ECE220 S21 Answer

Problem 1

Given the head of a linked list, and an integer k. Write a C function that removes the kth node (the list is 1-indexed) from the head and insert the removed node as the (2k)-th node of the linked list. Assume k > 1 and the length of the linked list is larger than k. If the length of the linked list is smaller than 2k, please insert the removed node to the end of the linked list.

```c
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>

typedef struct ListNode node;
struct ListNode{
    int data;
    struct ListNode* next;
};

void KthNode(node* head, int k){
    node* cur = head;
    int i;
    // Traverse the list to find the (k-1)-th node
    for (i = 1; i < k - 1; i++){
        cur = cur->next;
    } // 如果k = 2的话，这里就不会执行，但是k = 2意味着是第二个节点，所以下面要写cur->next而不是cur

    // Save the pointer that points to the (k)-th node
    node* k_ptr = cur->next;

    // Continue to traverse the list to find the right place for inserting the (k)-th node
    i = 0;
    while (cur->next != NULL){
        cur = cur->next;
        i = i + 1;
        if (i == k){
            break;
        }
    }
    // insert the (k)-th node after "cur" node
    node* tmp = cur->next;
    cur->next = k_ptr;
    k_ptr->next = tmp;
}
```

Problem 2

Already Known:

```
// tree node
struct Node{
    int data;
    struct Node *left, *right;
};
```

Part A: Given a binary tree, write a recursive C function to find its largest node and return a pointer to it. Please note that the tree does not have duplicate nodes.

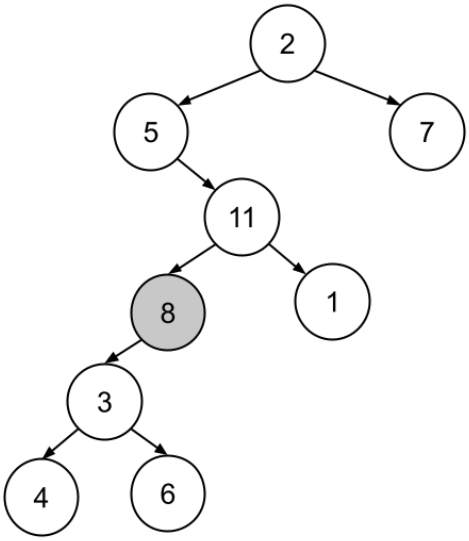
```
struct Node* findMax(struct Node* root){
    // Base case
    if (root == NULL){
        return NULL;
    }

    int lres = 0;
    int rres = 0;

    // Return maximum of 3 values: 1) Root's data, 2) Max in Left Subtree, 3) Max in right subtree
    int res = root->data;

    // Recursion for the left subtree
    struct Node* left_max = findMax(root->left);
    if (left_max != NULL){
        lres = left_max->data;
    }

    // Recursion for the right subtree
    struct Node* right_max = findMax(root->right);
    if (right_max != NULL){
        rres = right_max->data;
    }

    // Compare lres, rres and res then return the ptr to the max node
    // If the left subtree max node's data is greater than that of the right subtree's max and root
    if (lres > rres && lres > res){
        return left_max;
    }
    // If the right subtree max node's data is greater than that of the left subtree's max and root
    if (rres > lres && rres > res){
        return right_max;
    }
    return root;
}
```

Part B: Write a C function to insert a new node as the left child of the largest node (found in Part A). If there already exists a left child, insert the new node as the right child. If the left and right child already exists, insert the new node as a left child anyway. Move the existing left subtree of the largest node to be the left child of the new node that is inserted.

```
/* Assume helper function ‘struct Node* newnode(int val);’ is given and can be used to dynamically allocate a new
node, set its data to val, and left and right pointers to NULL */
void insertNode(struct Node* max_node, int val){
```

```
    if (max_node == NULL){
        printf("Cannot Insert New Node!");
        return;
    }

    // Case 1: max_node doesn't have a left child
    if (max_node->left == NULL){
        max_node->left = newnode(val);
        printf("Node inserted as left child.\n");
        return;
    }

    // Case 2: max_node has a left child but no right child
    if (max_node->right == NULL){
        max_node->right = newnode(val);
        printf("Node inserted as right child.\n");
        return;
    }

    // Case 3: max_node has both left and right children
    struct Node* temp = max_node->left;
    max_node->left = newNode(val);
    max_node->left->left = temp;
    printf("Node inserted as left child. Inserted node now has a left subtree.\n");
}

/* Main Function */
int main(){
    /* creation of the tree is omitted*/
    /* Function call for part A. Assume root to be the root of the binary tree */
    struct Node* max_node=findMax(root);
    if (root!=NULL){printf("Maximum node is %d \n", max_node->data);}
    /* Function call for part B. Assume 8 is to be inserted */
    insertNode(max_node, 8);
    return 0;
}
```

Part C: Suppose the binary tree after insertion looks like this:



What sequence would an in-order traversal (left, root, right) produce?

Answer: 5, 4, 3, 6, 8, 11, 1, 2, 7

Problem 3

In the following, **alternateList(struct Node* head)** represents a C function that prints alternate nodes of the given Linked List, first from head to tail, and then from tail to head. If Linked List has an even number of nodes, then **alternateList()** skips the last node.

Examples:

If given a linked list: 1->2->3->4->5, it prints 1 3 5 5 3 1.

If given a linked list: 1->2->3->4->5->6, it prints 1 3 5 5 3 1.

Replicate the given C function **alternateList(struct Node* head)** using LC3. Code snippet is provided. You just need to fill in the blanks.

```
/* A Linked List Node */
struct Node{
    int data;
    struct Node *next;
};
// Recursive function - this function has no local variables
void alternateList(struct Node* head){
    if (head == NULL){return;}
    printf("%d ", head->data);
    if(head->next != NULL){alternateList(head->next->next);}
    printf("%d ", head->data);
}
```

You **MUST** use and conform to the **LC-3 calling conventions** we have described **in class**. You may assume each data type will only occupy one memory location in LC-3.

```
; R6 is the stack pointer. R5 is the frame pointer.
; R7 contains the return address.
ALTERNATE_LIST
```

```
; complete callee set-up (push bookkeeping information and
; local variables) and comment on each line
    ADD R6, R6, #-3 ; reserve spaces (update stack pointer)
    STR R7, R6, #1 ; store return address
    STR R5, R6, #0 ; store caller's frame pointer
    ADD R5, R6, #-1 ; update frame pointer

    ; Load head to R0
    LDR R0, R5, #4 ; 此时R5在CFP上面一格，要获取head就要+4
    BRz FINISH

; printing omitted

; Check if head->next is NULL
    LDR R2, R0, #1 ; load head->next to R2
    BRz PRINT

RECURSIVE
    LDR R4, R2, #1 ; load head->next->next to R4
    ;Complete caller set-up (push arguments)
    STR R4, R6, #-1 ; store argument to run-time stack
    ADD R6, R6, #-1 ; update stack pointer
    ; 此时R6位于参数的位置，所以和初始状态一致
    ; make recursive call
    JSR ALTERNATE_LIST
    ; caller tear-down omitted

PRINT ; printing omitted

FINISH
; callee teardown
    LDR R5, R6, #1 ; restore caller's frame pointer
    LDR R7, R6, #2 ; restore return address
    ADD R6, R6, #3 ; update stack pointer
    RET
```

Problem 4

The following simple C++ program declares a class, creates some objects, prints them, and performs a “+” operation. Fill out the blanks in main.cpp for the program to match the provided output.

The expected output should look like the following:

```
This house has 5 rooms.
This house has 10 rooms.
This house has 15 rooms.
```

main.cpp

```
#include<iostream>
using namespace std;
class house {
```

```
int roomCount;
public :
    house(){}
    house(int r){
        this->roomCount = r;
    }
    int getRoomCount(void){
        return this->roomCount;
    }
    void setRoomCount(int newRoomCount){
        this->roomCount = newRoomCount;
    }
    void houseInfo(){
        cout << "This house has " << roomCount << " rooms." << endl;
    }
    house operator + (const house& other){
        int r1 = this->roomCount;
        int r2 = other.roomCount;
        return house(r1 + r2);
    }

int main() {
    house a(5);
    house b(10);
    a.houseInfo();
    b.houseInfo();
    house c = a+b;
    c.houseInfo();
    return 0;
}
```