ALGONQUIN COLLEGE

CST8130 – Data Structures

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

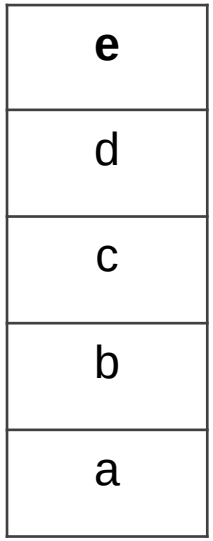
Stacks

Terminology - Stack

- Additions are done to “head” – called “*push*” onto stack
- Deletions are done at “head” – called “*pop*” off stack
- Hence……LIFO (last in, first out)…or FILO (first in, last out)

Why use stacks?

1. Reversing data - add a, b, c, d, e onto a stack



Comes off stack in reverse order…

e, d, c, b, a

Reversing Data - algorithm

```
create a stack
do {
        get an element
        push the element onto the stack
}  while (not end of data && stack not full)
while (stack not empty) {
        pop element from stack
        output element
}
```

Parsing Algorithm

```
create stack
while (more data) {
      get char from keyboard
      if (char is opening bracket or parenthesis)
            push char onto stack
      else if (char is closing bracket or parenthesis) {
            if (stack is empty)
                print – error condition…..closing not matched
            else {    pop char from stack
                  if (popped char does not match current char)
                   print – error condition – no match
                }
            }
} // while
if (stack not empty)
      print – error condition – opening not matched…
```

Using Arrays

```
public class Numbers  {
    protected int maxItems = 10;  // default value
    protected int numItems = 0;    // no elements added yet
    protected int [ ] numbers = new int[maxItems];


      // selected methods here……..
    public boolean add (int newOne) {
        if (numItems == maxItems)
            return false;
          numbers[numItems++] = newOne;
          return true;
    }
    public int deleteAtTop() {
        if (numItems == 0)
                return -1;
        else  return numbers[numItems--];
     }
```

Using Arrays

```
Class Stack   extends Numbers {   // Numbers is from Lab1

        public boolean push (int element) {
              return add(element);
        }

        public int pop () {
               return deleteAtTop()
        }

        public boolean isEmpty() {
                return numItems==0;

        public boolean isFull() {
                 return numItems == maxItems;
        }
}
```

Efficiency for Arrays implementation?

Algorithm

- push O(1) (constant)
- pop O(1) (constant)
- isEmpty O(1) (constant)
- isFull O(1) (constant)

Memory

- One extra reference for each element
- Block of memory of maxSize elements

Using LinkedList

```
public class Stack   extends LList {    // As I published in earlier notes

        public boolean push (int element) {
              return addAtHead(element);
        }

        public int pop () {
               return deleteAtHead()
        }

        public boolean isEmpty() {
                return head == null;

        public boolean isFull() {
                 return false;
        }
}
```

Efficiency for Linked List implementation?

Algorithm

- push O(1) (constant)
- pop O(1) (constant)
- isEmpty O(1) (constant)
- isFull O(1) (constant)

Memory

- One extra reference for each element
- One reference for head

Questions?