Lab N°04 : Trees

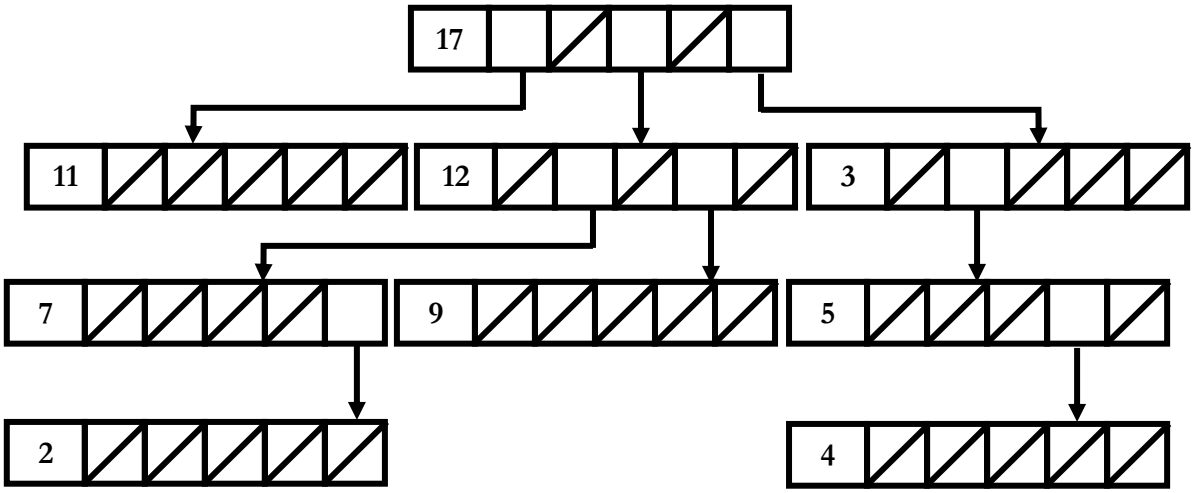
Exercise IV.1 (Generic Tree & Binary Tree)

A node in a generic (general) tree is defined as follows:

```
struct GenericNode {
      int  data;
      GenericNode* children[N];  //Maximum number of children
 };
 typedef GenericNode *GTREE;
```

Q1) Write a **Create_GNode()** function to create a generic node in a general (generic) tree.

Q2) Create the following generic tree.



Q3) Write a **Preorder_GT ()** function to display all the nodes of a generic tree for preorder traversal.

Basic Binary Tree Functions

void preorder(BTREE BTree): A **preorder** traversal procedure.

void inorder(BTREE BTree) : An **inorder** traversal procedure.

void postorder(BTREE BTree): A **postorder** traversal procedure.

void level_order(BTREE BTree): A level-order traversal procedure.

Node* search_BT(BTREE BTree, **int** k): Searching for a key in a BST.

void insert_BST(BTREE &BTree, **int** k): Inserting a key into a BST.

void remove_BST(BTREE &BTree, **int** data): Removes the node from the BST.

Also, all functions implemented in exercise series N° 3 on linked list can be use.

Init_BT(), Create_BTNode(), IsEmpty_BT(), Size_BT_ Rec(), Sum_BT(), Height_BT(), Find_Level(), Max_BT_Rec(), Max_BT_It(), Count_Leaves()

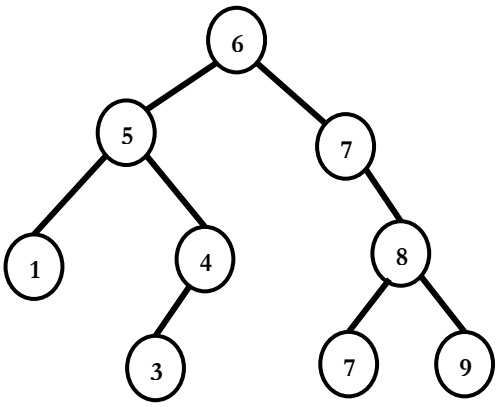
Exercise IV.2 (Binary Trees)

Q1) Place all the basic functions of a binary tree in a "**BINARY_TREE.hpp**" file and save it.

Q2) Write all the basic functions seen in the course to a "**Test_BT.cc**" file and Test all these functions.

Q3) Add all the functions implemented in exercise series N° 5 to a "**BINARY_TREE.hpp**" file and save it.

Q4) Create the following binary tree**.**



Q5) Write all the functions implemented in exercise series N° 5 to a "**Test_BT.cc**" file and Test all these functions.

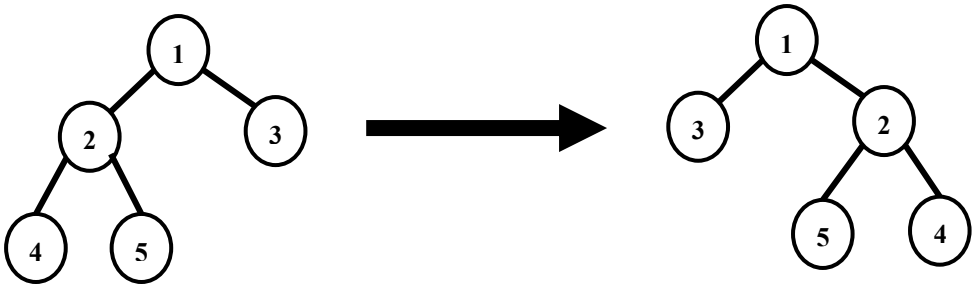
Q6) Test a **level_order()** function (Rewrite a structure of the new queue and adapt all queue functions to this structure)

Exercise IV.3 (Binary Search Tree)

A node in a binary tree is defined as follows:

```
struct BTNode {
    int data;
    BTNode *left;
    BTNode *right;
};
typedef BTNode *BTREE;
```

Q1) Write a **Mirror_BT()** function for converting a binary tree to its mirror. Mirror of a binary tree is another binary tree with left and right children of all non-leaf nodes interchanged. The binary trees below are mirrors to each other.



Q2) Given two binary trees, write a **Are_Mirrors_BT()** function for checking whether they are mirrors of each other.

Q3) write an **LCA_BT()** function for finding LCA (Least Common Ancestor) of two nodes in a Binary Tree.

Q4) Write a **Has_Path_Sum()** function for checking the existence of path with given sum. That means, given a sum, check whether there exists a path from root to any of the nodes.