1. Trees & Traversing Binary Trees

Przykład 1 (C++). Implementacja drzewa binarnego.

```
//tree.cpp
//demonstrates binary tree
//Robert Lafore - Teach Yourself Data Structures And Algorithms In 24 hours
#include <iostream>
#include <stack>
using namespace std;

class Node
{
    public:
        int iData;                  //data item (key)
        double dData;               //data item
        Node* pLeftChild;           //this node's left child
        Node* pRightChild;          //this node's right child

        //constructor
        Node() : iData(0), dData(0.0), pLeftChild(NULL),
        pRightChild(NULL)
        {  }

        ~Node()                         //destructor
        { cout << "X-" << iData << " "; }

        void displayNode()              //display ourself: {75, 7.5}
        {
            cout << "{" << iData << ", " << dData << "} ";
        }
};  //end class Node
//--------------------------------------------------------------
class Tree
{
    private:
        Node* pRoot;                //first node of tree

    public:
        Tree() : pRoot(NULL)        //constructor
        {  }

        Node* find(int key)         //find node with given key
        {                           //(assumes non-empty tree)
            Node* pCurrent = pRoot;             //start at root
            while(pCurrent->iData != key)       //while no match,
            {
                if(key < pCurrent->iData)        //go left?
                    pCurrent = pCurrent->pLeftChild;
                else                             //or go right?
                    pCurrent = pCurrent->pRightChild;
                if(pCurrent == NULL)             //if no child,
                    return NULL;                  //didn't find it
            }
            return pCurrent;                    //found it
        }  //end find()
```

```
void insert(int id, double dd)          //insert new node
{
    Node* pNewNode = new Node;          //make new node
    pNewNode->iData = id;               //insert data
    pNewNode->dData = dd;
    if(pRoot==NULL)                     //no node in root
        pRoot = pNewNode;
    else                                //root occupied
    {
        Node* pCurrent = pRoot;         //start at root
        Node* pParent;
        while(true)                     //(exits internally)
        {
            pParent = pCurrent;
            if(id < pCurrent->iData)    //go left?
            {
                pCurrent = pCurrent->pLeftChild;
                if(pCurrent == NULL)    //if end of the line,
                {                       //insert on left
                    pParent->pLeftChild = pNewNode;
                    return;
                }
            }  //end if go left
            else                        //or go right?
            {
                pCurrent = pCurrent->pRightChild;
                if(pCurrent == NULL)    //if end of the line
                {                       //insert on right
                    pParent->pRightChild = pNewNode;
                    return;
                }
            }  //end else go right
        }  //end while
    }  //end else not root
}  //end insert()

void traverse(int traverseType)
{
    switch(traverseType)
    {
        case 1: cout << "\nPreorder traversal: ";
                preOrder(pRoot);
                break;
        case 2: cout << "\nInorder traversal:  ";
                inOrder(pRoot);
                break;
        case 3: cout << "\nPostorder traversal: ";
                postOrder(pRoot);
                break;
    }
    cout << endl;
}

void preOrder(Node* pLocalRoot)
{
    if(pLocalRoot != NULL)
    {
        cout << pLocalRoot->iData << " ";    //display node
        preOrder(pLocalRoot->pLeftChild);    //left child
        preOrder(pLocalRoot->pRightChild);   //right child
    }
```

```
        }

        void inOrder(Node* pLocalRoot)
        {
            if(pLocalRoot != NULL)
            {
                inOrder(pLocalRoot->pLeftChild);      //left child
                cout << pLocalRoot->iData << " ";     //display node
                inOrder(pLocalRoot->pRightChild);     //right child
            }
        }

        void postOrder(Node* pLocalRoot)
        {
            if(pLocalRoot != NULL)
            {
                postOrder(pLocalRoot->pLeftChild);    //left child
                postOrder(pLocalRoot->pRightChild);   //right child
                cout << pLocalRoot->iData << " ";     //display node
            }
        }

        void displayTree()
        {
            stack<Node*> globalStack;
            globalStack.push(pRoot);
            int nBlanks = 32;
            bool isRowEmpty = false;
            cout
<<".....................................................";
            cout << endl;
            while(isRowEmpty==false)
            {
                stack<Node*> localStack;
                isRowEmpty = true;
                for(int j=0; j<nBlanks; j++)
                    cout << ' ';
                while(globalStack.empty()==false)
                {
                    Node* temp = globalStack.top();
                    globalStack.pop();
                    if(temp != NULL)
                    {
                        cout << temp->iData;
                        localStack.push(temp->pLeftChild);
                        localStack.push(temp->pRightChild);
                        if(temp->pLeftChild != NULL ||
                        temp->pRightChild != NULL)
                            isRowEmpty = false;
                    }
                    else
                    {
                        cout << "--";
                        localStack.push(NULL);
                        localStack.push(NULL);
                    }
                    for(int j=0; j<nBlanks*2-2; j++)
                    cout << ' ';
                }  //end while globalStack not empty
                cout << endl;
                nBlanks /= 2;
```

```
                while(localStack.empty()==false)
                {
                    globalStack.push( localStack.top() );
                    localStack.pop();
                }
            }  //end while isRowEmpty is false
            cout
<<".....................................................";
            cout << endl;
        }  //end displayTree()

        void destroy()                          //deletes all nodes
        { destroyRec(pRoot); }                  //start at root

        void destroyRec(Node* pLocalRoot)       //delete nodes in
        {                                       //   this subtree
            if(pLocalRoot != NULL)
            {                                   //uses postOrder
                destroyRec(pLocalRoot->pLeftChild);  //left subtree
                destroyRec(pLocalRoot->pRightChild); //right subtree
                delete pLocalRoot;                //delete this node
            }
        }
};  //end class Tree
//---------------------------------------------------------------
int main()
{
    int value;
    char choice;
    Node* found;
    Tree theTree;                           //create tree
    theTree.insert(50, 5.0);                //insert nodes
    theTree.insert(25, 2.5);
    theTree.insert(75, 7.5);
    theTree.insert(12, 1.2);
    theTree.insert(37, 3.7);
    theTree.insert(43, 4.3);
    theTree.insert(30, 3.0);
    theTree.insert(33, 3.3);
    theTree.insert(87, 8.7);
    theTree.insert(93, 9.3);
    theTree.insert(97, 9.7);
    while(choice != 'q')                     //interact with user
    {                                        //until user types 'q'
        cout << "Enter first letter of ";
        cout << "show, insert, find, traverse or quit: ";
        cin >> choice;
        switch(choice)
        {
            case 's':                        //show the tree
                theTree.displayTree();
                break;
            case 'i':                        //insert a node
                cout << "Enter value to insert: ";
                cin >> value;
                theTree.insert(value, value + 0.9);
                break;
            case 'f':                        //find a node
                cout << "Enter value to find: ";
                cin >> value;
                found = theTree.find(value);
```

```
            if(found != NULL)
            {
                cout << "Found: ";
                found->displayNode();
                cout << endl;
            }
            else
                cout << "Could not find " << value << endl;
                break;
            case 't':                          //traverse the tree
                cout << "Enter traverse type (1=preorder, "
                << "2=inorder, 3=postorder): ";
                cin >> value;
                theTree.traverse(value);
                break;
            case 'q':                          //quit the program
                theTree.destroy();
                cout << endl;
            break;
                default:
            cout << "Invalid entry\n";
        }  //end switch
    }  //end while
    return 0;
}  //end main()
```

Opracowano na podstawie:
Robert Lafore – "Teach Yourself Data Structures And Algorithms In 24 Hours"