Data Structures and Algorithms

Stacks and Queues

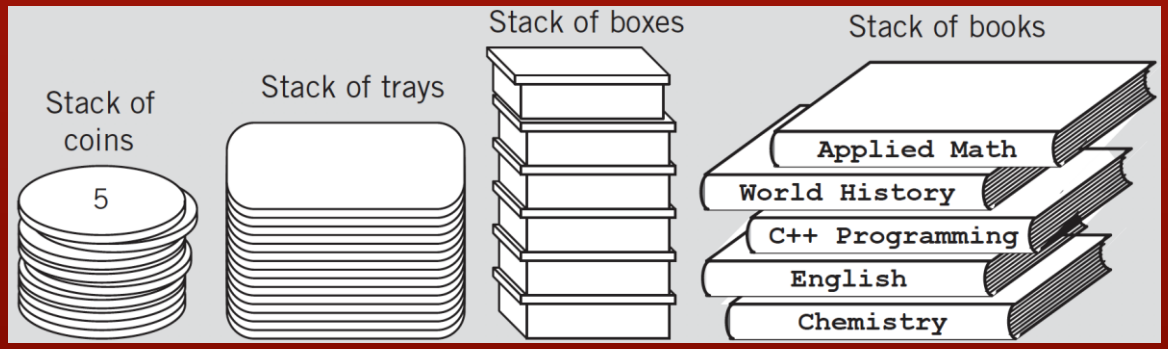
2.2.2 Implementing a Stack Using a Linked List

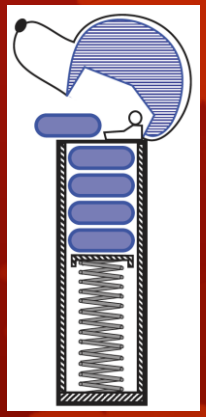
Introduction

- The array implementation of a stack has the advantages of simplicity and efficiency.
- However, one major disadvantage is the need to know the size to declare the array.
- Some reasonable guess has to be made, but this may turn out to be too small (and the program has to halt) or too big (and storage is wasted).
- To overcome this disadvantage, a linked list can be used. Now, we will allocate storage for an element only when it is needed.

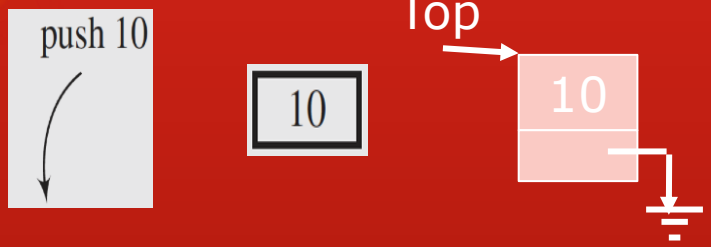
Implementation

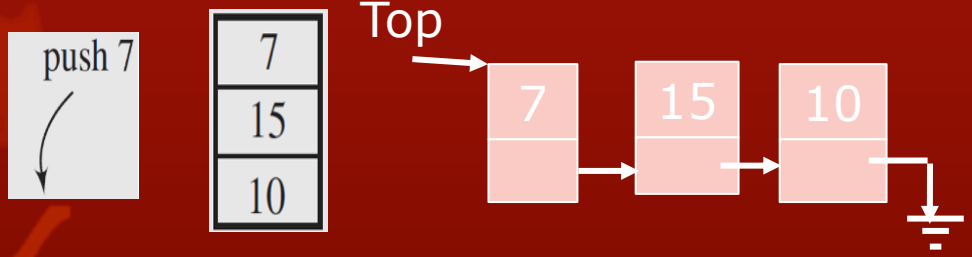
- The stack is implemented as a linked list.
- The main stack operations are:
  - push(*NodeData*): new item added at the head of the list
  - *NodeData* pop(): the item at the head is removed and returned

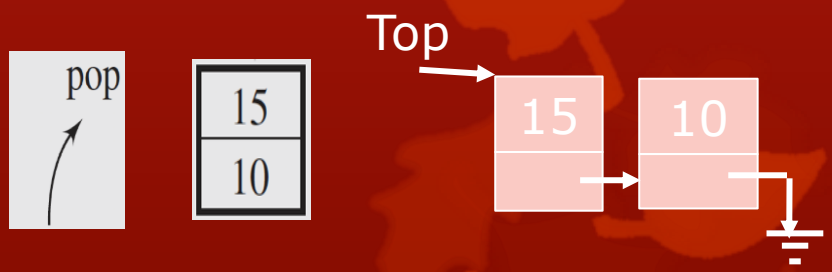




Stack: Linked-List Implementation







Implementation

We will use the same classes NodeData and Node explained in chapter 1.

```
public class Stack
{
    private Node top;

   //create an empty stack
    public Stack ()
    {
        top = null;
    }

    // check if the stack is empty
    public boolean isEmpty ()
    {
        return(top == null);
    }
```

Operation push

Same as addHead() in class LinkedList

```
// Pushes an element at the top
public void push(NodeData item)
{
    Node newNode = new Node(item); // create a new node
    newNode.next = top; // link the new node to the top node
    top = newNode; // make the new node the top node
} // end push
```

Simply adds an item at the top of the stack.

Operation pop()

To pop an item from the stack, we first check whether the stack is empty. If not, the item at the head of the list is returned, and the node containing the item is deleted from the list.

```
// Removes and returns the element at the top
 public NodeData pop()
 {
      if(isEmpty())  // if the list is empty output an error message
      {
          System.out.println("The stack is empty. Top operation failed.");
          System.exit(1);  // stop the program
      }
      NodeData nd = top. data;  // store the top element in nd
      top = top.next; // delete the top node
      return nd; // return the element that was at the top
 }  // end pop
```

Similar to deleteHead() in class LinkedList

Operation top()

This operation returns the top element without removing it.

```
// Returns the element at the top without removing it
public NodeData top()
{
    if(isEmpty()) // if the list is empty output an error message
    {
       System.out.println("The stack is empty. Top operation failed.");
       System.exit(1); // stop the program
    }
    return top.data; // return the top element
}  // end top
```