**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра АПУ**

отчет

**по лабораторной работе № 6**

**по дисциплине «Программирование»**

**Тема: Бинарное дерева поиска**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3371 |  | Манешов Д. В. |
| Преподаватель |  | Писарев А.С. |

Санкт-Петербург

2024

## Постановка задачи.

Необходимо реализовать структуру данных бинарного дерева поиска Tree

Данная структура должна обладать следующим набором функций:

* void add(int) - добавление элемента в бинарное дерево
* void remove(int) - удаляет из бинарного дерева элемент (если такого элемента нет ничего не делает)
* clear() - удаление всех элементов коллекции
* bool contains(int) - Определяет, содержит ли объект Tree указанный элемент.
* int count() - возвращает количество элементов в коллекции

Данная структура данных будет подвергнута тестированию автоматическими unit-тестами, поэтому наименование функций (и названия структуры) и сигнатуру изменять не следует.

## Ход решения.

Данная программа реализует структуру данных бинарного дерева поиска (БДП) для работы с целыми числами. В структуре Tree предусмотрен ряд основных операций: добавление, удаление и проверка наличия значения в дереве, а также очистка дерева и подсчет числа элементов.

Этот фрагмент кода (рис. 1.1) включает библиотеку <iostream>, которая предоставляет функциональность для ввода и вывода данных в C++. Благодаря этой библиотеке доступны такие объекты, как std::cin и std::cout, которые позволяют читать входные данные с консоли и выводить данные на экран. Использование выражения using namespace std; упрощает доступ к элементам из пространства имен std, избавляя от необходимости указывать префикс std:: перед каждым элементом, таким как cout и cin.



Рисунок 1.1 – Подключение библиотеки и пространство имён

В данном фрагменте кода (рис. 1.2) определяется структура Node, которая используется для представления узла в бинарном дереве поиска. Каждый узел содержит целочисленное значение data, а также два указателя: left и right, которые указывают на левое и правое поддеревья соответственно. Конструктор Node принимает значение value и присваивает его полю data. Указатели left и right изначально устанавливаются в nullptr, что означает отсутствие дочерних узлов.

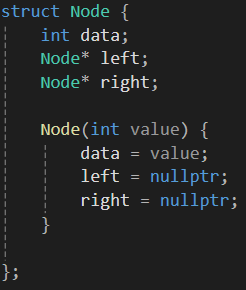


Рисунок 1.2 – Определение структуры Node

Данный фрагмент кода (рис. 1.3) представляет начало объявления класса Tree, который реализует структуру данных бинарного дерева поиска. В приватной области класса определены два члена: целочисленная переменная size и указатель root. Переменная size отслеживает общее количество узлов, присутствующих в дереве, что помогает быстро получать информацию о размере дерева. Указатель root указывает на корневой узел дерева, с которого начинаются все операции добавления, удаления и поиска. Изначально, при создании нового экземпляра класса, корень будет установлен в значение nullptr, что сигнализирует о пустом дереве.

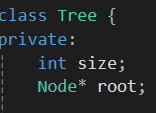


Рисунок 1.3 – Объявление класса Tree

Следующий фрагмент кода (рис. 1.4) содержит конструктор и деструктор класса Tree управляют инициализацией и освобождением ресурсов для структуры данных бинарного дерева поиска. Конструктор Tree() устанавливает начальные значения для приватных членов: переменная size устанавливается в ноль, указывая на то, что дерево изначально пусто, а указатель root устанавливается в nullptr, что означает отсутствие корневого узла.

Деструктор ~Tree() освобождает ресурсы, вызвав метод clear(), который удаляет все узлы в дереве, начиная с корневого и двигаясь рекурсивно по всем дочерним узлам.

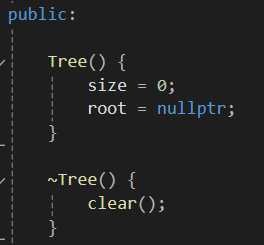


Рисунок 1.4 – Конструктор и деструктор класса Tree

Метод add (рис. 1.5) добавляет новый узел в бинарное дерево поиска. При добавлении нового значения value создается новый узел с этим значением. Если дерево пустое (корень root равен nullptr), новый узел становится корнем дерева. В противном случае, метод перемещается от корня к листьям, чтобы найти место для нового узла, сравнивая значение value с данными в узлах. Если value меньше, чем данные текущего узла, метод двигается влево по дереву. Если value больше, чем данные текущего узла, метод двигается вправо. Этот процесс продолжается до тех пор, пока не будет найдено место для вставки нового узла.

Если значение value уже присутствует в дереве, новый узел не добавляется, и память, выделенная для него, освобождается с помощью оператора delete.

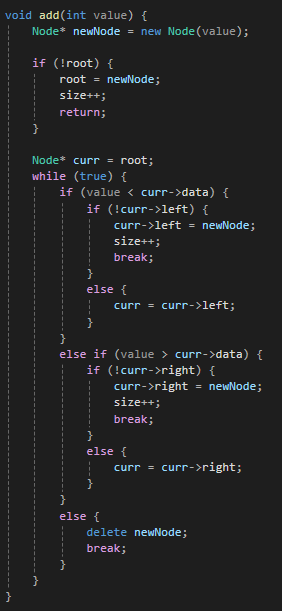


Рисунок 1.5 – Метод добавления элемента в дерево

Метод contains (рис. 1.6) в классе Tree проверяет наличие заданного значения value в бинарном дереве поиска. Метод начинает поиск с корневого узла дерева, хранящегося в переменной curr. Цикл продолжается, пока не будет найден узел с искомым значением или пока не будут проверены все узлы в возможном пути (т.е., пока curr не станет nullptr). Если значение меньше данных текущего узла (curr->data), поиск продолжается в левом поддереве, если больше – в правом. Когда значение равно данным в узле, метод возвращает true, подтверждая наличие значения в дереве. Если после обхода всех узлов нужное значение так и не было найдено, метод возвращает false, указывая на отсутствие значения в дереве.

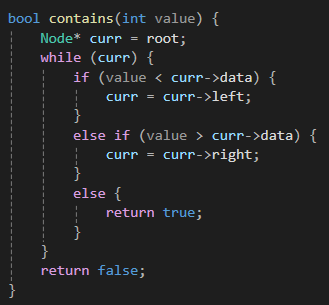


Рисунок 1.6 – Метод для проверки наличия значения value в классе Tree

Метод removeHelper (рис. 1.7) в классе Tree рекурсивно удаляет узел с заданным значением value из бинарного дерева поиска. Если текущий узел node равен nullptr, это означает, что узел с заданным значением отсутствует в дереве, и метод завершает выполнение. Если искомое значение меньше значения в текущем узле, метод вызывается рекурсивно для левого поддерева. Если значение больше — для правого поддерева. Если значение найдено (значение в узле равно искомому), происходит удаление: если у узла нет детей, он удаляется напрямую; если есть только левый или только правый ребенок, узел удаляется, а его ребенок занимает его место; если у узла два ребенка, находится минимальный узел в правом поддереве. Этот минимальный узел заменяет удаляемый, после чего удаляется рекурсивно, чтобы сохранить свойства бинарного дерева поиска. После удаления каждого узла переменная size уменьшается, что отражает общее количество узлов в дереве.

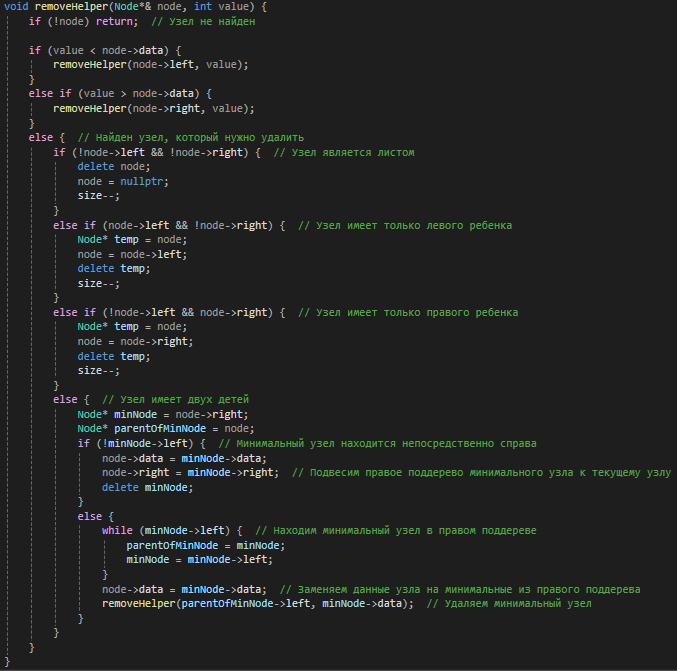


Рисунок 1.7 – Метод removeHelper для рекурсивного удаления узла с заданным значением value

Метод remove в классе Tree используется для удаления узла с заданным значением value из бинарного дерева поиска. Он делегирует выполнение этой задачи методу removeHelper, передавая ему корневой узел root и значение value.

Метод count возвращает текущее количество узлов в дереве, хранящееся в переменной size. Это позволяет легко получить общее число узлов в дереве без необходимости пересчитывать их.

Метод clear полностью очищает дерево, удаляя все его узлы и устанавливая значение root в nullptr, а переменную size обнуляет. Для удаления всех узлов вызывается вспомогательный метод clearNode, который рекурсивно обходит дерево, начиная с корневого узла. Метод clearNode вызывает сам себя для левого и правого поддеревьев каждого узла, а затем удаляет сам узел. Этот метод гарантирует, что все узлы дерева будут корректно удалены, предотвращая утечки памяти.

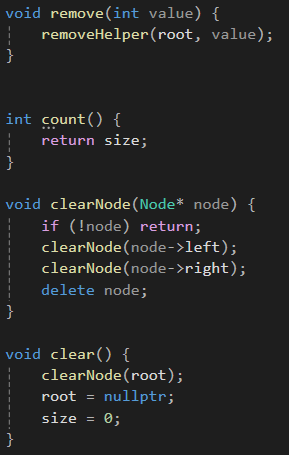


Рисунок 1.8 – Методы remove, count и clear в классе Tree

Функция main реализует текстовый интерфейс пользователя для управления экземпляром класса Tree, который позволяет добавлять и удалять элементы, очищать дерево, получать количество элементов в дереве и завершать программу. Пользовательский ввод считывается через cin, где программа ожидает команды в виде строк.

Команда "add": после этой команды программа ожидает целочисленное значение, которое добавляется в дерево методом add(data).

Команда "remove": следующее за этой командой целочисленное значение удаляется из дерева методом remove(data).

Команда "clear": при вызове этой команды дерево полностью очищается с помощью метода clear().

Команда "count": выводит текущее количество узлов в дереве на консоль.

Команда "stop" или любая другая команда, не соответствующая перечисленным: приводит к завершению бесконечного цикла и, соответственно, к завершению программы.

Программа организована таким образом, что она продолжает выполняться, пока пользователь не введет "stop" или не введет команду, не соответствующую ожидаемым. Это позволяет проводить серию операций над деревом без перезапуска программы.

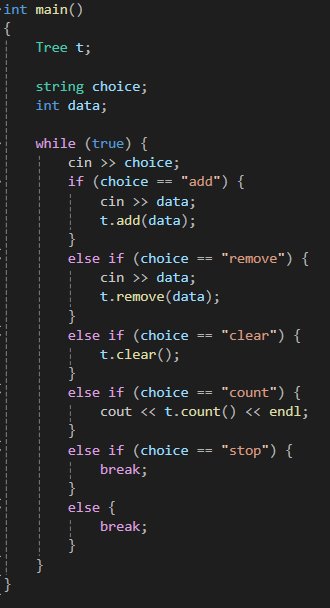


Рисунок 1.8 – Основная функция main для управления классом Tree

**Выводы.**

В ходе выполнения лабораторной работы была разработана программа, реализующая структуру данных бинарного дерева поиска Tree. Данная структура включает в себя основные функции, необходимые для управления коллекцией числовых значений: add(int), remove(int), clear(), contains(int) и count(). Функция add(int) позволяет добавлять элементы в дерево, обеспечивая поддержание его свойств бинарного дерева поиска. Метод remove(int) служит для удаления элемента из дерева, при этом если элемент отсутствует, функция ничего не делает, сохраняя стабильность структуры. Функция clear() эффективно очищает дерево от всех элементов, обнуляя его размер и структуру. Метод contains(int) предоставляет возможность проверить наличие конкретного элемента в дереве, что является важной функциональностью для любой коллекционирующей структуры данных. Возврат количества элементов в дереве осуществляется через функцию count(), что позволяет легко получить размер коллекции в любой момент времени.

# Список использованной литературы

1. Павловская Т.А., Чаевников В.В., Юрков Н.К. Программирование на языке С++. Электронное методическое пособие. СПб.: Издательство СПбГЭТУ «ЛЭТИ». 2014.
2. Практикум по процедурному программированию на языке C++ С. А. Ивановский, А. А. Лисс, В. П. Самойленко, О. М. Шолохова. Практикум по процедурному программированию на языке C++: учеб. пособие. СПб.: Издательство СПбГЭТУ«ЛЭТИ», 2016.
3. Справочник по стандартной библиотеке C++ (STL) // Справочный сайт learn.microsoft.com URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/cpp-standard-library-reference?view=msvc-170 (дата обращения 09.03.2024).

Документация по языку C++ // Справочный сайт learn.microsoft.com URL: https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения 10.03.2024).

# Приложение

## Код написанной программы

Файл «Lab6.cpp»:

// Lab6.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = nullptr;

right = nullptr;

}

};

class Tree {

private:

int size;

Node\* root;

public:

Tree() {

size = 0;

root = nullptr;

}

~Tree() {

clear();

}

void add(int value) {

Node\* newNode = new Node(value);

if (!root) {

root = newNode;

size++;

return;

}

Node\* curr = root;

while (true) {

if (value < curr->data) {

if (!curr->left) {

curr->left = newNode;

size++;

break;

}

else {

curr = curr->left;

}

}

else if (value > curr->data) {

if (!curr->right) {

curr->right = newNode;

size++;

break;

}

else {

curr = curr->right;

}

}

else {

delete newNode;

break;

}

}

}

bool contains(int value) {

Node\* curr = root;

while (curr) {

if (value < curr->data) {

curr = curr->left;

}

else if (value > curr->data) {

curr = curr->right;

}

else {

return true;

}

}

return false;

}

void removeHelper(Node\*& node, int value) {

if (!node) return; // Узел не найден

if (value < node->data) {

removeHelper(node->left, value);

}

else if (value > node->data) {

removeHelper(node->right, value);

}

else { // Найден узел, который нужно удалить

if (!node->left && !node->right) { // Узел является листом

delete node;

node = nullptr;

size--;

}

else if (node->left && !node->right) { // Узел имеет только левого ребенка

Node\* temp = node;

node = node->left;

delete temp;

size--;

}

else if (!node->left && node->right) { // Узел имеет только правого ребенка

Node\* temp = node;

node = node->right;

delete temp;

size--;

}

else { // Узел имеет двух детей

Node\* minNode = node->right;

Node\* parentOfMinNode = node;

if (!minNode->left) { // Минимальный узел находится непосредственно справа

node->data = minNode->data;

node->right = minNode->right; // Подвесим правое поддерево минимального узла к текущему узлу

delete minNode;

}

else {

while (minNode->left) { // Находим минимальный узел в правом поддереве

parentOfMinNode = minNode;

minNode = minNode->left;

}

node->data = minNode->data; // Заменяем данные узла на минимальные из правого поддерева

removeHelper(parentOfMinNode->left, minNode->data); // Удаляем минимальный узел

}

}

}

}

void remove(int value) {

removeHelper(root, value);

}

int count() {

return size;

}

void clearNode(Node\* node) {

if (!node) return;

clearNode(node->left);

clearNode(node->right);

delete node;

}

void clear() {

clearNode(root);

root = nullptr;

size = 0;

}

};

int main()

{

Tree t;

string choice;

int data;

while (true) {

cin >> choice;

if (choice == "add") {

cin >> data;

t.add(data);

}

else if (choice == "remove") {

cin >> data;

t.remove(data);

}

else if (choice == "clear") {

t.clear();

}

else if (choice == "count") {

cout << t.count() << endl;

}

else if (choice == "stop") {

break;

}

else {

break;

}

}

}

Файл data1-1.txt:

add 10

add 20

add 15

count

remove 20

count

stop

Файл report1-1.txt:

3

2

Файл data2-1.txt:

add 30

add 40

add 35

count

remove 30

count

clear

count

stop

Файл report2-1.txt:

3

2

0

Файл data3-1.txt:

add 50

add 60

add 70

add 80

count

remove 60

remove 70

count

stop

Файл report3-1.txt:

4

2