Лабораторная работа №6 (3-й семестр)

> Если отладка — процесс удаления ошибок, то программирование должно быть процессом их внесения.
>
> — Эдсгер Вайб Дейкстра

Теоретические сведения

Бинарное (двоичное) дерево (binary tree) — это упорядоченное дерево, каждая вершина которого имеет не более двух поддеревьев, причем для каждого узла выполняется правило: в левом поддереве содержатся только ключи, имеющие значения, меньшие, чем значение данного узла, а в правом поддереве содержатся только ключи, имеющие значения, большие, чем значение данного узла.

Бинарное дерево является рекурсивной структурой, поскольку каждое его поддерево само является бинарным деревом и, следовательно, каждый его узел в свою очередь является корнем дерева. Узел дерева, не имеющий потомков, называется листом. Графическое преставление бинарного дерева показано на рис. 1.

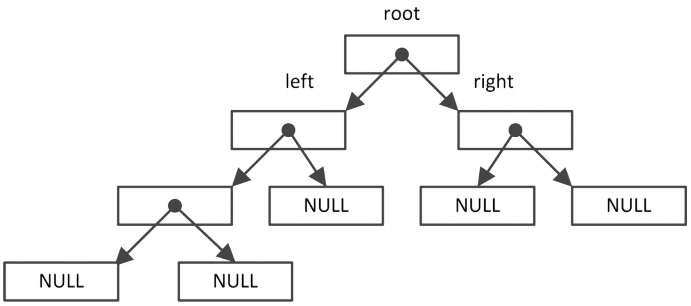
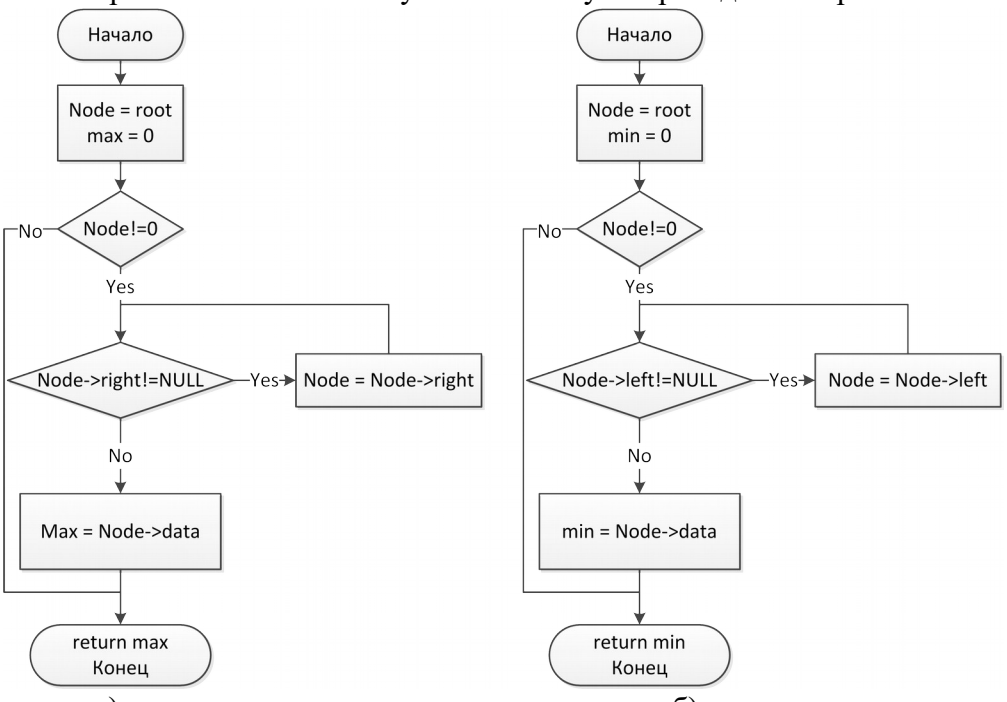


Рис. 1 — Графическое представление бинарного дерева

Так как бинарное дерево является упорядоченным, то нахождение максимального и минимального узла с водиться к задаче о поиске самого правого и самого левого узла соответственно. Алгоритмы поиска минимума и максимума приведены на рис. 2.



а) б)

Рис. 2 — Блок схема алгоритма поиска: а) максимального узла; б) минимального узла

Решение задачи о нахождении суммы элементов дерева целесообразно решать с помощью рекурсивных алгоритмов т. к. дерево является рекурсивной структурой. На рис. 3 приведена блок схема алгоритма нахождения суммы элементов дерева.

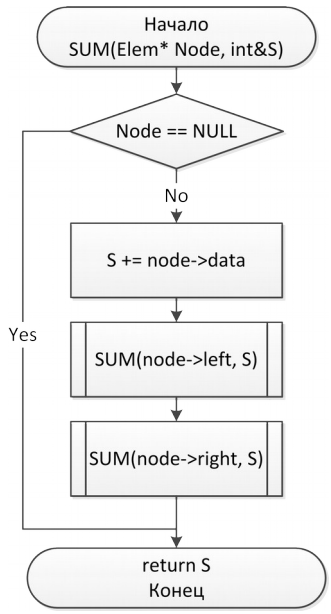


Рис. 3 — Блок схема алгоритма нахождения суммы элементов дерева

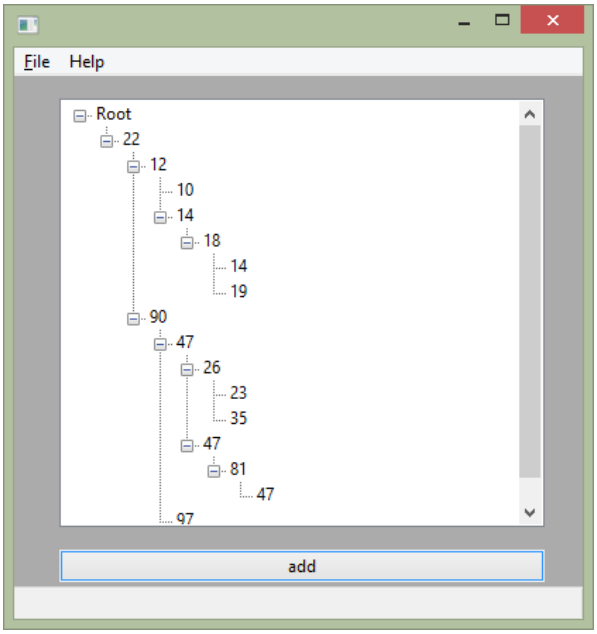


Рис. 4 — пример формы приложения

Пример. Написать программу, реализующею бинарное дерево для хранения целых чисел. Пользователь может помещать случайное число в дерево. Содержимое дерева связывается с wxTreeCtrl таким образом, чтобы пользователь мог видеть актуальное состояние дерева (рис. 4).

Файл: *btree.h*

```
#ifndef BTREE_H_INCLUDED
#define BTREE_H_INCLUDED
#include <wx/treectrl.h>
struct Elem
{
    int data;
    Elem *left, *right;
};
class Tree
{
    Elem * root;    // корень
    wxTreeCtrl* tv;
    void PrintTree(Elem * Node, wxTreeItemId trNode);
    void AddElem(Elem** node, Elem *data);
public:
    Tree();                  // Конструктор
    void Print();            // вывод дерева
    void Insert(int _data); // добавление элемента
    void SetTreeView(wxTreeCtrl *_tv); // Установка wxTreeCtrl
};
#endif // BTREE_H_INCLUDED
```

Файл: *btree.cpp*

```
#include "btree.h"
Tree::Tree()
{
    root = NULL;
}
```

```
void Tree::Print()
{
    if (tv == NULL)  return;
    tv->DeleteAllItems(); // Очистка ListView
    wxTreeItemId trNode = tv->AddRoot(wxT("Root"));
    PrintTree(root, trNode);  // Вызываем печать
    tv->ExpandAll();
}
void Tree::PrintTree(Elem * Node, wxTreeItemId trNode)
{
    if(Node == NULL)  return;
    wxTreeItemId trNode1 = tv->AppendItem(trNode,wxString()<<
(Node->data));

    PrintTree(Node->left,  trNode1);
    PrintTree(Node->right, trNode1);
}
void Tree::AddElem(Elem** node, Elem *data)
{
    if(*node == NULL)  // Если лист -- добавляем элемент
        *node = data;
    else
    {
        if ((*node)->data > data->data) // В какую ветвь?
            AddElem(&((*node)->left), data);
        else
            AddElem(&((*node)->right), data);
   }
}
void Tree::Insert(int _data)
{
    Elem * z = new Elem;
    z->left = NULL;
    z->right = NULL;
    z->data = _data;
    AddElem(&root, z);  // вызываем добавление
}
void Tree::SetTreeView(wxTreeCtrl *_tv)
{   tv = _tv;
}
```

Файл: *Unit1.cpp*

```
#include "btree.h"
Tree tree;
btreeFrame::btreeFrame(wxWindow* parent,wxWindowID id)
{    ...
    tree.SetTreeView(TreeCtrl1);
}
void btreeFrame::OnButton1Click(wxCommandEvent& event)
{
    tree.Insert(rand()%100);
    tree.Print();
}
```