## НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

#### иикс

Кафедра №22 «Кибернетика»

# Лабораторная работа №3

#### Выполнил:

студент гр. Б17-514 Жарков М. А.

### Преподаватель:

Трифоненков А.В.

### Вариант №9

Коллекция: Бинарное дерево поиска

Типы хранимых данных: вещественные числа, комплексные числа

**Операции над коллекцией**: map, where, слияние, извлечение поддерева по заданному элементу, поиск элемента на вхождение, сохранение в строку в соответствии с обходами ЛПК и ПЛК.

**Бинарное дерево поиска** - это двоичное дерево, для которого выполняются следующие дополнительные условия:

- Оба поддерева левое и правое являются двоичными деревьями поиска.
- У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.
- У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

**Операция тар и where:** Если  $l = [a_1, \dots, a_n]$  - некоторый список элементов типа T, а  $f: T \to T$ , то:

$$\max(f,l)\mapsto \Big[fig(a_1ig),\dots,fig(a_nig)\Big]$$
 Если, при тех же соглашениях,  $h:T\to \operatorname{Bool}$  - некоторая функция, возвращающая булево значение, то результатом  $\operatorname{where}(h,l)$  будет новый список  $l'$ , такой что:  $a_i'\in l'\Leftrightarrow hig(a_i'ig)=\operatorname{true}$ . Т.е. where фильтрует значения из списка  $l$  с помощью функции-фильтра  $h$ .

**Извлечение поддерева по заданному элементу** - нахождение элемента в в дереве и удаление поддерева, корнем которого является данный элемент.

**Поиск элемента на вхождение** - булева операция, возвращающая истину, если элемент присутствует в дереве.

**Сохранение в строку в соответствии с обходом** - перевод дерева в формат строки в виде  $[\Lambda][\Pi]$ К или  $[\Pi][\Lambda]$ К.

### Техническое задание

Создать заголовочный файл с объявлением класса бинарного дерева tree.hpp и класса комплексных complex.cpp

Объявить шаблон класса Tree<T> со следующими полями и разработать реализацию методов этого шаблона:

Название поля	Описание	
Private		
Node <t>* Root</t>	Указатель на корень дерева	
<pre>void remove_subtree(Node<t> *)</t></pre>	вспомогательная функция для удаления поддерева	
<pre>Node<t>* merge(Node<t>*, Tree<t> &amp;);</t></t></t></pre>	вспомогательная функция для merge	
<pre>void print(Node<t>*, int);</t></pre>	вспомогательная функция для print	

Название поля	Описание	
<pre>int get_depth(Node<t>*, int);</t></pre>	вспомогательная функция для get_depth	
Public		
Tree();	Конструктор	
~Tree();	Деструктор	
<pre>void add_node(T);</pre>	Добавление элемента в дерево	
<pre>void remove_node(T);</pre>	удаление элемента из дерево по значению	
<pre>void remove_node(Node<t>*);</t></pre>	удаление элемента из дерево по узлу	
<pre>void clear_node(Node<t>*);</t></pre>	Обнуляет указатель	
<pre>string post_order_to_str(Node<t>* x);</t></pre>	обход левый-правый-корень, начиная с х	
<pre>string post_order_to_str();</pre>	обход левый-правый-корень, начиная с корня	
<pre>string reverse_post_order_to_str(Node<t>* x);</t></pre>	обход правый-левый-корень, начиная с х	
<pre>string reverse_post_order_to_str();</pre>	обход правый-левый-корень, начиная с корня	
<pre>Node<t>* find_node(T);</t></pre>	находит узел по значению	
<pre>bool is_in_tree(T);</pre>	проверка элемента на вхождение	
<pre>Node<t>* find_min_node(Node<t>*);</t></t></pre>	Находит минимальное значение, начиная с узла	
<pre>Node<t>* find_max_node(Node<t>*);</t></t></pre>	Находит максимальное значение, начиная с узла	
<pre>Node<t>* copy_node(Node<t>*, Tree<t>&amp;);</t></t></t></pre>	копирует узел	
<pre>Tree<t> copy_tree();</t></pre>	копирует дерево	
<pre>Tree<t> copy_tree(Node<t>*);</t></t></pre>	Вспомогательная функция для копирования дерева	
<pre>Node<t>* get_root();</t></pre>	Возвращает корень	
<pre>void remove_subtree(T val);</pre>	Удаляет поддерево	
<pre>typedef bool (*where_func)(T);</pre>	Where функция	
<pre>Node<t>* where(where_func, Node<t>*, Tree<t>&amp;);</t></t></t></pre>	Вспомогательная функция для where	
<pre>Tree<t> where(where_func wf);</t></pre>	Where функция	
<pre>Tree<t> merge(Tree<t> &amp;sec_tree);</t></t></pre>	прибавляет второе дерево к первому (слияние)	
<pre>typedef T (*map_func)(T);</pre>	Мар функция	
<pre>void map(Node<t> *, map_func);</t></pre>	вспомогательная функция для тар	
<pre>void map(map_func mf);</pre>	Мар функция	

Название поля	Описание
<pre>void print();</pre>	Вывод дерева в консоль
<pre>int get_depth();</pre>	находит глубину дерева

# Так же объявляется шаблон структуры Node<T>:

Название поля	Описание	
T data;	Значение узла	
Node <t>* left;</t>	Указатель на левого потомка	
Node <t>* right;</t>	Указатель на правого потомка	
Node <t>* parent;</t>	Указатель на родителя	
Node();	Конструктор	
Node(const T);	Деструктор	

### И класс комплексных:

Название поля	Описание	
Private		
Re	Действительная часть комплексного	
Im	Мнимая часть комплексного	
Public		
Complex()	Конструктор по умолчанию	
Complex (double)	Конструктор	
Complex (double, double)	Конструктор	
~Complex();	Деструктор	
<pre>double return_re()</pre>	Возвращает значение ге	
<pre>double return_im()</pre>	Возвращает значение im	
<pre>Complex operator+ (Complex); Complex operator+= (Complex);</pre>	Перегрузка операторов сложения с комплексным	
<pre>Complex operator* (Complex);</pre>	Перегрузка оператора умножения на комплексное	
<pre>Complex operator+ (double); Complex operator+= (double);</pre>	Перегрузка операторов сложения с вещественным	
<pre>Complex operator* (double); Complex operator*= (double);</pre>	Перегрузка оператора умножения на вещественное	

Название поля	Описание
<pre>bool operator== (Complex); bool operator!= (Complex);</pre>	Перегрузка операторов сравнения с комплексным
<pre>bool operator== (double); bool operator!= (double);</pre>	Перегрузка операторов сравнения с вещественным
<pre>friend std::ostream &amp;   operator&lt;&lt;(std::ostream &amp;os, const   Complex &amp;c);</pre>	Перегрузка оператора вывода
<pre>bool operator&lt; (Complex); bool operator&lt;= (Complex);</pre>	Перегрузка операторов сравнения
<pre>bool operator&gt; (Complex); bool operator&gt;= (Complex);</pre>	Перегрузка операторов сравнения

# Пользовательский интерфейс

В программе реализован консольный интерфейс, с помощью которого пользователь создает дерево и работает с ним. Для работы с пользовательским интерфейсом необходимо подключить файл **UI.hpp**, а для его запуска необходимо вызвать метод **run\_UI()**.

С помощью пользовательского интерфейс программа может выполнять следующие команды:

Номер команды	Результат выполнение команды
1	Добавление элемента в дерево
2	Удаление элемента из дерева
3	Вывод дерева в консоль
4	Использование тар функции
5	Использование where функции
6	Слияние двух деревьев
7	Извлечение поддерева
8	Поиск элемента на вхождение
9	Вывод в формате ЛПК
10	Вывод в формате ПЛК
11	Запуск тестов

Номер команды	Результат выполнение команды
12	Выход из программы

При добавлении элемента пользователь выбирает тип значения, с которым будет работать дерево (вещественные или комплексные), а далее вводит это значение.

При удалении элемента, пользователь вводит значение, и оно (если оно есть в дереве) удаляется.

Использование функции тар возводит значения всех элементов дерева в квадрат.

Использование функции where удаляет все неположительные элементы.

При использовании merge программа предложит ввести второе дерево, после чего произведет слияние деревьев.

При использовании удаления поддерева программа попросит пользователя ввести значение, после чего удалит поддерево, корнем которого будет являться это значение.

При выводе обхода программа выведет дерево в формате [Левый потомок][Правый потомок]Корень или [Правый потомок][Левый потомок]Корень

### Программный интерфейс

Реализован тип данных Tree. Этот тип данных представляет собой концепцию бинарного дерева поиска.

Список основных методов класса Tree<T>:

Название метода	Аргументы	Описание
<pre>void add_node(T);</pre>	Значение типа Т	Добавление элемента в дерево
<pre>void remove_node(T);</pre>	Значение типа Т	удаление элемента из дерево по значению
<pre>string post_order_to_str();</pre>		Выводит строку в соответствии с обходом ЛПК
<pre>string reverse_post_order_to_str();</pre>		Выводит строку в соответствии с обходом ПЛК
<pre>Node<t>* find_node(T);</t></pre>	Значение типа Т	Находит узел со значением
<pre>bool is_in_tree(T);</pre>	Значение типа Т	проверка элемента на вхождение
<pre>void remove_subtree(T val);</pre>	Значение типа Т	Удалет поддерево
<pre>Tree<t> where(where_func wf);</t></pre>	wf – функция where	Where функция
<pre>void map(map_func mf);</pre>	mp — функция map	Мар функция

Название метода	Аргументы	Описание
<pre>void print();</pre>		Выводит дерево в консоль
<pre>int get_depth();</pre>		Возвращает глубину дерева

### Тестирование

Тестирование разбито на 2 раздела - тестирование класса комплексных (3 модуля) и класса дерева (5 модулей).

Файл с реализвацией тестирования tests. hpp находится в папке /Tests, а модули тестирования - в папках /Tests/Complex Tests/ и /Tests/Tree Tests/.

Для запуска тестирования нужно вызвать метод run\_all\_tests() или выбрать соответствующую команду в меню.

Список модулей тестирвания:

Раздел	Номер модуля	Файл модуля	Тестируемые методы
	1	/Tests/Complex Tests/ complex_unit_1.cpp	<pre>Complex(); Complex(double); Complex(double,double)</pre>
Complex	2	/Tests/Complex Tests/complex_unit_2.cpp	Перегрузка операторов сложения и произведения
	3	/Tests/Complex Tests/complex_unit_3.cpp	Перегрузка операторов сравнения
	1	/Tests/Tree Tests/ list_unit_1.cpp	Создание дерева и удаление элементов
	2	/Tests/Tree Tests/ list_unit_2.cpp	Удаление из дерева
Tree	3	/Tests/Tree Tests/ list_unit_3.cpp	Вывод дерева в соответствии с обходом
	4	/Tests/Tree Tests/ list_unit_4.cpp	Удаление поддерева и поиск элемента на вхождение
	5	/Tests/Tree Tests/ list_unit_5.cpp	map, where, merge

# Приложение

```
ptr1->right=n;
tree.cpp:
                                                 }
                                             }
#include "tree.hpp"
                                             template <typename T>
using namespace std;
                                             void Tree<T>::remove node(Node<T>* x){
                                                 if (root == nullptr || x ==
template <typename T>
                                             nullptr) return;
Node<T>::Node(){
                                                 if (x == root)
    data = NULL;
    left = nullptr;
                                                     if (x->left \&\& x->right) {
    right = nullptr;
                                                         Node<T> *max_node =
}
                                             find_max_node(x->left);
                                                         x->data = max_node->data;
template <typename T>
                                                          remove_node(max_node);
Node<T>::Node(const T x){
                                                          return;
    data = x;
    left = nullptr;
                                                     else if (x->left) {
    right = nullptr;
                                                         root = x->left;
}
                                                     else if (x->right) {
template <typename T>
                                                         root = x->right;
Tree<T>::Tree() {
    root = nullptr;
                                                     else {
}
                                                         root = nullptr;
//template <typename T>
                                                     free(x);
//Tree<T>::Tree(Node<T> &rt) {
                                                     clear_node(x);
//
      root = copy_node(&rt);
                                                 }
//}
                                                 else
//
//template <typename T>
                                                     if (x->left \&\& x->right){
//Tree<T>::Tree(Tree<T> &tr) {
                                                         Node<T> *max_node =
//
//}
      root = copy_node(tr.root);
                                             find_max_node(x->left);
                                                         x->data = max_node->data;
                                                         remove_node(max_node);
template <typename T>
                                                          return;
Tree<T>::~Tree() {
                                                     }
      delete root;
                                                     else if (x->left){
                                                         if (x == x->parent->left)
                                             x->parent->left = x->left;
template <typename T>
                                                         else x->parent->right = x-
void Tree<T>::add node(T x){
                                             >left;
    Node<T>* n = \overline{\text{new Node}} < T > (x);
    Node<T>* ptr;
                                                     else if (x->right){
    Node<T>* ptr1 = nullptr;
                                                         if (x == x->parent->right)
                                             x->parent->right = x->right;
    n->left = 0;
                                                         else x->parent->left = x-
    n->right = 0;
                                             >right;
    n-parent = 0;
    ptr = root;
                                                     else{
    while (ptr != 0) {
                                                         if (x == x-\text{-parent-->left})
        ptr1 = ptr;
                                             x->parent->left = nullptr;
        if (x < ptr->data)
                                                         else x->parent->right =
             ptr = ptr->left;
                                             nullptr;
        else
             ptr = ptr->right;
                                                     free(x);
    }
                                                     clear_node(x);
                                                 }
    n->parent = ptr1;
                                             }
    if (ptr1 == 0)
        root = n;
                                             template <typename T>
    else{
                                             void Tree<T>::remove_node(T val){
        if(x < ptr1->data)
                                                 Node<T>* n = find_node(val);
            ptr1->left=n;
                                                 remove_node(n);
        else
                                             }
```

```
template <typename T>
                                                while(current_node != nullptr){
void Tree<T>::clear_node(Node<T> *n){
                                                     if (val < current_node->data)
    n = nullptr;
                                            current_node = current_node->left;
    return;
                                                     else if (val > current_node-
}
                                            >data) current_node = current_node-
                                            >right;
template <typename T>
                                                     else if (val == current node-
                                            >data){
Tree<T>::post_order_to_str(Node<T>* x)
                                                         return current_node;
                                                         break:
    stringstream ss;
    if (x != 0){
        if (x->left != NULL || x-
                                                     if (current_node == nullptr)
>right != NULL) ss<<"[";
                                            return nullptr;
        ss<<this->post_order_to_str(x-
                                                 return nullptr;
>left);
                                            }
        if (x->left != NULL || x-
>right != NULL) ss<<"][";
                                            template <typename T>
        ss<<this->post_order_to_str(x-
                                            bool Tree<T>::is_in_tree(const T val){
>right);
                                                 if (find_node(val) == NULL) return
        if (x->left != NULL || x-
                                            false;
>right != NULL) ss<<"]";</pre>
                                                else return true;
        ss<<x->data;
                                            }
    string s = ss.str();
                                            template <typename T>
    return s;
}
                                            Node<T>*
                                            Tree<T>::find_min_node(Node<T> *n){
                                                Node<T>* current_node = n;
template <typename T>
                                                while (current_node->left !=
string Tree<T>::post_order_to_str() {
                                            nullptr)
    return post_order_to_str(root);
                                                     current_node = current_node-
                                            >left;
template <typename T>
                                                 return current_node;
string
Tree<T>::reverse_post_order_to_str(Nod
e<T>* x) {
    stringstream ss;
                                            template <typename T>
    if (x != 0){
                                            Node<T>*
        if (x->left != NULL || x-
                                            Tree<T>::find_max_node(Node<T> *n){
>right != NULL) ss<<"[";
                                                Node<T>* current_node = n;
                                                while (current_node->right !=
        ss<<this-
>reverse_post_order_to_str(x->right);
    if (x->left != NULL || x-
                                            nullptr)
                                                     current_node = current_node-
>right != NULL) ss<<"][";
                                            >right;
        ss<<this-
                                                 return current_node;
>reverse_post_order_to_str(x->left);
        if (x->left != NULL || x-
                                            }
>right != NULL) ss<<"]";</pre>
                                            template <typename T>
        ss<<x->data;
                                            Node<T>* Tree<T>::copy_node(Node<T>
                                            *node_to_copy, Tree<T> &new_tree){
    string s = ss.str();
                                                if (node_to_copy != nullptr)
    return s;
template <typename T>
                                            new_tree.add_node(node_to_copy->data);
string
                                                     copy_node(node_to_copy->left,
Tree<T>::reverse_post_order_to_str() {
                                            new_tree);
                                                     copy_node(node_to_copy->right,
    return
reverse_post_order_to_str(root);
                                            new_tree);
                                                 return new_tree.root;
                                            }
template <typename T>
Node<T>* Tree<T>::find_node(T val){
```

Node<T>\* current node = root;

```
template <typename T>
Tree<T> Tree<T>::copy tree(){
                                             template <typename T>
    Tree<T> new_tree;
                                            Node<T>* Tree<T>::merge(Node<T> *n,
    Node<T>* n = root;
                                             Tree<T> &new_tree){
    copy_node(n,new_tree);
                                                 if (n != nullptr)
    return new_tree;
                                                     new tree.add node(n->data);
}
                                                     merge(n->left, new_tree);
                                                     merge(n->right, new_tree);
template <typename T>
Tree<T> Tree<T>::copy_tree(Node<T>
                                                 return new_tree.root;
*node_to_copy){
                                             }
    Tree<T> new_tree;
    copy_node(node_to_copy, new_tree);
                                             template <typename T>
    return new_tree;
                                             Tree<T> Tree<T>::merge(Tree<T>
                                            &sec_tree){
    Tree<T> new_tree;
}
template <typename T>
                                                 new_tree = copy_tree(root);
Node<T>* Tree<T>::get_root(){
                                                 Node<T> *n = sec_tree.root;
                                                 merge(n, new_tree);
    return root;
                                                 return new_tree;
                                             }
template <typename T>
void Tree<T>::remove_subtree(Node<T>
                                             template <typename T>
                                             void Tree<T>::map(Node<T> *n, map_func
*n){
    if (n != nullptr)
                                             mf){
                                                 if (n != nullptr)
        remove_subtree(n->left);
                                                 {
                                                     n->data = mf(n->data);
        remove_subtree(n->right);
        remove_node(n);
                                                     map(n->left, mf);
    }
                                                     map(n->right, mf);
}
                                                 }
                                             }
                                             template <typename T>
template <typename T>
                                             void Tree<T>::map(map_func mf){
void Tree<T>::remove_subtree(const T
                                                 map(root, mf);
val){
    Node<T>* n;
    n = find_node(val);
    remove_subtree(n);
                                             template <typename T>
}
                                             void Tree<T>::print(Node<T>* n, int
                                             level){
template <typename T>
Node<T>* Tree<T>::where(where_func wf,
                                                 if (n != nullptr){
                                                     print(n->right, level + 1);
for (int i = 0; i < level; i+</pre>
Node<T>* n, Tree<T> &new_tree) {
    if (n != nullptr)
                                             +) cout<<"
    {
                                                     cout<<n->data<<endl;
        if (wf(n->data))
                                                     print(n->left, level + 1);
                                                 }
             new_tree.add_node(n-
                                             }
>data);
        where(wf,n->left, new_tree);
                                             template <typename T>
        where(wf,n->right, new_tree);
                                             void Tree<T>::print(){
                                                 if (root == nullptr) cout<<"\nTree</pre>
                                             is empty.\n\n";
    return new_tree.root;
}
                                                 else{
                                                     print(root, get_depth());
                                                     cout<<endl;
template <typename T>
Tree<T> Tree<T>::where(where_func wf){
                                                 }
                                             }
    Tree<T> new_tree;
    Node<T>* n = root;
    where(wf, n, new_tree);
                                             template <typename T>
                                             int Tree<T>::get_depth(){
    return new_tree;
                                                 int length = 0;
}
                                                 return get_depth(root,length);
```

```
}
                                               void clear_node(Node<T>*);
                                               string post_order_to_str(Node<T>*
template <typename T>
                                           x);
int Tree<T>::get depth(Node<T> *n, int
                                                          //обход левый-правый-
                                           корень, начиная с х
depth){
                                               string post_order_to_str();
    if (n == nullptr)
                                           //обход левый-правый-корень, начиная с
        return depth;
    return max(get_depth(n->left,
                                           корня
                                               strina
depth + 1), get_depth(n->right, depth
                                           reverse_post_order_to_str(Node<T>* x);
+ 1));
                                           //обход правый-левый-корень, начиная с
tree.hpp
                                           reverse post order to str();
                                           //обход правый-левый-корень, начиная с
#ifndef three hpp
                                           корня
#define three_hpp
                                               Node<T>* find_node(T);
                                                                                //
#include <stdio.h>
                                           находит узел по значению
#include <iostream>
                                               bool is_in_tree(T);
                                                                                 //
#include <string>
                                           проверка элемента на вхождение
#include <sstream>
                                               Node<T>* find_min_node(Node<T>*);
                                               Node<T>* find_max_node(Node<T>*);
using namespace std;
                                               Node<T>* copy_node(Node<T>*,
template <typename T>
                                           Tree<T>&);
                                                          //копирует узел
struct Node{
                                               Tree<T> copy_tree();
    T data;
                                           //копирует дерево
    Node<T>* left;
                                               Tree<T> copy_tree(Node<T>*);
    Node<T>* right;
                                               Node<T>* get_root();
    Node<T>* parent;
                                               void remove_subtree(T
    Node();
                                           val);
                                                        //удаляет поддерево по
    Node(const T);
                                           значению
};
                                               typedef bool (*where func)(T);
template <typename T>
                                               Node<T>* where(where func,
class Tree{
                                           Node<T>*, Tree<T>&); //вспомогательная
private:
                                           функция для where
    Node<T>* root;
                                               Tree<T> where(where func wf);
    void remove_subtree(Node<T> *);
                                               Tree<T> merge(Tree<T> &sec_tree);
//вспомогательная функция для удаления
                                           //прибавляет второе дерево к первому
поддерева
                                           (слияние)
    Node<T>* merge(Node<T>*, Tree<T>
&);
       //вспомогательная функция для
                                               typedef T (*map_func)(T);
merge
                                               void map(Node<T> *, map_func);
    void print(Node<T>*, int);
                                           //вспомогательная функция для тар
//вспомогательная функция для print
                                               void map(map_func mf);
    int get_depth(Node<T>*, int);
//вспомогательная функция для
                                               void print();
get_depth
                                               int get_depth(); //находит глубину
                                           дерева
public:
    Tree();
      Tree(Node<T> &rt);
                                           #include "tree.cpp"
      Tree(Tree<T> &tr);
    ~Tree();
                                           #endif /* three_hpp */
    void add node(T);
                                 //
                                           UI.cpp
добаление элемента в дерево
    void remove_node(T);
                                 //
                                           #include "UI.hpp"
удаление элемента из дерево по
значению
                                           int cin_int(){
    void remove_node(Node<T>*); //
                                               int N;
удаление элемента из дерево по узлу
```

```
while (true) {
                                                  int type = 0;
        cout<<flush;//очистка потока
                                                  int menu item = 0;
        if ((cin >> N).good()) return
                                                  while (menu_item != 12){
N;
                                                       cout<<"\nPress\n";</pre>
        if (cin.fail()) {
                                                       cout<<" 1. to add item in
             cin.clear();//очищает флаг
                                              tree\n";
ошибки, но часть введенной строки при
этом остается
                                                       cout<<" 2.
                                                                   to remove item\n";
                                                      cout<<" 3.
                                                                   to print tree\n";
             cout << "Invalid input.</pre>
                                                       cout<<" 4.
\n":
                                                                   use map function
                                              (square)\n";
                                                       cout<<" 5.
                                                                   use where function
cin.ignore(numeric limits<streamsize>:
                                              (remove all non-positive)\n";
                                                       cout<<" 6.
                                                                   merge\n";
:max(), '\n');//пропустить все символы
                                                       cout<<" 7.
в потоке до \n, после этого можно
                                                                    remove subtree\n";
                                                       cout<<" 8.
                                                                   search for a
начинать считывать новое значение
                                              value\n";
                                                       cout<<" 9.
}
                                                                   post-order
                                              traversal (LEFT-RIGHT-ROOT) \n";
                                              cout<<" 10. reverse post-order
traversal (RIGHT-LEFT-R00T) \n";</pre>
double cin_double(){
    double N;
                                                      cout<<" 11. to run tests\n";
cout<<" 12. to exit\n";</pre>
    while (true) {
        cout<<flush;//очистка потока
        if ((cin >> N).good()) return
                                                      menu_item = cin_int();
N;
        if (cin.fail()) {
                                                       switch (menu_item) {
             cin.clear();//очищает флаг
                                                           case 1:
ошибки, но часть введенной строки при
этом остается
                                                               cout<<endl<<"Select
                                              type of tree:\n";
             cout << "Invalid input.</pre>
                                                               cout<<" 1. Real\n";
\n";
                                                               cout<<" 2. Complex\n";
        }
                                                               type = cin_int();
cin.ignore(numeric_limits<streamsize>:
                                                               while (type != DOUBLE
:max(), '\n');//пропустить все символы
в потоке до \n, после этого можно
                                              && type != COMPLEX){
начинать считывать новое значение
                                              cout<<endl<<"Incorrect type."<<endl;</pre>
}
                                              cout<<endl<<"Select type of tree:\n";</pre>
                                                                    cout<<" 1.
bool db_check_values(double db){
                                              Real\n";
    if (db > 0) return true;
                                                                    cout<<" 2.
    else return false;
                                              Complex\n";
}
                                                                    type = cin_int();
bool com_check_values(Complex db){
    if (\overline{db} > 0) return true;
                                                               if (type == DOUBLE) {
    else return false;
                                                                    cout<<"\nEnter
}
                                              value:\n";
double db_square(double db){
                                              dbTree.add_node(cin_double());
    return db*db;
                                                               else {
                                                                    double re = 0;
Complex com_square(Complex com){
                                                                    double im = 0;
cout<<"\nEnter</pre>
    return com*com;
                                              real part of value:\n";
                                                                    re = cin_double();
void run_UI(){
                                                                    cout<<"\nEnter
    Tree<double>::map_func db_mf =
                                              imaginary part of value:\n";
db_square;
                                                                    im = cin_double();
    Tree<Complex>::map_func com_mf =
com_square;
                                              comTree.add node(Complex(re,im));
    Tree<double> dbTree;
    Tree<Complex> comTree;
                                                           }
```

```
break;
                                                          case 5:
             case 2:
                                                              if (type == DOUBLE) {
                 if (type == DOUBLE) {
                                             Tree<double>::where func wf =
                     cout<<"\nEnter
                                             db_check_values;
value you want to remove:\n";
                                                                  dbTree.where(wf);
                     double val =
cin double();
                                                              else if (type ==
                                             COMPLEX) {
dbTree.remove_node(val);
                                             Tree<Complex>::where func wf =
                 else if (type ==
                                             com_check_values;
COMPLEX) {
                                                                  comTree.where(wf);
                     cout<<"\nEnter
value you want to remove:\n";
                                                              else cout<<"\nTree
                                             doesn't exist\n";
                     cout<<"\n Enter
real part:\n";
                     double re =
                                                              break;
cin_double();
                     cout<<"\n Enter
                                                          case 6:
imaginary part:\n";
                     double im =
                                                              if (type == 0)
cin double();
                                             cout<<"\nTree doesn't exist\n";</pre>
                                                              else{
comTree.remove_node(Complex(re,im));
                                                                  int menu_item = 0;
                                                                  Tree<double>
                 else cout<<"\nTree
                                             sec_dbTree;
doesn't exist\n"
                                                                  Tree<Complex>
                                             sec_comTree;
                                                                  cout<<"\nEnter
                 break;
                                             second three:\n";
             case 3:
                                                                  cout<<"Do you want
                                             to enter new item?\n";
                 if (type == DOUBLE) {
                                                                  while (menu_item !
                     cout<<"\nYour
                                             = 2) {
tree:\n";
                                                                       cout<<"
                     dbTree.print();
                                             1.Yes\n";
                                                                       cout<<"
                 else if (type ==
                                             2.No\n";
COMPLEX) {
                                                                       menu_item =
                     cout<<"\nYour
                                             cin_int();
tree:\n";
                                                                       while
                     comTree.print();
                                             (menu_item != 1 && menu_item != 2) {
                 }
                 else cout<<"\nTree
                                             cout<<"Incorrect input.\n";</pre>
doesn't exist\n"
                                                                           menu_item
                                             = cin_int();
                 break;
                                                                       if (menu_item
                                             == 1) {
             case 4:
                                                                           if (type
                 if (type == DOUBLE) {
                                             == DOUBLE) {
                     dbTree.map(db_mf);
                                             cout<<"\nEnter value:\n";</pre>
                 else if (type ==
COMPLEX) {
                                             sec_dbTree.add_node(cin_double());
                                                                           }
                                                                           else{
comTree.map(com_mf);
                                                                               double
                                             re = 0;
                 else cout<<"\nTree
                                                                               double
doesn't exist\n
                                             im = 0;
                                             cout<<"\nEnter real part of value:\n";</pre>
                 break;
```

```
re =
                                                                   cout<<"\nEnter
cin double();
                                             value you want to check:\n";
                                                                   double re = 0;
                                                                   double im = 0;
cout<<"\nEnter</pre>
cout<<"\nEnter imaginary part of
value:\n";
                                             real part of value:\n";
                                  im =
cin_double();
                                                                   re = cin_double();
                                                                   cout<<"\nEnter
                                             imaginary part of value:\n";
sec_comTree.add_node(Complex(re,im));
                                                                   im = cin_double();
                                                                   if
                     }
                                             (comTree.is_in_tree(Complex(re,im)))
                                             cout<<"\nValue enters the tree\n";
                     if (type ==
                                                                   else
DOUBLE) dbTree =
                                             cout<<"\nValue doesn't enter the
                                             tree\n";
dbTree.merge(sec_dbTree);
                     else comTree =
                                                               else cout<<"\nTree</pre>
comTree.merge(sec_comTree);
                                             doesn't exist\n";
             }
                                                          ł
                                                               break:
                 break;
             case 7:
                                                          case 9:
                 if (type == DOUBLE) {
                                                               if (type == DOUBLE) {
                     cout<<"\nEnter
value:\n";
                                             cout<<dbTree.post_order_to_str();</pre>
dbTree.remove_subtree(cin_double());
                                                               else if (type ==
                                             COMPLEX) {
                 else if (type ==
COMPLEX) {
                                             cout<<comTree.post_order_to_str();</pre>
                     cout<<"\nEnter
value:\n";
                                                               else cout<<"\nTree
                     double re = 0;
                                             doesn't exist\n";
                     double im = 0;
                     cout<<"\nEnter
                                                               break:
real part of value:\n";
                     re = cin double();
                                                          case 10:
                     cout<<"\nEnter
imaginary part of value:\n";
                                                               if (type == DOUBLE) {
                     im = cin_double();
                                             cout<<dbTree.reverse_post_order_to_str</pre>
comTree.remove_subtree(Complex(re,im))
                                             ();
                                                               else if (type ==
                 else cout<<"\nTree
                                             COMPLEX) {
doesn't exist\n";
                                             cout<<comTree.reverse_post_order_to_st</pre>
                 break;
                                             r();
             case 8:
                                                               else cout<<"\nTree
                                             doesn't exist\n";
                 if (type == DOUBLE) {
                     cout<<"\nEnter
                                                               break;
value you want to check:\n";
                     if
                                                          case 11:
(dbTree.is_in_tree(cin_double()))
cout<<"\nValue enters the tree\n";
                                                               run_all_tests();
                     else
cout<<"\nValue doesn't enter the
                                                               break;
tree\n";
                                                          case 12:
                 else if (type ==
                                                               break;
COMPLEX) {
                                                          default:
```

```
cout<<"Invalid
                                                     run complex unit 2();
command\n\n";
                                                     run complex unit 3();
                  break:
         }
     }
                                                void run_tree_tests(){
                                                     run_tree_unit_1();
run_tree_unit_2();
run_tree_unit_3();
run_tree_unit_4();
}
UI.hpp
                                                     run_tree_unit_5();
#ifndef UI hpp
#define UI_hpp
                                                Tests/tests.hpp
#define DOUBLE 1
#define COMPLEX 2
                                                #ifndef tests hpp
                                                #define tests_hpp
#include <stdio.h>
#include <iostream>
#include \lostream>
#include "tree.hpp"
#include "complex.hpp"
#include "Tests/tests.hpp"
                                                #include <stdio.h>
                                                #include <iostream>
#include "Complex Tests/
                                                complex_unit_1.hpp"
                                                #include "Complex Tests/
void run_UI();
                                                complex_unit_2.hpp"
                                                #include "Complex Tests/
int cin_int();
                                                complex_unit_3.hpp"
double cin_double();
                                                #include "Tree Tests/tree_unit_1.hpp"
                                                #include "Tree Tests/tree_unit_2.hpp"
bool check_values(double);
                                                #include "Tree Tests/tree_unit_3.hpp"
bool check_values(Complex);
                                                #include "Tree Tests/tree_unit_4.hpp"
double square(double);
                                                #include "Tree Tests/tree_unit_5.hpp"
Complex square(Complex);
#endif /* UI_hpp */
                                                using namespace std;
Tests/tests.cpp
                                                void run_all_tests();
                                                void run_complex_tests();
#include "tests.hpp"
                                                void run_tree_tests();
                                                void test_prepare(int *test_id, bool
using namespace std;
                                                *test_ok);
                                                void test_result(bool test_ok);
void test_prepare(int *test_id, bool
*test ok){
                                                #endif /* tests hpp */
     (*test ok) = false;
     (*test id)++;
                                                Tests/Tree Tests/tree_unit_1.cpp
     cout<<"[TESTING] Test
                                                #include "tree unit 1.hpp"
#"<<*test_id<<": ";
                                                using namespace std;
void test_result(bool test_ok){
                                                /*
     if (test_ok == true){
         cout<<"0K.\n";
                                                 UNIT 1: Тестирование создания дерева
     } else {
                                                и добавления элементов
         cout<<"ERROR.\n";</pre>
                                                 Список тестов:
                                                 Тест 1: Добавление одного элемента
Тест 2: Добавление пяти элементов
}
                                                 Тест 3: Создание пустого дерева
void run_all_tests(){
                                                 */
     run_complex_tests();
     run_tree_tests();
                                                void run_tree_unit_1(){
}
                                                     cout<<"[TREE CLASS TESTING] UNIT 1
                                                :\n\n";
                                                     int test_id = 0;
void run_complex_tests(){
                                                     bool test_ok = true;
     run_complex_unit_1();
```

```
//TECT 1
                                                     test_prepare(&test_id,
    if (test ok == true){
                                             &test ok);
        test prepare(&test id,
                                                     Tree<double> tr:
&test_ok);
        Tree<double> tr;
                                                     if (tr.get root() == NULL)
        tr.add_node(5);
                                             test_ok = true;
                                                     else test_ok = false;
        if (tr.get_root()->data == 5
&& tr.get root() \rightarrow \overline{l}eft == nullptr &&
                                                     test_result(test_ok);
                                                 }
tr.get_root()->left == nullptr)
test_ok = true;
                                                 if (test_id == 3 && test_ok ==
        else test_ok = false;
                                             true) {
                                                     cout<<"\n[TESTING] Unit 1</pre>
        test_result(test_ok);
                                             testing SUCCESSEDED.\n\n\n";
    }
                                                 } else {
                                                     cout<<"\n[TESTING] Unit 1</pre>
    //TECT 2
                                             testing FAILED on TEST #"<<test_id<<".
    if (test ok == true){
                                             n\n';
        test_prepare(&test_id,
                                                 }
&test_ok);
                                             }
        Tree<double> tr;
        tr.add_node(5);
        tr.add_node(2);
                                             Tests/Tree Tests/tree_unit_1.hpp
        tr.add_node(3);
        tr.add_node(-5);
                                             #ifndef tree_unit_1_hpp
        tr.add_node(1);
                                             #define tree_unit_1_hpp
        Node<double>* current_node =
                                             #include <stdio.h>
tr.get root();
                                             #include "tree.hpp"
if (current_node->data == 5 &&
current_node->right == nullptr){
                                             #include "tests.hpp"
            current_node =
                                             using namespace std;
current_node->left;
            if (current_node->data ==
                                             void run_tree_unit_1(void);
2){
                 current node =
                                             #endif /* tree unit 1 hpp */
current_node->right;
                 if (current node->data
                                             Tests/Tree Tests/tree unit 2.cpp
== 3 && current node->left == nullptr
&& current_node->right == nullptr){
                                             #include "tree unit 2.hpp"
                     current node =
current_node->parent->left;
                     if (current_node-
                                             using namespace std;
>data == -5 && current_node->left ==
nullptr){
                                             /*
                         current_node =
                                              UNIT 2: Тестирование удаления числа
current_node->right;
                                             из дерева
                         if
(current_node->data == 1 &&
                                              Список тестов:
current_node->left == nullptr &&
                                              Тест 1: Удаление корня из дерева
current_node->right == nullptr)
                                              Тест 2: Удаление элемента без
test_ok = true;
                                             потомков
                                              Тест 3: Удаление элемента с одним
                 }
                                             потомком
            }
                                             Тест 4: Попытка удаления
                                             несуществующего элемента
        else test_ok = false;
                                              */
        test_result(test_ok);
    }
                                             void run_tree_unit_2(){
                                                 cout<<"[TREE CLASS TESTING] UNIT 2
    //TECT 3
                                             :\n\n";
    if (test_ok == true){
                                                 int test_id = 0;
                                                 bool test_ok = true;
```

```
current node =
                                            current node->left;
    //TECT 1
    if (test ok == true){
                                                         if (current node->data ==
        test_prepare(&test_id,
                                            2){
&test ok);
                                                             current node =
                                            current node->right;
        Tree<double> tr;
                                                             if (current_node->data
        tr.add_node(5);
                                            == 3 && current_node->left == nullptr
        tr.add_node(2);
                                            && current_node->right == nullptr){
        tr.add_node(3);
                                                                 current_node =
        tr.add_node(-5);
                                            current_node->parent->left;
        tr.add node(1);
                                                                 if (current node-
        tr.add_node(6);
                                            >data == -5 && current node->left ==
                                            nullptr && current_node->right ==
                                            nullptr) test_ok = true;
        tr.remove_node(5);
                                                             }
        Node<double>* current_node =
tr.get_root();
                                                    else test_ok = false;
        if (current_node->data == 3){
            current node =
                                                    test_result(test_ok);
current_node->right;
                                                }
            if (current_node->data ==
6 && current node->left == nullptr &&
                                                //TECT 3
current_node->right == nullptr){
                                                if (test_ok == true){
                 current_node =
                                                     test_prepare(&test_id,
current_node->parent->left;
                 if (current_node->data
                                            &test ok);
== 2 && current_node->right ==
                                                    Tree<double> tr;
nullptr){
                                                    tr.add_node(5);
                     current_node =
                                                    tr.add_node(2);
current_node->left;
                                                    tr.add_node(3);
                     if (current_node-
                                                    tr.add_node(-5);
>data == -5 && current_node->left ==
                                                    tr.add_node(1);
nullptr){
                         current_node =
                                                    tr.remove node(-5);
current_node->right;
                                                    Node<double>* current_node =
                                            tr.get_root();
(current_node->data == 1 &&
                                                     if (current node->data == 5 &&
current_node->left == nullptr &&
                                            current_node->right == nullptr){
current_node->right == nullptr)
                                                         current_node =
test_ok = true;
                                            current node->left;
                     }
                                                         if (current_node->data ==
                 }
                                            2){
            }
                                                             current_node =
        }
                                            current_node->right;
                                                             if (current_node->data
        test_result(test_ok);
                                            == 3 && current_node->left == nullptr
    }
                                            && current_node->right == nullptr){
                                                                 current_node =
    //TECT 2
                                            current_node->parent->left;
    if (test_ok == true){
                                                                 if (current node-
        test_prepare(&test_id,
                                            >data == 1 && current_node->left ==
nullptr && current_node->right ==
&test_ok);
                                            nullptr) test_ok = true;
        Tree<double> tr;
        tr.add_node(5);
        tr.add_node(2);
        tr.add_node(3);
                                                    else test_ok = false;
        tr.add_node(-5);
        tr.add_node(1);
                                                    test_result(test_ok);
                                                }
        tr.remove_node(1);
        Node<double>* current_node =
                                                //TECT 4
tr.get_root();
                                                if (test_ok == true){
        if (current_node->data == 5 &&
current_node->right == nullptr){
```

```
test_prepare(&test_id,
&test ok);
                                             Tests/Tree Tests/tree_unit_3.cpp
        Tree<double> tr;
        tr.add node(5);
                                             #include "tree_unit_3.hpp"
        tr.add_node(2);
        tr.add_node(3);
                                             using namespace std;
        tr.add_node(1);
        tr.remove_node(10);
                                              UNIT 3: Тестирование вывода дерева в
                                             строку в соответствии с обходом
        Node<double>* current node =
tr.get root();
                                              Список тестов:
        if (current_node->data == 5 &&
                                              Тест 1: Вывод по обходу ЛПК
current node->right == nullptr){
                                              Тест 2: Вывод по обходу ПЛК
             current_node =
                                              */
current_node->left;
             if (current node->data ==
                                             void run_tree_unit_3(){
2){
                                                  cout<<"[TREE CLASS TESTING] UNIT 3
                 current_node =
                                             :\n\n";
current_node->right;
                                                  int test_id = 0;
                 if (current_node->data
                                                  bool test_ok = true;
== 3 && current_node->left == nullptr
&& current_node->right == nullptr){
                                                  //TECT 1
                     current_node =
                                                  if (test_ok == true){
current_node->parent->left;
                                                      test_prepare(&test_id,
                     if (current_node-
                                             &test_ok);
>data == 1 && current_node->left ==
nullptr && current_node->right ==
                                                      Tree<double> tr;
nullptr) test_ok = true;
                                                      tr.add_node(5);
tr.add_node(2);
                                                      tr.add_node(3);
        }
                                                      tr.add_node(-5);
        else test_ok = false;
                                                      tr.add_node(1);
                                                      tr.add_node(6);
        test_result(test_ok);
    }
                                                      string s =
                                             tr.post_order_to_str();
     if (s == "[[[][1]-5][3]2]
    if (test_id == 4 && test_ok ==
true) {
                                             [6]5") test_ok = true;
        cout<<"\n[TESTING] Unit 2
testing SUCCESSEDED.\n\n\n";
                                                      test_result(test_ok);
    } else {
                                                 }
        cout<<"\n[TESTING] Unit 2
testing FAILED on TEST #"<<test_id<<".
                                                  //TECT 2
\n\n\n";
                                                  if (test_ok == true){
    }
                                                      test_prepare(&test_id,
}
                                             &test_ok);
                                                      Tree<double> tr;
                                                      tr.add_node(5);
tr.add_node(2);
tr.add_node(3);
Tests/Tree Tests/tree_unit_2.hpp
                                                      tr.add_node(-5);
#ifndef tree_unit_2_hpp
                                                      tr.add_node(1);
#define tree_unit_2_hpp
                                                      tr.add_node(6);
#include <stdio.h>
#include "tree.hpp"
                                                      string s =
#include "tests.hpp"
                                             tr.reverse_post_order_to_str();
                                                      if (s == "[6][[3][[1]]
                                             []-5]2]5") test_ok = true;
using namespace std;
                                                      test_result(test_ok);
void run tree unit 2(void);
                                                 }
#endif /* tree_unit_2_hpp */
```

```
if (test id == 2 && test ok ==
                                                     Node<double>* current node =
true) {
        cout<<"\n[TESTING] Unit 3</pre>
                                            tr.get root();
testing SUCCESSEDED.\n\n\n";
                                                     if (current_node->data == 5){
                                                         current node =
    } else {
        cout<<"\n[TESTING] Unit 3
                                            current_node->right;
testing FAILED on TEST #"<<test_id<<".
                                                         if (current_node->data ==
n\n';
                                            6 && current_node->left == nullptr &&
    }
                                            current_node->right == nullptr){
}
                                                              current_node =
                                            current_node->parent->left;
                                                              if (current node->data
                                            == 2 && current_node->left == nullptr)
Tests/Tree Tests/tree_unit_3.hpp
                                            {
                                                                  current node =
#ifndef tree_unit_3_hpp
                                            current_node->right;
#define tree_unit_3_hpp
                                                                  if (current_node-
                                            >data == 3 && current_node->left ==
#include <stdio.h>
#include "tree.hpp"
                                            nullptr && current_node->right ==
                                            nullptr) test_ok = true;
#include "tests.hpp"
                                                         }
using namespace std;
                                                     }
void run_tree_unit_3(void);
                                                     test_result(test_ok);
                                                 }
#endif /* tree_unit_3_hpp */
                                                 //TECT 2
                                                 if (test_ok == true){
Tests/Tree Tests/tree unit 4.cpp
                                                     test_prepare(&test_id,
                                            &test_ok);
#include "tree unit 4.hpp"
                                                     Tree<double> tr;
using namespace std;
                                                     tr.add_node(5);
                                                     tr.add_node(2);
                                                     tr.add node(3);
                                                     tr.add node(-5);
UNIT 4: Тестирование удаления
                                                     tr.add node(1);
поддерева и поиска на вхождение
                                                     tr.add_node(6);
элемента
                                                     if (tr.is_in_tree(3) && !
 Список тестов:
 Тест 1: Извлечение поддерева из
                                            tr.is_in_tree(10)) test_ok = true;
 Тест 2: Поиск на вхождение элемента
                                                     test_result(test_ok);
                                                 }
                                                 if (test_id == 2 && test_ok ==
void run tree unit 4(){
                                            true) {
    cout<<"[TREE CLASS TESTING] UNIT 4
                                                     cout<<"\n[TESTING] Unit 4
:\n\n";
                                            testing SUCCESSEDED.\n\n\n";
    int test_id = 0;
                                                 } else {
    bool test_ok = true;
                                                     cout<<"\n[TESTING] Unit 4
                                            testing FAILED on TEST #"<<test_id<<".
    //TECT 1
                                            n\n';
    if (test_ok == true){
                                                 }
        test_prepare(&test_id,
                                            }
&test_ok);
        Tree<double> tr;
        tr.add_node(5);
                                            Tests/Tree Tests/tree_unit_4.hpp
        tr.add_node(2);
tr.add_node(3);
                                            #ifndef tree_unit_4_hpp
        tr.add_node(-5);
                                            #define tree_unit_4_hpp
        tr.add_node(1);
        tr.add_node(6);
                                            #include <stdio.h>
                                            #include "tree.hpp"
        tr.remove_subtree(-5);
```

```
if (current node->data ==
#include "tests.hpp"
                                            36 && current_node->left == nullptr &&
                                            current node->right == nullptr){
using namespace std;
                                                              current_node =
                                            current_node->parent->left;
void run_tree_unit_4(void);
                                                              if (current node->data
                                            == 4){
#endif /* tree_unit_4_hpp */
                                                                  current_node =
                                            current_node->right;
Tests/Tree Tests/tree_unit_5.cpp
                                                                  if (current_node-
                                            >data == 9 && current_node->left ==
#include "tree_unit_5.hpp"
                                            nullptr && current node->right ==
                                            nullptr) {
using namespace std;
                                                                      current_node =
                                            current node->parent->left;
                                                                      if
UNIT 5: Тестирование map, where и
                                             (current_node->data == 25 &&
merge
                                            current node->left == nullptr){
 Список тестов:
                                            current_node = current_node->right;
 Tест 1: map
 Тест 2: where (результат функции
                                             (current_node->data == 1 &&
равен истине только когда число
                                            current_node->left == nullptr &&
положительно)
                                            current_node->right == nullptr)
 Tест 3: merge
                                            test_ok = true;
                                                                      }
                                                                  }
bool check_values(double db){
                                                             }
    if (db > 0) return true;
                                                         }
    else return false;
                                                     }
}
                                                     test_result(test_ok);
double square(double db){
    return db*db;
                                                 //TECT 2
                                                 if (test ok == true){
void run_tree_unit_5(){
                                                     test_prepare(&test_id,
    cout<<"[TREE CLASS TESTING] UNIT 5
                                            &test_ok);
:\n\n";
    int test id = 0;
                                                     Tree<double>::where func wf =
                                            check_values;
    bool test_ok = true;
                                                     Tree<double> tr;
                                                     tr.add_node(5);
    //TECT 1
    if (test_ok == true){
                                                     tr.add_node(2);
                                                     tr.add_node(3);
        test_prepare(&test_id,
                                                     tr.add_node(-5);
&test_ok);
                                                     tr.add_node(-1);
        Tree<double>::map func mf =
                                                     tr.add_node(6);
square;
        Tree<double> tr;
                                                     Tree<double> new_tree =
        tr.add_node(5);
                                            tr.where(wf);
        tr.add_node(2);
tr.add_node(3);
tr.add_node(-5);
                                                     Node<double>* current_node =
                                            new_tree.get_root();
                                                     if (current_node->data == 5){
    current_node =
        tr.add_node(1);
        tr.add_node(6);
                                            current_node->right;
                                                         if (current_node->data ==
        tr.map(mf);
                                            6 && current_node->left == nullptr &&
                                            current_node->right == nullptr){
        Node<double>* current_node =
                                                             current_node =
tr.get_root();
        if (current_node->data == 25){
                                            current_node->parent->left;
            current_node =
                                                              if (current_node->data
current_node->right;
                                            == 2 && current_node->left == nullptr)
```

```
current node =
                                                                               if
                                             (current node->data == 0 &&
current node->right;
                     if (current_node-
                                             current node->right == nullptr){
>data == 3 && current_node->left ==
nullptr && current_node->right ==
                                             current node = current node->left;
nullptr) test_ok = true;
                                                                                   if
                                             (current_node->data == -2 &&
            }
                                             current_node->left == nullptr &&
                                             current_node->right == nullptr)
                                             test_ok = true;
        test_result(test_ok);
                                                                               }
    }
                                                                           }
                                                                      }
                                                                  }
    //TECT 3
                                                              }
    if (test ok == true){
                                                          }
        test_prepare(&test_id,
                                                     }
&test_ok);
        Tree<double> first_tr;
                                                     test_result(test_ok);
        first_tr.add_node(\overline{5});
                                                 }
        first_tr.add_node(2);
        first_tr.add_node(3);
                                                 if (test_id == 2 && test_ok ==
        first_tr.add_node(1);
                                             true) {
                                                     cout<<"\n[TESTING] Unit 5
        Tree<double> second_tr;
                                             testing SUCCESSEDED.\n\n\n";
        second_tr.add_node(7);
second_tr.add_node(0);
second_tr.add_node(-2);
                                             cout<<"---
                                                ----"<<"\n\n\n";
        second_tr.add_node(4);
                                                 } else {
                                                     cout<<"\n[TESTING] Unit 5
                                             testing FAILED on TEST #"<<test_id<<".
        Tree<double> tr =
first_tr.merge(second_tr);
                                             n\n';
                                             cout<<"--
        Node<double>* current_node =
                                                      -"<<"\n\n\n";
tr.get_root();
                                                 }
        if (current_node->data == 5){
            current node =
                                             }
current_node->right;
            if (current node->data ==
7 && current node->left == nullptr &&
                                             Tests/Tree Tests/tree_unit_5.hpp
current_node->right == nullptr){
                 current_node =
                                             #ifndef tree_unit_5_hpp
current_node->parent->left;
                                             #define tree_unit_5_hpp
                 if (current_node->data
== 2){
                                             #include <stdio.h>
                     current_node =
                                             #include "tree.hpp"
current_node->right;
                                             #include "tests.hpp"
                     if (current_node-
>data == 3 && current_node->left ==
                                             using namespace std;
nullptr){
                         current node =
                                             void run_tree_unit_5(void);
current node->right;
                                             bool check_values(double);
(current_node->data == 4 &&
                                             double square(double);
current_node->left == nullptr &&
current_node->right == nullptr){
                                             #endif /* tree unit 5 hpp */
current_node = current_node->parent-
                                             Tests/Complex Tests/complex unit 1.cpp
>parent->left;
                                             #include "complex_unit_1.hpp"
(current node->data == 1 &&
current_node->right == nullptr){
                                             using namespace std;
current node = current node->left;
                                             /*
```

```
UNIT 1: Тестирование создания
                                                     test_result(test_ok);
                                                 }
комплексного числа
 Список тестов:
                                                 if (test_id == 3 && test_ok ==
 Тест 1: Создание комплексного числа с
                                            true) {
нулевой действительной частью и
                                                     cout<<"\n[TESTING] Unit 1</pre>
нулевой мнимой частью
                                            testing SUCCESSEDED.\n\n\n";
Тест 2: Создание комплексного числа с
                                                 } else {
ненулевой действительной частью и
                                                     cout<<"\n[TESTING] Unit 1</pre>
нулевой мнимой частью
                                            testing FAILED on TEST #"<<test_id<<".
Тест 3: Создание комплексного числа с
                                             \n\n\n";
ненулевой действительной частью и
                                                 }
                                            }
ненулевой мнимой частью
*/
                                            Tests/Complex Tests/complex_unit_1.hpp
void run_complex_unit_1(){
                                            #ifndef complex_unit_1_hpp
    cout<<"[COMPLEX CLASS TESTING]</pre>
                                            #define complex_unit_1_hpp
UNIT 1 :\n\n";
    int test id = 0;
                                            #include <stdio.h>
                                            #include "complex.hpp"
#include "tests.hpp"
    bool test_ok = true;
    //TECT 1
    if (test_ok == true){
                                            using namespace std;
        test_prepare(&test_id,
&test ok);
                                            void run_complex_unit_1(void);
        Complex com = Complex();
                                            #endif /* complex_unit_1_hpp */
                                            Tests/Complex Tests/complex_unit_2.cpp
        if (com.return_re() == 0 &&
com.return_im() == 0){
                                            #include "complex unit 2.hpp"
            test_ok = true;
                                            using namespace std;
        test_result(test_ok);
    }
                                             UNIT 2: Тестирование математических
                                            операций с классом Complex
    //TECT 2
    if (test_ok == true){
        test_prepare(&test_id,
                                             Список тестов:
                                             Тест 1: Сложение комплексного и
&test_ok);
                                            комплексного (операции + и +=)
                                             Тест 2: Сложение комплексного и
        Complex com = Complex(5);
                                            вещественного (операции + и +=)
                                             Тест 3: Произведение комплексного и
                                            вещественного (операции * и *=)
        if (com.return_re() == 5 &&
                                             Тест 4: Произведение комплексного и
com.return_im() == 0){
                                            комплексного
            test_ok = true;
                                             */
        test_result(test_ok);
                                            void run_complex_unit_2(){
    }
                                                 cout<<"[COMPLEX CLASS TESTING]</pre>
                                            UNIT 2 :\n\n";
    int test_id = 0;
    //TECT 3
    if (test ok == true){
                                                 bool test_ok = true;
        test_prepare(&test_id,
&test_ok);
                                                 //TECT 1
                                                 if (test_ok == true){
        Complex com = Complex(5, -2);
                                                     test_prepare(&test_id,
                                            &test_ok);
                                                     Complex com1 = Complex(2, 5);
        if (com.return_re() == 5 &&
com_return_im() == -2){
                                                     Complex com2 = Complex(3, 4);
            test_ok = true;
                                                     Complex test_com_1 = com1 +
                                            com2;
```

```
Complex test_com_2 = com1;
        test com 2 += com2;
                                                     Complex test com = com1 *
                                            com2;
        if (test_com_1.return_re() ==
5 && test_com_1.return_im() == 9 &&
                                                     if (test com == Complex(-14,
test_com_2.return_re() == 5 &&
                                            23)){
test_com_2.return_im() == 9){
                                                         test_ok = true;
            test_ok = true;
                                                     }
                                                     test_result(test_ok);
                                                }
        test_result(test_ok);
    }
                                                if (test id == 4 && test ok ==
    //TECT 2
                                            true) {
    if (test ok == true){
                                                     cout<<"\n[TESTING] Unit 2
                                            testing SUCCESSEDED.\n\n\n";
        test_prepare(&test_id,
&test_ok);
                                                } else {
                                                     cout<<"\n[TESTING] Unit 2
        Complex com1 = Complex(2, 5);
                                            testing FAILED on TEST #"<<test id<<".
        double db = 5;
                                            n\n';
                                                }
        Complex test_com_1 = com1 +
                                            }
db;
        Complex test_com_2 = com1;
        test_{com_2} += db;
                                            Tests/Complex Tests/complex_unit_2.hpp
        if (test com 1.return re() ==
                                            #ifndef complex_unit_2_hpp
7 && test_com_1.return_im() == 5 && test_com_2.return_re() == 7 &&
                                            #define complex_unit_2_hpp
test_com_2.return_im() == 5){
                                            #include <stdio.h>
            test_ok = true;
                                            #include "complex.hpp"
                                            #include "tests.hpp"
        test_result(test_ok);
                                            using namespace std;
    }
                                            void run complex unit 2(void);
    //TECT 3
    if (test ok == true){
                                            #endif /* complex_unit_2_hpp */
        test_prepare(&test_id,
&test_ok);
                                            Tests/Complex Tests/complex_unit_3.cpp
        Complex com1 = Complex(2, 5);
        double db = 5;
                                            #include "complex unit 3.hpp"
        Complex test_com_1 = com1 *
                                            using namespace std;
db;
        Complex test_com_2 = com1;
                                            /*
        test_com_2 *= db;
                                             UNIT 3: Тестирование операций
                                            сравнения
        if (test_com_1.return_re() ==
10 && test_com_1.return_im() == 25 &&
                                             Список тестов:
test_com_2.return_re() == 10 &&
                                             Тест 1: Сравнение двух совпадающих
test_com_2.return_im() == 25){
                                            комплексных (для == и !=)
            test_ok = true;
                                             Тест 2: Сравнение двух несовпадающих
        }
                                            комплексных (для == и !=)
                                             Тест 3: Сравнение совпадающих
        test_result(test_ok);
                                            комплексного и вещественного (для == и
    }
                                             Тест 4: Сравнение несовпадающих
    //TECT 4
                                            комплексного и вещественного (для == и
    if (test_ok == true){
                                            !=)
        test_prepare(&test_id,
                                             */
&test_ok);
                                            void run_complex_unit_3(){
        Complex com1 = Complex(2, 5);
        Complex com2 = Complex(3, 4);
```

```
cout<<"[COMPLEX CLASS TESTING]</pre>
                                                     test_prepare(&test_id,
UNIT 3 :\n\n";
    int test_id = 0;
                                            &test ok);
    bool test_ok = true;
                                                     Complex com = Complex(3, 7);
                                                     double db = 3;
    //TECT 1
    if (test_ok == true){
                                                     bool test1_ok = (com == db);
                                                     bool test2_ok = (com != db);
        test_prepare(&test_id,
&test ok);
                                                     if (!test1_ok && test2_ok){
        Complex com1 = Complex(3, 4);
                                                         test_ok = true;
        Complex com2 = Complex(3, 4);
        bool test1 ok = (com1 ==
                                                     test_result(test_ok);
                                                }
com2);
        bool test2_ok = (com1 !=
com2);
                                                 if (test_id == 4 && test_ok ==
                                            true) {
        if (test1_ok && !test2_ok){
                                                     cout<<"\n[TESTING] Unit 3</pre>
                                            testing SUCCESSEDED.\n";
            test_ok = true;
                                            cout<<"----
        test_result(test_ok);
                                                 ----"<<"\n\n\n";
    }
                                                } else {
                                                     cout<<"\n[TESTING] Unit 3</pre>
                                            testing FAILED on TEST #"<<test_id<<".
    //TECT 2
    if (test ok == true){
        test_prepare(&test_id,
                                            cout<<"----
&test_ok);
                                            ----"<<"\n\n\n";
        Complex com1 = Complex(3, 4);
                                                }
        Complex com2 = Complex(5, -4);
        bool test1 ok = (com1 ==
com2);
                                            Tests/Complex Tests/complex_unit_3.hpp
        bool test2 ok = (com1 !=
com2):
                                            #ifndef complex_unit_3_hpp
                                            #define complex_unit_3_hpp
        if (!test1 ok && test2 ok){
            test ok = true;
                                            #include <stdio.h>
                                            #include "complex.hpp"
                                            #include "tests.hpp"
        test_result(test_ok);
    }
                                            using namespace std;
    //TECT 3
                                            void run_complex_unit_3(void);
    if (test_ok == true){
        test_prepare(&test_id,
                                            #endif /* complex unit 3 hpp */
&test_ok);
        Complex com = Complex(3, 0);
        double db = 3;
        bool test1_ok = (com == db);
        bool test2_ok = (com != db);
        if (test1_ok && !test2_ok){
            test_ok = true;
        }
        test_result(test_ok);
    }
    //TECT 4
    if (test_ok == true){
```