Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

Informatics II
Exercise 8 / **Solution**

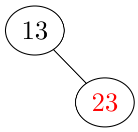
April 20, 2020

Binary Search Trees

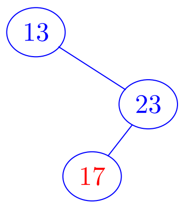
Task 1. Binary search tree is created by inserting nodes 13, 23, 17, 40, 16, 3, 10, 5, 2, 49, 20. Show structure of the tree after insertion of each node.

Insert 13

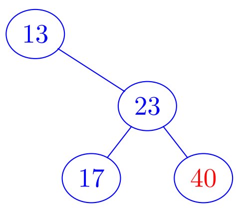
Insert 23



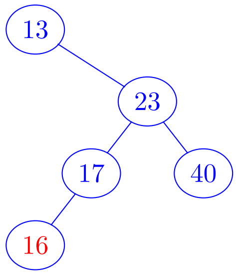
Insert 17



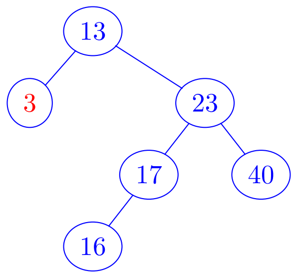
Insert 40



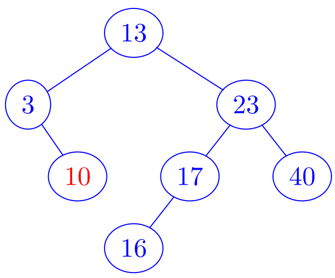
Insert 16



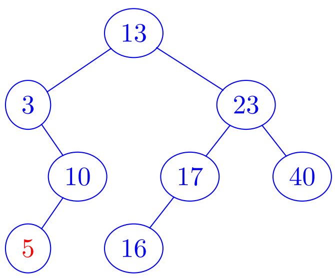
Insert 3



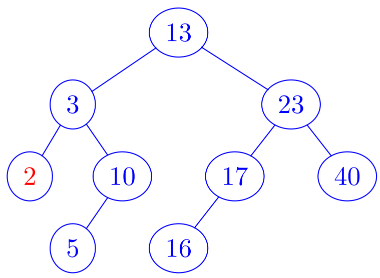
Insert 10



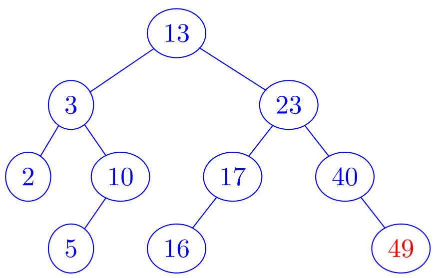
Insert 5



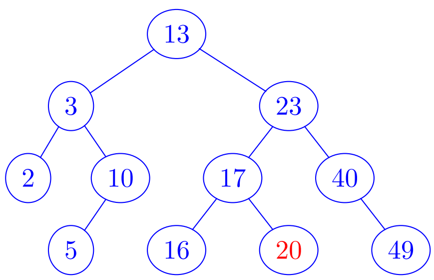
Insert 2



Insert 49

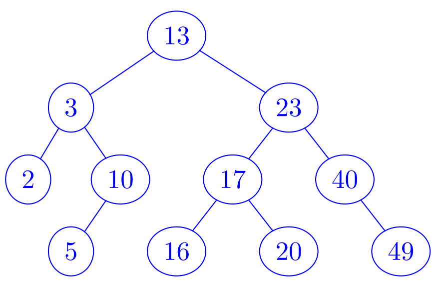


Insert 20

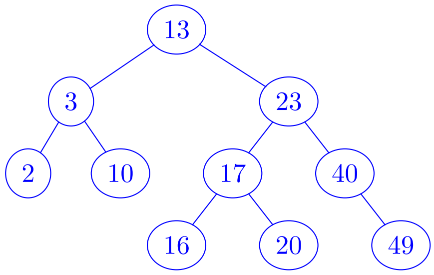


Task 2. Considering binary search tree created in `Task 1`, show the structure of the tree after deleting each of the following nodes 5, 40, 17, 23, 13.
(When the node that is going to be deleted has two children, find the child with the largest value in the left subtree.)

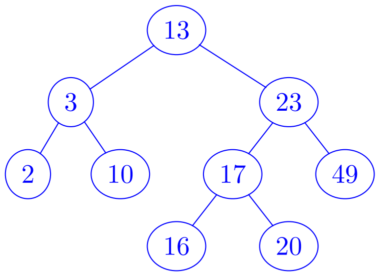
Initial tree



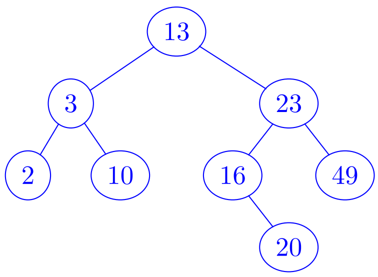
Delete 5



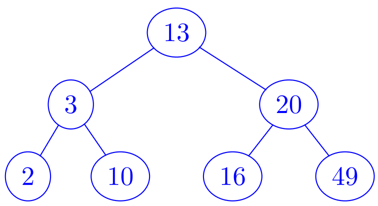
Delete 40



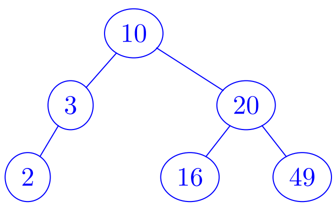
Delete 17



Delete 23



Delete 13



Task 3.

```c
struct TreeNode {
  int val;
  struct TreeNode* left;
  struct TreeNode* right;
};
```

Given the above definition of a binary search tree with positive integer values, create a C program that contains the following functions:

a) the function *void insert(struct TreeNode** root, int val)*, which gets a root node and a value `val` as an input and inserts a node with the value `val` into the proper position of the binary search tree.

```c
void insert(struct TreeNode** root, int val) {
  struct TreeNode* newTreeNode = NULL;
  struct TreeNode* prev = NULL;
  struct TreeNode* current = *root;
  newTreeNode = malloc(sizeof(struct TreeNode));
  newTreeNode->val = val;
  newTreeNode->left = NULL;
  newTreeNode->right = NULL;
  while (current != NULL) {
    prev = current;
    if (val < current->val){
      current = current->left;
    } else{
      current = current->right;
    }
  }
  if (prev == NULL) {
    *root = newTreeNode;
  } else if (val < prev->val) {
    prev->left = newTreeNode;
  } else {
    prev->right = newTreeNode;
  }
}
```

b) the function *struct TreeNode* search(struct TreeNode* root, int val)*, which finds the tree node with value `val` and returns the node.

```c
struct TreeNode* search(struct TreeNode* root, int val) {
  struct TreeNode* current = root;
  while (current != NULL && current->val != val) {
    if (val < current->val){
        current = current->left;
    } else{
        current = current->right;
    }
  }
  return current;
}
```

c) the function *void delete(struct TreeNode** root, int val)*, which deletes the node with value `val` from the tree.

```c
void delete(struct TreeNode** root, int val) {
  struct TreeNode* x = search(*root, val);
  if (x == NULL){
    return;
  }
  struct TreeNode* u = *root;
  struct TreeNode* v = NULL;
  while (u != x) {
    v = u;
    if (x->val < u->val){
      u = u->left;
    } else{
      u = u->right;
    }
  }
  if (u->right == NULL) {
    if (v == NULL){
      *root = u->left;
    } else if (v->left == u){
      v->left = u->left;
    } else{
      v->right = u->right;
    }
  } else if (u->left == NULL) {
    if (v == NULL){
      *root = u->right;
    } else if (v->left == NULL){
      v->left = u->right;
    } else{
      v->right = u->right;
    }
  } else{
    struct TreeNode* p = x->left;
    struct TreeNode* q = p;
    while (p->right != NULL) {
      q = p;
      p = p->right;
    }
    if (v == NULL){
      *root = p;
    } else if (v->left == u){
      v->left = p;
    } else{
      v->right = p;
    }
    p->right = u->right;
    if (q != p) {
      q->right = p->left;
      p->left = u->left;
    }
  }
}
```

d) *void printTree(tree *root)* which prints the tree with root `root` in the console in the format `graph g` { (all the edges in the form `NodeA -- NodeB`) }, where each edge should be printed on a separate line. The ordering of the edges is not relevant and may vary based on your implementation.

```c
void printTreeRecursive(struct TreeNode *root) {
  if (root == NULL)
    return;
  if (root->left != NULL) {
    printf("  %d -- %d\n", root->val, root->left->val);
    printTreeRecursive(root->left);
  }
  if (root->right != NULL) {
    printf("  %d -- %d\n", root->val, root->right->val);
    printTreeRecursive(root->right);
  }
}

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

e) *struct TreeNode* maximum(struct TreeNode* node)*, which returns the node with the largest value in the subtree with root node `n`.

```c
struct TreeNode* maximum(struct TreeNode* root) {
  struct TreeNode* current = root;
  while (current->right != NULL) {
        current = current->right;
  }
  return current;
}
```

f) *struct TreeNode* minimum(struct TreeNode* node)*, which returns the node with the smallest value in the subtree with root node `n`.

```c
struct TreeNode* minimum(struct TreeNode* root) {
  struct TreeNode* current = root;
  while (current->left != NULL) {
        current = current->left;
  }
  return current;
}
```

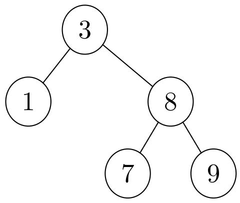
g) *int distanceToRoot(struct TreeNode* root, int val)* that returns the distance of the node with value `val` from the root node `root`.

```c
int distanceToRoot(struct TreeNode* root, int val) {
  if (root->val == val) return 0;
  else if (root->val>val) return 1+distanceToRoot(root->left,val);
  else return 1+distanceToRoot(root->right,val);
}
```

For example, if the values 3, 8, 7, 1 and 9 are inserted into an empty tree, your program should produce the binary tree shown below.

Print Form	**Binary Tree Form**

```
  graph g {
3 -- 1
3 -- 8
8 -- 7
8 -- 9
  }
```



Test your program by performing the following operations:

- Create a root node `root` and insert the values 4, 2, 3, 8, 6, 7, 9, 12, 1.
- Print tree to the console.
- Print the minimum value of the tree.
- Print the distanceToRoot of node 7.
- Delete the values 4, 12, 2 from the tree.
- Print tree to the console.
- Print the maximum value of the tree.
- Print the distanceToRoot of node 6.

```
struct TreeNode* root= NULL;
printf("Inserting: 4, 2, 3, 8, 6, 7, 9, 12, 1\n");
insert(&root, 4);
insert(&root, 2);
insert(&root, 3);
insert(&root, 8);
insert(&root, 6);
insert(&root, 7);
insert(&root, 9);
insert(&root, 12);
insert(&root, 1);
printTree(root);
printf("Minimum Value: %d\n", minimum(root)->val);
printf("Distance to root of node 7: %d\n",distanceToRoot(root,7));
printf("Deleting: 4, 12, 2\n");
delete(&root, 4);
delete(&root, 12);
delete(&root, 2);
printTree(root);
printf("Maximum Value: %d\n", maximum(root)->val);
printf("Distance to root of node 6: %d\n",distanceToRoot(root,6));
```

Task 4. Given a rooted tree T, the *lowest common ancestor* `(LCA)` between two nodes *n1* and *n2* is defined as the lowest node in T that has both *n1* and *n2* as descendants (where we allow a node to be a descendant of itself). Consequently, the LCA of *n1* and *n2* in T is the shared ancestor of *n1* and *n2* that is located farthest from the root.

For example, given a tree as in Figure 1, the LCA of nodes *9* and *21* is node *15*.

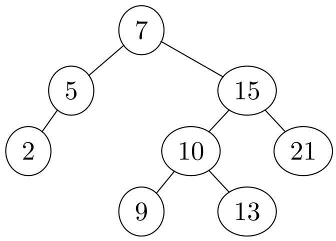


Figure 1: LCA of nodes 9 and 21 is node 15

Given values of two nodes in a Binary Search Tree, implement in C the function `struct TreeNode* lca(TreeNode* root, int n1, int n2)` that finds the *Lowest Common Ancestor (LCA)*. You may assume that both the values exist in the tree.

Write in C a program to test your implementation by performing the following operations:

- Create an empty tree and insert the nodes 7, 5, 6, 1, 9, 10, 8.
- Print the resulting tree using method described in **Task 3**.
- Print out the LCA of node *8* and node *9*

```c
struct TreeNode *lca(struct TreeNode *root, int n1, int n2)
{
    if (root == NULL) return NULL;

    // If both n1 and n2 are smaller than root, then LCA lies in left
    if (root->val > n1 && root->val > n2)
        return lca(root->left, n1, n2);

    // If both n1 and n2 are greater than root, then LCA lies in right
    if (root->val < n1 && root->val < n2)
        return lca(root->right, n1, n2);

    return root;
}
```

Initialization and Function Call:

```c
int n1, n2;
struct TreeNode *root = NULL;

insert(&root, 7);
insert(&root, 5);
insert(&root, 6);
insert(&root, 1);
insert(&root, 9);
insert(&root, 10);
insert(&root, 8);

printTree(root);

n1 = 8;
n2 = 9;
struct TreeNode *t = lca(root, n1, n2);
printf("LCA of %d and %d is %d \n", n1, n2, t->val);
```