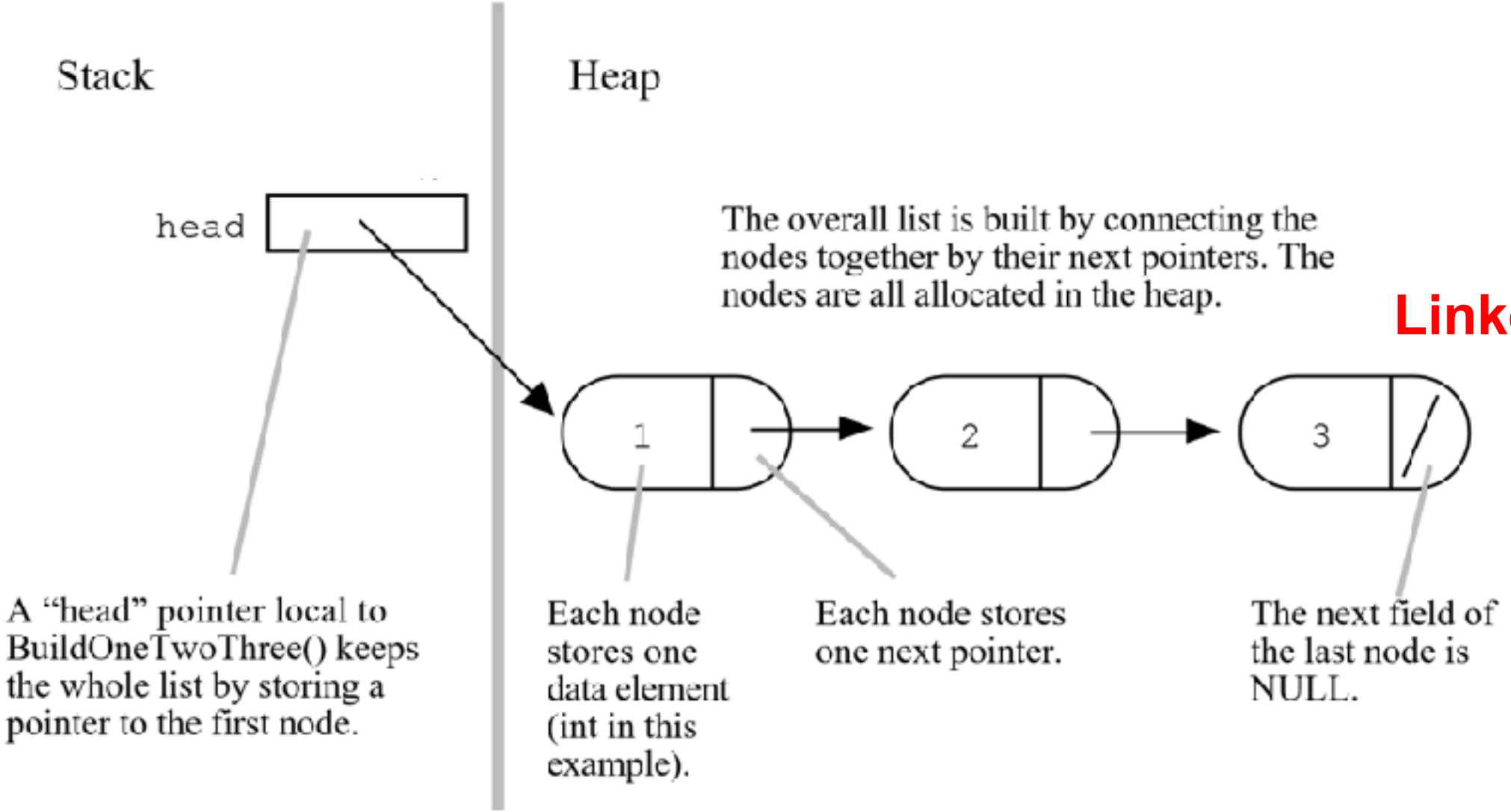
LINKED LISTS (CONTD)
DYNAMIC MEMORY PROBLEMS

Problem Solving with Computers-I

Linked Lists

Array List

The Drawing Of List {1, 2, 3}



Linked List

Creating a small list

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

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

Basic LinkedList Functions

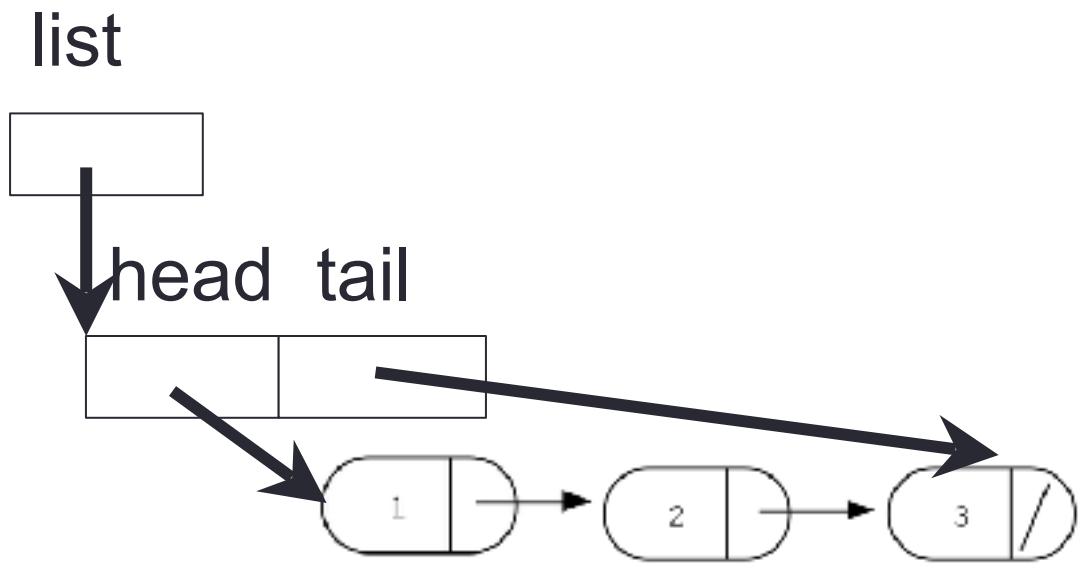
```
LinkedList* createLinkedList();
void insert(LinkedList* list, int value);
```

Iterating through the list

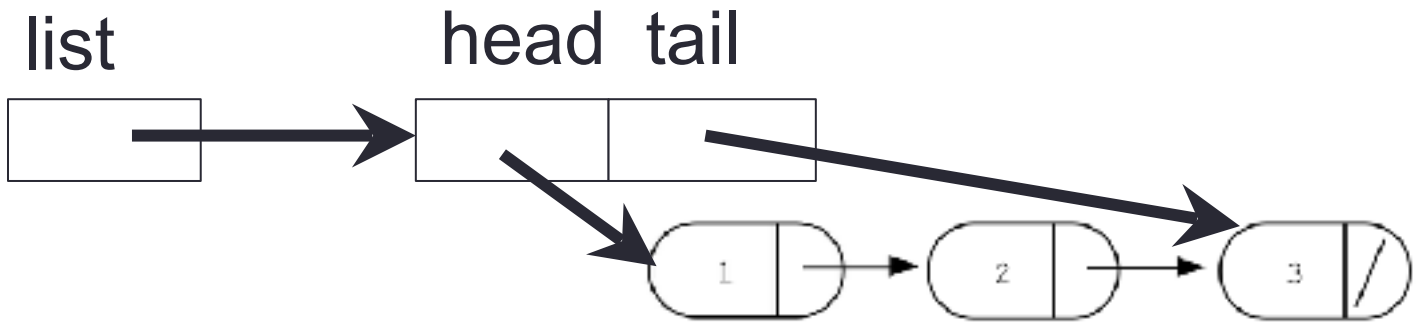
```
int lengthOfList(LinkedList * list) {
   /* Find the number of elements in the list */




}
```



Review:



What is a linked-list?

What are the nodes in a linked list?

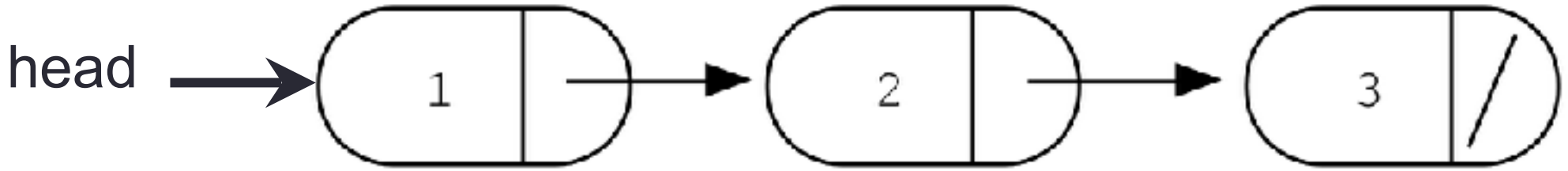
What is stored in each node and why?

What are the links in the above diagram?

How do we access the first element in the list?

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

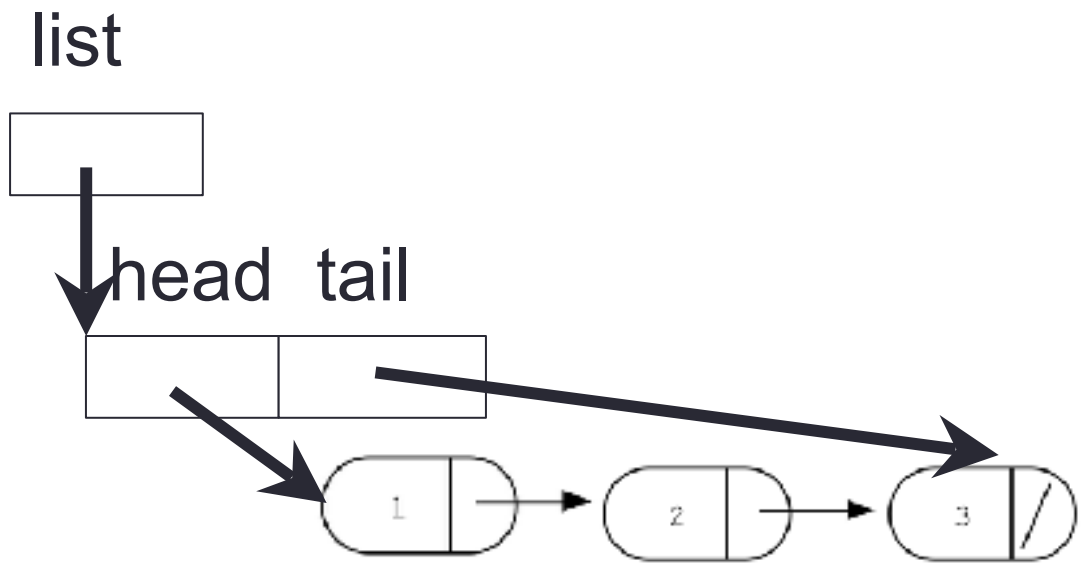
A. 1
B. 2
C. 3
D. NULL
E. Run time error

Searching for an element in the list

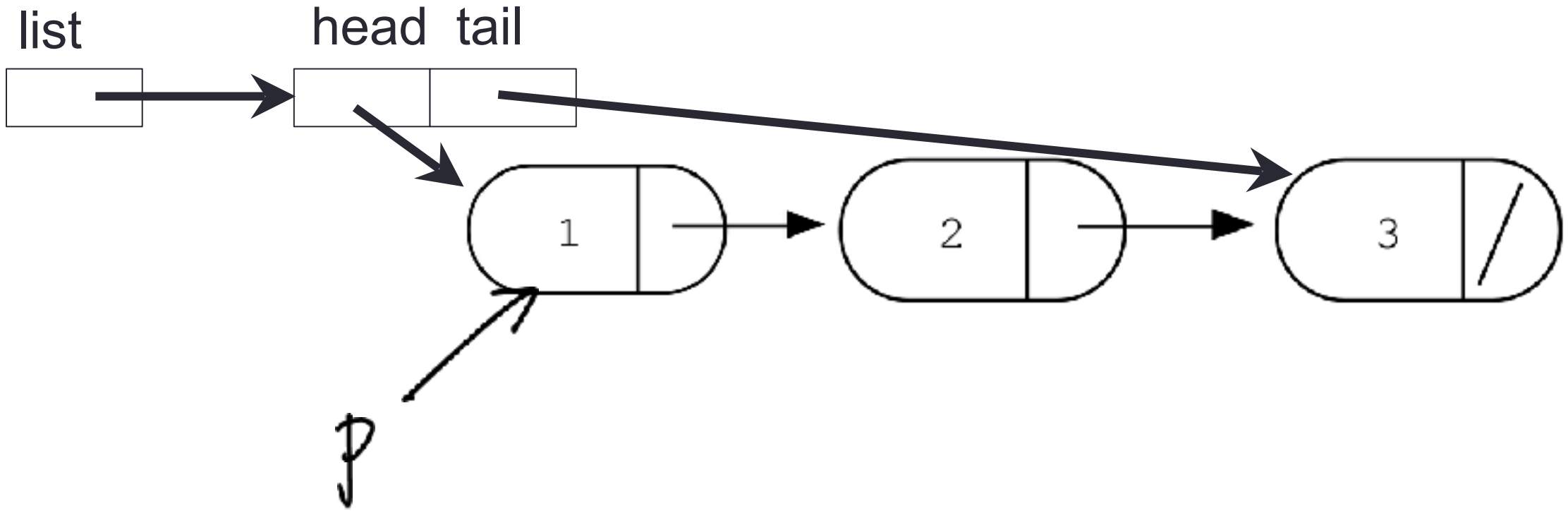
```
bool search(int value) {
    // returns true if the element is in the list
    // false otherwise.




}
```

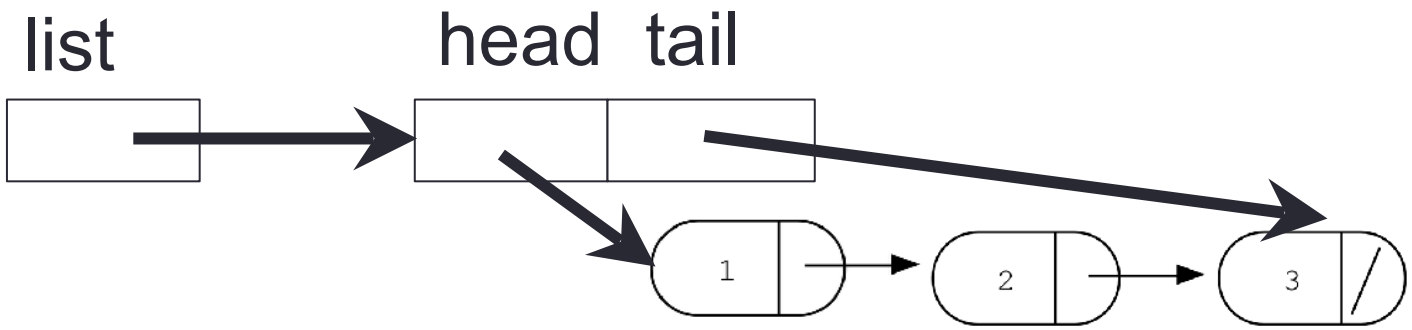


Delete node 2 in the list



Delete the list

```
int freeLinkedList(LinkedList * list);
```



Dynamic memory allocation

- To allocate memory on the heap use the ‘new’ operator
- To free the memory use delete

```
int *p= new int;
delete p;
```

Dangling pointers and memory leaks

- Dangling pointer: Pointer points to a memory location that no longer exists
- Memory leaks (tardy free)
  - Heap memory not deallocated before the end of program (more strict definition, potential problem)
  - Heap memory that can no longer be accessed (definitely a leak , must be avoided!)

Dynamic memory pitfall: Memory Leaks

- Memory leaks (tardy free)

Does calling foo() result in a memory leak? A. Yes B. No

```
void foo(){
     int * p = new int;

}
```

Q: Which of the following functions results in a dangling pointer?

```
int * f1(int num){
    int *mem1 =new int[num];
    return(mem1);
}
```

```
int * f2(int num){
     int mem2[num];
     return(mem2);
}
```

A. f1

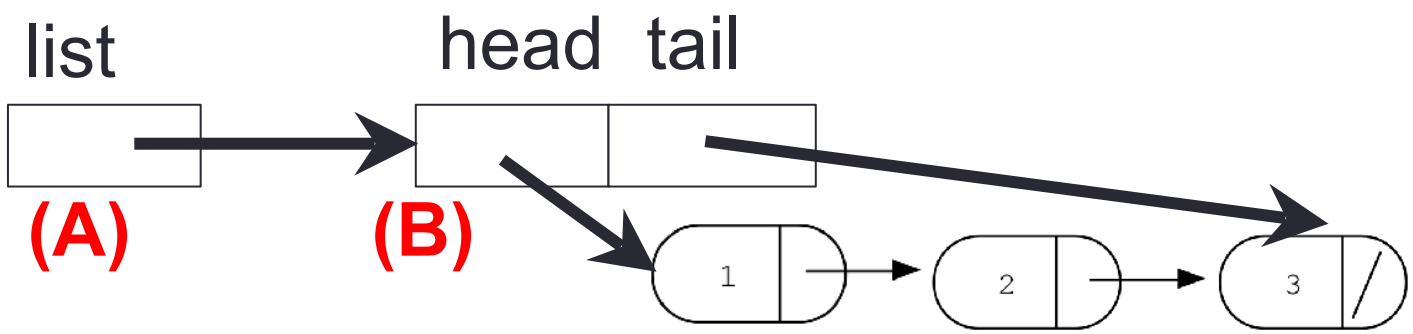
B. f2

C. Both

Deleting the list

```
int freeLinkedList(LinkedList * list){…}
```

Which data objects are deleted by the statement: `delete list;`



(C) All nodes of the linked list

(D) B and C
(E) All of the above

Does this result in a memory leak?

Next time

- Recursion
- Strings