C++ MEMORY MODEL
LINKED LISTS

Problem Solving with Computers-I

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

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

The case of the disappearing data!

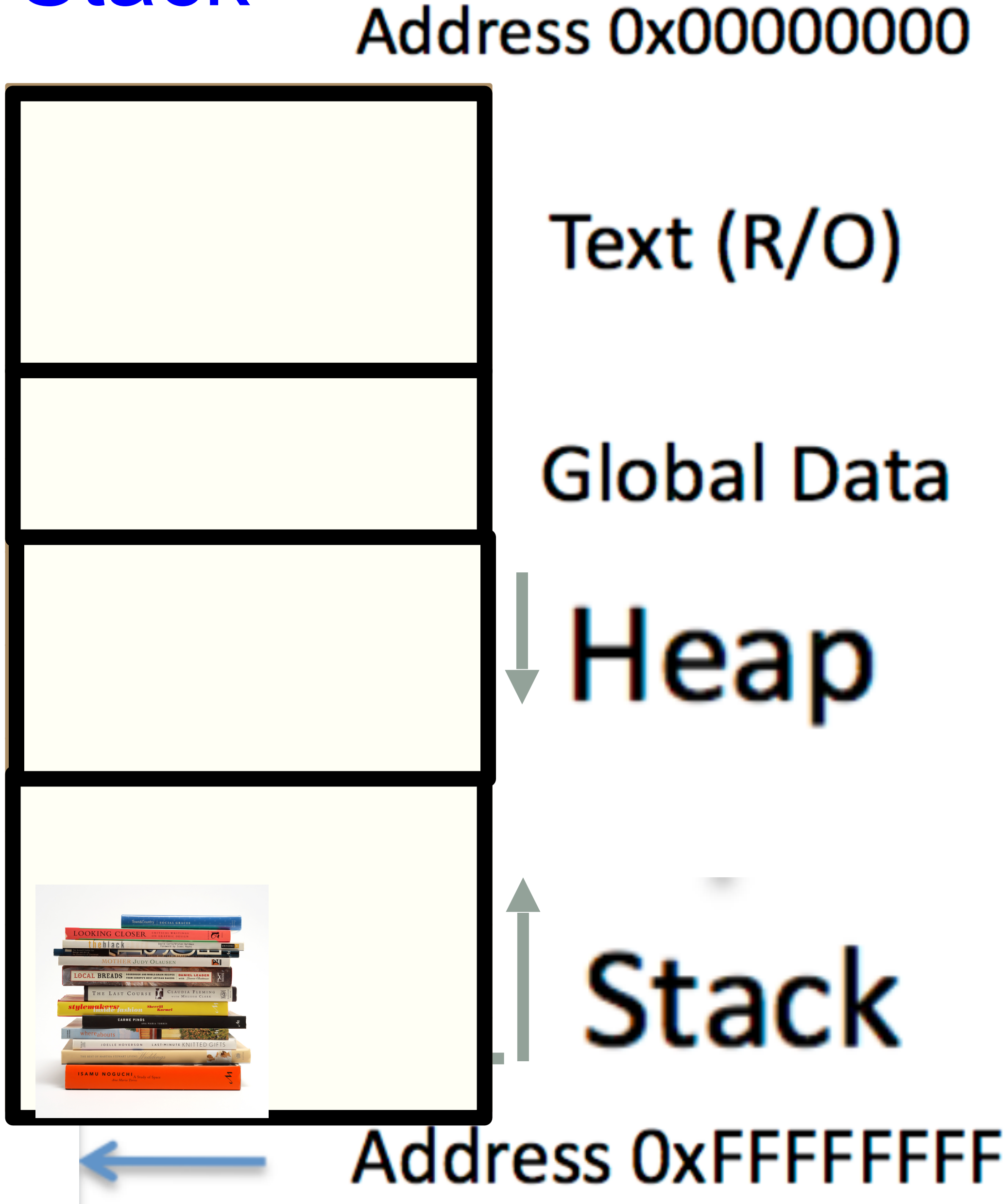
```
int getInt(){
     int x=5;
     return x;
}
int* getAddressOfInt(){
     int x=10;
     return &x;
}
int main(){
     int y=0, *p=nullptr, z=0;
     y = getInt();
     p = getAddressOfInt();
     z = *p;
    cout<<y<<", "<<z<<", "<<*p<<endl;
}
```

What is the output?

A. 5, 0, 10
B. 5, 10, 10
C. Something else

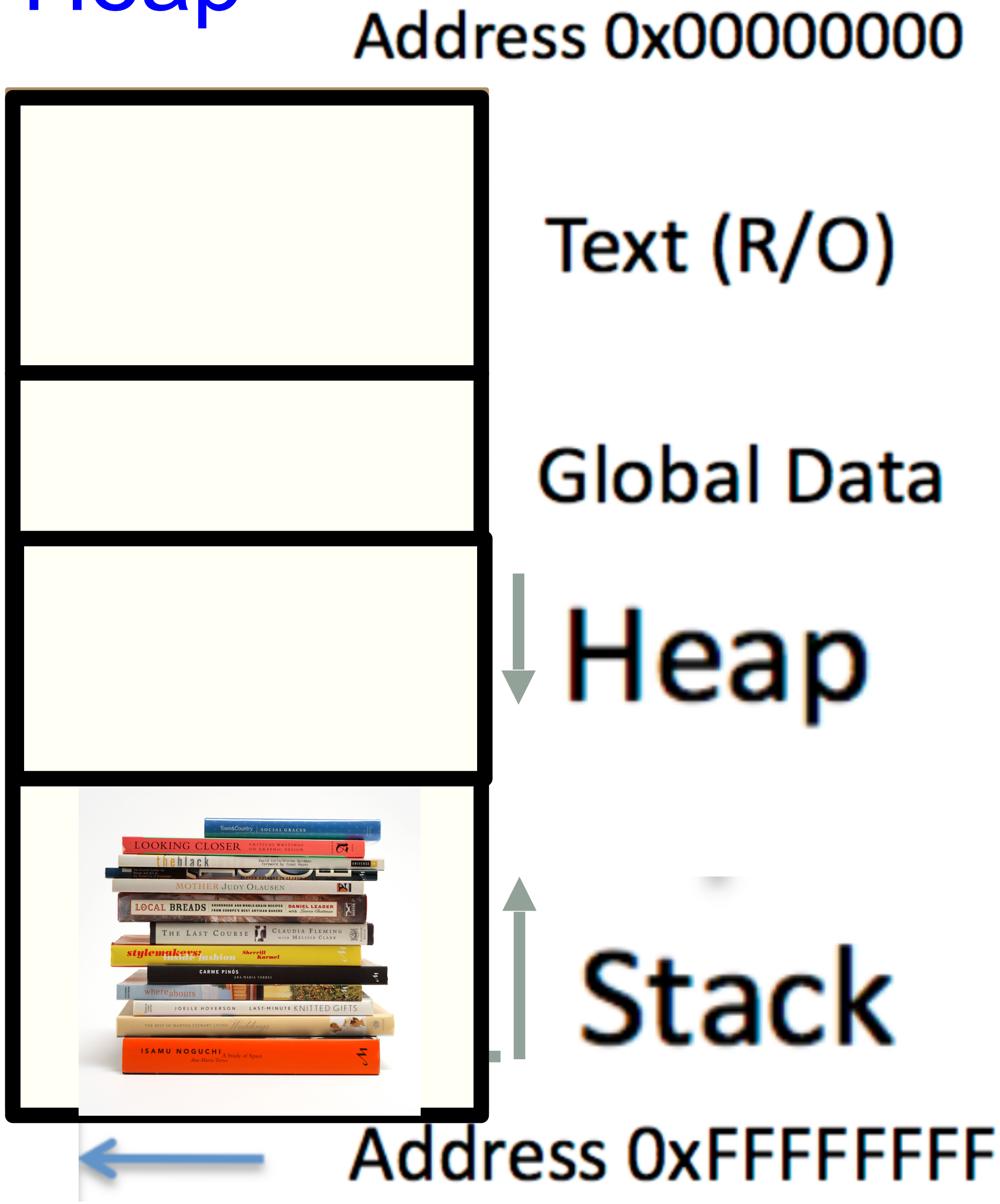
C++ Memory Model: Stack

- Stack: Segment of memory managed automatically using a Last in First Out (LIFO) principle
- Think of it like a stack of books!



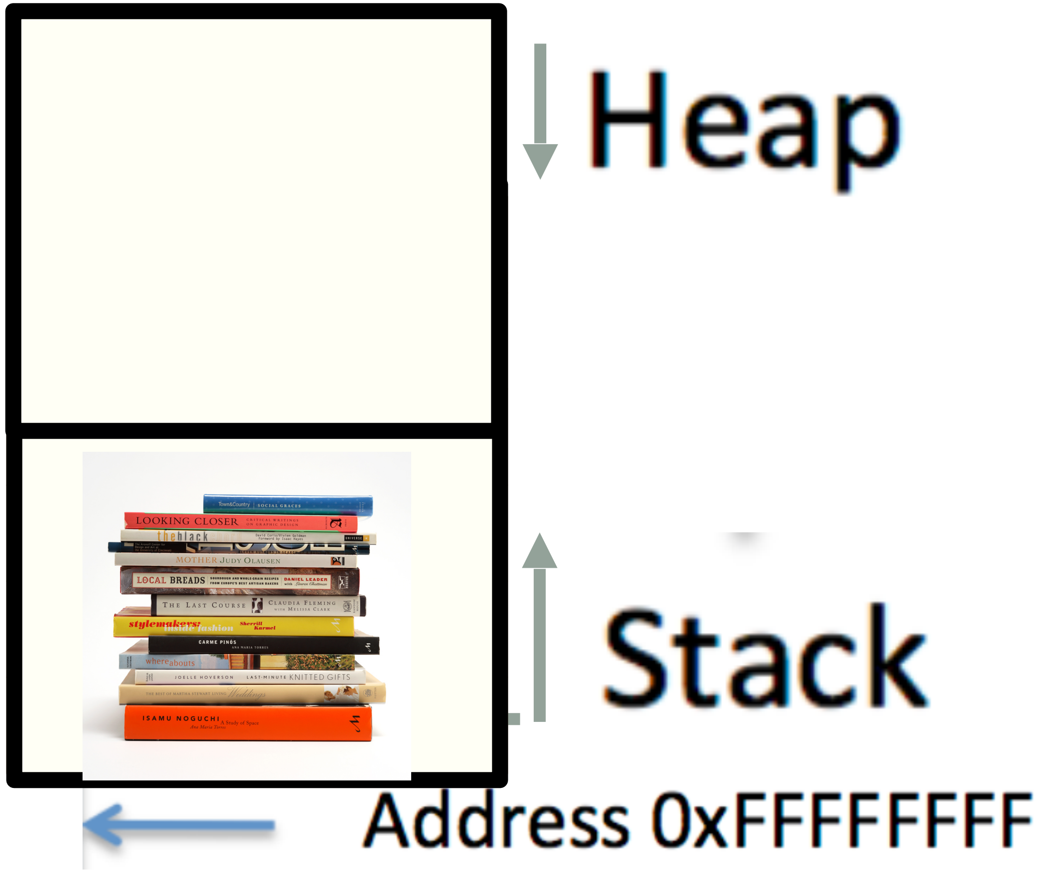
C++ Memory Model: Heap

- Heap: Segment of memory managed by the programmer
- Data created on the heap stays there
  - FOREVER or
  - until the programmer explicitly deletes it



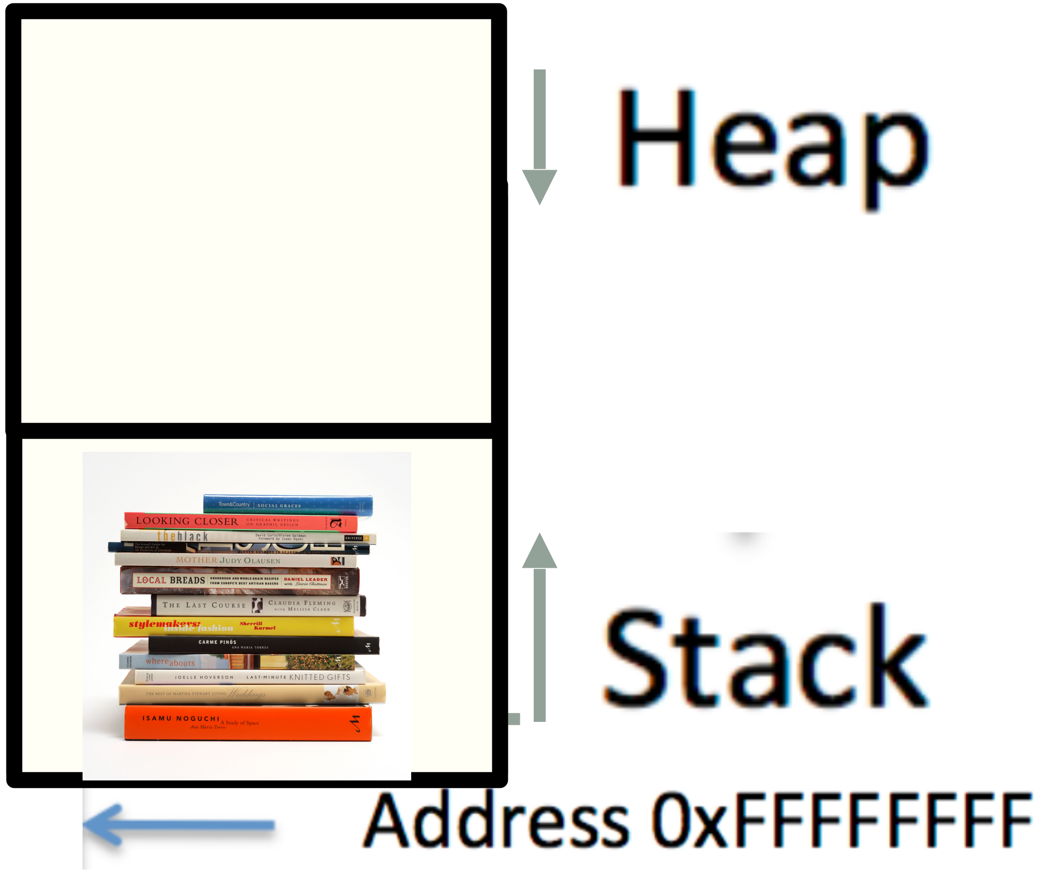
Creating data on the Heap: new

To **allocate** memory on the heap use the **new** operator



Deleting data on the Heap: delete

To **free** memory on the heap use the **delete** operator



Dynamic memory management = Managing data on the heap

```
int *p= new int; //creates a new integer on the
heap

SuperHero *n = new SuperHero;

                //creates a new Student on the
heap

delete p; //Frees the integer

delete n; //Frees the Student
```

Pointer pitfalls and memory errors

- **Segmentation faults**: Program crashes because it attempted to access a memory location that either doesn’t exist or doesn’t have permission to access
- Examples of code that results in undefined behavior and potential segmentation fault

```
int arr[] = {50, 60, 70};

for(int i=0; i<=3; i++){
   cout<<arr[i]<<endl;
}
```

```
int x = 10;
int *p;
cout<<*p<<endl;
```

Dynamic memory pitfalls

Memory leaks (tardy free):

Heap memory not deallocated before the end of program

Heap memory that can no longer be accessed

Example

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

}
```

Heap vs. stack

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

int* createAnIntArray(int len){

    int arr[len];
    return arr;

}
```

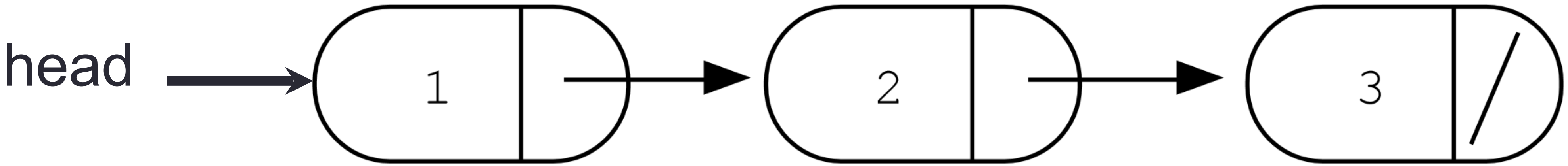
Does the above function correctly return an array of integers?

A. Yes

B. No

Accessing elements of a linked 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

Creating a small list

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

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

Heap vs. stack

```
Node* createSmallLinkedList(int x, int y){
    Node* head = NULL;
    Node n1 ={x, NULL};
    Node n2 ={y, NULL};
    head = &n1;
    n1.next = &n2;
    return head;
}
```

Does the above function correctly return a two-node linked list?

A. Yes

B. No

Creating a small list

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

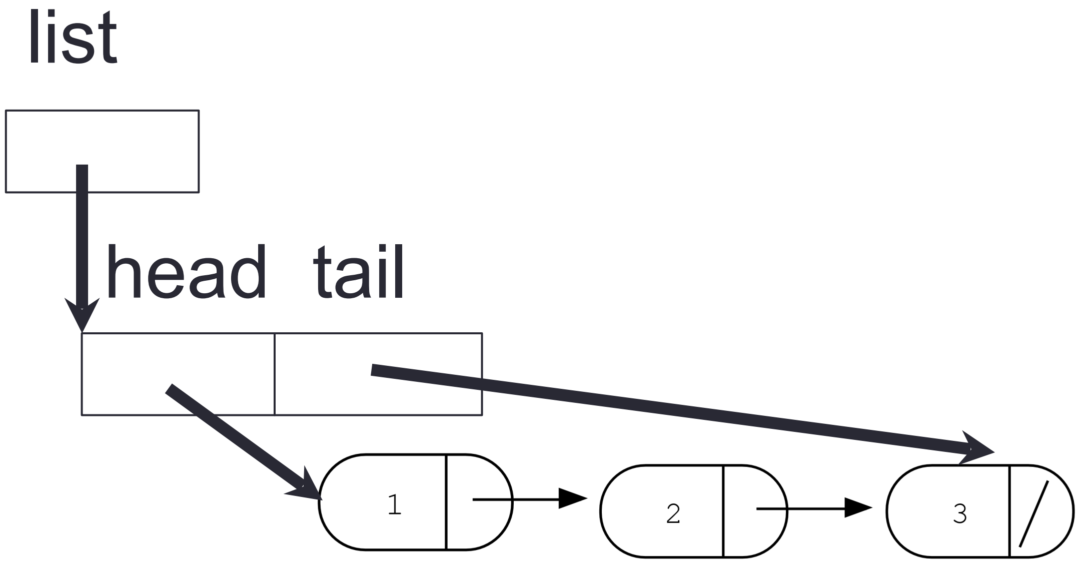
```
struct Node {
     int data;
     Node* next;
};

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

Inserting a node in a linked list

```
void insert(LinkedList* h, int value) ;
```

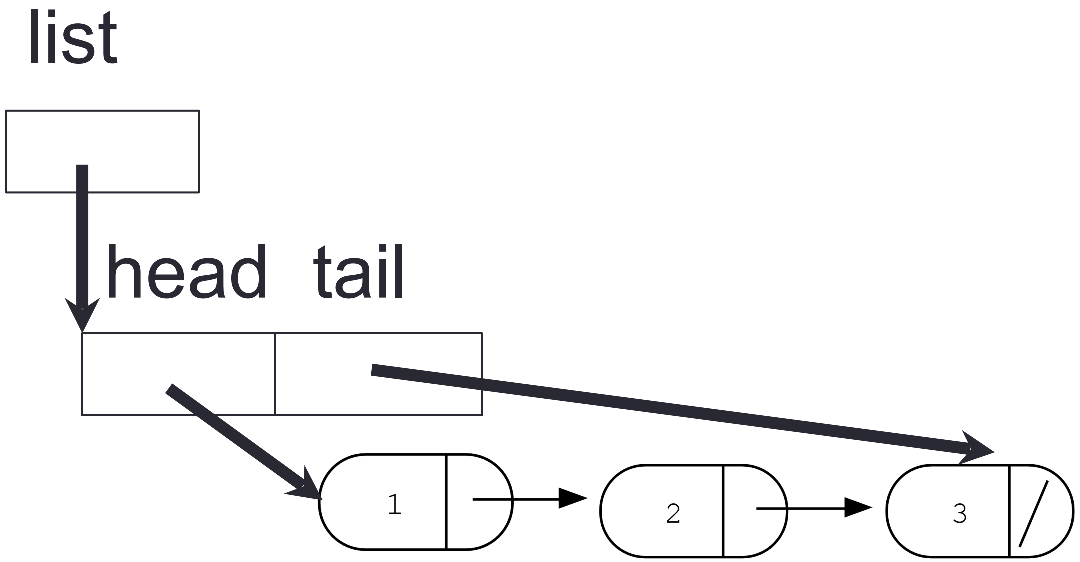
Iterating through the list

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









}
```



Deleting the list

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







}
```



Next time

- Memory-related errors
- Double-linked lists