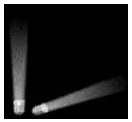
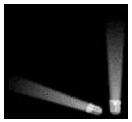
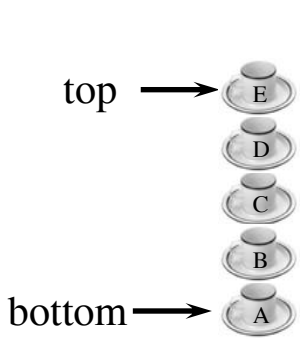
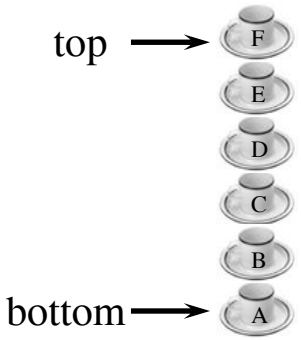
Stacks

- Linear list.
- One end is called top.
- Other end is called bottom.
- Additions to and removals from the top end only.

Stack Of Cups





- Add (Push) a cup to the stack.
- Remove (Pop) a cup from new stack.
- A stack is a LIFO (last in and first out) list.

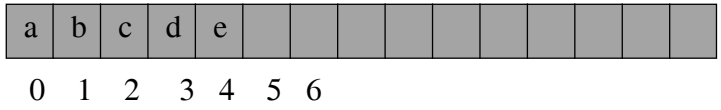
Stacks

- Standard operations:
  - IsEmpty … return true iff stack is empty
  - Top … return top element of stack
  - Push … add an element to the top of the stack
  - Pop … delete the top element of the stack

Stacks

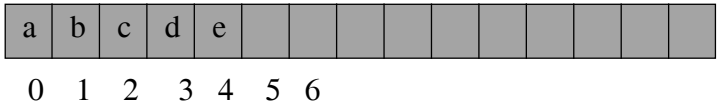
- Use a 1D array to represent a stack.
- Stack elements are stored in stack[0] through stack[top].

Stacks



- stack top is at element e
- IsEmpty() => check whether top >= 0
  - O(1) time
- Top() => If not empty return stack[top]
  - O(1) time

Derive From arrayList



- Push(theElement) => if array full (top == capacity – 1) increase capacity and then add at stack[top+1]
- O(capacity) time when full; otherwise O(1)
- pop() => if not empty, delete from stack[top]
- O(1) time

The Class Stack

```
template<class T>
class Stack
{
   public:
    Stack(int stackCapacity = 10);
    ~Stack() {delete [] stack;}
    bool IsEmpty() const;
    T& Top() const;
    void Push(const T& item);
    void Pop();
   private:
    T *stack;     // array for stack elements
    int top;      // position of top element
    int capacity; // capacity of stack array
};
```

Constructor

```
template<class T>
Stack<T>::Stack(int stackCapacity)
               :capacity(stackCapacity)
{
   if (capacity < 1)
     throw "Stack capacity must be > 0";
   stack = new T[capacity];
   top = -1;
}
```

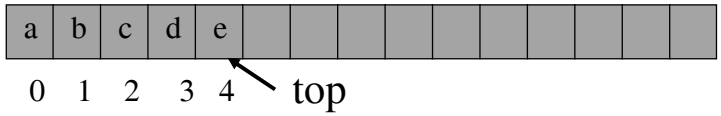
IsEmpty

```
template<class T>
inline bool Stack<T>::IsEmpty() const
   {return top == -1}
```

Top

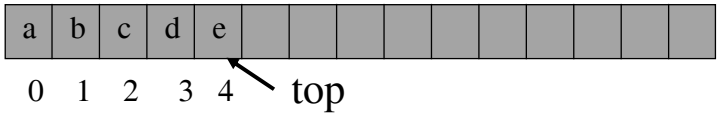
```
template<class T>
inline T& Stack<T>::Top() const
{
   if (IsEmpty())
       throw "Stack is empty";
   return stack[top];
}
```

Push




```
template<class T>
void Stack<T>::Push(const T& x)
{// Add x to the stack.
   if (top == capacity - 1)
      {ChangeSize1D(stack, capacity,
                           2*capacity);
       capacity *= 2;
      }
   // add at stack top
   stack[++top] = x;
}
```

Pop




```
void Stack<T>::Pop()
{
   if (IsEmpty())
       throw "Stack is empty. Cannot delete.";
   stack[top--].~T(); // destructor for T
}
```