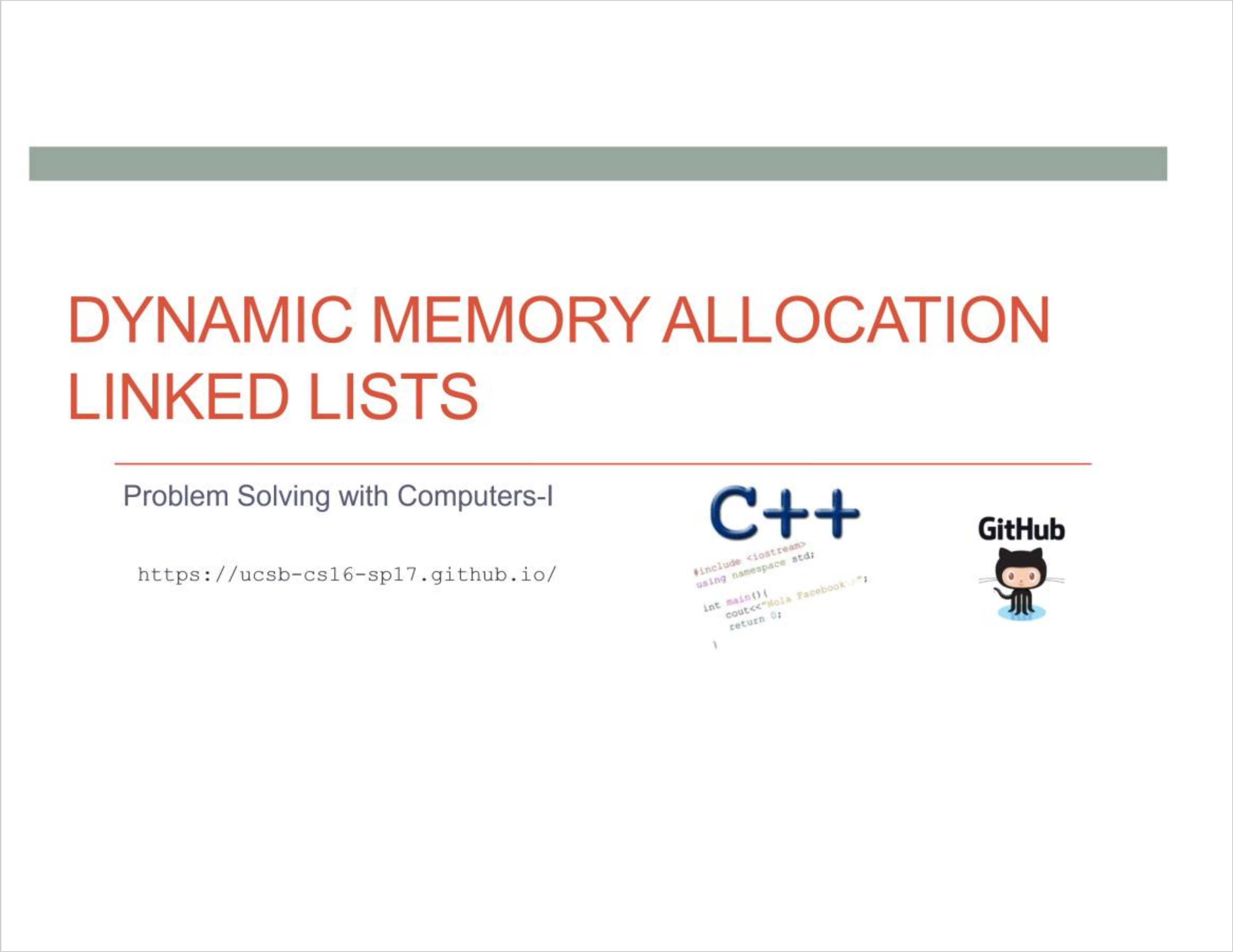
Lecture 12

Sunday, May 14, 2017 6:05 PM



DYNAMIC MEMORY ALLOCATION
LINKED LISTS

Problem Solving with Computers-I

https://ucsb-cs16-sp17.github.io/

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook :)";
    return 0;
}
```

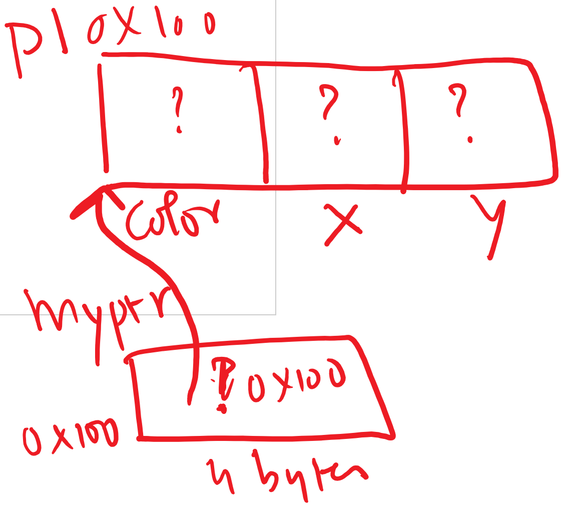
Review: Structs, arrays of structs

```
struct Point {
    string color;
    double x;
    double y;
};
```

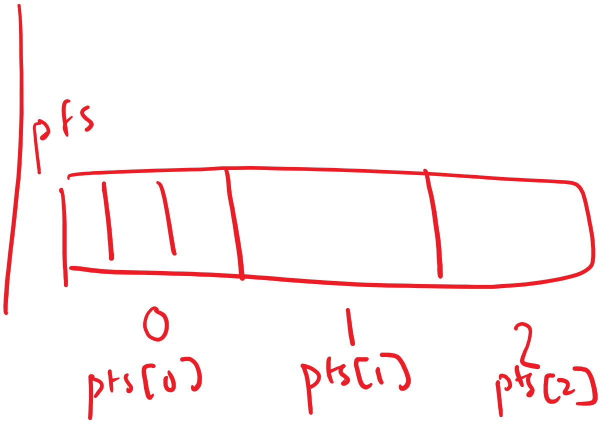
```
Point    p1;

Point * myptr;
```

```
p1.color = "red";
p1.x = 100;
myptr = &p1;
```

```
cout<< (*myptr).x;

cout << myptr->x;
```

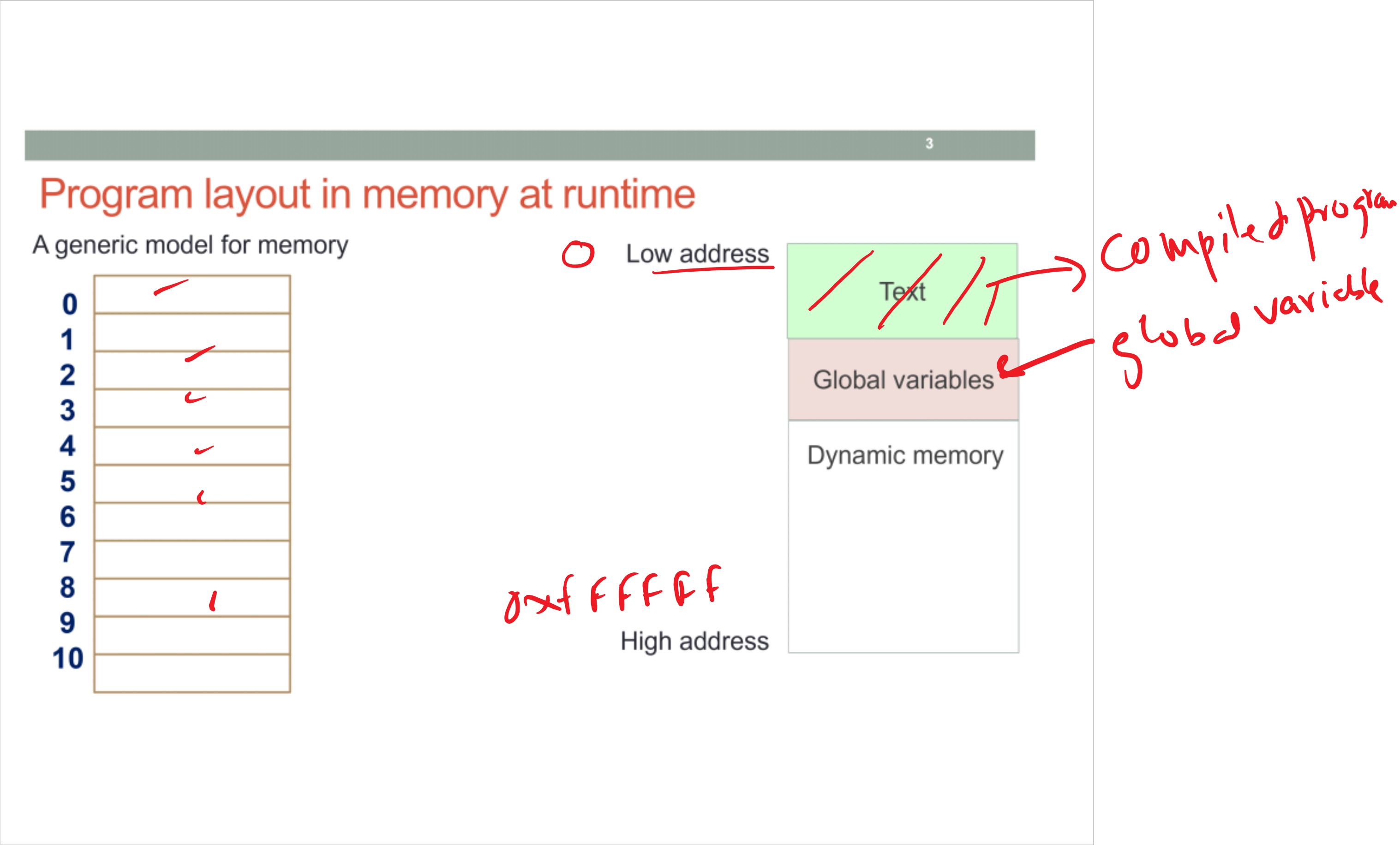


```
Point pts[3];
```



Page 3

Sunday, May 14, 2017 6:05 PM



Program layout in memory at runtime

A generic model for memory

0
1
2
3
4
5
6
7
8
9
10

0 Low address

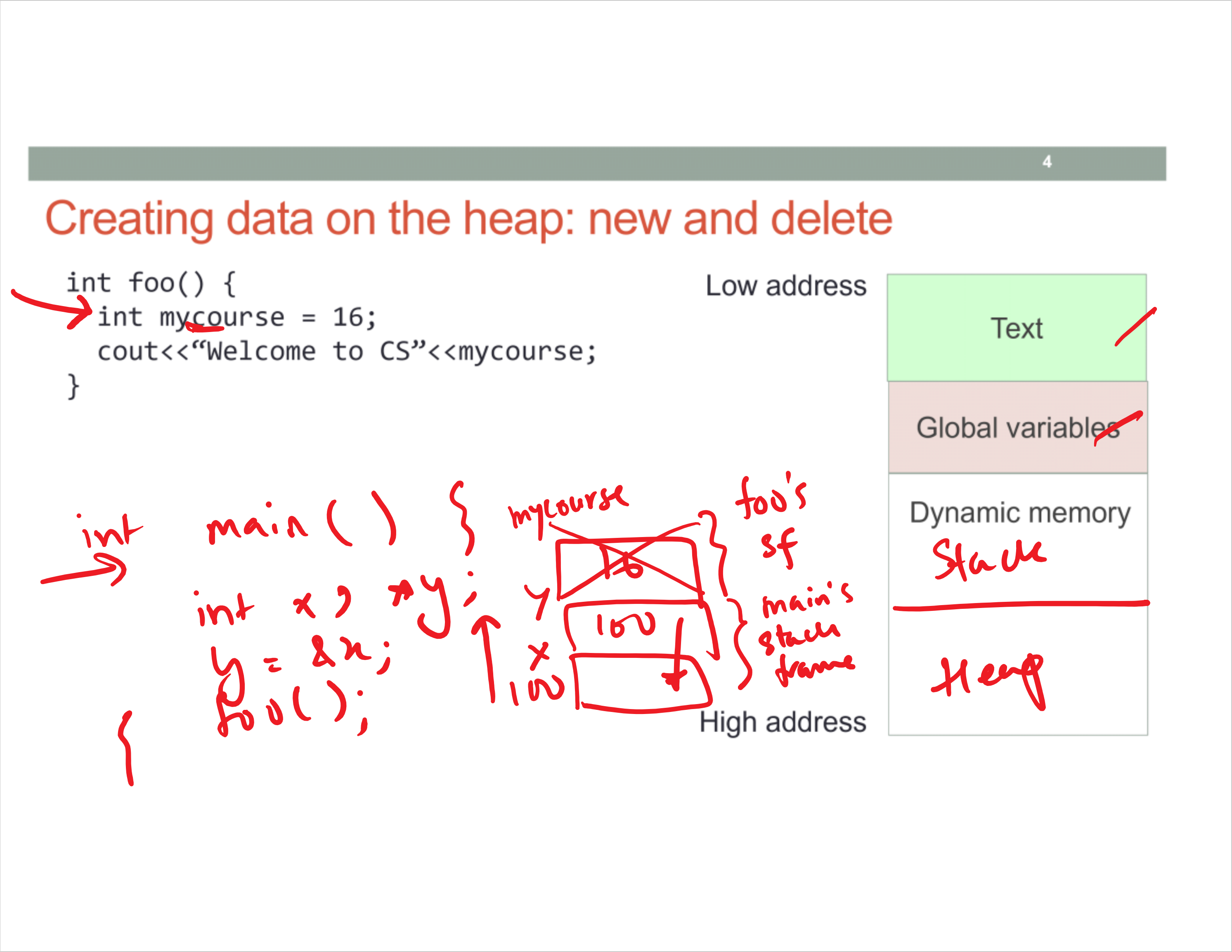
Text → Compiled program

Global variables ← global variable

Dynamic memory

0xfFFFFF

High address

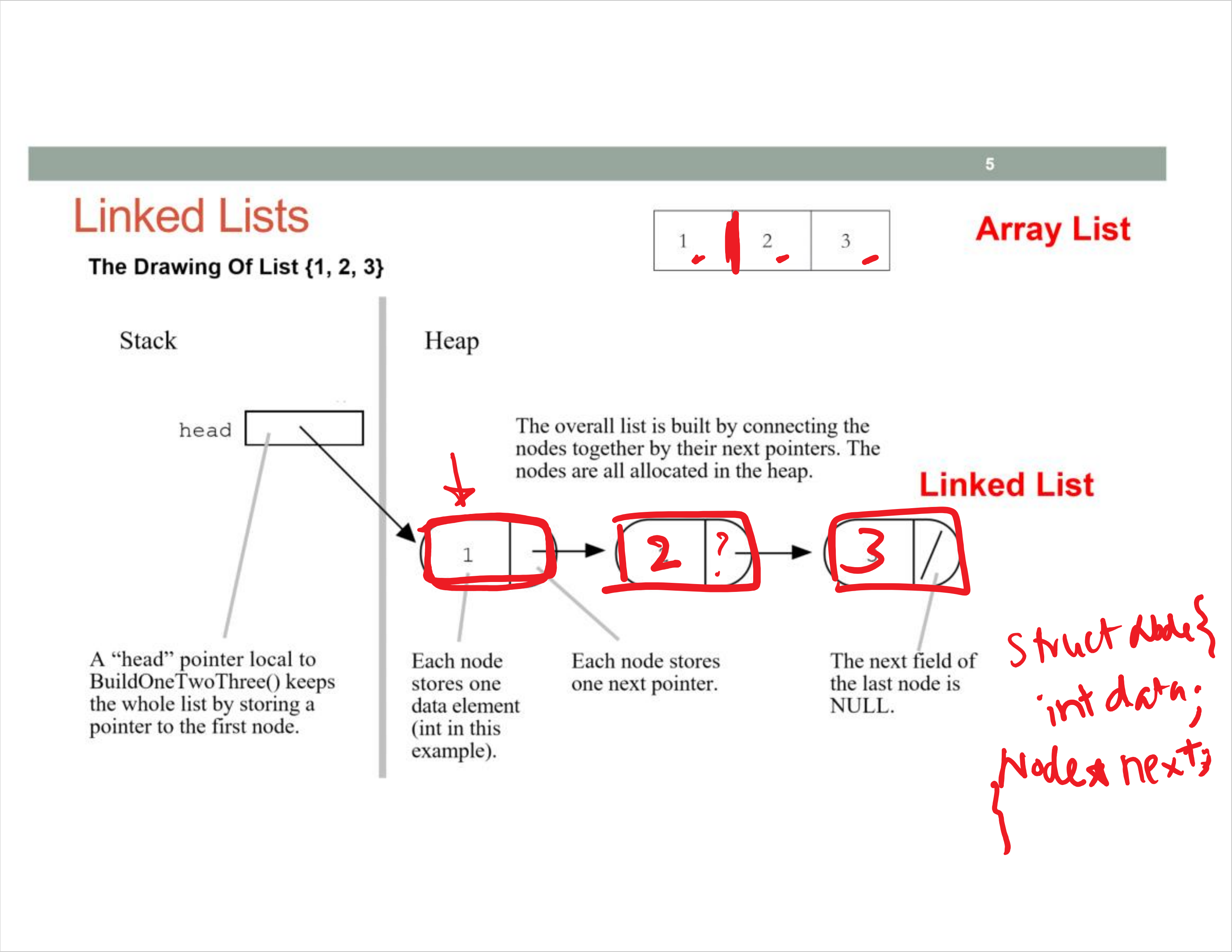


Creating data on the heap: new and delete

```
int foo() {
  int mycourse = 16;
  cout<<"Welcome to CS"<<mycourse;
}
```

```
int main() {
  int x, *y;
  y = &x;
  foo();
}
```

Sunday, May 14, 2017 6:05 PM



Linked Lists

The Drawing Of List {1, 2, 3}

The overall list is built by connecting the nodes together by their next pointers. The nodes are all allocated in the heap.

A "head" pointer local to BuildOneTwoThree() keeps the whole list by storing a pointer to the first node.

Each node stores one data element (int in this example).

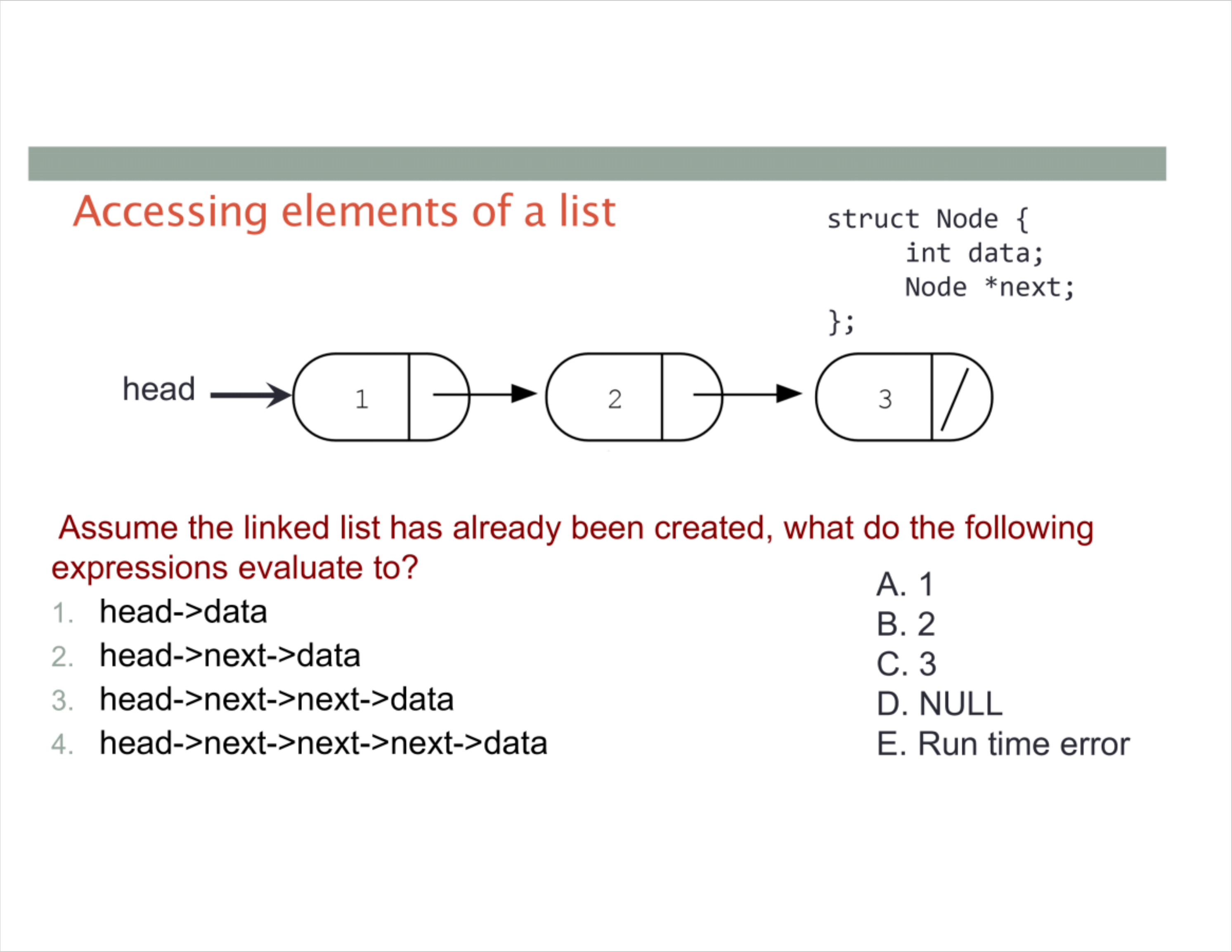
Each node stores one next pointer.

The next field of the last node is NULL.

```
struct Node{
int data;
Node* next;
}
```

Page 6

Sunday, May 14, 2017 6:05 PM



Accessing elements of a list

```
struct Node {
     int data;
     Node *next;
};
```

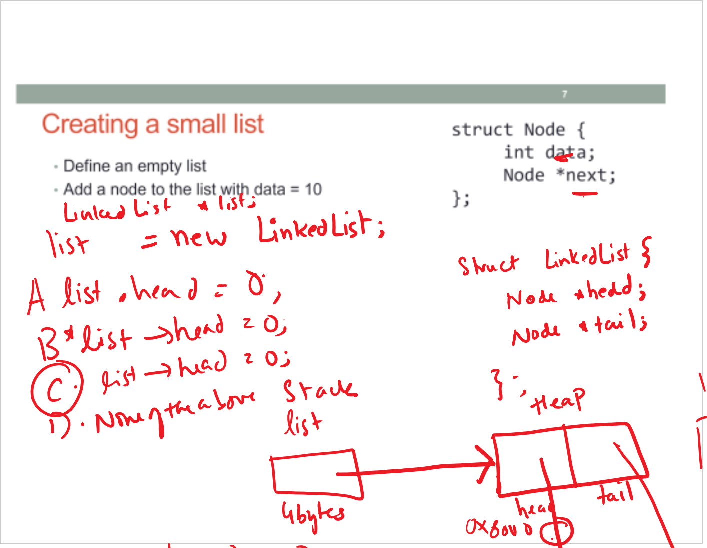
Assume the linked list has already been created, what do the following expressions evaluate to?

1. head->data
2. head->next->data
3. head->next->next->data
4. head->next->next->next->data

A. 1
B. 2
C. 3
D. NULL
E. Run time error

Page 7

Sunday, May 14, 2017 6:05 PM



7

Creating a small list

- Define an empty list
- Add a node to the list with data = 10

```
LinkedList *list;
list = new LinkedList;
```

A list.head = 0;

B *list->head = 0;

[✓ C.] list->head = 0;

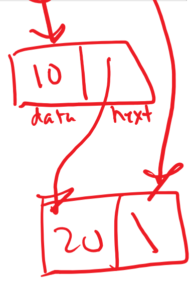
D. None of the above

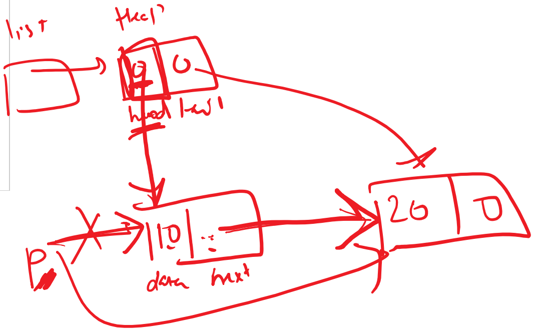
```
struct Node {
      int data;
      Node *next;
};
```

```
struct LinkedList {
      Node *head;
      Node *tail;
};
```

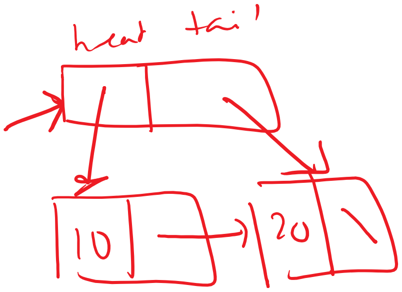
Stack
list
4bytes
heap
head
tail
0x8000
0

```
list->head = 0;
list->tail = 0;
```





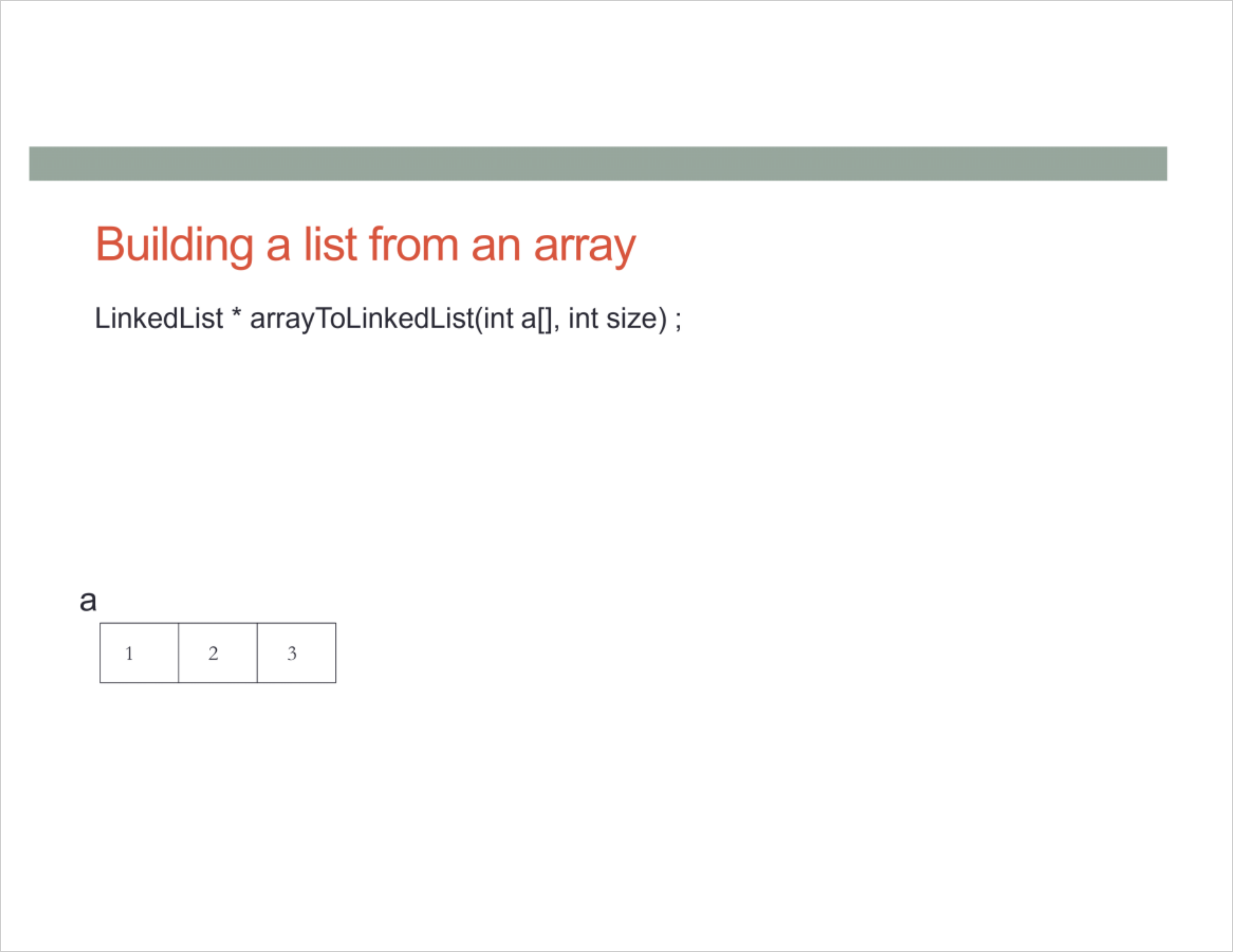
```
Node *p = new Node;
p->data = 10;
p->next = 0;
list->head = p;
list->tail = p;
```



```
p = new Node
p->data = 20;
p->next = 0;
list->head->next = p;
list->tail = p;
```

Page 8

Sunday, May 14, 2017 6:06 PM



Building a list from an array

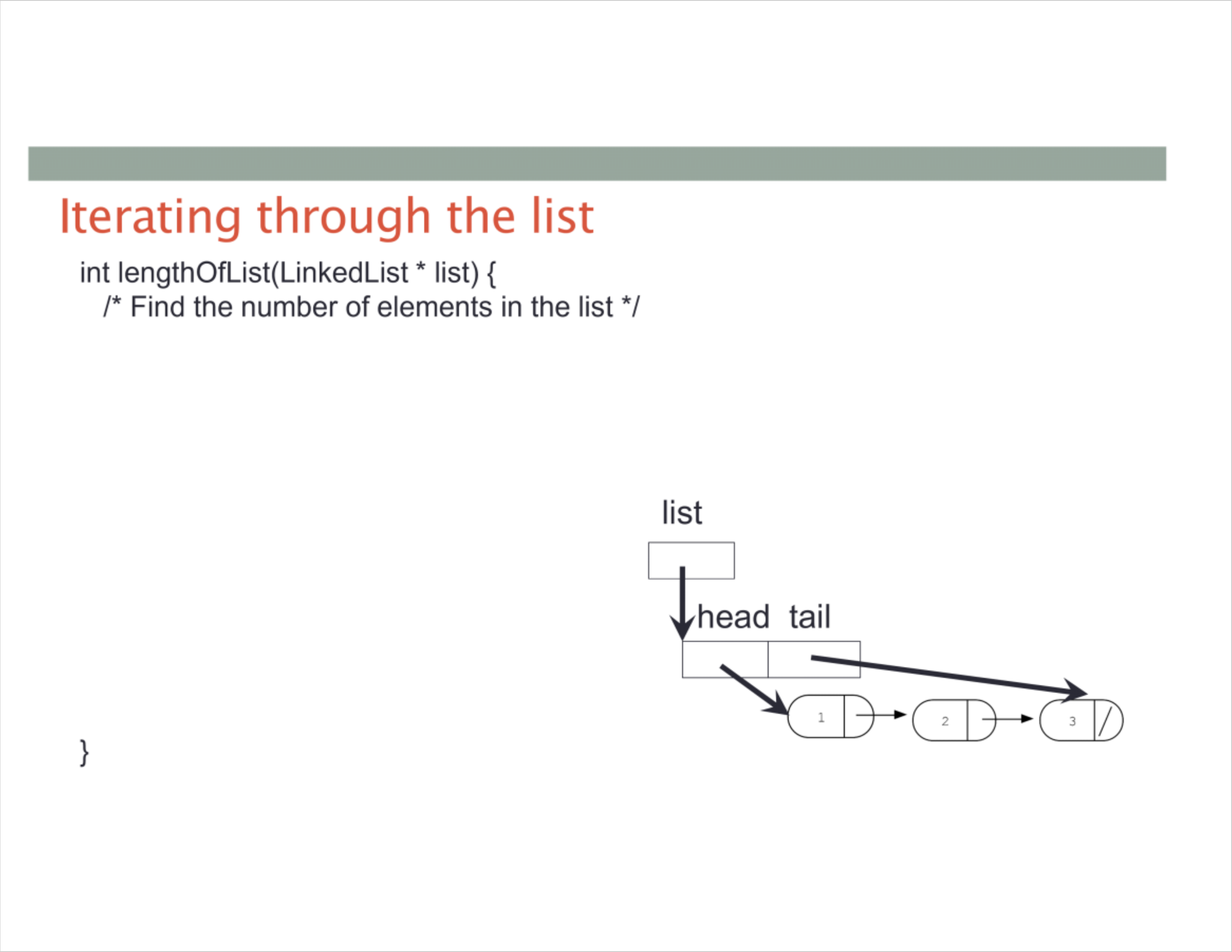
LinkedList * arrayToLinkedList(int a[], int size) ;

a

1	2	3

Page 9

Sunday, May 14, 2017 6:06 PM

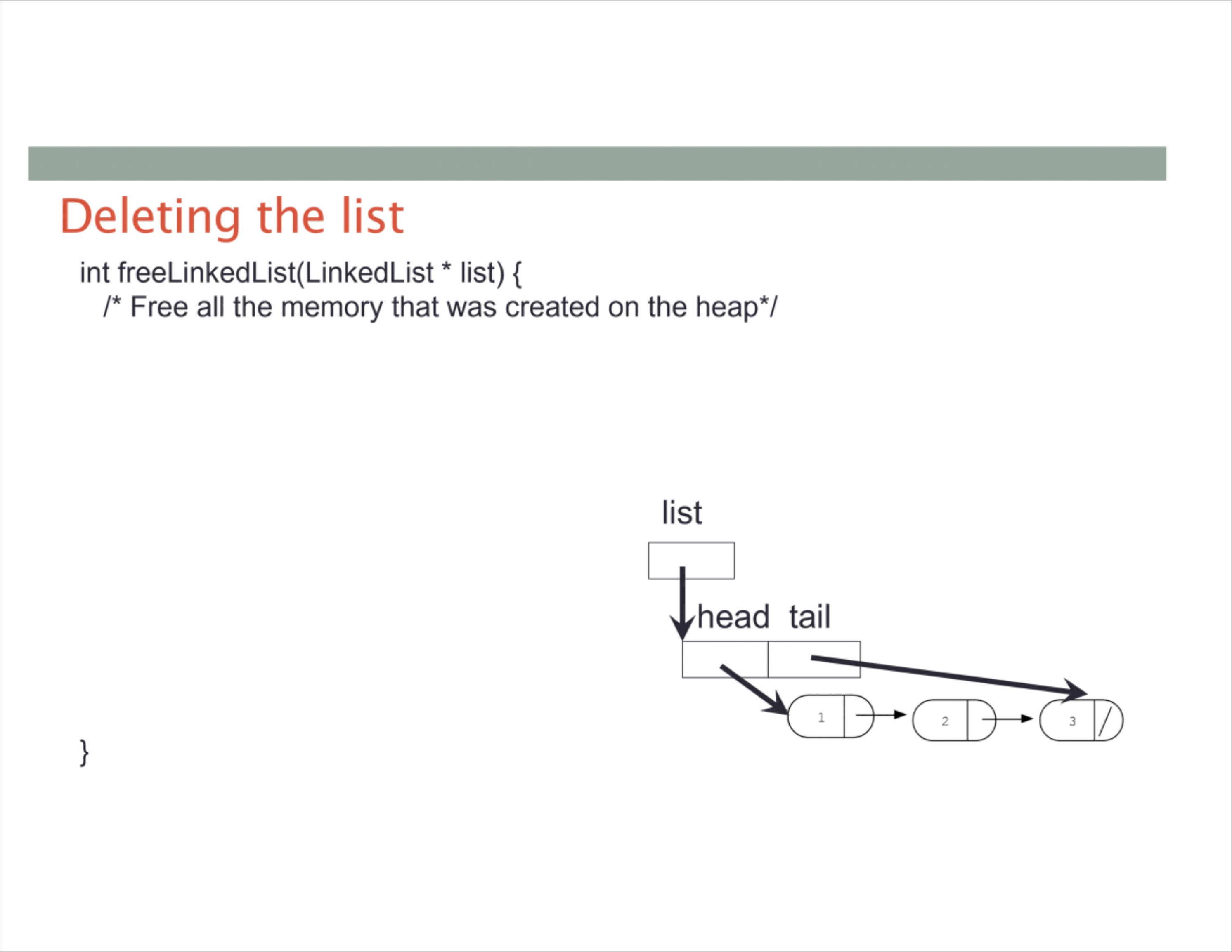


Iterating through the list

```
int lengthOfList(LinkedList * list) {
  /* Find the number of elements in the list */









}
```

Page 10

Sunday, May 14, 2017 6:06 PM

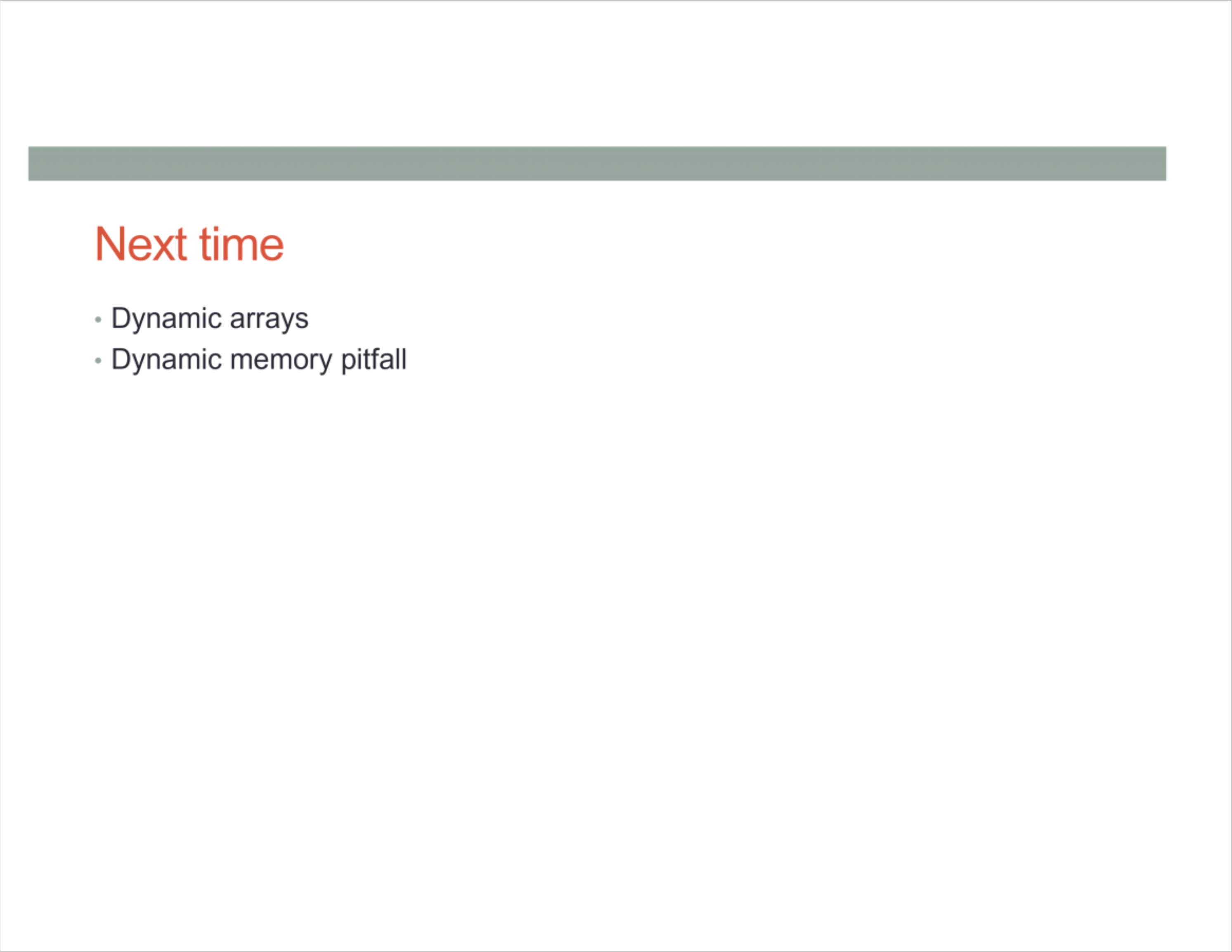


Deleting the list

```
int freeLinkedList(LinkedList * list) {
   /* Free all the memory that was created on the heap*/




}
```

Page 11

Sunday, May 14, 2017 6:06 PM



Next time

- Dynamic arrays
- Dynamic memory pitfall