**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**  
Федеральное государственное бюджетное образовательное учреждение высшего образования  
**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»  
 РУТ (МИИТ)**  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Кафедра: «Вычислительные системы и сети»  
  
  
  
  
  
  
  
  
  
  
**О Т Ч Ё Т  
П О Л А Б О Р А Т О Р Н О Й Р А Б О Т Е № 5**  
по дисциплине «Структуры и алгоритмы обработки данных»  
на тему: «Методы обработки иерархических структур (деревьев)»  
  
  
  
  
  
  
  
  
  
 Выполнил: студент группы УИБ-311   
 Владыка П.А.

Москва 2018

**Задание**

Разработать и отладить программу, реализующую обработку иерархической структуры. Обеспечить возможность вставки и удаления элементов, просмотр дерева, используя меню для выбора операции. Вид иерархической структуры и дополнительные операции задаются по вариантам.

|  |  |  |
| --- | --- | --- |
| Вариант 4 | Бинарное дерево | Обратный обход, преобразовать в дерево поиска |

**Таблица имён**

|  |  |
| --- | --- |
| **Исходные данные** | |
| Knot | Указатель на корень дерева (поле класса USETREE) |
| **Промежуточные данные** | |
| Root | Указатель на корень дерева |
| Element | Корень дерева (входной параметр функции) |
| Parent | Родитель элемента Element |
| Value | Имя переменной, хранящее вводимое с клавиатуры значение (поле класса USETREE) |
| **Поля структуры** ELEMENT | |
| inf | Информационное поле |
| pLeft | Указатель на левое поддерево (элемент) |
| pRight | Указатель на правое поддерево (элемент) |

**Блок-схема**



**Исходный код**

/\* Лабораторная работа #5. Вариант-4. Бинарное дерево. Обратный обход, преобразовать в дерево поиска. \*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<windows.h>

#include <iomanip>

#include <malloc.h>

using namespace std;

/\* Класс создан для работы с бинарным дервом \*/

class BINTREE {

public:

template<typename ClassName, typename Type>

inline static void AddItemToTree(ClassName\*\* Element, Type value) {

if (\*Element == NULL) {

\*Element = new ClassName;

(\*Element)->inf = value; }

else {

ClassName \*Root = \*Element; // Сохранение корня дерева

for (;;) {

if ((\*Element)->inf == value)

break;

if (((\*Element)->inf > value) && ((\*Element)->pLeft == NULL)){

(\*Element)->pLeft = new ClassName;

(\*Element)->pLeft->inf = value;

break; }

else if (((\*Element)->inf < value) && ((\*Element)->pRight == NULL)){

(\*Element)->pRight = new ClassName;

(\*Element)->pRight->inf = value;

break;}

else if ((\*Element)->inf > value) {

\*Element = (\*Element)->pLeft;}

else {

\*Element = (\*Element)->pRight;}}

\*Element = Root; // Восстановление корня }}

template<typename ClassName, typename Type>

inline static bool DelItemFromTree(ClassName\*\* Element, Type value) {

if (\*Element != NULL) {

return true;}

return false;}

template<typename ClassName>

inline static ClassName\* CopyTree(ClassName\* Element) // Возвращает ссылку на корень скопированного скопированное дерево{

ClassName\* CopiedTree = NULL;

if (Element != NULL) {

CopiedTree = new ClassName(Element->inf);

CopiedTree->pLeft = CopyTree(Element->pLeft);

CopiedTree->pRight = CopyTree(Element->pRight);}

return CopiedTree;}

template<typename ClassName>

inline static void DelTree(ClassName\* Element) { // Рекурсивное полное удаление дерева

if (Element != NULL) {

if ((Element->pLeft == NULL) && (Element->pRight == NULL)) {

delete Element;

Element = NULL; }

else {

DelTree(Element->pLeft);

DelTree((Element)->pRight);}}}};

/\* Класс для работы с бинарным деревом \*/

template<typename T>

class USETREE

{

public:

USETREE() : Knot(NULL) {}

~USETREE(){

BINTREE::DelTree(Knot);}

void menu() {

char choice;

char secondchoise;

system("cls");

cout << " {{>================================================<}}\n"

<< " || Бинарное дерево ||\n"

<< " {{>================================================<}}\n"

<< " || 1. Добавить элемент в дерево ||\n"

<< " || 2. Просмотр элементов дерева (Обратный обход) ||\n"

<< " || 3. Поиск элемента по дереву ||\n"

<< " || 4. Выход ||\n"

<< " {{>================================================<}}\n"

<< "\n Выбранная опция: ";

(cin >> choice).get();

switch (choice) {

case 49: {

system("cls");

cout << "Введите новый элемент: ";

(cin >> Value).get();

AddElement(Value);

cout << "Добавлено. ";

system("pause");

menu();

break;}

case 50:{

system("cls");

if (!PrintTheStucture(&Knot))

cerr << "No structure";

cout << endl;

system("pause");

menu();

break; }

case 51: {

system("cls");

cout << "Введите элемент для поиска: ";

(cin >> Value).get();

if (StructureSearch(&Knot, Value)) {

for (;;) {

system("cls");

cout << "Элемент найден\n"

<< "Удалить?\n"

<< "1. Да\n"

<< "2. Нет\n" << endl;

(cin >> secondchoise).get();

if (secondchoise == 49) {

system("cls");

if (DelElement(Value))

cout << "Удалено" << endl;

break;}

else if (secondchoise == 50) {

break;}}}

else { cout << "Элемент не найден" << endl;}

system("pause");

menu();

break;}

case 52:{break;}

default: menu();}}

private:/\* Структура элемента дерева \*/

template<typename Ty>

struct ELEMENT {

ELEMENT() : pLeft(NULL), pRight(NULL), inf(-1) {}

ELEMENT(int val) : pLeft(NULL), pRight(NULL), inf(val) {}

Ty inf;

ELEMENT\* pLeft = NULL;

ELEMENT\* pRight = NULL;};

T Value;

ELEMENT<T>\* Knot;

public:

template<typename ClassName>

bool PrintTheStucture(ClassName\*\* Element) {

if (\*Element != NULL) {

ELEMENT<T>\* Root = \*Element; // Сохраняем корень

ELEMENT<T>\* Copy = BINTREE::CopyTree(\*Element);

cout << "Элементы в порядке обхода:\n";

for (;;) {

if (Root->pLeft != NULL) {

if ((Root->pLeft->pLeft == NULL) && (Root->pLeft->pRight == NULL)) {

cout << setw(4) << Root->pLeft->inf;

delete Root->pLeft;

Root->pLeft = NULL;

Root = \*Element; }

else { Root = Root->pLeft;}}

else if (Root->pRight != NULL) {

if ((Root->pRight->pLeft == NULL) && (Root->pRight->pRight == NULL)) {

cout << setw(4) << Root->pRight->inf;

delete Root->pRight;

Root->pRight = NULL;

Root = \*Element;}

else { Root = Root->pRight;}}

else {

cout << setw(4) << Root->inf;

break;}}

delete Root;

\*Element = Copy;

return true;}

return false;}

template<typename ClassName, typename Type>

inline bool StructureSearch(ClassName\*\* Element, Type value){

if (\*Element != NULL) {

ClassName\* Root = \*Element;

if (value == (\*Element)->inf) {return true;}

else {

return (StructureSearch(&(Root->pLeft), value) > StructureSearch(&(Root->pRight), value) ? StructureSearch(&(Root->pLeft), value) : StructureSearch(&(Root->pRight), value));}}

return false;

}

template<typename Type>

inline void AddElement(Type value){

BINTREE::AddItemToTree(&Knot, value);}

template<typename Type>

inline bool DelElement(Type value) {

return BINTREE::DelItemFromTree(&Knot, value);}

};

int main() {

SetConsoleCP(1251);

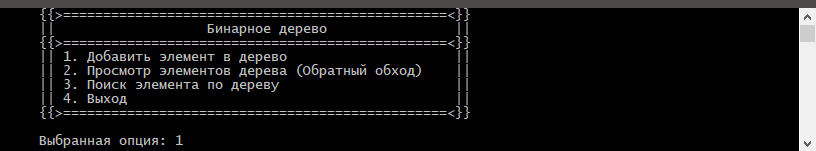
SetConsoleOutputCP(1251);

USETREE<int> list;

list.menu();

return 0;}

**Отладка**

Добавление 8 элементов в бинарное дерево поиска через меню выбора:









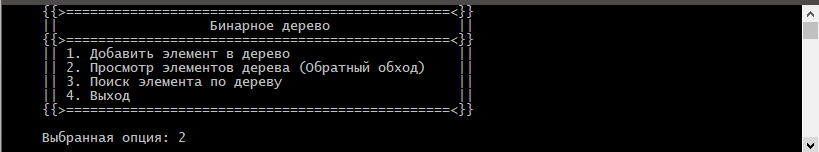








Просмотр элементов дерева в обратном порядке обхода:





Поиск элемента по дереву:

