UNSORTED LINKED LISTS

1. FALL 2007 #2: FIND_REPLACE

   ![Diagram of FIND_REPLACE]

   1: find
   2: replace
   newNode

2. FALL 2006 #2: IS_LIST_EVEN

   ![Diagram of IS_LIST_EVEN]

   1: iterate
   2: b=!b

3. SPRING 2006 #3: REMOVE_EVERY_OTHER_NODE

   ![Diagram of REMOVE_EVERY_OTHER_NODE]

   1: relink
   2: iterate

void rm_every_other(node_ptr p) {
    while (p && p->next) {
        p->next = p->next->next;
        p = p->next;
    }
}

}
5. FALL 2005 #2: REVERSE.DOUBLY.LINKED.LIST (REVISED 4/4/08)

2: relink

tmp

1: relink
3: tmp

6. FALL 2004 #1: REVERSE.SINGLY.LINKED.LIST

2: relink

c
A
F
R
Q

prevPtr
p
1: nextPtr
3: prevPtr
4: p

SORTED LINKED LISTS

1. SPRING 2007 #2: REVERSE_ELEMENTS.DOUBLY.LINKED.LIST

1: swap

3
4
6
8
9

p1
2: iterate
p2
2. SPRING 2003 #2: REVERSE_INTERSECTION

While neither list is exhausted:
   if one value is smaller : increment its pointer
   if both values are equal: push value onto L3, increment both L1, L2

[2]->[9]->[17]
L1
L2

Assume 9 has already been pushed, now push 17:

2: link

17

1: newNode

9

3: L3

L3
3. SPRING 2002 #2: ELIMINATE_DUPLICATES_ACROSS_LISTS

init previous p to L1’s dummy node [*]
increment L1, L2
While neither list is exhausted:
  if L1 value is larger : increment L1 pointer
  if L2 value is larger : increment L2 pointer
    increment p
  if both values are equal: increment both L1, L2
    relink using p
    do NOT increment p

  L1
[*, 17]->[12]->[9]->[3]
  L2
p

  L1
[*, 17]->[12]->[9]->[3]
  L2
p

Since L1, L2 values are both 17, relink around L2’s 17:

2: relink

<table>
<thead>
<tr>
<th>2</th>
<th>12</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: L1</td>
<td>17</td>
<td>L1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>12</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: L1</td>
<td>17</td>
<td>L1</td>
</tr>
</tbody>
</table>

Since L1, L2 values are both 17, relink around L2’s 17:
4. FALL 1999 #2: ELIMINATE_DUPLICATES_WITHIN_LIST

if adjacent values are NOT equal: increment p
if adjacent values ARE equal : relink
    free q
    do NOT increment p

STACK LINKED LISTS
1. FALL 2001 #2: STACK_ADT_AS_LIST

PUSH:
    2: link

    7
    1: newNode
top
    3: top

POP:

    7
top
    2: top
    1: q

    3: free
    4: rtn 7
QUEUE ARRAY LISTS
1. SPRING 2005 #1: CIRCULAR_QUEUE_ASARRAY

iterate from front to rear looking for x
need a successor function to handle both cases: i++ with wrap-around

```
   7 1 4

  front  rear
```

```
5 1 6 2

  rear  front
```

GENERAL TREES
1. FALL 2001 #1: IS_TREE_BINARY

iterate across child list:
  count and if > 2 then abort
  recursion on subtree
  if not binary then abort
success

```
A
| v
v
B---C---D
|   | v
v v
E->F G
```
**BINARY TREES**

1. **FALL 2007 #3: COPY_TREE**

   ![Binary Tree Diagram]

   - 7
   - subtree A
   - subtree B
   - copy of A
   - copy of B
   - p
   - new p

3. **FALL 2006 #3: IS_TREE_ALLSAME**

   1: left does not exist
   - right exists and agrees
   - continue with recursion
   2: left and right exist and agree
   - continue with recursion
   3: left and right exist but do NOT agree
   - abort

   ![Binary Tree Diagram for IS_TREE_ALLSAME]

   - 4
   - 4
   - 4
   - 4
   - 4
   - 2
   - 5
   - 5
5. SPRING 2005 #2: SWAP_WITH_LEFT_CHILD

```
A                  B     B
/ \                / \    / \    / \  
/   \              /   \  /   \  /   
B     G  ... ==>  ... A     G  ... ==>  ... C     G
/ \   \               / \   \               / \   \  
C     D     H        C     D     H        A     E     I
/ \   /               / \   /               / \   /  
E     F     I        E     F     I        D     F     H
```

Initial Tree  Intermediate Tree  Final Tree

swap A and B
recursion left, recursion right

6. SPRING 2004 #2: FIND_PARENT_OF_X

left exists but is NOT x
right exists and IS x
otherwise recursion A
if not found recursion B
7. **SPRING 2003 #1: COUNT_INTERIOR_NODES**

1: neither left or right exist: NOT interior
2: otherwise node IS interior: 1 + A + B

```
2:  p
   |  
   V  
subtree A  subtree B
```

8. **FALL 2002 #1: IS_TREE_EVEN**

A, B EVEN: subtree p is ODD
A, B ODD : subtree p is ODD
A \(\leftrightarrow\) B  : subtree p is EVEN

```
p
   |  
   V  
subtree A  subtree B
```

9. **SPRING 2000 #1: NO_LEFT_CHILDREN**

1: found a LEFT : FALSE
2: never found a LEFT: TRUE

```
1:  
   |  
   V  
2:  
   |  
   V  
```
10. SPRING 1999 #2: COUNT_FULL_NODES

1: A AND B exist : 1 + A + B
2: A OR B do NOT exist: 0 + A + B

Alternatively: COUNT_LEAF_NODES:

```c
int count_leaf_nodes(BINARY_TREE T) {
    if (T == NULL)
        return 0;
    else
        if (T->left == NULL && T->right == NULL)
            return 1;
        else
            return count_leaf_nodes(T->left) + count_leaf_nodes(T->right);
}
```