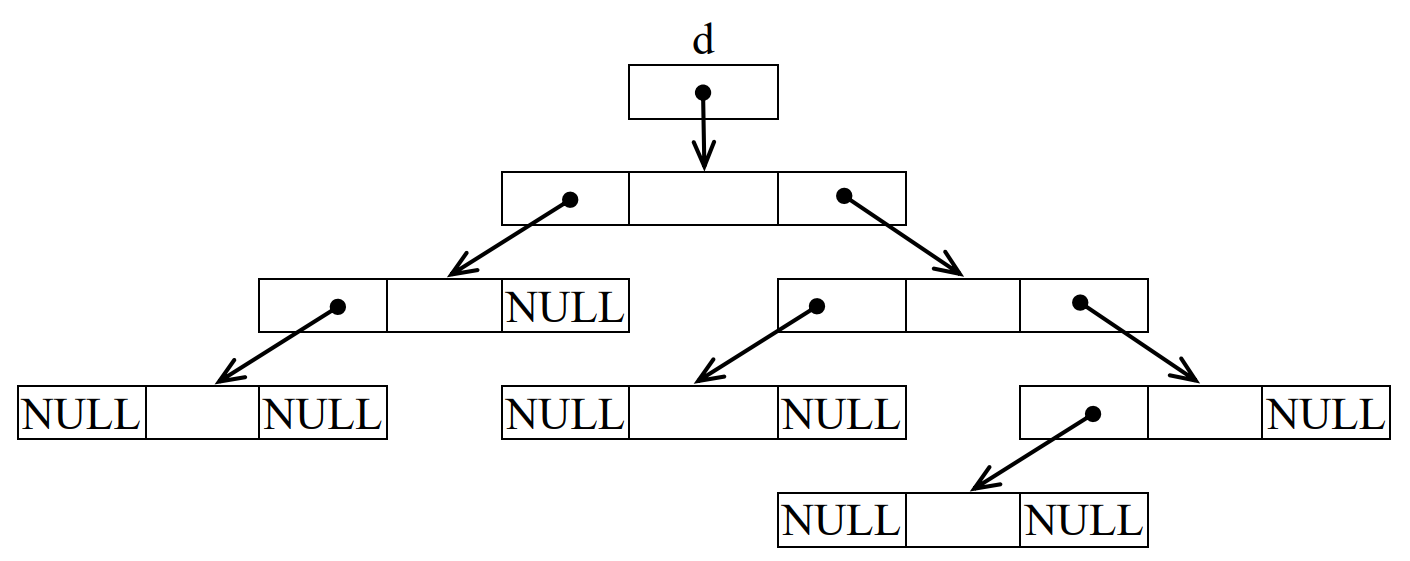
|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Основы программирования» | | |
| **Структуры данных: деревья** | | |
|  | | |
|  | Бригада | ФИО |
| №2 | Веселый Денис Андреевич |
|  | Ворончук Илья Игоревич |
| Группа | ПМИ-32 |
|  |  |
|  |  |
| Преподаватель | Дворецкая виктория константиновна |
|  |  |
| Новосибирск,2024 | | |

**1) Цель работы**

Изучить основные алгоритмы работы с деревьями; получить практические навыки разработки и использования этих структур и алгоритмов для решения задач**.**

**2) Условие задачи**

Используя очередь или стек (считать уже описанными их типы и операции над ними), опишите на C++ тип *бинарное дерево* (считая, что все элементы имеют некоторый простой тип) и реализуйте в виде процедуры или функции вычисление суммы всех элементов дерева.

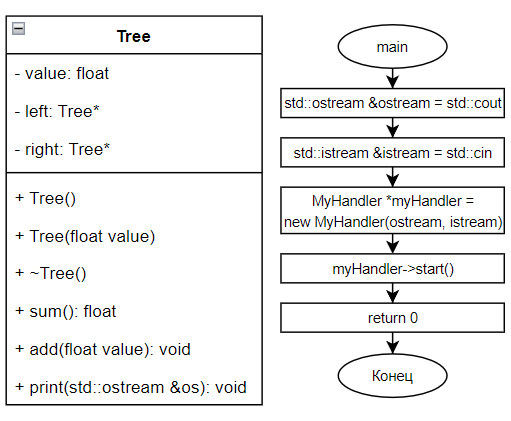
**Алгоритм решения задачи**

Все элементы из множества положительных вещественных чисел.

1. Суммирование элементов дерева реализовано через прямой итерационный обход дерева в глубину.

/\* Добавление нового элемента в дерево происходит следующим образом. Пусть добавляемый элемент – K, тогда любой элемент дерева, после которого новый будет помещён, обозначим как M. Если при обходе дерева K > M, то K помещается в *левое* поддерево элемента M, в противном случае – K помещается в *правое* поддерево элемента M; и в том, и в другом случае K становится листом дерева.

В силу того, что необходимо было кэшировать только один элемент, для реализации данного алгоритма был использован итерационный метод неполного обхода в глубину дерева, вместо рекурсивного. \*/

**3) Стандартная схема программы**

**4) Код программы**

**Tree.cpp**

#include <functional>

#include "proto/Tree.h"

#include "proto/Stack.h"

Tree::Tree()

{

left = nullptr;

right = nullptr;

value = 0;

}

Tree::Tree(float value)

{

left = nullptr;

right = nullptr;

this->value = value;

}

Tree::~Tree()

{

delete left, right;

left, right = nullptr, nullptr;

}

// суммирует все элементы дерева

float Tree::sum()

{

float res = 0;

Stack \*stack = new Stack;

Tree \*cur = this;

while (cur)

{

res += cur->value;

if (cur->left && cur->right)

{

stack->push(cur->right);

cur = cur->left;

}

else if (!cur->left && !cur->right)

if (stack->size() > 1)

cur = stack->pop();

else

cur = nullptr;

else

cur = cur->left ? cur->left : cur->right;

}

delete stack;

stack = nullptr;

return res;

}

// добавляет новое значение в дерево

void Tree::add(float value)

{

Tree \*t = new Tree(value);

Tree \*cur = this;

Tree \*buff;

while (cur)

{

buff = cur;

cur = value > cur->value ? cur->left : cur->right;

}

value > buff->value ? buff->left = t : buff->right = t;

}

// выводит дерево в поток вывода

void Tree::print(std::ostream &os)

{

// рекурсивная лямбда-функция

std::function<void(Tree \*, std::string, std::ostream &)> func;

func = [&](Tree \*tree, std::string str, std::ostream &os)

{

if (tree->right)

func(tree->right, str + "-", os);

os << str << tree->value << std::endl;

if (tree->left)

func(tree->left, str + "-", os);

};

func(this, "", os);

}

**5) Тестирование программы**

|  |  |  |
| --- | --- | --- |
| № | Команды | Комментарий |
| 1 | add 10 4 6 20 4  sum | 1. Добавление элементов ‘10’ ‘4’ ‘6’ ‘20’ ‘4’ в дерево. 2. Суммирование элементов и вывод результата в консоль – 44. |
| 2 | add 30 1 7 1000 8  prt | 1. Добавление элементов ‘30’ ‘1’ ‘7’ ‘1000’ ‘8’ в дерево. 2. Вывод дерева в консоль. |
| 3 | add 88 11 22 3  sum  prt | 1. Добавление элементов ‘88’ ‘11’ ‘22’ ‘3’ в дерево. 2. Суммирование элементов и вывод результата в консоль – 124. 3. Вывод дерева в консоль. |

**6) Результаты отладки**

Результаты работы программы показали, что программа работает успешно.