C++ MEMORY MODEL
DYNAMIC MEMORY
HEAP VS STACK
LINKED LIST

Problem Solving with Computers-I

Don’t miss these events!

UCSB Computer Science:

Speed Advising

Meet with a CS faculty member!

- Learn about CS UCSB
- Get valuable career advice
- Find research opportunities
- Discover elective choices

Friday, November 9th

10:00 AM – 1:00 PM

Outside Harold Frank Hall

(by the CSIL labs)

[array of talks]

Talks[16] “How to Start a Startup”:
A fireside conversation with Randy Modos and
UCSB Professor Giovanni Vigna

Time: Nov 14, 2018 3:30p - 5:00p
Location: 1132 Harold Frank Hall

Randy Modos, the president and co-founder of PayJunction, provides vision and leadership for the company as it pioneers green payment technology for businesses, helping to reduce costs and eliminate fraud. Modos met his co-founders at UCSB while pursuing his B.S. in Computer Science. Together, they took an inspired idea and grew it into a company that process over $4 billion annually.

Giovanni Vigna is a Professor in the Department of Computer Science at the University of California in Santa Barbara. His current research interests include malware analysis, web security, vulnerability assessment, and mobile phone security. He also edited a book on Security and Mobile Agents and authored one on Intrusion Correlation.

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

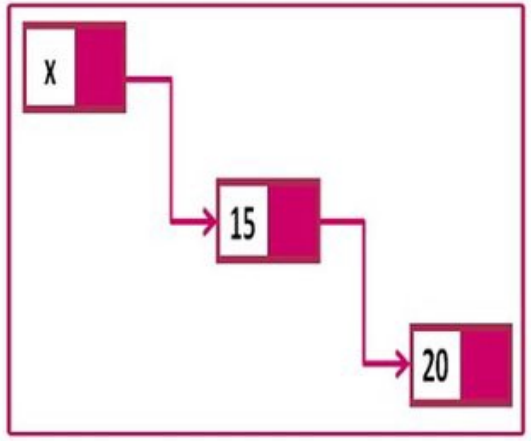
```
int arr[] = {50, 60, 70};

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

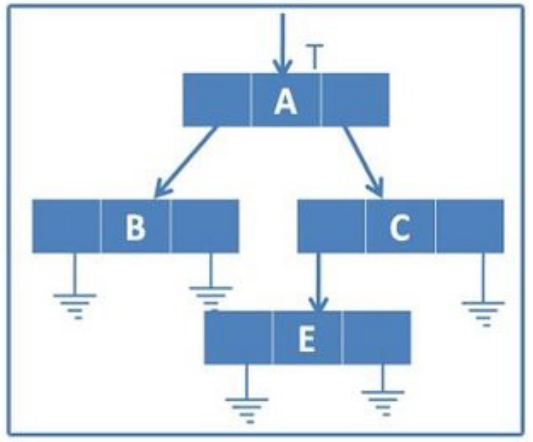
```
int x = 10;
int* p;
cout<<*p<<endl;
```

Where are we going? Data Structures!

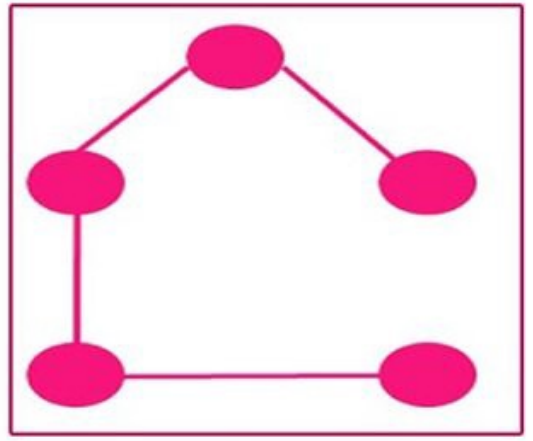
Arrays



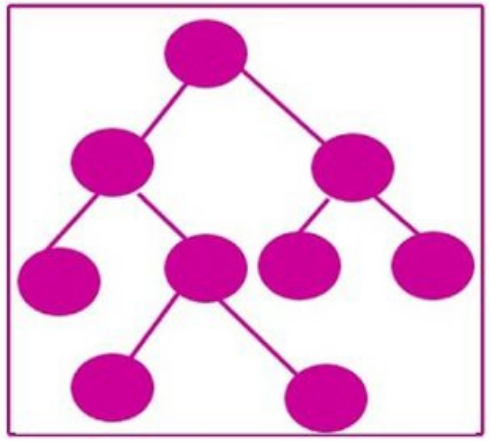
Link list



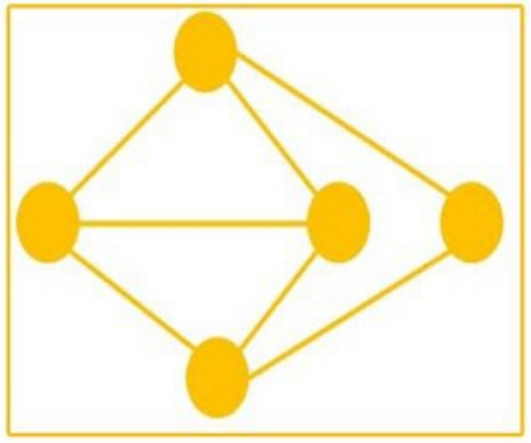
list



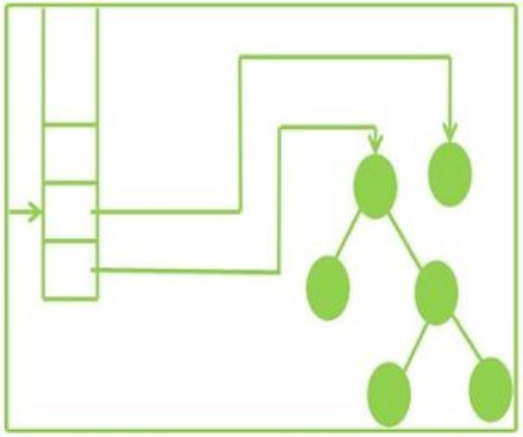
spanning tree



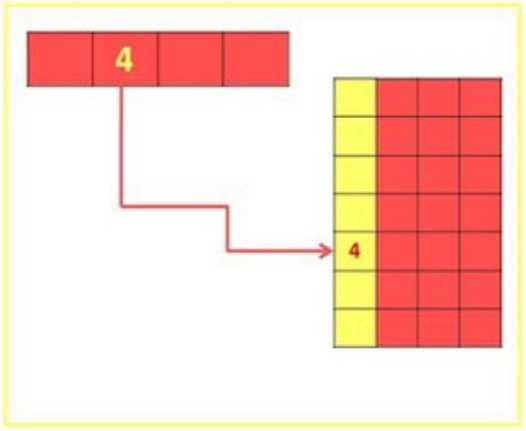
Tree



Graph



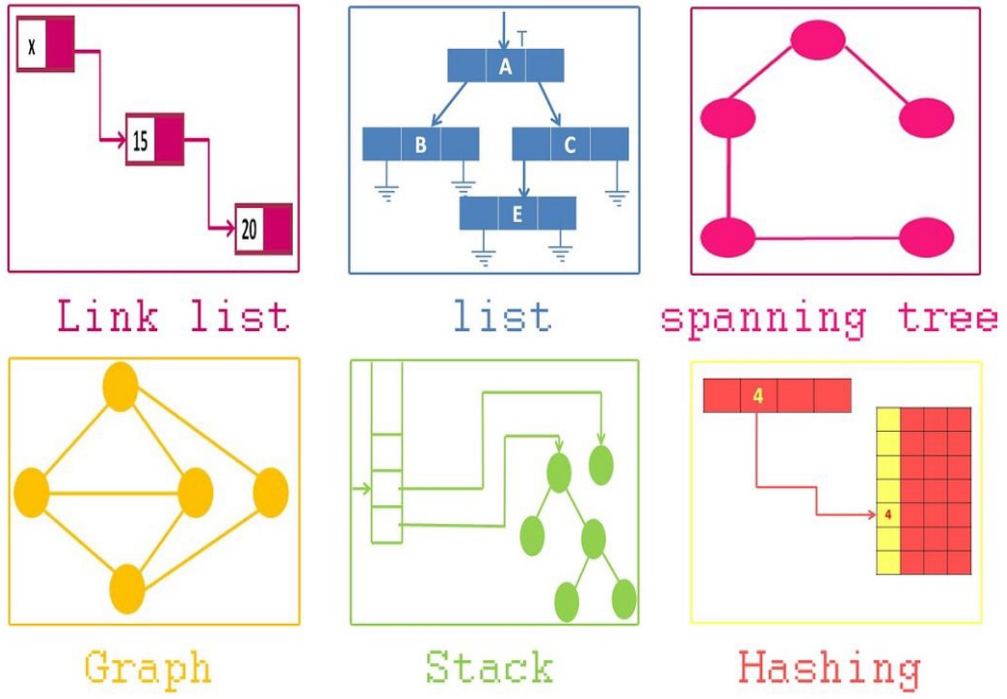
Stack



Hashing

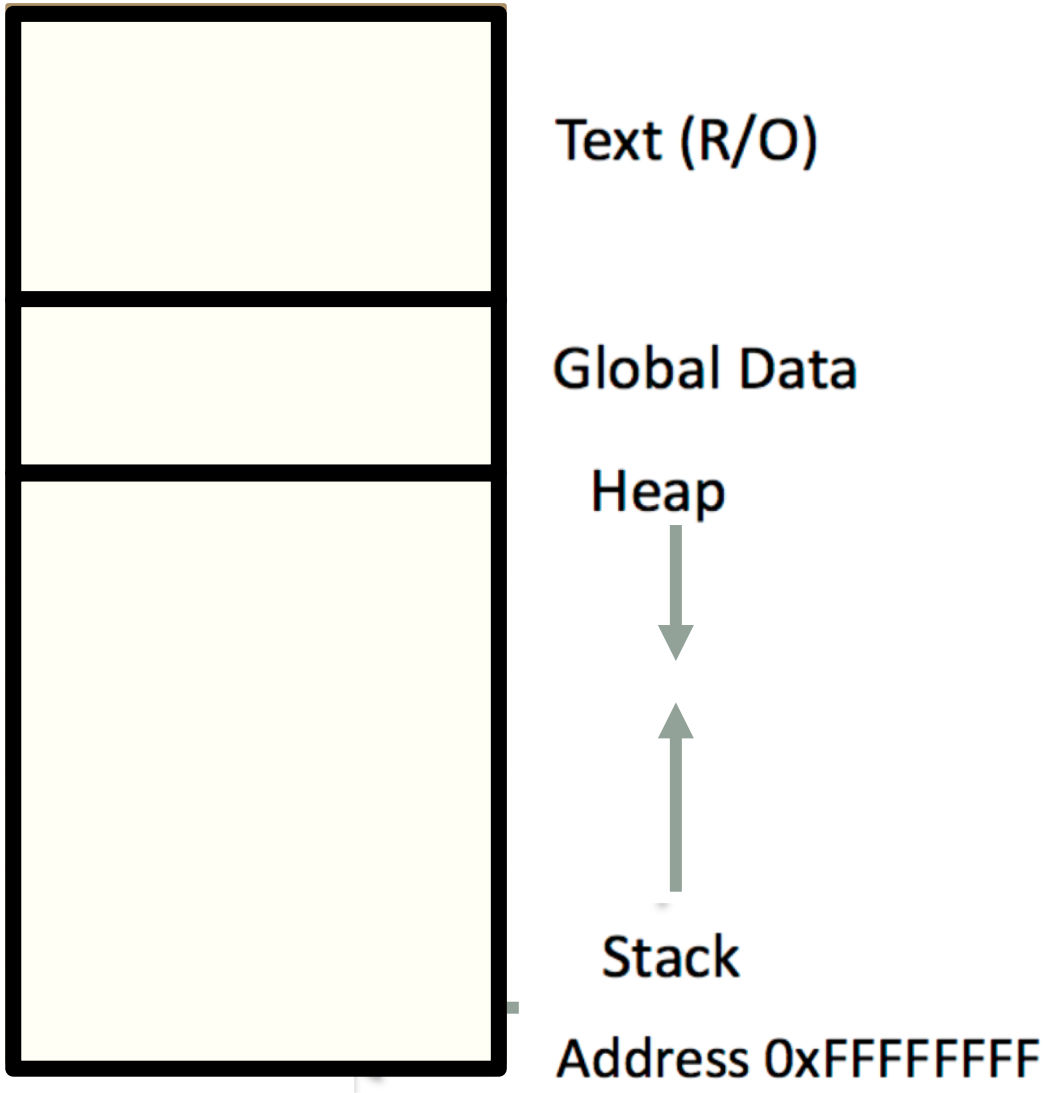
Where are we going? Data structures!!

Arrays



(CS24/32)

It all boils down to 1's and 0's



Dynamic memory management

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

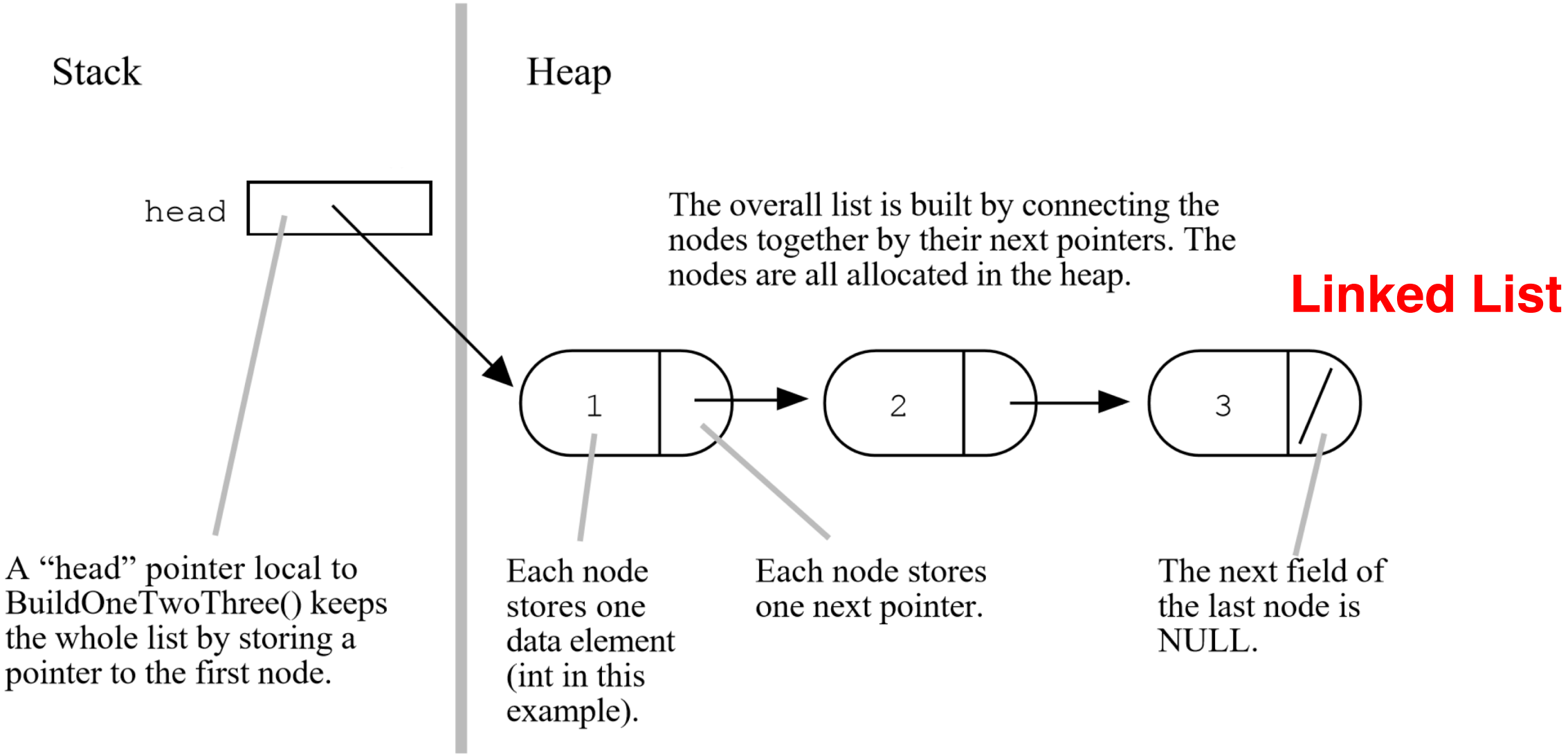
  (data persists in the heap until the programmer frees it)
- To free the memory use **delete**
- Identify the location of all the data (Heap or stack?)

```
int* p= new int; //creates a new integer on the heap
Student* n = new Student; //creates a new Student on the heap
delete p; //Frees the integer
delete n; //Frees the Student
```

Linked Lists

Array List

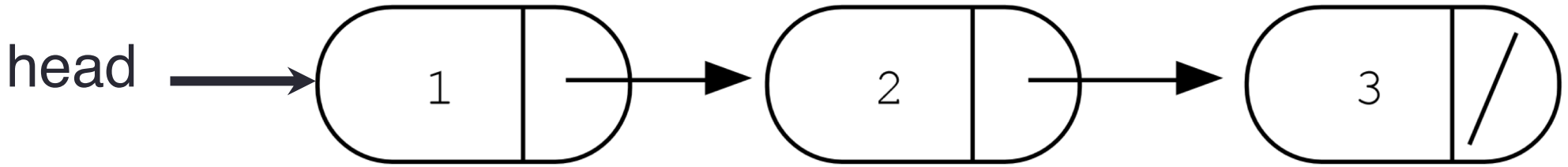
The Drawing Of List {1, 2, 3}



Linked List

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

Create a small list – use only the stack

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

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

Heap vs Stack

```
1 #include <iostream>
2 using namespace std;
3
4 int* createAnInt(){
5
6     int x;
7     return &x;
8
9 }
```

Does the above function correctly create a new integer and return its address? Why or why not?

A. Yes
B. No

Heap vs. stack

```
1 #include <iostream>
2 using namespace std;
3
4 int* createAnIntArray(int len){
5
6     int arr[len];
7     return arr;
8
9 }
```

Does the above function correctly create an array of integers?

A. Yes

B. No

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 create an array of integers?

A. Yes
B. No

Dynamic memory pitfalls

Dangling pointer: Pointer points to a memory location that no longer exists

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
D. Neither

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;

}
```

Next time

- More Linked Lists