**Лабораторная работа №11. Проект по бинарным деревьям**

**Вариант 10**

Условие:

Добавить к проекту  функции смешанного и нисходящего обхода дерева с выводом  на консоль, функцию в соответствии с вариантом из таблицы, представленной в лабораторной работе № 10, изменив ее так, чтобы функция соответствовала проекту данной лабораторной работы.

Код (ConsoleAplication2.cpp):

// ConsoleApplication2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "Tree.h"

#include <fstream>

using namespace std;

struct NodeTree

{

int key;

};

fstream file\_in,file\_out;

btree::CMP cmpfnc(void\* x, void\* y) // Сравнение

{

btree::CMP rc = btree::EQUAL;

if (((NodeTree\*)x)->key < ((NodeTree\*)y)->key)

rc = btree::LESS;

else if (((NodeTree\*)x)->key > ((NodeTree\*)y)->key)

rc = btree::GREAT;

return rc;

}

void Print(void\* x) // Вывод при обходе

{

cout <<((NodeTree\*)x)->key <<endl;

}

bool BuildTree(char \*FileName, btree::Object& tree) //Построение дерева из файла

{

bool rc = true;

file\_in.open("in.txt");

if (file\_in.fail())

{

cout<<"Ошибка открытия входного файла"<<endl;

rc = false;

}

cout<<" Исходные данные"<< endl; // заполнение дерева и вывод исходных данных

while (!file\_in.eof())

{

int num;

file\_in>>num;

NodeTree \*a = new NodeTree();

a->key = num;

tree.Insert(a);

cout<<num<<endl;

}

file\_in.close();

return rc;

}

void SaveToFile(void \*x) // Запись одного элемента в файл

{

NodeTree \*a = (NodeTree\*)x;

int q = a->key;

file\_out<<q<<" ";

}

void SaveTree(btree::Object &tree, char \*FileName) //Сохранение дерева в файл

{ file\_out.open("out.txt");

if (file\_out.fail())

{ cout<<"Ошибка открытия выходного файла"<<endl; return; }

tree.Root->ScanDown(SaveToFile);

file\_out.close();

}

int \_tmain(int argc, \_TCHAR\* argv[])

{ setlocale (LC\_CTYPE, "Russian");

btree::Object t1 = btree::Create(cmpfnc); int k, choise;

NodeTree a1 = {1}, a2 = {2}, a3 = {3}, a4 = {4}, a5 = {5}, a6 = {6};

bool rc = t1.Insert(&a4); //true 4

rc = t1.Insert(&a1); //true 1

rc = t1.Insert(&a6); //true 6

rc = t1.Insert(&a2); //true 2

rc = t1.Insert(&a3); //true 3

rc = t1.Insert(&a5); //true 5

for(;;)

{ NodeTree \*a = new NodeTree;

cout<<"\t\t Меню\n";

cout<<"\t1 - Вывод дерева на экран"<<endl;

cout<<"\t2 - Добавление элемента"<<endl;

cout<<"\t3 - Удаление элемента"<<endl;

cout<<"\t4 - Сохранить в файл"<<endl;

cout<<"\t5 - Загрузить из файла"<<endl;

cout<<"\t6 - Очистить дерево"<<endl;

cout<<"\t7 - Смешанный обход"<<endl;

cout<<"\t8 - Нисходящий обход"<<endl;

cout<<"\t0 - Выход"<<endl;

cout<<"\tCделайте выбор "; cin>>choise;

switch(choise)

{ case 0: exit(0);

case 1: t1.Root->Vyvod(Print,0);

break;

case 2: cout<<"введите ключ"<<endl; cin>>k;

a->key = k; t1.Insert(a); break;

case 3: cout<<"введите ключ"<<endl; cin>>k;

a->key = k; t1.Delete(a); break;

case 4: SaveTree(t1, "G.txt"); break;

case 5: BuildTree("G.txt", t1); break;

case 6: { while (t1.Root) t1.Delete(t1.Root); } break;

case 7: t1.Root->obxod\_smesh(Print);

cout<<endl;

break;

case 8:t1.Root->obxod\_niz(Print);

}

}

return 0;

}

}

Код (Tree.cpp):

#include "stdafx.h"

#include "Tree.h"

using namespace std;

namespace btree // бинарное дерево, не допускается дублирование ключей

{

Object Create(CMP (\*f)(void\*, void\*))

{

return \*(new Object(f));

}

Node\* Node::Min()

{

Node\* rc = this; if (rc->Left != NULL)

rc = rc->Left->Min();

return rc;

}

Node\* Node::Next()

{ Node\* rc = this, \*x = this;

if (rc->Right != NULL)

rc = rc->Right->Min();

else

{

rc = this->Parent;

while (rc != NULL && x == rc->Right)

{

x = rc;

rc = rc->Parent;

}

}

return rc;

}

void Node::ScanDown(void (\*f)(void\* n))

{

f(this->Data);

std::cout<<endl;

if (this->Left != NULL)

this->Left->ScanDown(f);

if (this->Right != NULL)

this->Right->ScanDown(f);

}

Node\* Object::Search(void\* d, Node\* n)

{

Node\* rc = n;

if (rc != NULL)

{ if (isLess(d, n->Data))

rc = Search(d, n->Left);

else if (isGreat(d, n->Data))

rc = Search(d, n->Right);

} return rc;

}

bool Object::Insert(void\* d)

{

Node\* x = this->Root, \*n = NULL;

bool rc = true;

while (rc == true && x != NULL)

{ n = x;

if (isLess(d, x->Data))

x = x->Left; //выбор куда идти - влево или вправо

else if (isGreat(d, x->Data))

x = x->Right;

else rc = false;

}

if (rc == true && n == NULL)

this->Root = new Node(NULL, NULL, NULL, d);

else if (rc == true && isLess(d, n->Data))

n->Left = new Node(n, NULL, NULL, d);

else if (rc == true && isGreat(d, n->Data))

n->Right = new Node(n, NULL, NULL, d);

return rc;

};

bool Object::Delete(Node\* n )

{

bool rc = true;

if (rc = (n != NULL))

{

if (n->Left == NULL && n->Right == NULL) //если потомков нет

{

if (n->Parent == NULL)

this->Root = NULL; //обнуление корня

else if (n->Parent->Left == n)

n->Parent->Left = NULL;

else n->Parent->Right = NULL;

delete n;

}

else if (n->Left==NULL && n->Right!=NULL) //только правый по-томок

{

if (n->Parent == NULL)

this->Root = n->Right;

else if (n->Parent->Left == n)

n->Parent->Left = n->Right;

else n->Parent->Right = n->Right;

n->Right->Parent = n->Parent;

delete n;

}

else if (n->Left!=NULL &&n->Right==NULL)//только левый по-томок

{

if (n->Parent == NULL)

this->Root = n->Left;

else if (n->Parent->Right == n)

n->Parent->Left = n->Left;

else n->Parent->Right = n->Left;

n->Left->Parent = n->Parent;

delete n;

}

else if (n->Left != NULL && n->Right != NULL) //если есть оба потомка

{

Node\* x = n->Next();

n->Data = x->Data;

rc = Delete(x);

}

} return rc;

}

void Node::ScanLevel(void (\*f)(void\* n), int i) //Вывести вершины уровня

{

if (this->Left != NULL)

this->Left->ScanLevel(f, i);

if(this->GetLevel() == i)

f(this->Data);

if (this->Right != NULL)

this->Right->ScanLevel(f, i);

}

int Node::GetLevel()

{

Node \*rc = this;

int q = 0;

while(rc->Parent != NULL)

{

rc = rc->Parent;

q++;

}

return q;

}

void Node::ScanByLevel(void (\*f)(void\* n))

{

for(int i = 0; i < 10; i++)

{

std::cout<<'\t';

this->ScanLevel(f,i);

std::cout<<'\n';

}

}

void Node::Vyvod (void (\*f)(void\* n), int l) //Вывод на экран

{ int i;

if (this != NULL) //\*w - указатель на корень

{ this->Right->Vyvod (f, l + 1);

for (i = 1; i <= l; i++) cout<<" ";

f(this->Data);

this->Left->Vyvod (f, l + 1);

}

}

void Node::obxod\_smesh(void (\*f)(void\* n))

{

if(this!=NULL)

{

this->Left->obxod\_smesh(f);

f(this->Data);

this->Right->obxod\_smesh(f);

}

}

void Node::obxod\_niz(void (\*f)(void\* n))

{

if(this!=NULL)

{

f(this->Data);

this->Left->obxod\_smesh(f);

this->Right->obxod\_smesh(f);

}

}

}

Код(Tree.h):

#include "stdafx.h"

#include <iostream>

namespace btree

{

enum CMP

{

LESS = -1, EQUAL = 0, GREAT = 1

};

struct Node //Узел бинарного списка

{

Node\* Parent; // указатель на родителя

Node\* Left; // указатель на левую ветвь

Node\* Right; // указатель на правую ветвь

void\* Data; // данные

Node(Node\* p, Node\* l, Node\* r, void\* d) // конструктор

{ Parent = p; Left = l; Right = r; Data = d; }

Node\* Next(); // следующий по ключу

Node\* Prev(); // предыдущий по ключу

Node\* Min(); // минимум в поддереве

Node\* Max(); // максимум в поддереве

void ScanDown(void (\*f)(void\* n)); // обход поддерева сверху вниз

void Scan(int (\*f)(void\* n));

void ScanLevel(void (\*f)(void\* n), int );

int GetLevel();

void Vyvod (void (\*f)(void\* n), int l); //Вывод на экран

void obxod\_smesh(void (\*f)(void\* n));//смешанный обход

void obxod\_niz(void (\*f)(void\* n));//нисходящий обход

void ScanByLevel(void (\*f)(void\* n));

};

struct Object // Интерфейс бинарного дерева

{ Node\* Root; // указатель на корень

CMP (\*Compare)(void\*, void\*); // функция сравнения

Object(CMP (\*f)(void\*, void\*))

{ Root = NULL; Compare = f; };

Node\* Max() { return Root->Max(); };

Node\* Min() { return Root->Min(); };

bool isLess(void\* x1, void\* x2)

const { return Compare(x1, x2) == LESS; };

bool isGreat(void\* x1, void\* x2)

const { return Compare(x1, x2) == GREAT; };

bool isEqual(void\* x1, void\* x2)

const { return Compare(x1, x2) == EQUAL; };

bool Insert(void\* data); // добавить элемент

Node\* Search(void\* d, Node\* n ); // найти по ключу

Node\* Search(void\* d) { return Search(d, Root); };

bool Delete(Node\* e); // удалить по адресу элемента

bool Delete(void\* data)

{ return Delete(Search(data)); }; //удалить по ключу

void ScanDown(void (\*f)(void\* n))

{ Root->ScanDown(f); }; // обход дерева

btree::Object BuildTree(char \*);

void SaveToFile(void \*);

void SaveTree(btree::Object tree, char \*);

};

Object Create(CMP (\*f)(void\*, void\*)); // Создать бинарное дерево

};

Скриншот:

