Stacks

CS110C
Max Luttrell, CCSF

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces

- example: the user types in "recie<-<-eive" where each "<-" is a backspace

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces

- example: the user types in "recie<-<-eive" where each "<-" is a backspace

r

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

re

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

rec

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

reci

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

recie

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

reci

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

rec

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

rece

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces

- example: the user types in "recie<-<-eive" where each "<-" is a backspace

recei

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces

- example: the user types in "recie<-<-eive" where each "<-" is a backspace

receiv

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces
- example: the user types in "recie<-<-eive" where each "<-" is a backspace

receive

reading in a line of text

- suppose you need to implement some code that inputs a line of text into some ADT from the keyboard, and the user can type backspaces

- example: the user types in "recie<-<-eive" where each "<-" is a backspace

```
//read in a line, allowing mistakes
while (not the end of the line)
{
  read in a character c
  if (c is not a backspace)
    add c to the ADT
  else
    remove the last character from the ADT
}
```

operations
Add an item
Remove last item

reading in a line

- we need to handle the case where the user types in a backspace first:

```
//read in a line, allowing mistakes
while (not the end of the line)
{
  read in a character c
  if (c is not a backspace)
    add c to the ADT
  else if (the ADT is not empty)
    remove the last character from the ADT
  else
    ignore the backspace
}
```

operations
Add an item
Remove last item
Check if empty

writing out the line

- we now have the following line read in: "receive" -- how could we print it out using our operations?
- the pseudocode is an initial guess, but it has some problems...

```
// display the line
while (the ADT isn't empty)
{
  remove from the ADT the item that was added most recently
  display the character // this doesn't work!
}
```

- Problems:
  1. removes the item from ADT before displaying it!
  2. prints the string in reverse

operations
Add an item
Remove last item
Check if empty
Peek at the top

ADT stack

- the ADT we have come up with is a well-known ADT called a **stack,** which is defined by our four operations

stack ADT operations

- is the stack empty?
- add a new item to the stack
- remove the item that was added most recently to the stack
- get the item that was added most recently to the stack (without changing the stack)

Last in - First Out

stack methods

- **`isEmpty(): boolean`**
  - returns true if stack is empty, false if not.
- **`push(newEntry: ItemType): boolean`**
  - put newEntry on the top of the stack
  - returns true if successful, false if not
- **`pop(): boolean`**
  - remove the entry at the top of the stack
  - returns true if successful, false if not
- **`peek(): ItemType`**
  - returns the entry at the top of the stack. does not change the stack.

ADT stack UML

```
Stack
------------------------------------------

------------------------------------------
+isEmpty(): boolean
+push(newEntry: ItemType): boolean
+pop(): boolean
+peek(): ItemType
```

reading in a line pseudocode using Stack

```
// read an input line, handling backspaces
// return a stack with the corrected characters read in
readLine(): Stack
  aStack = a new empty stack
  do
  {
    read newChar
    if (newChar is a backspace)
      aStack.pop()
    else
      aStack.push(newChar)
  } while (newChar is not end-of-line)
  return aStack
```