DYNAMIC MEMORY ALLOCATION LINKED LISTS

Problem Solving with Computers-I

Different ways of organizing data!

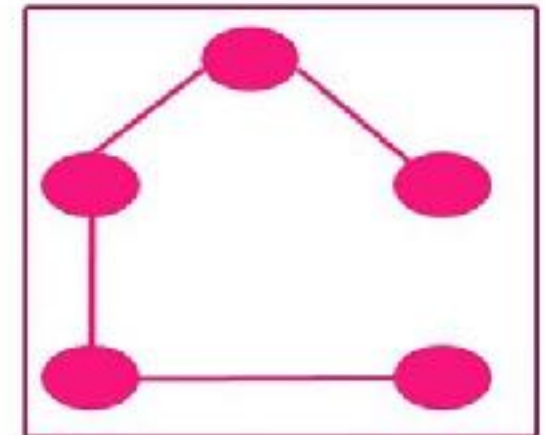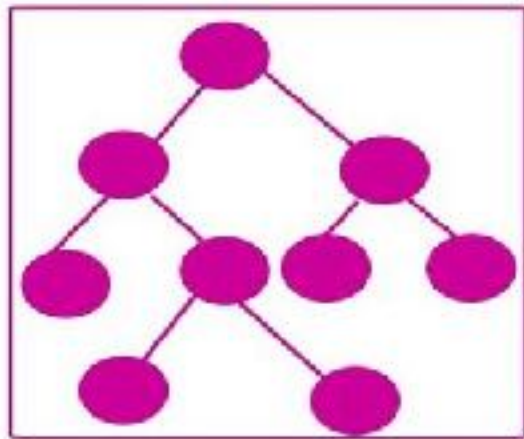| 15	20	30

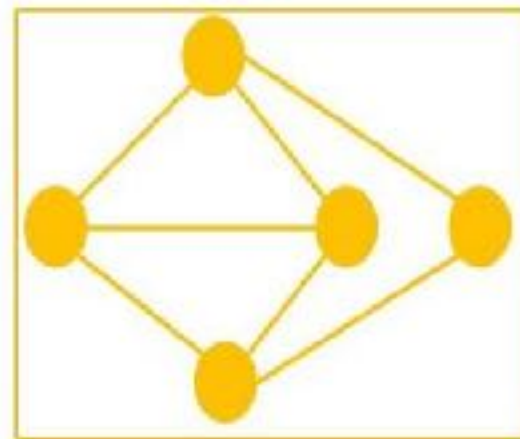**Array List**
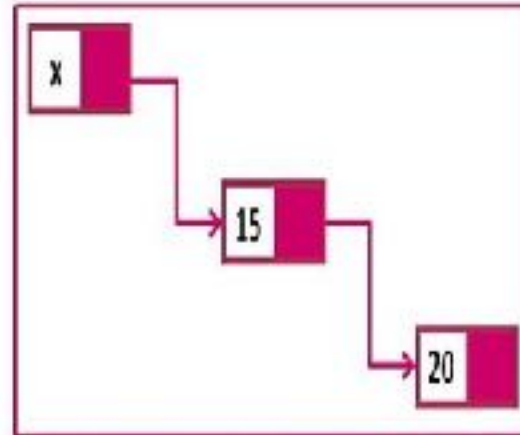
Link list
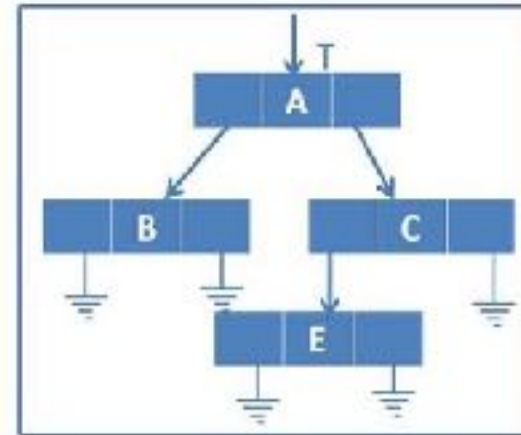
list

spanning tree

Tree

Graph
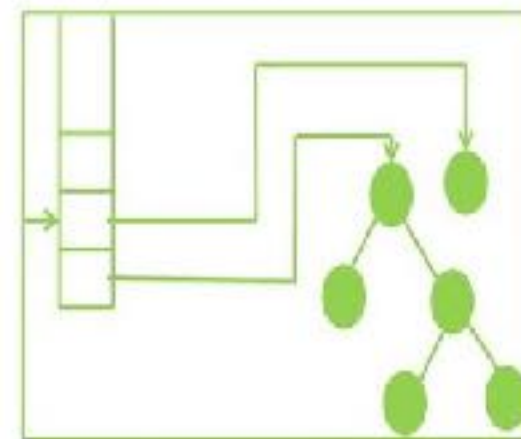
Stack
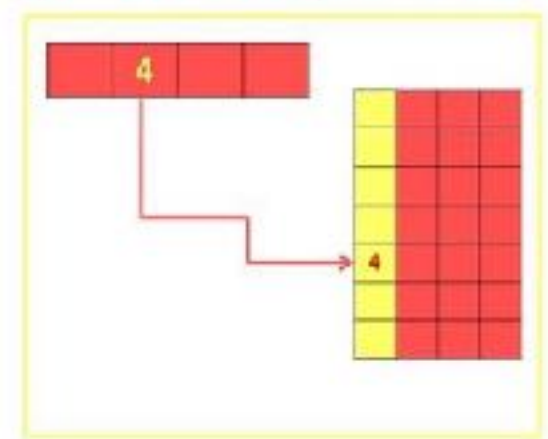
Hashing

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;
```

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

Dynamic memory pitfalls

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

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

}
```

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);
}
```
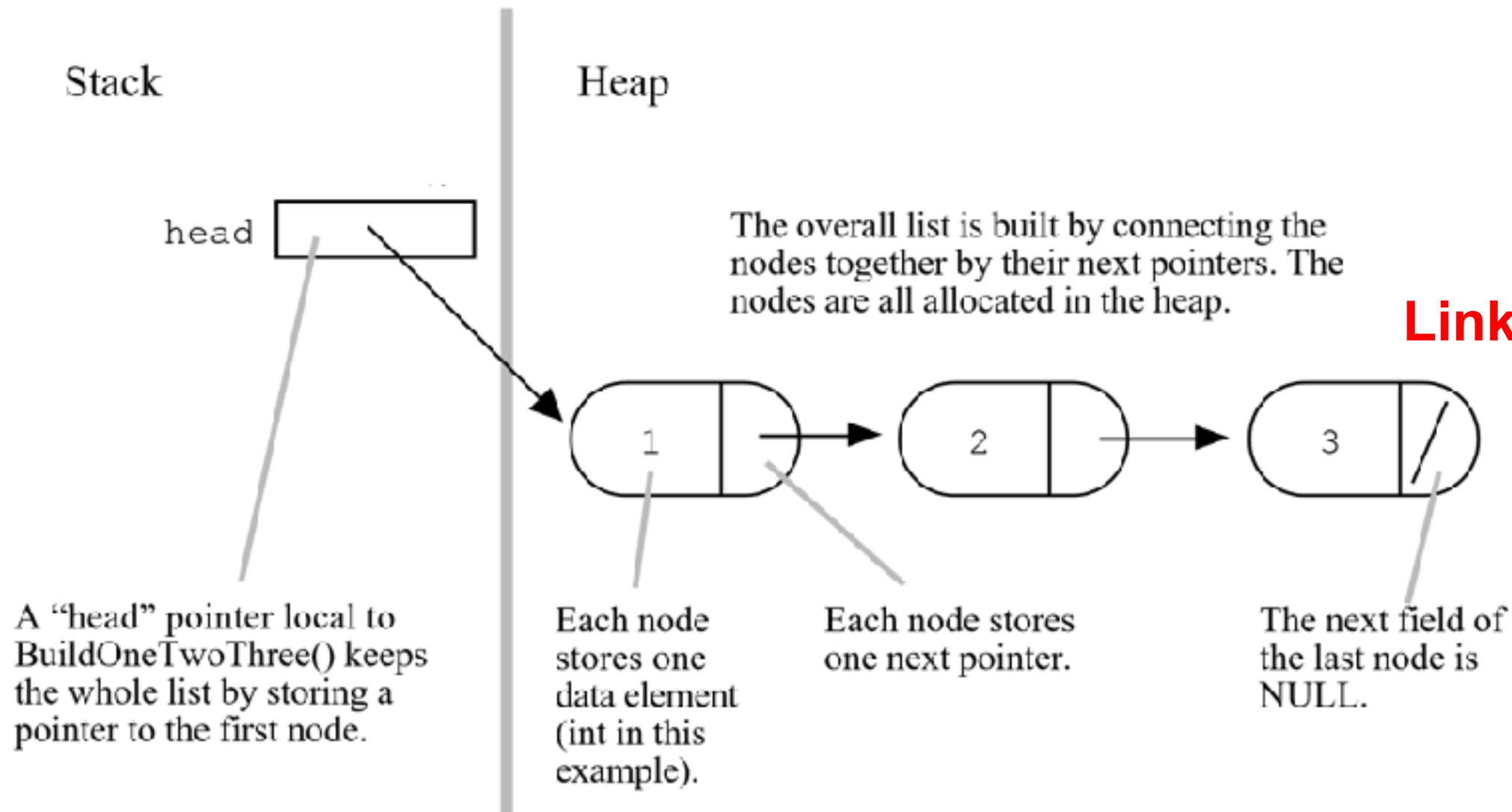
A. f1

B. f2

C. Both

Linked Lists

Array List

1	2	3

The Drawing Of List {1, 2, 3}

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

1 → 2 → 3

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).

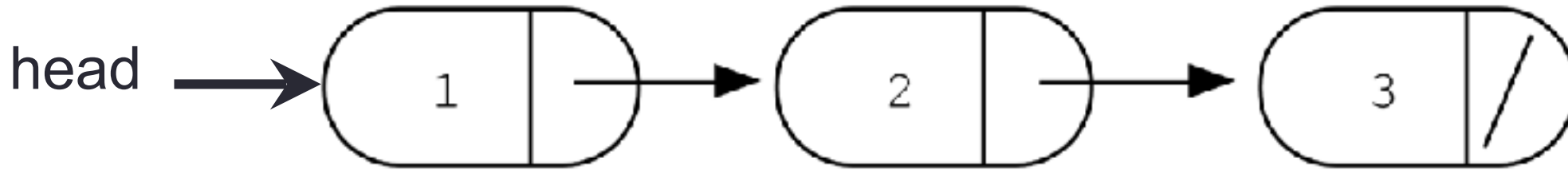Each node stores one next pointer.

The next field of the last node is NULL.

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
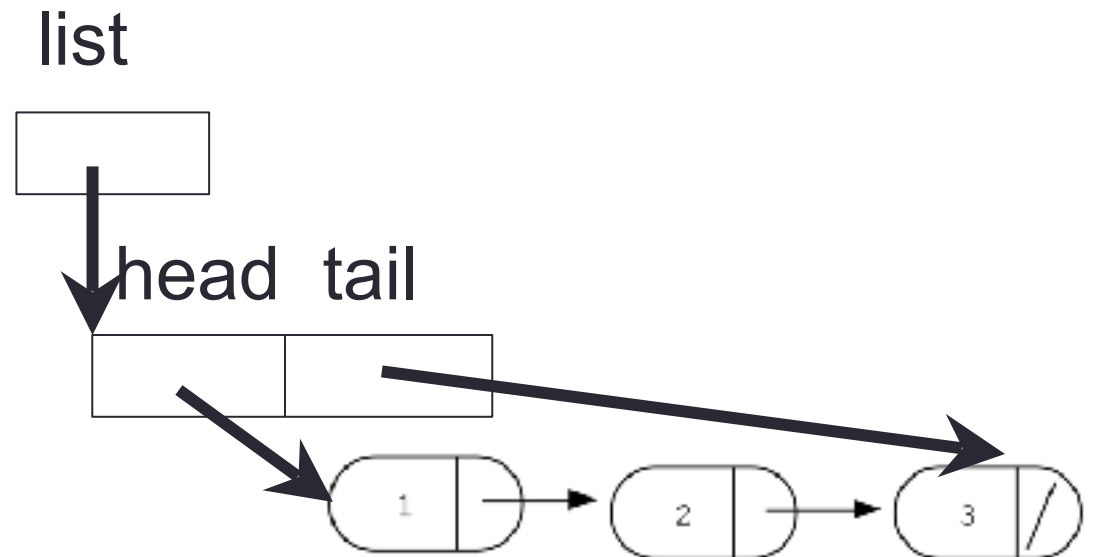- Add a node to the list with data = 10

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

Inserting a node in a linked list

```
Void insertToHeadOfList(LinkedList* h, int value) ;
```
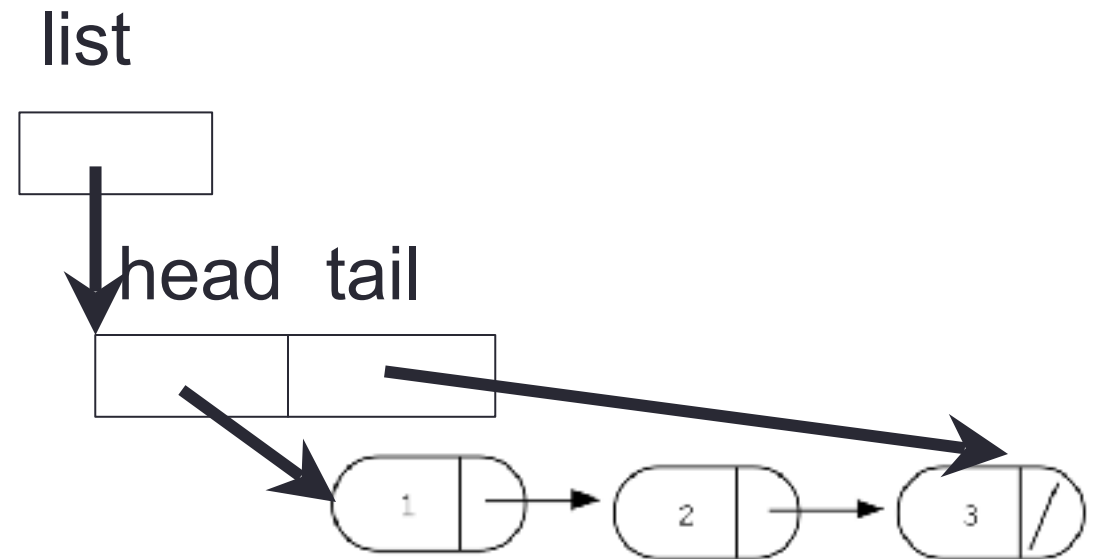
Iterating through the list

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

}
```

list

head tail

Deleting the list

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



}
```

list

head tail

Next time

- More linked lists
- Dynamic arrays
- Dynamic memory pitfall