

DYNAMIC MEMORY ALLOCATION LINKED LISTS

Problem Solving with Computers-I

<https://ucsb-cs16-wi17.github.io/>

C++

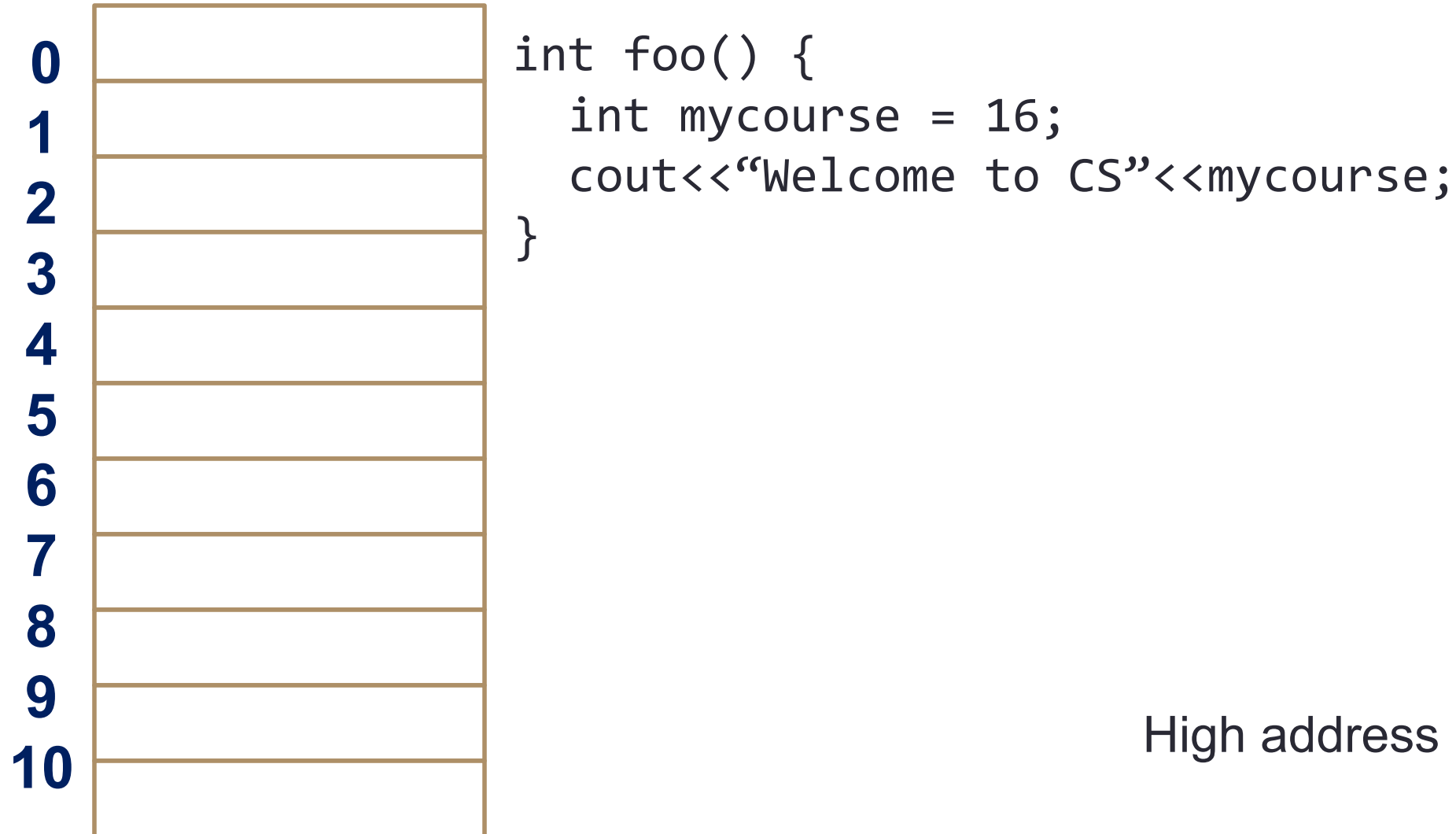
```
#include <iostream>
using namespace std;

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

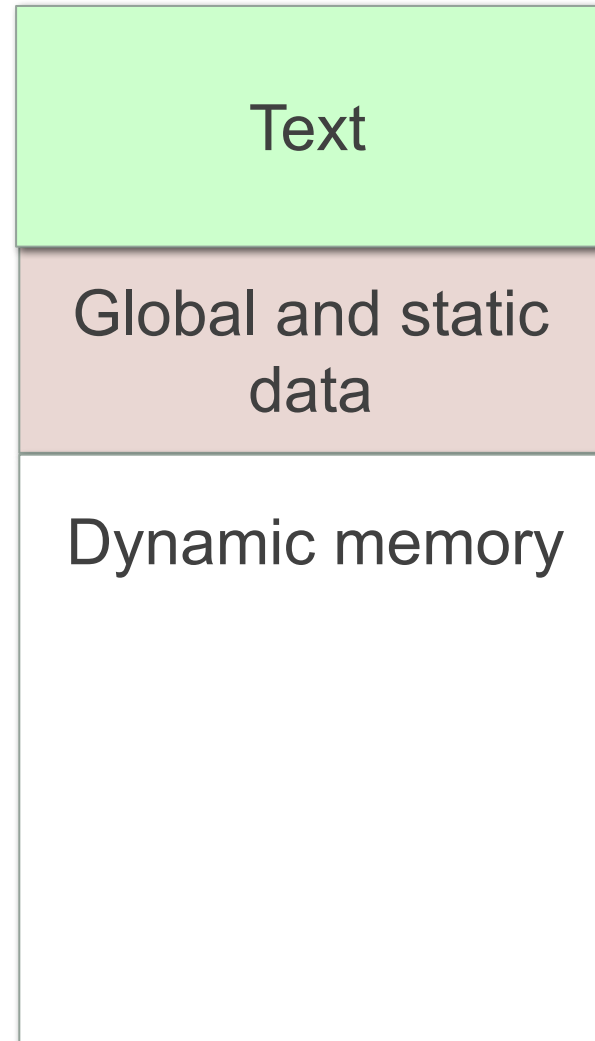


Program layout in memory at runtime

A generic model for memory



Low address



High address

Creating data on the heap: new and delete

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

Low address

Text

Global and static
data

Dynamic memory

High address

Linked Lists

The Drawing Of List {1, 2, 3}

1	2	3
---	---	---

Array List

Stack

Heap

head

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

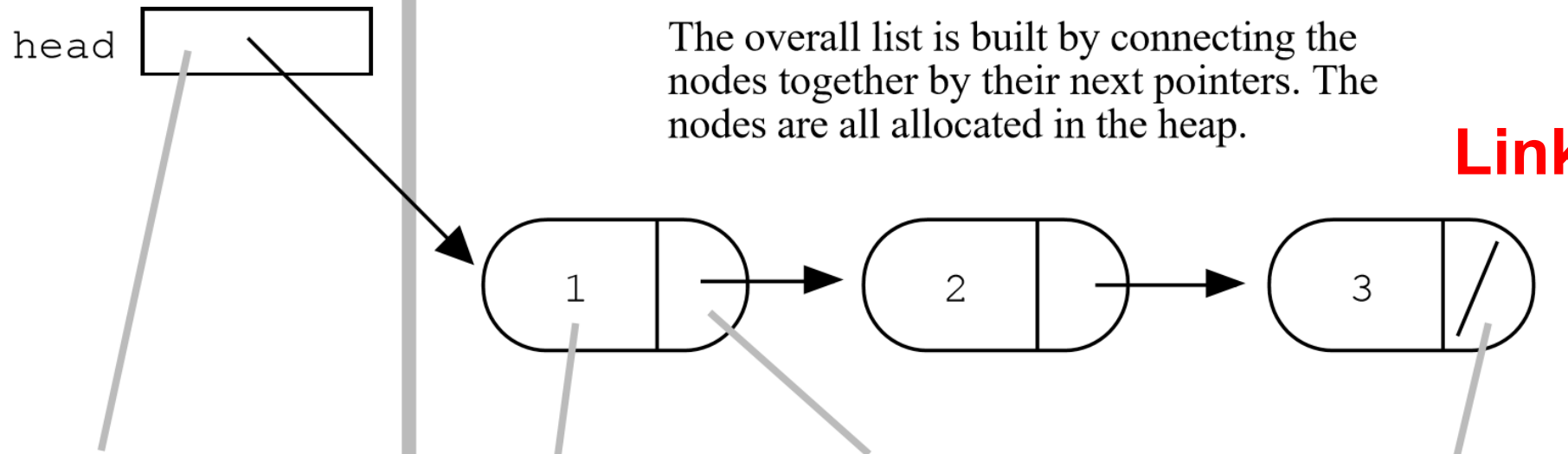
Linked List

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).

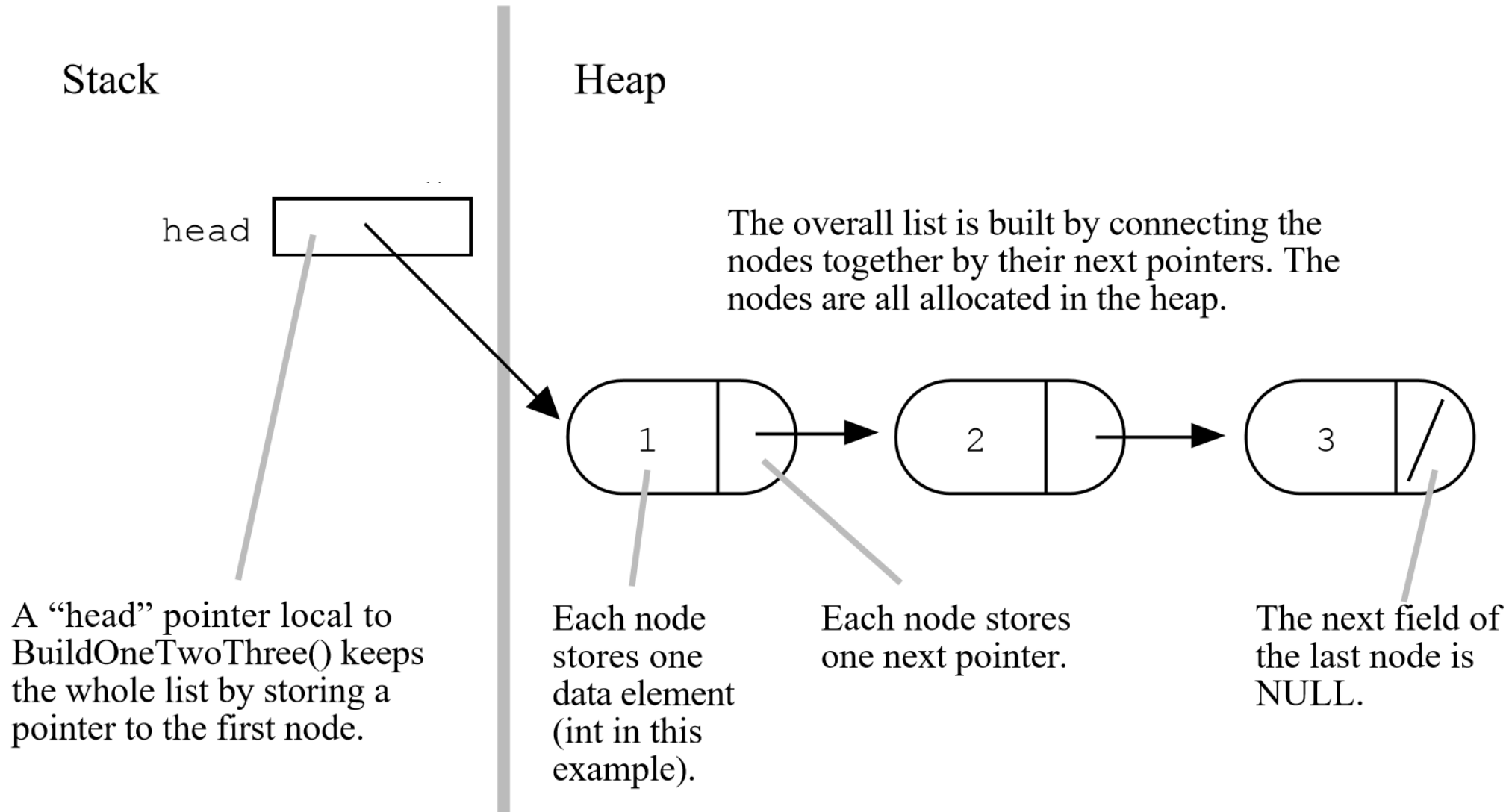
Each node stores one next pointer.

The next field of the last node is NULL.



Declaring a node in the list

The Drawing Of List {1, 2, 3}



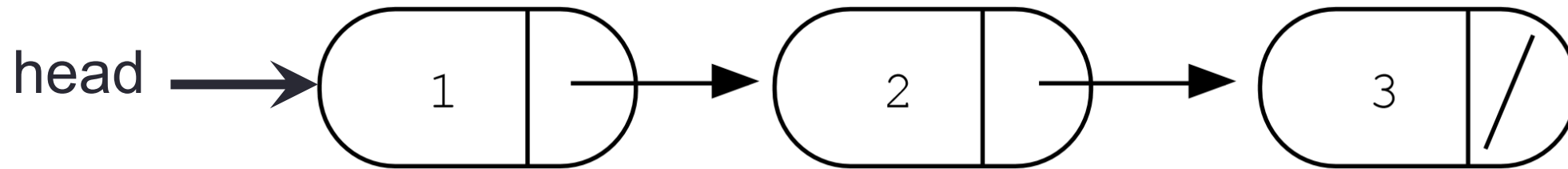
Creating a small list

- Define an empty list
- Add a node to the list

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

Accessing elements of a list

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



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

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

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

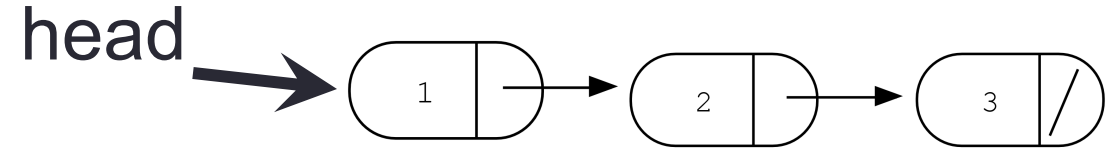
Building a list from an array

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

a

1	2	3
---	---	---

Iterating through the list



```
int length(Node* head) {
    /* Find the number of elements in the list */

```

}

Complex declarations in C/C++

How do we decipher declarations of this sort?

```
int **arr[];
```

Read

- * as “pointer to” (always on the left of identifier)
- [] as “array of” (always to the right of identifier)
- () as “function returning” (always to the right ...)

For more info see:

http://ieng9.ucsd.edu/~cs30x/rt_lt.rule.html

Complex declarations in C/C++

Right-Left Rule

```
int **arr [];
```

Step 1: Find the identifier

Step 2: Look at the symbols to the right of the identifier. Continue right until you run out of symbols *OR* hit a *right* parenthesis ")"

Step 3: Look at the symbol to the left of the identifier. If it is not one of the symbols '*', '(', '[' just say it. Otherwise, translate it into English using the table in the previous slide. Keep going left until you run out of symbols *OR* hit a *left* parenthesis "(".

Repeat steps 2 and 3 until you've formed your declaration.

Illegal combinations include:

[]() - cannot have an array of functions

()() - cannot have a function that returns a function

()[] - cannot have a function that returns an array

Complex declarations in C/C++

```
int i;  
int *i;  
int a[10];  
int f( );  
int **p;  
int (*p)[ ];  
int (*fp)( );  
int *p[ ];  
int af[ ]( );  
int *f( );  
int fa()[ ];  
int ff()( );  
int (**ppa)[ ];  
int (*apa[ ])[ ] ;
```

Next time

- Dynamic arrays
- Pointer arithmetic
- Dynamic memory pitfall