CS 241
Data Organization
Solution for Lab 5: Linked Lists and Binary Trees

Brooke Chenoweth

University of New Mexico

Fall 2014

Linked List

```
struct ListNode
{
  int data;
  struct ListNode* next;
};
```

createNode

```
struct ListNode* createNode(int data)
{
  struct ListNode* node =
     malloc(sizeof(struct ListNode));
  node->data = data;
  node->next = NULL;
}
```

insertSorted

```
struct ListNode* insertSorted(struct ListNode* head, int data)
{
  struct ListNode* current = head;
  struct ListNode* newNode = createNode(data);
  if(current == NULL || data < current->data)
  {
    newNode->next = current;
    return newNode;
  }
  else
  {
    while(current->next != NULL &&
          current->next->data < data)
    {
      current = current->next;
    }
    newNode->next = current->next;
    current->next = newNode;
    return head;
  }
}
```

insertSorted – recursive

```
struct ListNode* insertSorted(struct ListNode* head,
                              int data)
{
  struct ListNode* newNode;
  if(head == NULL || data < head->data)
  {
    newNode = createNode(data);
    newNode->next = head;
    return newNode;
  }
  else
  {
    head->next = insertSorted(head->next, data);
    return head;
  }
}
```

removeItem

```
int removeItem(struct ListNode** headRef, int data)
{
  struct ListNode* node;
  if(*headRef == NULL)
  {
    return 0;
  }
  else if((*headRef)->data == data)
  {
    node = *headRef;
    *headRef = (*headRef)->next;
    free(node);
    return 1;
  }
  else
  {
    return removeItem(&((*headRef)->next), data);
  }
}
```

push

```
struct ListNode* push(struct ListNode* head,
                      int data)
{
  struct ListNode* newNode = createNode(data);

  newNode->next = head;
  return newNode;
}
```

pop

```
int pop(struct ListNode** headRef)
{
  struct ListNode* node = *headRef;
  int data = node->data;
  *headRef = node->next;
  free(node);
  return data;
}
```

listLength

```
int listLength(struct ListNode* head)
{
  struct ListNode* current = head;
  int length = 0;
  while(current != NULL)
  {
    length++;
    current = current->next;
  }
  return length;
}
```

printList

```
void printList(struct ListNode* head)
{
  struct ListNode* current = head;

  while(current != NULL)
  {
    printf("%d ", current->data);
    current = current->next;
  }
  printf("\n");
}
```

freeList

```
void freeList(struct ListNode* head)
{
  struct ListNode* current = head;

  if(current != NULL)
  {
    freeList(current->next);
    current->next = NULL;
    free(current);
  }
}
```

reverseList

```
void reverseList(struct ListNode** headRef)
{
  struct ListNode* first;
  struct ListNode* rest;
  if(*headRef != NULL)
  {
    first = *headRef;
    rest = first->next;
    if(rest != NULL)
    {
      reverseList(&rest);
      first->next->next = first;
      first->next = NULL;
      *headRef = rest;
    }
  }
}
```

Binary Tree

```
struct TreeNode
{
  int data;
  struct TreeNode* left;
  struct TreeNode* right;
};
```

createNode

```
struct TreeNode* createNode(int data)
{
   struct TreeNode* node =
      malloc(sizeof(struct TreeNode));
   node->data = data;
   node->left = NULL;
   node->right = NULL;
}
```

insertBST

```
struct TreeNode* insertBST(struct TreeNode* root,
                           int data)
{
  if(root == NULL)
  {
    root = createNode(data);
  }
  else if(data < root->data)
  {
    root->left = insertBST(root->left, data);
  }
  else
  {
    root->right = insertBST(root->right, data);
  }
  return root;
}
```

minValueBST

```
int minValueBST(struct TreeNode* node)
{
  if(node == NULL)
  {
    printf("NULL node in minValue!\n");
  }
  else if(node->left == NULL)
  {
    return node->data;
  }
  else
  {
    return minValueBST(node->left);
  }
}
```

removeBST – Part 1

```
int removeBST(struct TreeNode** rootRef, int data)
{
  struct TreeNode* node;
  if(*rootRef == NULL)
  {
    return 0;
  }
  else if(data < (*rootRef)->data)
  {
    return removeBST(&((*rootRef)->left), data);
  }
  else if(data > (*rootRef)->data)
  {
    return removeBST(&((*rootRef)->right), data);
  }
  else /* Found node to remove, see next slide */
}
```

removeBST – Part 2

```
  else
  {
    if((*rootRef)->left == NULL)
    {
      node = *rootRef;
      *rootRef = (*rootRef)->right;
      free(node);
    }
    else if ((*rootRef)->right == NULL)
    {
      node = *rootRef;
      *rootRef = (*rootRef)->left;
      free(node);
    }
    else
    {
      (*rootRef)->data = minValueBST((*rootRef)->right);
      return removeBST(&((*rootRef)->right), (*rootRef)->data);
    }
    return 1;
  }
}
```

maxDepth

```
int maxDepth(struct TreeNode* root)
{
  int leftDepth, rightDepth;
  if(root == NULL)
  {
    return 0;
  }
  else
  {
    leftDepth = maxDepth(root->left);
    rightDepth = maxDepth(root->right);
    return 1 + ((leftDepth > rightDepth)
                ? leftDepth : rightDepth);
  }
}
```

isBalanced – Inefficient

```
int isBalanced(struct TreeNode* root)
{
  int balanceFactor = 0;
  if(root == NULL) return 1;
  else
  {
    balanceFactor =
      maxDepth(root->left) - maxDepth(root->right);
    if(-1 <= balanceFactor && balanceFactor <= 1
       && isBalanced(root->left)
       && isBalanced(root->right))
    {
      return 1;
    }
    else return 0;
  }
}
```

isBalanced – Traverses once

```
int isBalancedHelp(struct TreeNode* root, int* height)
{
  int leftH, rightH, leftBalanced, rightBalanced, balanceFactor;
  if(root == NULL)
  {
    *height = 0;
    return 1;
  }
  else
  {
    leftBalanced = isBalancedHelp(root->left, &leftH);
    rightBalanced = isBalancedHelp(root->right, &rightH);
    balanceFactor = leftH - rightH;
    *height = 1 + ((leftH > rightH) ? leftH : rightH);
    return leftBalanced && rightBalanced
      && -1 <= balanceFactor && balanceFactor <= 1;
  }
}
int isBalanced(struct TreeNode* root)
{
  int height = 0;
  return isBalancedHelp(root, &height);
}
```

isBST

```
int isBSThelper(struct TreeNode* root,
                int min, int max)
{
  if(root == NULL) return 1;
  else if(root->data < min) return 0;
  else if(root->data > max) return 0;
  else return
         isBSThelper(root->left, min, root->data) &&
         isBSThelper(root->right, root->data, max);
}

int isBST(struct TreeNode* root)
{
  return isBSThelper(root, INT_MIN, INT_MAX);
}
```

printTree

```
void printTreeHelper(struct TreeNode* root)
{
  if (root != NULL)
  {
    printTreeHelper(root->left);
    printf("%d ", root->data);
    printTreeHelper(root->right);
  }
}

void printTree(struct TreeNode* root)
{
  printTreeHelper(root);
  printf("\n");
}
```

printLeaves

```
void printLeavesHelper(struct TreeNode* root)
{
  if(root != NULL)
  {
    printLeavesHelper(root->left);
    printLeavesHelper(root->right);
    if(root->left == NULL && root->right == NULL)
    {
      printf("%d ", root->data);
    }
  }
}

void printLeaves(struct TreeNode* root)
{
  printLeavesHelper(root);
  printf("\n");
}
```

freeTree

```
void freeTree(struct TreeNode* root)
{
  if(root != NULL)
  {
    freeTree(root->left);
    freeTree(root->right);
    free(root);
  }
}
```