And Linked Lists

Stacks

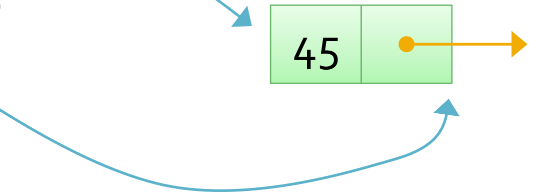
Outline

- Linked lists
- Linked list stack implementation

Linked Lists

A Dream Data Structure

- It would be nice to have a data structure that is
  - Dynamic
  - Performs fast insertions / removals in the middle
- We can achieve this with a linked list
  - A dynamic data structure that consists of nodes linked together
- A *node* is a data structure that contains
  - data
  - the location of the next node

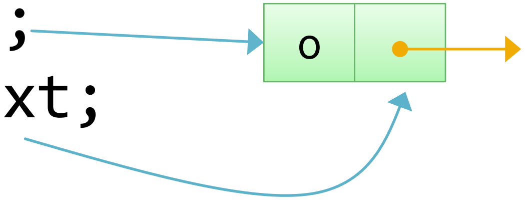


Node Pointers

- A node contains a pointer to the next node
  - Nodes are created in dynamic memory
  - May not be adjacent to the previous node
- The data attribute of a node varies depending on what the node is intended to store

```
class Node {
public:
    int data;
    Node* next;
}
```

Nodes point to other nodes, so the pointer must be of type Node

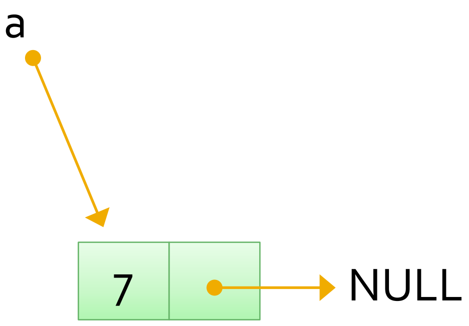


Building a Linked List

```
Node* a = new Node(7, nullptr);
```

Assumes a constructor in the Node class

```
Node(int value, Node* nd){
    data = value;
    next = nd;
}
```

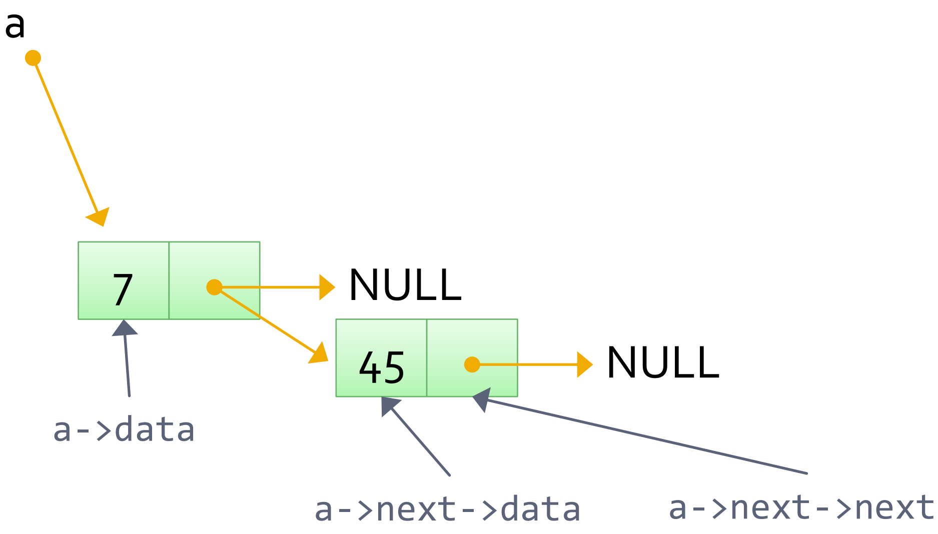


Building a Linked List

```
Node* a = new Node(7, nullptr);
a->next = new Node(45, nullptr);
```

Assumes a constructor in the Node class

```
Node(int value, Node* nd){
    data = value;
    next = nd;
}
```

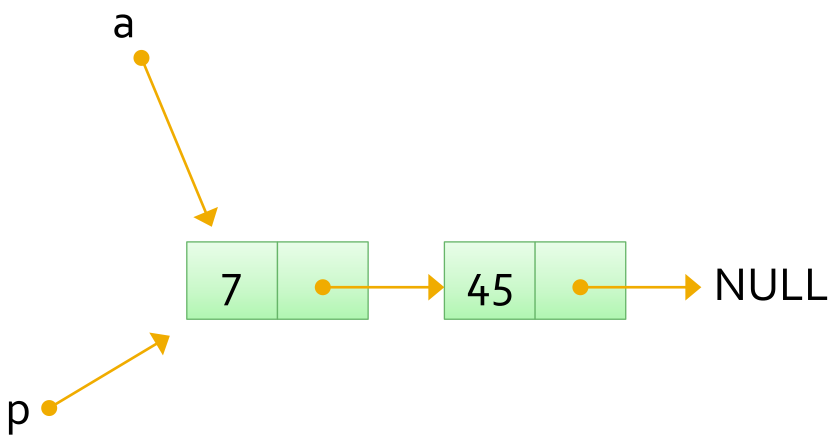


Traversing a Linked List

```
Node* a = new Node(7, nullptr);
a->next = new Node(45, nullptr);
Node* p = a;
```

Assumes a constructor in the Node class

```
Node(int value, Node* nd){
    data = value;
    next = nd;
}
```

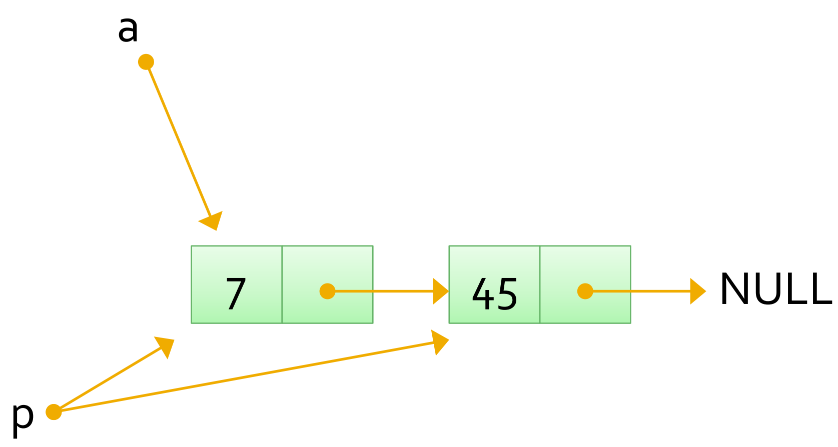


Traversing a Linked List

```
Node* a = new Node(7, nullptr);
a->next = new Node(45, nullptr);
Node* p = a;
p = p->next;  // go to next node
```

Assumes a constructor in the Node class

```
Node(int value, Node* nd){
    data = value;
    next = nd;
}
```

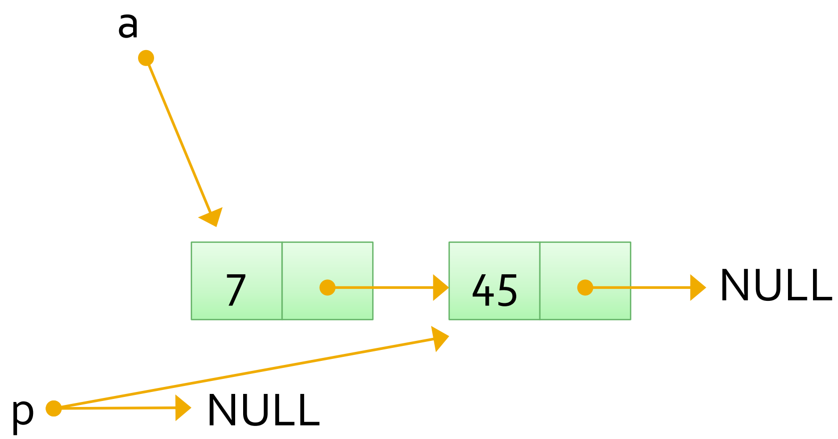


Traversing a Linked List

```
Node* a = new Node(7, nullptr);
a->next = new Node(45, nullptr);
Node* p = a;
p = p->next;  // go to next node
p = p->next;  // go to next node
```

Assumes a constructor in the Node class

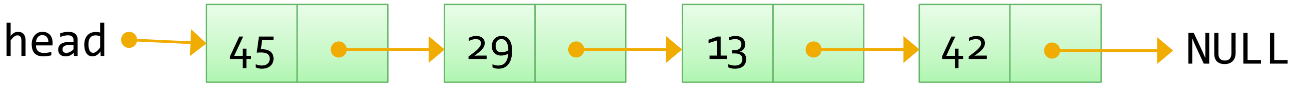
```
Node(int value, Node* nd){
    data = value;
    next = nd;
}
```



In practice insertion and traversal would be methods of a linked list class

Linked Lists

- A linked list is a *chain* of nodes where each node stores the address of the next node



- In practice a linked list is encapsulated in its own class
  - Allowing new nodes to be inserted and removed as desired
  - The linked list class has a pointer to the node at the head of the list
- Implementations of linked lists vary

Implementing a Stack

With a Linked List

Stack: Linked List

- Nodes should be inserted and removed at the head of the list
  - New nodes are pushed onto the front of the list, so that they become the top of the stack
  - Nodes are popped from the front of the list
- Straight-forward linked list implementation
  - Both *push* and *pop* affect the front of the list
    - There is therefore no need for either algorithm to traverse the entire list

Linked List Implementation

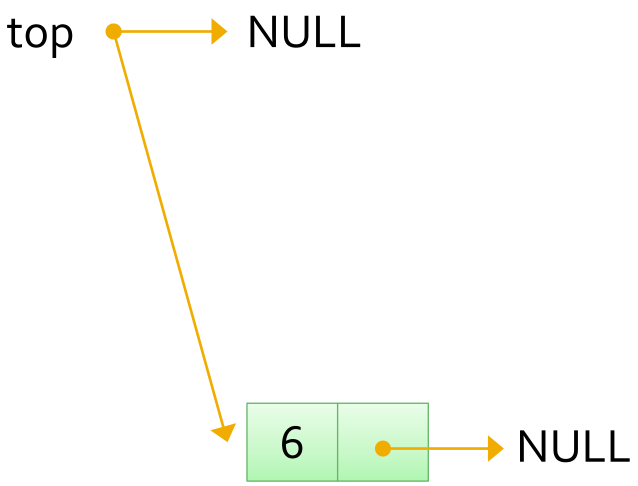
```
void push(int x){
// Make a new node whose next pointer is the
// existing list
      Node* newNode = new Node(x, top);
      top = newNode; //head points to new node
}
```

```
int pop(){
// Return the value at the head of the list
      int temp = top->data;
      Node* p = top;
      top = top->next;
      delete p; // deallocate old head
      return temp;
}
```

What happens if the list to be popped is empty?

List Stack Example

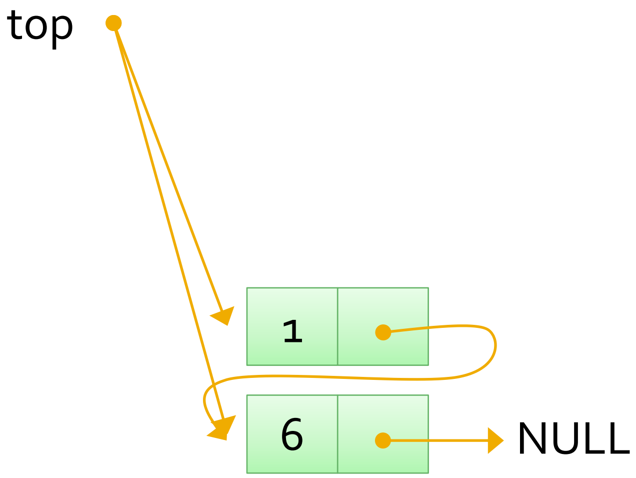
```
Stack st;
st.push(6);
```



```
void push(int x){
        Node* newNode = new Node(x, top);
        top = newNode;
}
```

List Stack Example

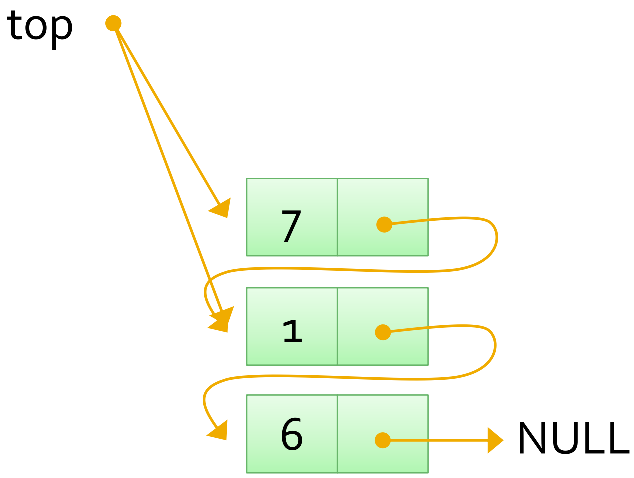
```
Stack st;
st.push(6);
st.push(1);
```



```
void push(int x){
        Node* newNode = new Node(x, top);
        top = newNode;
}
```

List Stack Example

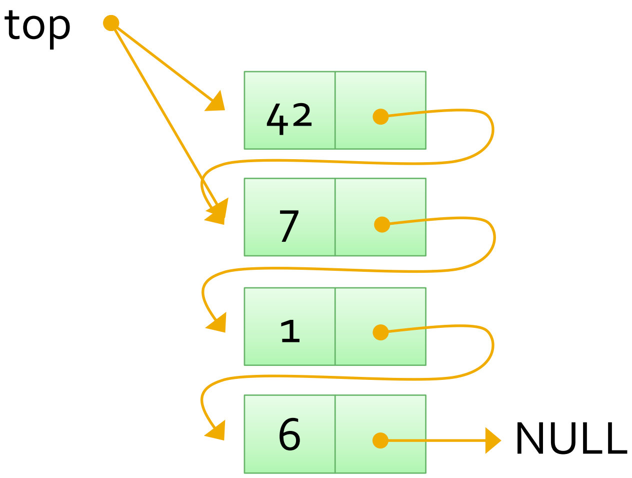
```
Stack st;
st.push(6);
st.push(1);
st.push(7);
```



```
void push(int x){
        Node* newNode = new Node(x, top);
        top = newNode;
}
```

List Stack Example

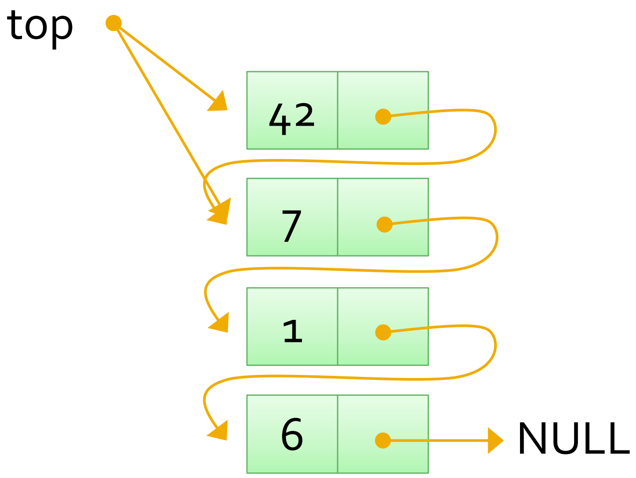
```
Stack st;
st.push(6);
st.push(1);
st.push(7);
st.push(42);
```



```
void push(int x){
	Node* newNode = new Node(x, top);
	top = newNode;
}
```

List Stack Example

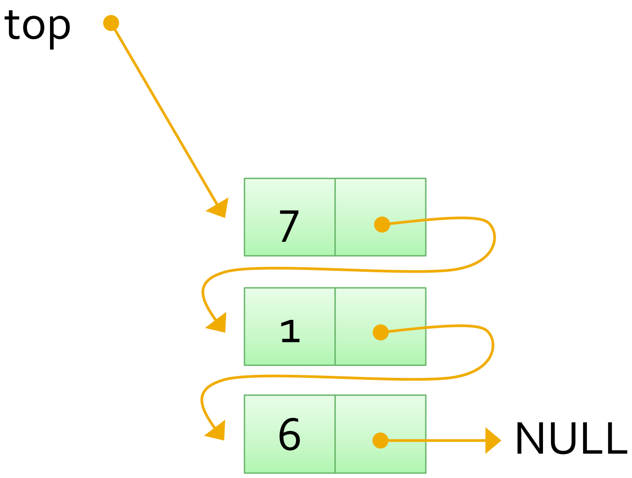
```
Stack st;
st.push(6);
st.push(1);
st.push(7);
st.push(42);
st.pop();
```



```
int pop(){
// Return the value at the head of the list
        int temp = top->data;
        Node* p = top;
        top = top->next;
        delete p; // deallocate old head
        return temp;
}
```

List Stack Example

```
Stack st;
st.push(6);
st.push(1);
st.push(7);
st.push(42);
st.pop();
```



```
int pop(){
// Return the value at the head of the list
        int temp = top->data;
        Node* p = top;
        top = top->next;
        delete p; // deallocate old head
        return temp;
}
```