**Лабораторная работа № 28**

**Классы для работы с динамическими структурами данных**

**Теоретическое введение**. Объекты классов могут храниться в виде массивов или динамических связных списков. Если класс содержит конструктор, массив может быть инициализирован, причем конструктор вызывается столько раз, сколько задается элементов массива:

**samp ob[4]={1,2,3,4};**

Список инициализации – это сокращение общей конструкции:

**samp ob[4]={samp(1,2),samp(3,4),samp(5,6),samp(7,8)};**

При создании динамических объектов используется оператор ***new***, который вызывает конструктор и производит инициализацию. Для разрушения динамического объекта используется оператор ***delete***, который может помещаться в деструкторе.

Кроме указателей на классы используются ссылки. Ссылка является скрытым указателем и работает как другое имя переменной. При передаче объекта через ссылку в функцию передается адрес объекта и не делается его копия. Это уменьшает вероятность ошибок, связанных с выделением динамической памяти и вызовом деструктора. Так, при передаче в функцию параметра-объекта может возникнуть ошибка из-за разрушения деструктором на выходе копии объекта, которая должна быть исправлена созданием конструктора копирования. В такой ситуации лучше передать в функцию ссылку на объект и возвратить ссылку на объект.

**Пример.** Создается класс **Tree** (бинарное дерево). Информационная часть узла дерева содержит целое число. Тестирование класса выполняется с помощью меню, которое позволяет сформировать дерево, вывести содержимое его узлов в порядке возрастания, найти узел по ключу, вывести содержимое листьев дерева (вершин, не имеющих потомков).

#include "pch.h"

#include <iostream>

#include <conio.h>

#include <string.h>

using namespace std;

struct node

{

int n; //информационное поле узла дерева

int count;

node\*left, \*right;

};

class Tree

{

public: node\*root;

Tree() { root = 0; }

Tree(int t); // Формирование дерева из t случайных чисел

void CopyTree(node\*&rootnew,node\*rootold);

/\* Копирует дерево с корнем rootold в дерево с корнем rootnew. В результате деревья находятся в различных динамических участках памяти.\*/

Tree(const Tree&ob); //конструктор копирования

// Рекурсивная функция, используемая в деструкторе (освобождение памяти)

void DelTree(node \*wer);

~Tree(){DelTree(root);}

void Push(node\*&wer,int data);// Вставка элемента в дерево

void Look(node\*wer); //- Вывод дерева на экран

node\*Find(node\*wer,int key); // Поиск по ключу

void PrintLeaves(node \*wer); // Вывод листьев дерева на экран

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Tree::Tree(int t) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tree::Tree(int t)

{

root = 0;

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

Push(root, rand() % 10 - 5);

}

void Tree::CopyTree(node\*&rootnew, node\*rootold)

{

if (rootold->left != 0)

{

Push(rootnew, (rootold->left)->n);

CopyTree(rootnew, rootold->left);

}

if (rootold->right != 0)

{

Push(rootnew, (rootold->right)->n);

CopyTree(rootnew, rootold->right);

}

}

Tree::Tree(const Tree&ob)

{

if (ob.root == 0)root = 0;

else {

root = new node;

root->n = ob.root->n;

root->count = 1;

root->left = 0;

root->right = 0;

CopyTree(root, ob.root);

}

}

void Tree::DelTree(node \*wer)

{

if (wer->left != 0)DelTree(wer->left);

if (wer->right != 0)DelTree(wer->right);

delete wer;

}

void Tree::Push(node\*&wer, int data)

{

if (wer == 0)

{

wer = new node;

wer->n = data;

wer->left = 0; wer->right = 0;

wer->count = 1;

}

else if (data < wer->n)Push(wer->left, data);

else if (data > wer->n)Push(wer->right, data);

else wer->count++;

}

void Tree::Look(node\*wer)

{

if (wer != 0)

{

Look(wer->left);

cout << "Число: " << wer->n << " - " << wer->count;

cout << " штук" << endl;

Look(wer->right);

}

}

node\* Tree::Find(node\*wer, int key)

{

if (wer == 0) return 0; else if (key < wer->n) return Find(wer->left, key); else if (key > wer->n) return Find(wer->right, key);

else return wer;

}

void Tree::PrintLeaves(node \*wer)

{

if (wer == 0) return;

else if ((wer->left == 0) && (wer->right == 0)) {

cout << "Число: " << wer->n << " - " << wer->count;

cout << "штук" << endl;

}

else

{

PrintLeaves(wer->left);

PrintLeaves(wer->right);

}

}

//-------------------------------- MAIN ----------------------------------------

int main()

{

Tree tr;

node \*u;

int k = 0, max, kol;

char menu[][100] = { {" PushElement "}, {" ShowTree "}, {" FindElement "}, {" PrintLeaves "}, {" EXIT "}, };

kol = 5;//КОЛИЧЕСТВО СТРОК МЕНЮ. Используется в выравнивании строк меню по центру.

while (1) {

switch (kol)

{

case 0: {

int data;

cout << "Введите число:";

cin >> data;

tr.Push(tr.root, data);

k = 0; break;

}

case 1: {

if (tr.root == 0) cout << "Дерево пустое";

else

{

cout << "Наше дерево:" << endl;

tr.Look(tr.root);

}

while (!\_kbhit());

k = 1; break;

}

case 2: {

if (tr.root == 0)cout << "Дерево пустое";

else {

int key;

cout << "Введите искомое число:";

cin >> key;

if ((u = tr.Find(tr.root, key)) != 0) {

cout << "Элементов: " << key << " найдено ";

cout << u->count << " штук";

}

else cout << "Таких элементов нет!";

}

while (!\_kbhit());

k = 2;

break;

}

case 3: {

if (tr.root == 0)cout << "Дерево пустое";

else {

cout << "Листья:" << endl;

tr.PrintLeaves(tr.root);

}

while (!\_kbhit());

k = 3; break;

}

case 4: {

exit(0);

}

}

}

return 0;

}

# Задание для самостоятельного решения

При решении задач необходимо описать класс, который используется для представления элементов динамической структуры данных. Затем разрабатывается класс для работы с используемой динамической структурой данных, которая при тестировании класса может быть построена путем ввода данных: a) с клавиатуры; б) из файла.

Возможны два варианта решения:

а) динамическая структура данных постоянно хранится в памяти;

б) динамическая структура данных хранится в файле.

1. Построить класс для работы с односвязным списком. Элементы списка – целые числа. Сформировать список, упорядочить элементы списка по возрастанию, используя сортировку: a) методом выбора; б) методом пузырька; в) методом вставки.