  return -1;
}
```



Typical Splice out pattern

Success!

```
[       OK ] InsertTest.InsertInto_EmptyLL (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_1EltLL
[       OK ] InsertTest.InsertAsReplace_1EltLL (0 ms)
[ RUN      ] InsertTest.InsertNew_1EltLL
[       OK ] InsertTest.InsertNew_1EltLL (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_MidElt
[       OK ] InsertTest.InsertAsReplace_MidElt (0 ms)
[ RUN      ] InsertTest.InsertAsReplace_TailElt
[       OK ] InsertTest.InsertAsReplace_TailElt (0 ms)
[----------] 5 tests from InsertTest (2 ms total)

[----------] 6 tests from DeleteTest
[ RUN      ] DeleteTest.Delete_w_EmptyLL
[       OK ] DeleteTest.Delete_w_EmptyLL (0 ms)
[ RUN      ] DeleteTest.Delete_1EltLL_Match
[       OK ] DeleteTest.Delete_1EltLL_Match (0 ms)
[ RUN      ] DeleteTest.Delete_1EltLL_NoMatch
[       OK ] DeleteTest.Delete_1EltLL_NoMatch (0 ms)
[ RUN      ] DeleteTest.Delete_MultiEltLL_Head
[       OK ] DeleteTest.Delete_MultiEltLL_Head (0 ms)
[ RUN      ] DeleteTest.Delete_MultiEltLL_NonHead
[       OK ] DeleteTest.Delete_MultiEltLL_NonHead (0 ms)
[ RUN      ] DeleteTest.Delete_MultiEltLL_NoMatch
[       OK ] DeleteTest.Delete_MultiEltLL_NoMatch (0 ms)
[----------] 6 tests from DeleteTest (1 ms total)

[----------] Global test environment tear-down
[==========] 14 tests from 3 test cases ran. (9 ms total)
[  PASSED  ] 14 tests.
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents/classes/20
```