Государственное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ   
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Донецкий национальный технический университет»

Д09.03.04-ПОИС.21-20/5847.ЛР

***Кафедра*** искусственного интеллекта  
 и системного анализа

Лабораторная работа №1

по дисциплине " Объектно-ориентированное программирование"

на тему: " Реализация контейнерного класса динамической структуры данных. Инкапсуляция. Перегрузка функций и операторов."

Проверили:

\_\_\_\_\_\_\_\_\_\_\_\_ асс. А.П. Семёнова

(дата, подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ асс. И.В. Савицкая

(дата, подпись)

Выполнил:

\_\_\_\_\_\_\_\_\_\_\_\_ ст.гр.ПИ-20г М. А. Евсеев

(дата, подпись)

Донецк – 2021

Тема: Реализация контейнерного класса динамической структуры данных. Инкапсуляция. Перегрузка функций и операторов.

Цель работы: Изучить основы программирования на языке С++. Получить практические навыки составления программ на С++, описания классов, перегрузки операторов и функций.

Вариант 11

Задание: Реализовать и протестировать контейнерный класс динамической структуры данных, содержащей строки. Класс должен иметь интерфейс АТД для добавления, удаления и поиска элементов, а также содержать следущие функции-члены:

А) перегруженные конструкторы:

по умолчанию;

копирования;

с параметрами по умолчанию;

б) деструктор;

в) перегруженные операции:

добаввление элемента +;

удаление элемента -;

индексирование [];

копирование =;

отношение равенства ==;

отношение порядка >;

Вывод значения контейнера в стандартный поток вывода <<;

г) объявление и реализация дружественной функции.

Листинг программы

#include <iostream>

#include <windows.h>

using namespace std;

int tabs=0;

int compare(string s1, string s2);

class A {

friend A\* tree(const string& s, A\* a);

friend bool find(A\* a, const string& s);

//friend void del\_tree(A\* a);

friend A\* del\_tree(A\* a);

friend void print(A\* a);

private:

int k;

int count;

string s;

A\* right = 0, \* left = 0;

void c\_plus(A\* a) {

if (a == 0) return;

a->count++;

c\_plus(a->right);

c\_plus(a->left);

}

public:

A() :s("empty"), count(1),k(1) {}//по умолчанию

A(const string& s, int c) :s(s), count(c) {}// с параметрами

A(const A& a) :s(a.s), right(a.right), left(a.left) {} // копирования

string get\_s() {

return s;

}

int get\_c() {

return count;

}

//увеличивает кол-во повторений всех значений веток данного дерева

A\* operator++(int) {

c\_plus(this);

return this;

}

//увеличивает кол-во повторений всех значений веток данного дерева

A\* operator++() {

return (\*this)++;

}

//удаляет самый низший правый элемент

A\* operator--() {

if (this->right != 0) {

if (this->right->right != 0) {

--(\*this->right);

}

else {

this->right = del\_tree(this->right);

this->right = 0;

return this;

}

}

return this;

}

//удаляет самый низший левый элемент

A\* operator--(int) {

if (this->left != 0) {

if (this->left->left != 0) {

--(\*this->left);

}

else {

this->left = del\_tree(this->left);

this->left = 0;

return this;

}

}

return this;

}

//есть ли строка в дереве

bool operator[](const string& a) {

return find(this, a);

}

//удаляет ветку с этим значением

A\* operator-(const string& a) {

if (this) {

A\* t = this;

int x = compare(a, this->get\_s());

cout << x<<endl;

if (!x) return del\_tree(this);

if (x > 0) t->right = \*this->right - a;

if (x < 0) t->left = \*this->left - a;

}

return this;

}

//добавление элемента

A\* operator+(const string& a) {

return tree(a, this);

}

bool operator>(A\* a) {

if (this->k > a->k)

return true;

return false;

}

bool operator<(A\* a) {

if (this->k < a->k)

return true;

return false;

}

bool operator== (A\* a) {

if (this->k == a->k)

return true;

return false;

}

//коприрование

A& operator = (A& a) {

cout << a.count;

k = a.k;

count = a.count;

s = a.s;

left = new A(\*a.left);

right = new A(\*a.right);

return \*this;

}

~A() { del\_tree(this); } //деструктор

};

//увеличивает кол-во повторений всех значений веток данного дерева

//сравнение 2 строк

int compare(string s1, string s2) {

int a = s1.length() + 1, b = s2.length() + 1;

char\* text1 = new char[a];

char\* text2 = new char[b];

for (size\_t i = 0; i < a; i++)

text1[i] = s1[i];

for (size\_t i = 0; i < b; i++)

text2[i] = s2[i];

int x = strcmp(text1, text2);

free(text1);

free(text2);

return x;

}

//вывод дерева

void print(A\* a){

if (!a) return; //Если ветки не существует - выходим. Выводить нечего

tabs += 5; //Иначе увеличим счетчик рекурсивно вызванных процедур

//Который будет считать нам отступы для вывода

print(a->left); //Выведем ветку и ее подветки слева

for (int i = 0; i < tabs; i++) cout << " "; //Потом отступы

cout << a->count<<" \_ "<<a->k<<" \_ " << " " << a->s << endl; //Данные этой ветки

print(a->right);//И ветки, что справа

tabs -= 5; //После уменьшим кол-во отступов

}

bool find(A\* a,const string& s) {

if (!a) return false;

int x = compare(s, a->get\_s());

if (!x) return true;

if (x > 0) return find(a->right, s);

if (x < 0) return find(a->left, s);

return false;

}

A\* tree(const string& s, A\* a) {

if (!a) {

a = new A();

a->s = s;

}

else {

a->k++;

int x = compare(s, a->get\_s());

if (!x) {

a->count++;

}

else if (x > 0) {

a->right = tree(s, a->right);

}

else {

a->left = tree(s, a->left);

}

}

return a;

}

A\* del\_tree(A\* a) {

if (a) {

del\_tree(a->left);

del\_tree(a->right);

free(a);

}

return 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

A\* node = 0, \* p = 0;

int n;

string s;

cout << "Введите кол-во веток бинарного дерева #1:\n";

cin >> n;

for (size\_t i = 0; i < n; i++) {

cin >> s;

node = (\*node) + s;

}

cout << "Введите кол-во веток бинарного дерева #2:\n";

cin >> n;

for (size\_t i = 0; i < n; i++) {

cin >> s;

p = (\*p) + s;

}

print(node);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

print(p);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nСравниваем дерево: \t" << ((\*node) == (p)) << endl;

cout << "Введите элемент который будет проверен на хранение в дереве:\n";

cin >> s;

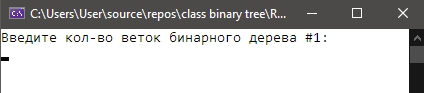
cout << ((\*node)[s]);

return 0;

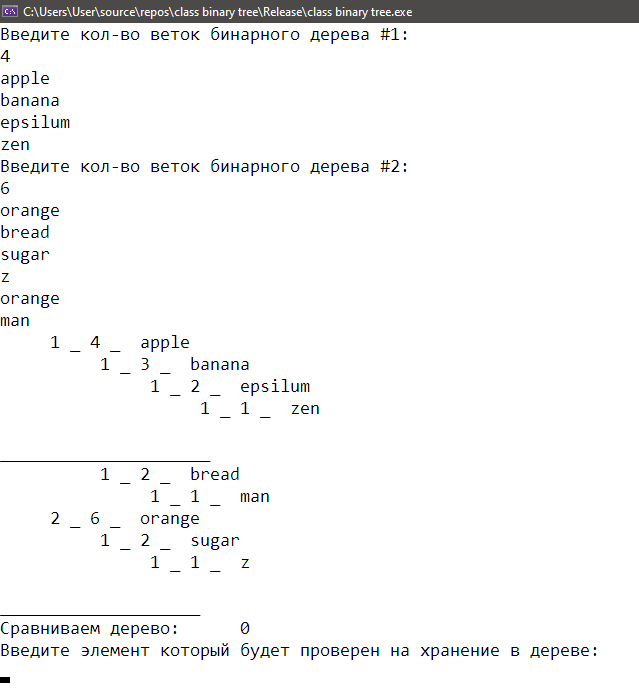
}

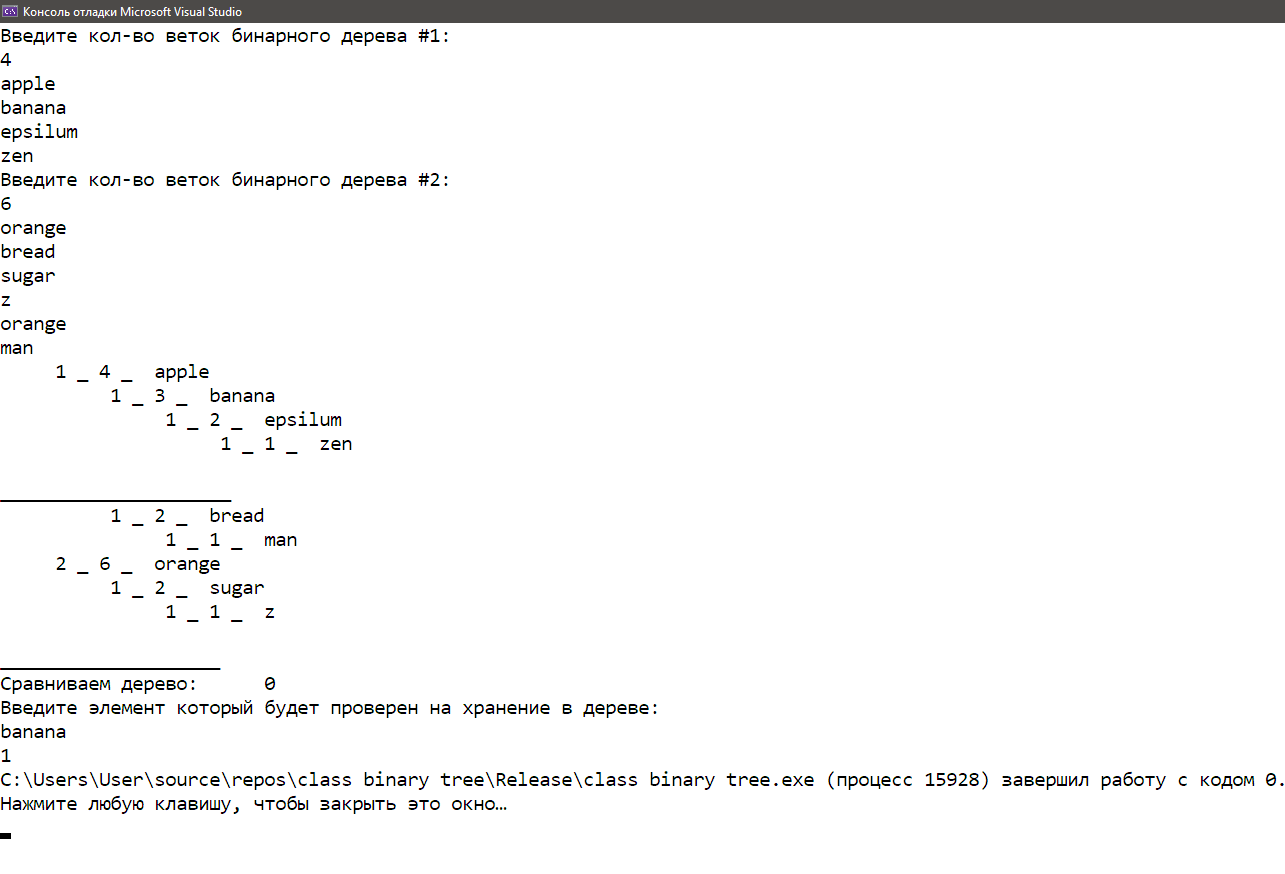
Скриншоты программы

После запуска программы мы должны ввести 2 бинарных дерева – количество элементов и сами строки:



После ввода, программа выведет и сравнит эти деревья по количеству элементов и попросит нас ввести элемент для проверки на его существование в первом дереве.



После ввода строки, программа выдаст 1 или 0 – существует она или нет. 

Вывод: Изучил основы программирования на языке С++. Получил практические навыки составления программ на С++, описания классов, перегрузки операторов и функций.