CS1110

Lecture 23: **Prelim 2 Review Session**

Announcements

All regrade requests for prelim 1 and A2 have been processed and the hardcopies are back in the homework handback room, Gates 216 (open noon-4pm on weekdays; bring ID).

No change in CMS grade means that we elected not to change your grade.

Slides by D. Gries, L. Lee, S. Marschner, W. White

http://www.keepcalm-o-matic.co.uk/p/keep-calm-and-read-the-specs-again-1/

Notes

1. Always carefully read the specs (and class invariants, and ...), and re-read them after finishing a problem. In doubt? *Ask!*
2. Check your code against any examples we give you.
3. When ask you to solve a problem a certain way (i.e., recursively), the intent is for us to see if you understand that implementation method.

   (Ex: don't use a loop if we ask for recursion.)
4. If we don't ask for an invariant, you do not need to provide one.

Provide a *recursive* implementation

```
def merge(s1,s2):
    """Given s1 & s2 strings with characters in alphabetical order,
        return a string equivalent to the sorted concatenation.
      Examples: merge('ab', '') → 'ab'
                merge('abbce', 'cdg') →  'abbccdeg' """
    # Compare characters with =, >, and <.
```

Provide a *recursive* implementation

```
def merge(s1,s2):
    """Given s1 & s2 strings with characters in alphabetical order,
        return a string equivalent to the sorted concatenation.
      Examples: merge('ab', '') → 'ab'
                merge('abbce', 'cdg') →  'abbccdeg' """
    # Compare characters with =, >, and <.

    if s1 == ''  or  s2 == '':
        return s1 + s2
    if s1[0] <=  s2[0]:        # Pick first from s1 and merge the rest
        return s1[0]+merge(s1[1:],s2)
    else:                      # Pick first from s2 and merge the rest
        return s2[0]+merge(s1,s2[1:])
```

Provide a recursive implementation

```
def skip(s):
    """Returns: copy of string s, odd letters (i.e., 1st, 3rd, 5th) dropped.
    Example: 'abcd' -> 'bd'.   '' -> ''   'abc' -> 'b', 'zzz' -> 'z'   """
```

Provide a recursive implementation

```
def skip(s):
    """Returns: copy of string s, odd letters (i.e., 1st, 3rd, 5th) dropped.
    Example: 'abcd' -> 'bd'.   '' -> ''   'abc' -> 'b', 'zzz' -> 'z'   """

     if len(s) <= 1:    # One base case
         return ''
     else:  # s >= 2  characters (if exactly 2, another base case)
         return s[1] + (skip(s[2:]) if len(s) > 2 else '')
```

Provide a for-loop implementation

```
def skip(s):
    """Returns: copy of string s, odd letters (i.e., 1st, 3rd, 5th) dropped.
    Example: 'abcd' -> 'bd'.   '' -> ''   'abc' -> 'b', 'zzz' -> 'z'   """
```

Provide a for-loop implementation

```
def skip(s):
    """Returns: copy of string s, odd letters (i.e., 1st, 3rd, 5th) dropped.
    Example: 'abcd' -> 'bd'.   '' -> ''   'abc' -> 'b', 'zzz' -> 'z'   """

     out = ''  # progress towards output
     # Inv: chars s[0..i-1] have been processed, s[i] is next to check
     for i in range(len(s)): # i in 0..len(s) - 1
         if  i % 2 == 1:
             out += s[i]
     return out
```

Provide a while-loop implementation

```
def skip(s):
    """Returns: copy of string s, odd letters (i.e., 1st, 3rd, 5th) dropped.
    Example: 'abcd' -> 'bd'.   '' -> ''   'abc' -> 'b', 'zzz' -> 'z'   """





     # Inv: chars s[0..i-1] have been processed. Done when i is len(s)
```

Provide a while-loop implementation

```
def skip(s):
    """Returns: copy of string s, odd letters (i.e., 1st, 3rd, 5th) dropped.
    Example: 'abcd' -> 'bd'.   '' -> ''   'abc' -> 'b', 'zzz' -> 'z'   """

    out = ''  # progress towards output
    ~~if len(s) <=1:~~  #  these two lines are optional
    ~~    return out~~
    i = 1
    # Inv: chars s[0..i-1] have been processed. Done when i is len(s)
    while i  < len(s): # don't need parens around loop condition
        out += s[i]
        i += 2
    return out
```

Defining a class

```
class Paper(object):
    """An instance is a  scientific paper.
    Class variables:
    number [int]: number of papers that have been created. >= 0

    Instance variables:
    title [string]: title of this paper.  At least one char long.
    cites [list of Papers]: papers that this paper cites
    cited_by [list of Papers]: papers that this paper is cited by
    """

    number = 0  # initial value is 0
```

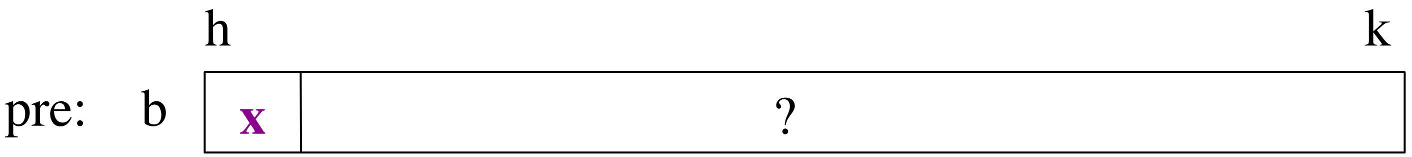
```
def __init__(self, title, cites=None):
 """Initializer.  A new paper with title <title>,  citing the papers in list
<cites> (set to [] if <cites> is None, and should be a copy of <cites>
otherwise), and with cited_by set to []. This initializer should also
update the  relevant attributes of any papers in the list <cites>. Pre:
arg values as in class specification."""
 # Don't forget to update the class variable.
```

Write the body of __init__

```
def __init__(self, title, cites=None):
   # spec on previous slide
    self.title = title

    self.cites = ([] if cites is None else cites[:])
    for p in self.cites:
        p.cited_by.append(self)

    self.cited_by = []

    Paper.number += 1   # note how to reference the class variable.
```

Implement according to invariant

- Given a sequence b[h..k] with some value x in b[h]:

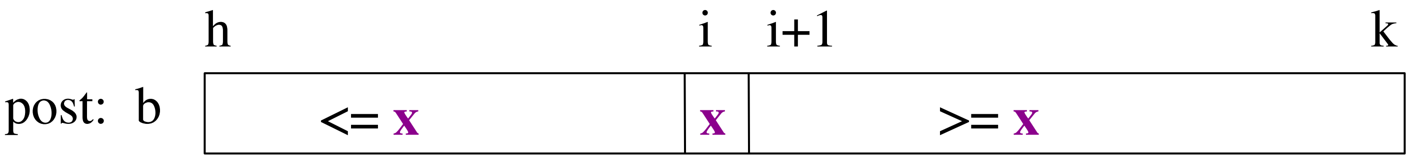


h k

pre: b

x ?

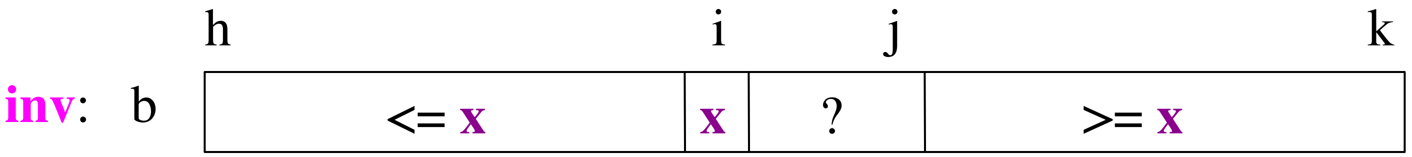
- Swap elements of b[h..k] and store in i to truthify post:



h i i+1 k

post: b

<= x x >= x



h i j k

inv: b

<= x x ? >= x

Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]. Return i s.t. b[i] is x."""

    # invariant: b[h..i-1] <= b[i], b[j+1..k] >= b[i], b[i] is x
```

Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]. . Return i s.t. b[i] is x"""
    i = h; j = k
    # inv: b[h..i-1] <= b[i], b[j+1..k] >= b[i], b[i] is x
    while i < j:
        if b[i+1] >= b[i]:
            # Move to end of block.
            b[i+1], b[j] = b[j], b[i+1]
            j = j - 1
        else:   # b[i+1] < b[i]
            b[i], b[i+1] = b[i+1], b[i]
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

```
def evaluate(p, x): """Returns: The evaluated polynomial p(x).
We represent polynomials as a list of coefficients (as floats):
        [1.5, –2.2, 3.1, 0, –1.0] is 1.5 – 2.2x + 3.1x**2 + 0x**3 – x**4
We evaluate by substituting in for the value x.  For example
  evaluate([1.5,–2.2,3.1,0,–1.0], 2) = 1.5–2.2(2)+3.1(4)–1(16) = –6.5
   evaluate([2], 4) = 2
 Precondition: p is a list (len > 0) of floats, x is a float"""
```

One implementation

```
def evaluate(p, x):
    """(spec on previous slide)"""
    sum = 0     # sum of all the coeffs*x**y for coeffs seen so far
    xval = 1    # x**0 == 1; value to multiply with next coeff yet unseen
    for c in p:  # c is next unseen coefficient
        sum = sum + c*xval
        xval = xval * x
    return sum
```

Alternate implementation

```
def evaluate(p, x):
    """(spec on previous slide)"""
    i=0; sum=0
   # Inv: sum is eval of p[0..i-1], i is next power to do
    while i < len(p):
            sum += p[i]*(x**i)
            i +=  1
    return sum
```

Alternate implementation

```
def evaluate(p, x):
    """(spec on previous slide)"""
    i=0; xval = 1; sum = p[i]  # no point in multiplying by 1; showing
                               # i for clarity; it's not really necessary here
    i = 1
    while i < len(p):
        # Invariant: xval = x**(i-1); sum = eval(p[..i-1], x)
        xval *= x                     # or, xval = xval*x
        sum += p[i]*xval              # or, sum = sum + p[i]*xval
        i +=  1                       # or, i = i + 1
    return sum
```