Exercise 3

csd4406 Mike Bastakis

ASK1.1

```
int isLeaf(struct node *node){
        if(node->lc==NULL && node->rc==NULL){
                return 1;
        }
        return 0;
}
int Depth(struct node *node){
        struct node *temp=node;
        int depth = 0;
        while(Parent(temp)!=NILL){
                temp = Parent(temp);
                depth++;
        }
        return depth;
}

int Height(struct node *node){
        if(node==NULL){
                return 0;
        }
        int lc_height = Height(node->lc);
        int rc_height = Height(node->rc);
        if(lc_height > rc_height){
                return lc_height+1;
        }else{
```

```
                return rc_height+1;
        }
}
struct node* Parent(struct node *node){
        return node->parent;
}

int PerfectLeaves(struct node* node,int height){

        if(node==NULL){
                return 0;
        }
        if(isLeaf(node)&&(height-Depth(node)==1 || Parent(nod
e)==NULL)){

                return 1;
        }
        int lc = PerfectLeaves(node->lc,height);
        int rc = PerfectLeaves(node->rc,height);
        return lc+rc;
}
/*Checks if the Binary Tree is perfect*/
bool PerfectBT(struct node* node){

        int tree_height=Height(node)-1;
        int perfect_leaves = PerfectLeaves(node,tree_height+
1);
        int expected_leaves = 1;
        for(int i=0;i<tree_height;i++){
                expected_leaves = expected_leaves * 2;
```

```
        }
        return expected_leaves == perfect_leaves;
}
```

ASK1.2

```
struct node{
        struct node *lc;
        int data;
        struct node *rs;
};

int expected_degree = -1;
bool checkDegree(struct node *node,int k){
        static int depth = 0;
        int degree = 0;


        if(node==NULL) return true;

        if(k==depth){
                struct node *temp = node->lc;
                while(temp!=NULL){
                        temp=temp->rs;
                        degree++;
                }

                if(expected_degree==-1) expected_degree=degre
e;
                else if(degree!=expected_degree) return false;
```

```
        }
        depth++;
        bool lc = checkDegree(node->lc,k);
        depth--;
        bool rs = checkDegree(node->rs,k);
        return lc&&rs;
}
```

ASK2.1

```
struct node{
        int key;
        int info;
        int cnt;
        struct node* lc;
        struct node* rc;
};

int CountOfNodes(struct node *node){
        if(node==NULL) return 0;
        return CountOfNodes(node->rc)+node->cnt+1;
}
```

ASK2.2

```
int insertBST(struct node* node,int key,int info){
        if(node==NULL){
                return 1;
        }

        if(node->key>key){
                if(insertBST(node->lc,key,info)){
```

```
                        node->lc=newNode(key,info);
                        node->lc->cnt=0;
                }
                else node->cnt++;
        }else if(node->key<key){
                if(insertBST(node->rc,key,info)){
                        node->rc=newNode(key,info);
                        node->rc->cnt=0;
                }
        }
        else if(node->key==key){
                node->info=info;
        }

        return 0;
}
```

ASK2.3

ASK3

```
struct node{
        int key;
        int info;
        struct node* lc;
        struct node* rc;
};

struct node* newNode(int key,int info){
        struct node *newNode = (struct node*)malloc(sizeof(str
uct node));
```

```
        newNode->info=info;
        newNode->key=key;
        newNode->lc=NULL;
        newNode->rc=NULL;

        return newNode;
}

void storeToArray(struct node* node, int arr[])
{
        static int i = 0;
    if (node == NULL) return;

    storeToArray(node->lc, arr,i);
    arr[i++] = node->info;
    storeToArray(node->rc,arr,i);
    return;
}

int *mergeArrays(int a[],int b[],int size_a,int size_b){
        int *merge = (int*)malloc(sizeof(int)*(size_a+size_
b));
        int i=0,j=0,k=0;
        while(i<size_a && j<size_b){
                if(a[i] > b[j]) merge[k++] = a[i++];
                else merge[k++] = b[j++];
        }
        while(i<size_a) merge[k++] = a[i++];
        while(j<size_b) merge[k++] = b[j++];
```

```
        return merge;
}

struct node* arrayToBST(int start,int end,int[] nums){
    if(start<=end){
        int middle=(start+end)/2;
        struct node* node=newNode(nums[middle]);
        node.left=sortedArrayToBST(start,middle-1,nums);
        node.right=sortedArrayToBST(middle+1,end,nums);
        return node;
    }else{
        return null;
    }
}

struct node* mergeBST(struct node* T1,struct node* T2,int T1_s
ize, int T2_size){
        int* a = malloc(sizeof(int)*T1_size);
        int* b = malloc(sizeof(int)*T2_size);
        storeToArray(T1,a);
        storeToArray(T2,b);

        int* merge= mergeArrays(a,b,T1_size,T2_size);
        return arrayToBST(0,T1_size+T2_size-1,merge);
}
```

ASK4

```
int findBlackHeight(struct node* node){
        if(node==NULL) return 0;
```

```
        int lc = findBlackHeight(node->lc);
        int rc = findBlackHeight(node->rc);

        if(lc==-1 || rc==-1 || lc!=rc) return -1;
        else return lc + node->color==BLACK ? 1 : 0;
}

bool checkBlackHeight(struct node* root){
      return findBlackHeight(root) != 1;
}
```