Stacks

Stack ADT: Last-In First-Out

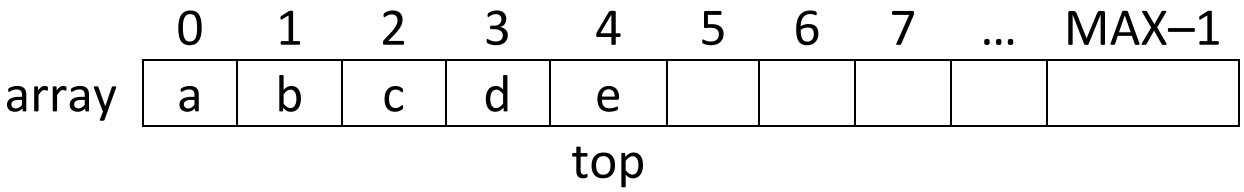
```
void push (ElementType  x)
ElementType  pop( )
-----------------------------------
ElementType  top( )
boolean isEmpty( )
int size( )
```

Standard data structures for Stack ADT:

Array
Singly-linked list

In efficient implementations of a Stack, every operation is O(1) time

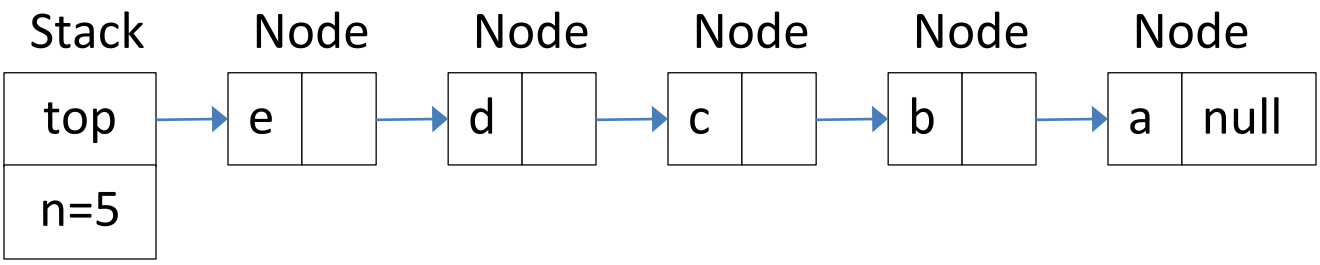
Stack implemented as an Array




```
class Stack {
        ElementType  array[MAX];
        int top;
        Stack( ) { top = –1; }
        void push (ElementType  x) {
                if (isFull( )) throw exception;
                top += 1;
                array[top] = x;
        }
        ElementType  pop( ) {
                if (isEmpty( )) throw exception;
                ElementType x = array[top];
                top –= 1;
                return x;
        }
        ElementType  top( ) {
                if (isEmpty( )) throw exception;
                return array[top];
        }
```

```
    boolean isEmpty( ) { return top == –1; }
    boolean isFull( ) { return top == MAX–1; }
    int size( ) { return top+1; }
}
```

Stack implemented as a Singly-Linked List




```
class Node {
    ElementType  data;
    Node next;
    Node (ElementType x, Node q) { data = x;  next = q; }
}
class Stack {
    Node top;
    int n;
    Stack( ) { top = null;  n = 0; }
    void push (ElementType  x) {
        top = new Node (x, top);
        n += 1;
    }
```

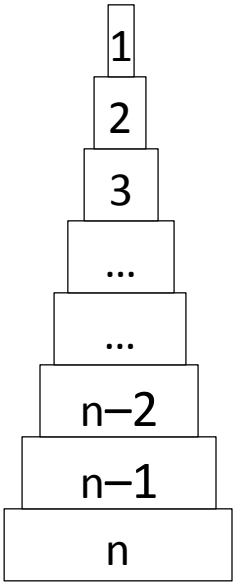
```
        ElementType  pop( ) {
                if (isEmpty( )) throw exception;
                ElementType x = top.data;
                top = top.next;
                n –= 1;
                return x;
        }
        ElementType  top( ) {
                if (isEmpty( )) throw exception;
                return top.data;
        }
        boolean isEmpty( ) { return top == null; }
        int size( ) { return n; }
}
```

Applications of Stacks:

- Read values from input and print them in reverse order

```
void reverse( ) {
        Stack S( );
        while (not endOfFile( )) {
                data = read( );
                S.push (data);
        }
        while (not S.isEmpty( )) {
                data = S.pop( );
                print (data);
        }
}
```

- Towers of Hanoi problem



Initially stack A contains n disks of different sizes.
Initially stacks B and C are empty.
Goal: Move all n disks from stack A to stack B.
But a larger disk can never be placed on a smaller disk.
So use stack C to hold some disks temporarily.

```
void towersOfHanoi (int n, Stack A, Stack B, Stack C) {
        if (n > 1)  towersOfHanoi (n–1, A, C, B);
        int disk = A.pop( );
        B.push (disk);
        if (n > 1)  towersOfHanoi (n–1, C, B, A);
}
```

- Eliminating recursion

```
int fibonnacci (int n) {        // recursive
        if (n <= 1) return 1;
        else return fibonnacci (n–2) + fibonnacci (n–1);
}

int fibonnacci (int n) {        // non-recursive
        Stack S( );
        S.push (n);
        int result = 0;
        while (not  S.isEmpty( )) {
                int k = S.pop( );
                if (k <= 1)
                        result += 1;
                else {
                        S.push (k–2);
                        S.push (k–1);
                }
        }
        return result;
}
```

Compilers translate recursive high-level functions into non-recursive stack-based machine code

- Web browser history using the Back button
- Text editor using the Undo button
- Expression evaluation (later in this course)
- Traversing the nodes of a tree or graph (later in this course)