**Лабораторная работа № 6**

**Тема**: Алгоритмы работы с бинарными деревьями

**Цель**: Научиться реализовывать алгоритмы построения и обхода двоичных деревьев

Задача: Реализовать программно абстрактный тип данных двоичное дерево и его экземпляр, структура которого соответствует изображенной в Вашем варианте, заполнить узлы дерева случайными числами. Реализовать процедуры обхода двоичного дерева в прямом, обратном и симметричном порядке. При посещении каждого узла дерева выводить его значение на экран.

вариант 1

1

2

5

10

11

3

4

6

12

0

3

7

9

7

17

# include <iostream>

# include <conio.h>

# include <ctime>

*using* *namespace* std;

*//Структура* *дерева*

*template*<*class* T>

*struct* node\_struct

{

T information; *//Поле* *данных*

node\_struct \*r; *//Правая* *часть* *дерева*

node\_struct \*l; *//Левая* *часть* *дерева*

};

*//Объявляем* *указатель,* *тип* *которой* *структура* *Дерево*

node\_struct<int> \* tree=NULL;

*//* *Заполнение* *дерева*

*template*<*class* T>

void push\_element(T info,node\_struct<T> \*\*tmp)

{

*if* ((\*tmp)== NULL) *//Если* *дерева* *не* *существует*

{

(\*tmp)= *new* node\_struct<T>; *//Выделяем* *память*

(\*tmp)->information = info; *//Кладем* *в* *выделенное* *место* *аргумент* *info*

(\*tmp)->l=(\*tmp)->r=NULL; *//Очищаем* *память* *для* *следующего* *роста*

*return*;

}

*if* (info >(\*tmp)->information)

*//Если* *аргумент* *info* *больше* *чем* *текущий* *элемент,* *кладем* *его* *вправо*

push\_element(info,&(\*tmp)->r);

*else* *//Иначе* *кладем* *его* *влево*

push\_element(info,&(\*tmp)->l);

}

*//* *Отображение* *дерева*

*template* <*class* T>

void print\_prefix (node\_struct<T> \*tmp)

{

*if* (tmp==NULL) *return*; *//Если* *дерево* *пустое,* *то* *отображать* *нечего,* *выходим*

*else* *//Иначе*

cout << tmp->information << endl; *//показываем* *элемент*

print\_prefix(tmp->l); *//С* *помощью* *рекурсии* *посещаем* *левоее* *поддерево*

print\_prefix(tmp->r); *//С* *помощью* *рекурсии* *посещаем* *правое* *поддерево*

}

*template* <*class* T>

void print\_postfix (node\_struct<T> \*tmp)

{

*if* (tmp==NULL) *return*; *//Если* *дерево* *пустое,* *то* *отображать* *нечего,* *выходим*

*else* *//Иначе*

print\_postfix(tmp->l); *//С* *помощью* *рекурсии* *посещаем* *левоее* *поддерево*

print\_postfix(tmp->r); *//С* *помощью* *рекурсии* *посещаем* *правое* *поддерево*

cout << tmp->information << endl; *//показываем* *элемент*

}

*template* <*class* T>

void print\_infix (node\_struct<T> \*tmp)

{

*if* (tmp==NULL) *return*; *//Если* *дерево* *пустое,* *то* *отображать* *нечего,* *выходим*

*else* *//Иначе*

print\_infix(tmp->l); *//С* *помощью* *рекурсии* *посещаем* *левоее* *поддерево*

cout << tmp->information << endl; *//показываем* *элемент*

print\_infix (tmp->r); *//С* *помощью* *рекурсии* *посещаем* *правое* *поддерево*

}

*//* *главная* *программа*

int main (){

srand(time(0));

int n; *//Количество* *элементов*

int s; *//Число,* *передаваемое* *в* *дерево*

cout<<"Entered element count ";

cin>>n; *//Вводим* *количество* *элементов*

*for* (int i=0;i<n;++i) {

s = 1+rand()%10; *//Считываем* *элемент* *за* *элементом*

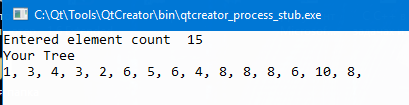
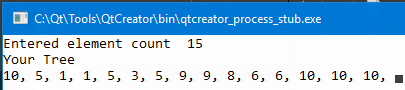
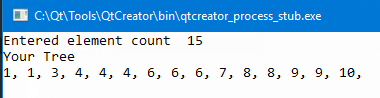
push\_element(s,&tree);} *//И* *каждый* *кладем* *в* *дерево*

cout<<"Your Tree\n";

print\_postfix(tree);

getch();

}

 Postfix Prefix

Infix