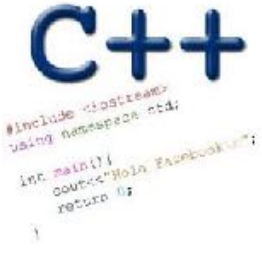
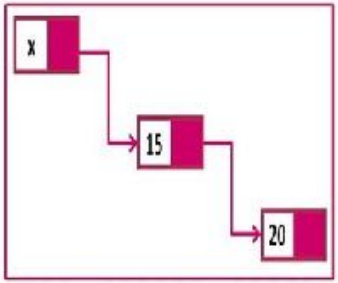
DYNAMIC MEMORY ALLOCATION LINKED LISTS

Problem Solving with Computers-I

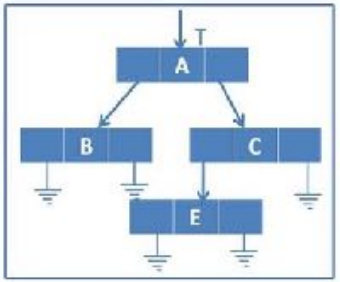


Different ways of organizing data!

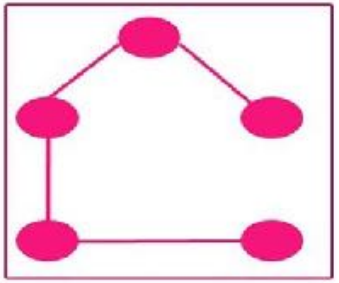
Array List



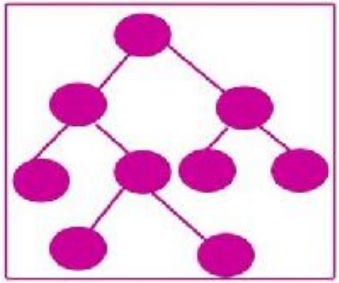
Link list



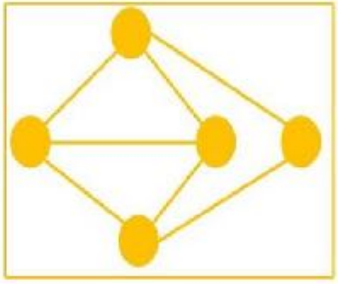
list



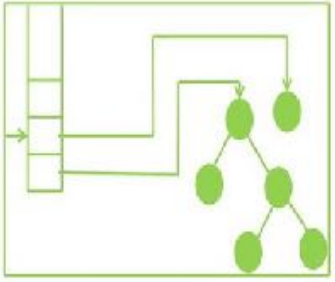
spanning tree



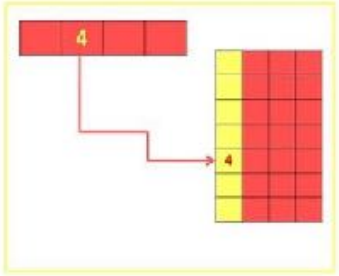
Tree



Graph



Stack



Hashing

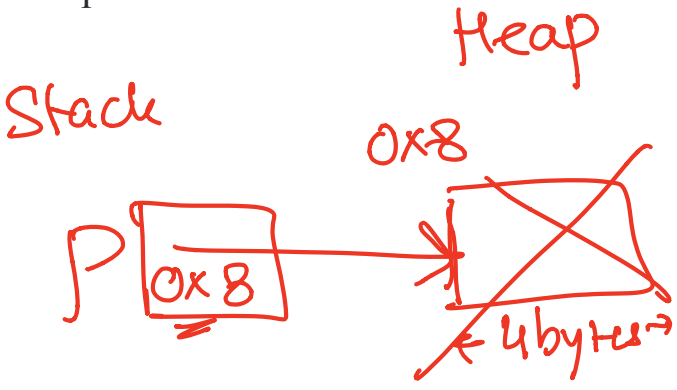
Dynamic memory management

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

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

same as

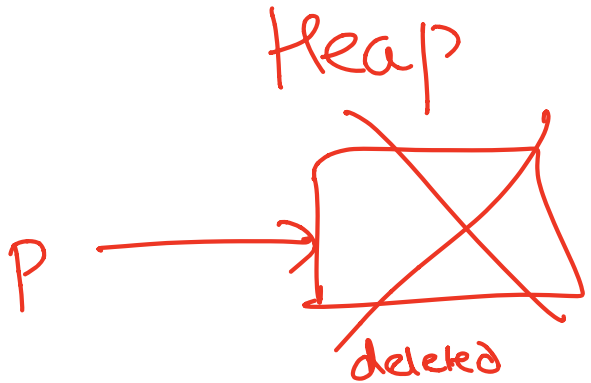
```
int *p;
p = new int;
...
delete p; // deletes the object that 'p' points to
```



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
  - Heap memory that can no longer be accessed

```
int *p;
p = new int;
delete p;
// p is now dangling
// set p to null
p = 0;
```



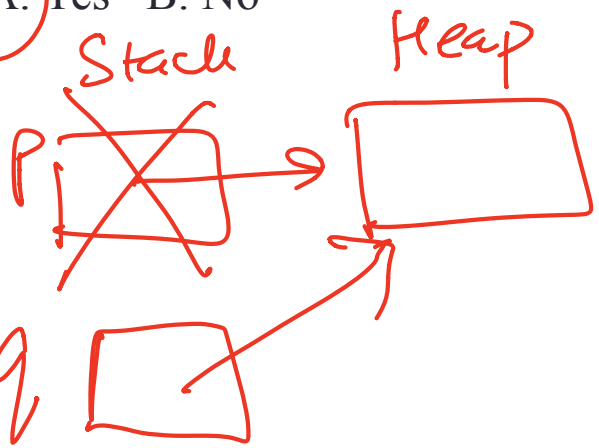
Dynamic memory pitfalls

When the function returns p is removed from the stack & there is no way to get to the heap object

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

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

}
```



change to

```
int* foo() {
  int *p = new int;
  return p
}
```

```
int * q;
q = foo();
```

Q: Which of the following functions returns 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

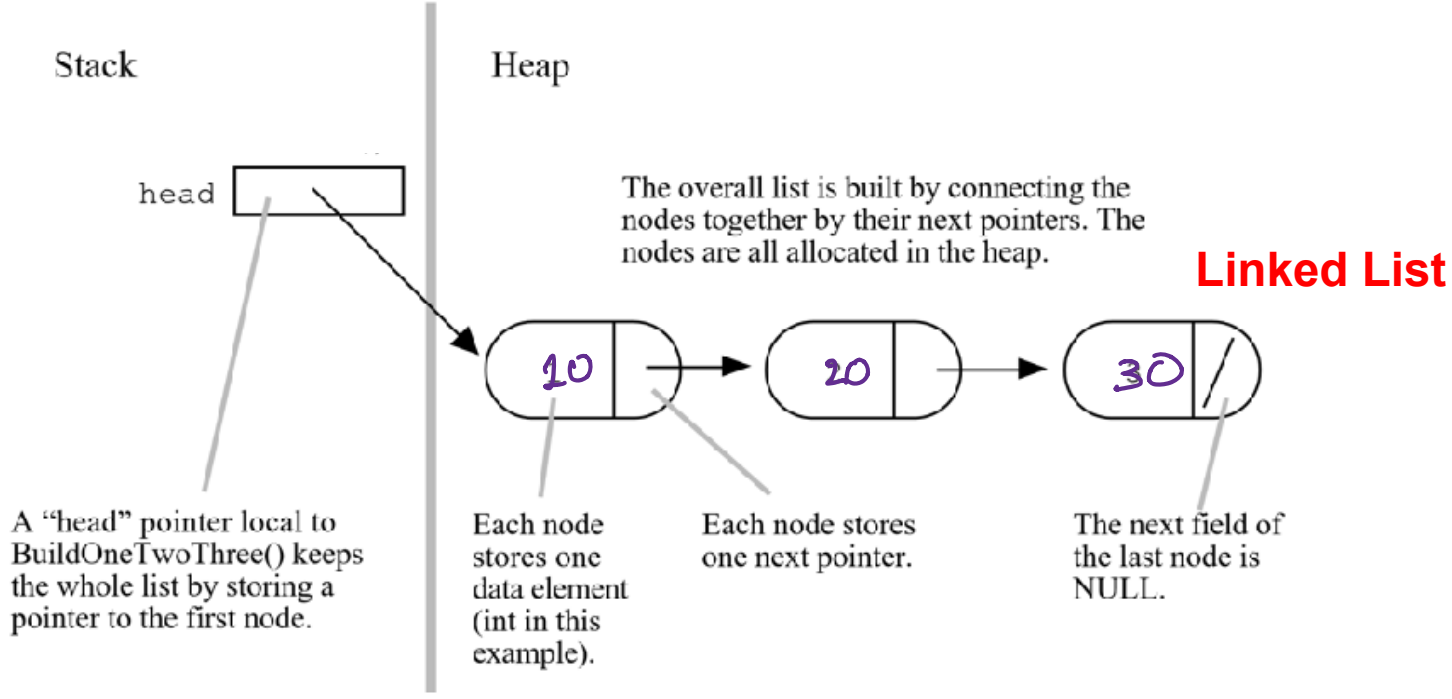
C. Both

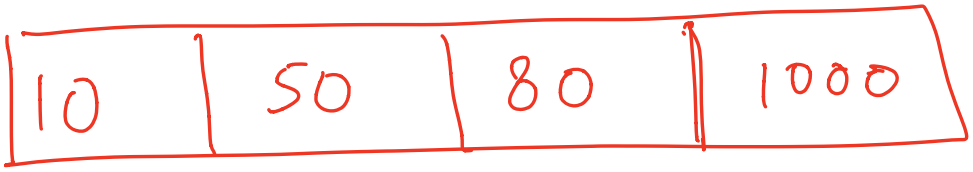
Linked Lists

10	20	30

Array List

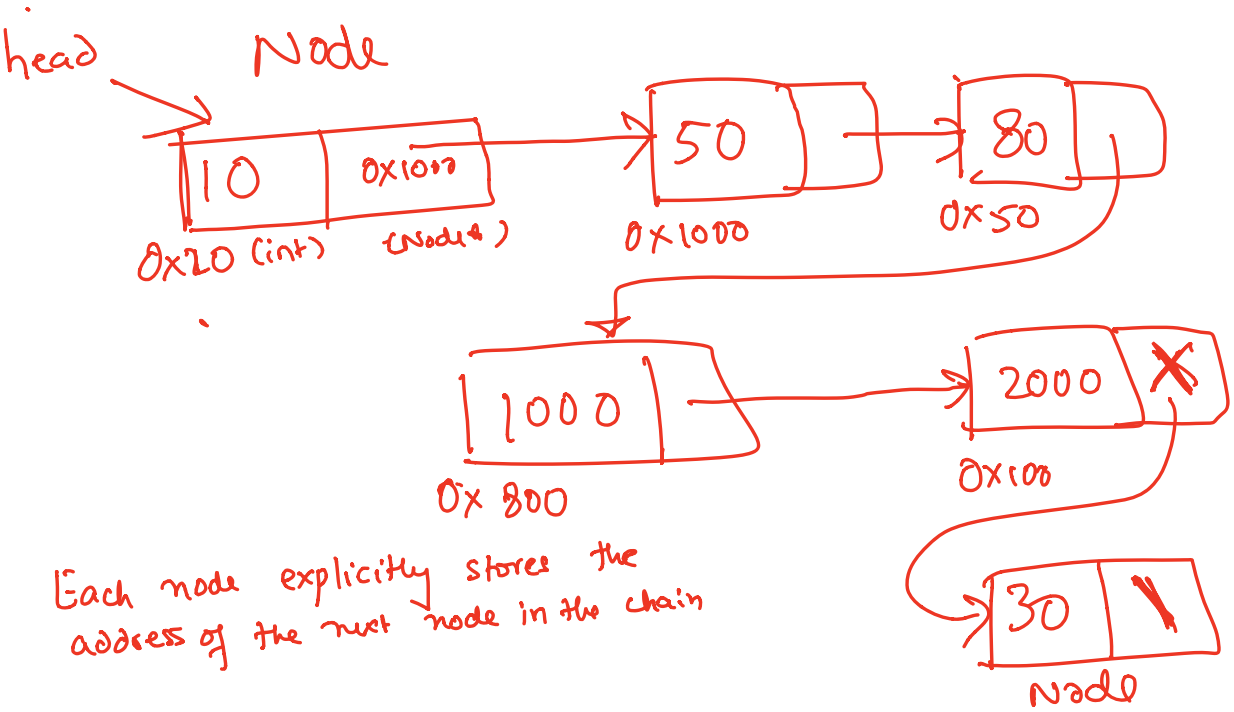
The Drawing Of List {1, 2, 3}



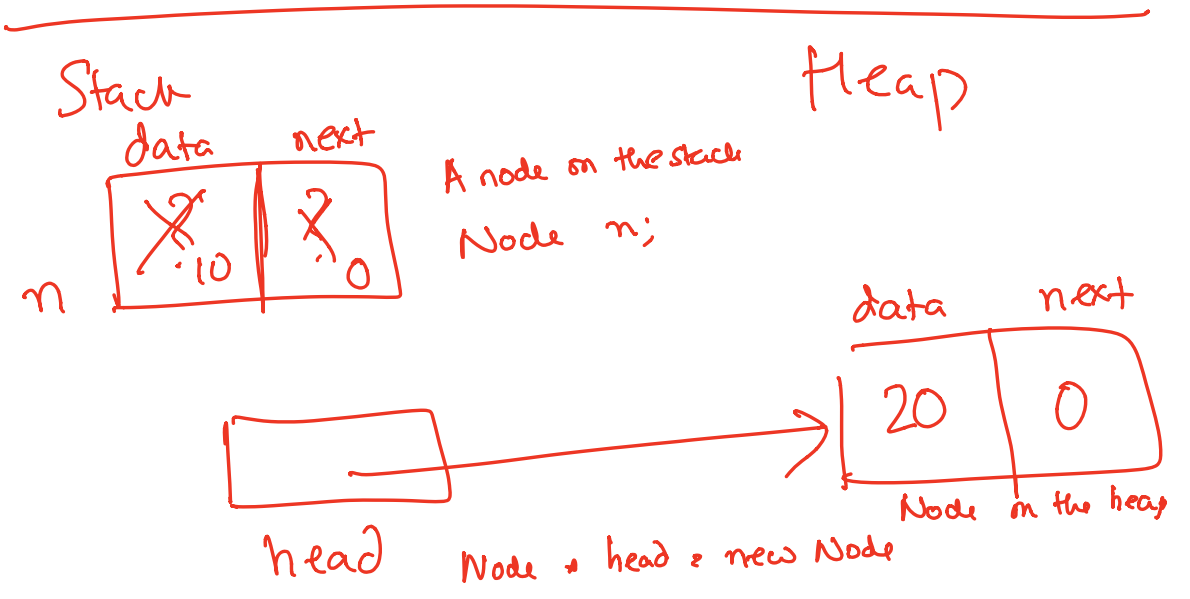


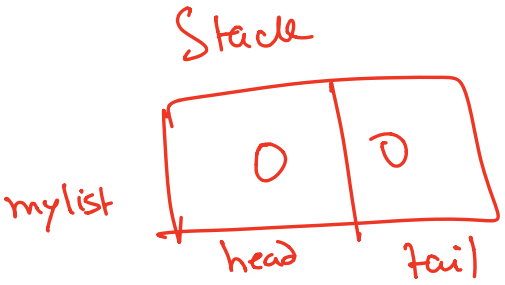
arr

int arr[] = {10, 50, 80, 1000};



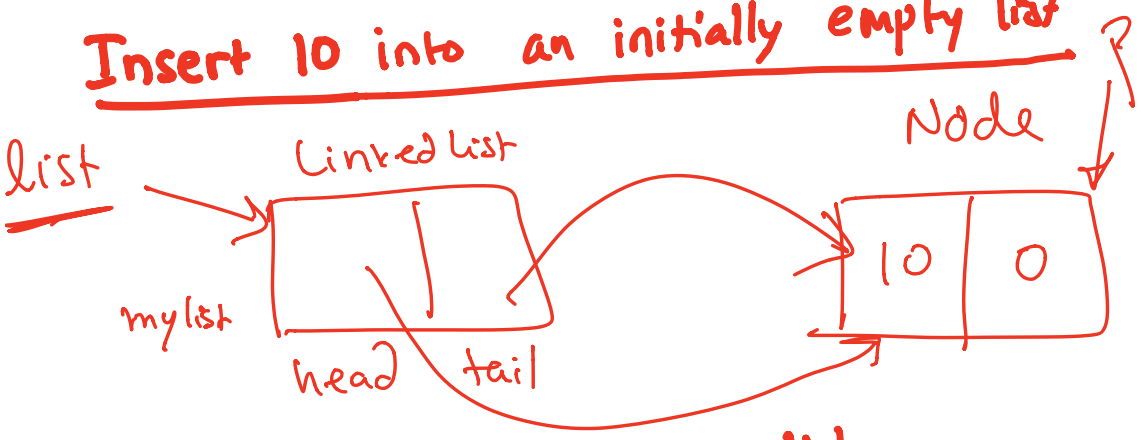
Each node explicitly stores the address of the next node in the chain



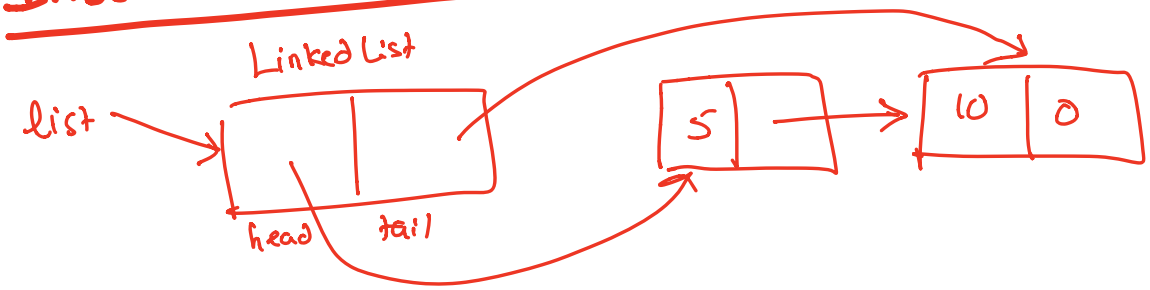


Empty linked list

Insert 10 into an initially empty list

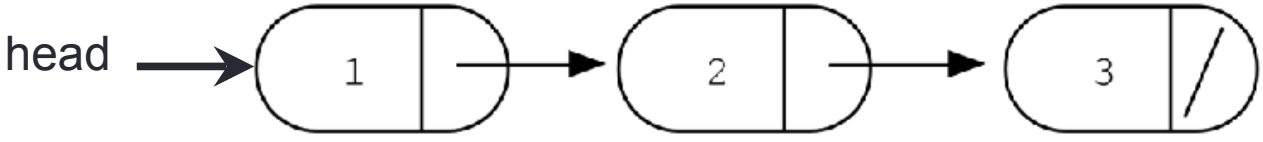


Insert 5 to the front of the list



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

Creating a small list

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

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

Inserting a node in a linked list

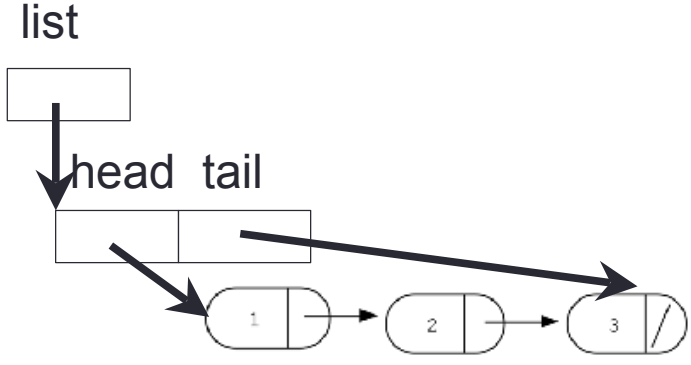
```
Void insertToHeadOfList(LinkedList* h, int value) ;
```

Iterating through the list

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




}
```

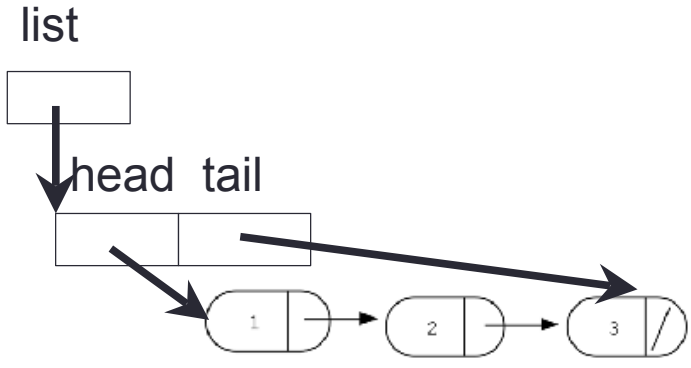


Deleting the list

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




}
```



Next time

- More linked lists
- Dynamic arrays
- Dynamic memory pitfall