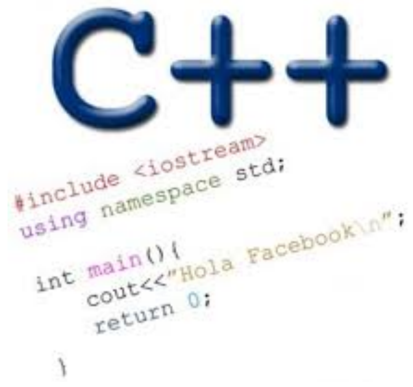
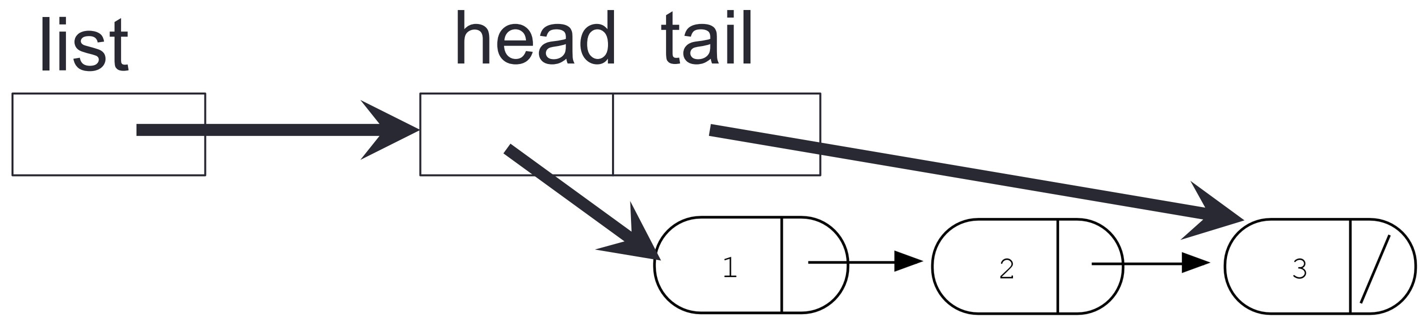
LINKED LISTS (CONTD) DYNAMIC MEMORY PROBLEMS

Problem Solving with Computers-I



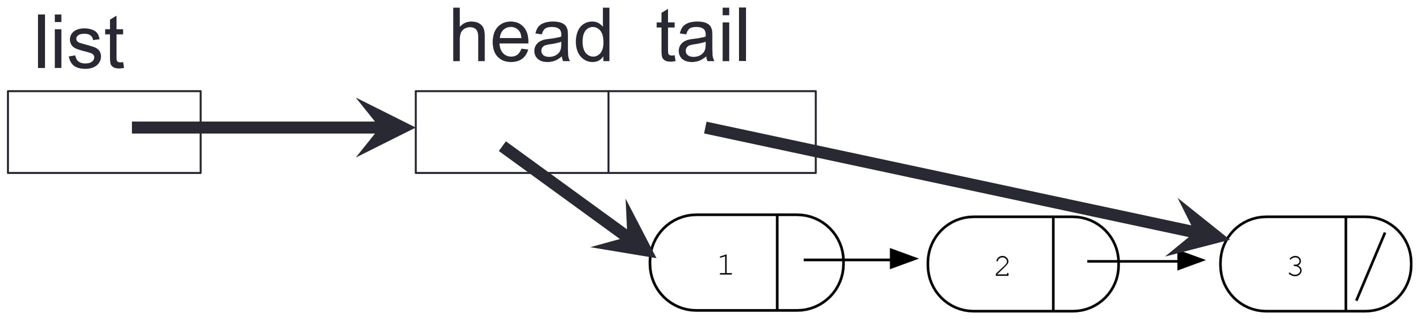
```
#include <iostream>
using namespace std;

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

Review:
What are the ‘links’ in a linked-list?

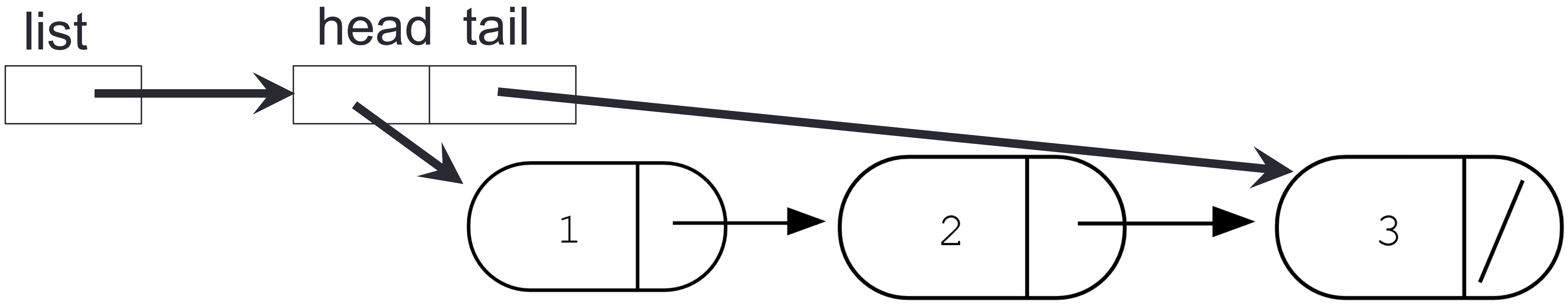


Iterating through the list

```
int lengthOfList(LinkedList *list) {
    ???
}
```



Delete node 2 in the 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 (premature free—you freed the memory too early)
- Memory leaks (tardy free—you're freeing the memory too late, or not at all)
  - 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

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

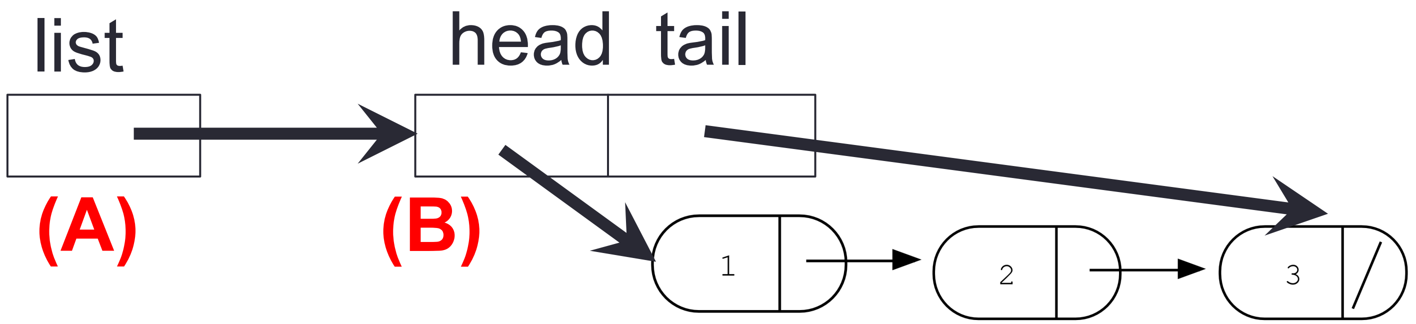
B. f2

C. Both

Deleting the list

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

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



(A)

(B)

(C) All nodes of the linked list

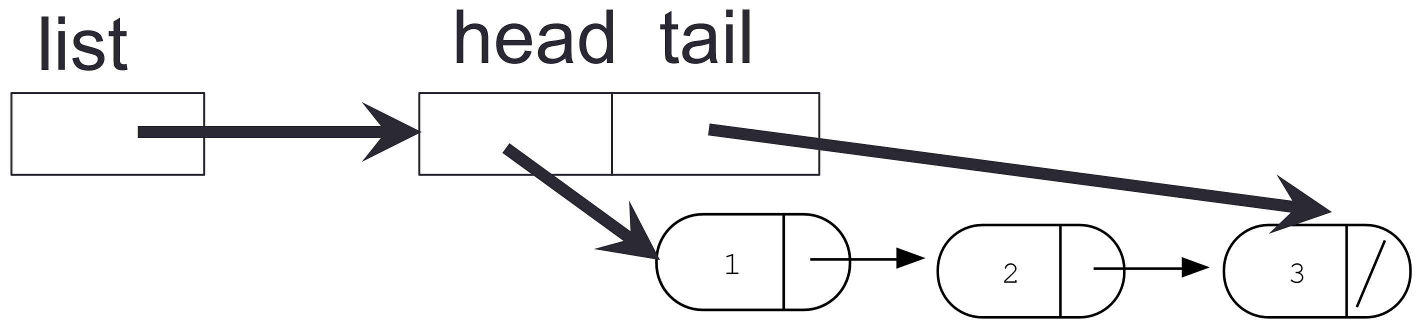
(D) B and C

(E) All of the above

Does this result in a memory leak?

Delete the list

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


list
head
tail
1
2
3