ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

по дисциплине «Системы управления базами данных»

Выполнил: ст. гр. ТКИ – 441

Гаврилин-Кирьянов И.А.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2022

ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc124969078)

[ЗАДАНИЕ 3](#_Toc124969079)

[СТРУКТУРА ПРОЕКТА 3](#_Toc124969080)

[1 ЛИСТИНГ ПРОГРАММЫ 4](#_Toc124969081)

[1.1 Файл Node.cpp» 4](#_Toc124969082)

[1.2 Файл «Binary\_tree.cpp» 9](#_Toc124969083)

[1.3 Файл «test\_tree.cpp» 14](#_Toc124969084)

[1.4 Файл «Makefile» 16](#_Toc124969085)

[2 ДИАГРАММЫ структур 17](#_Toc124969086)

[3 РЕЗУЛЬТАТЫ ПРОХОЖДЕНИЙ ТЕСТОВ 18](#_Toc124969087)

ЗАДАНИЕ

1. Для структуры «Двоичное дерево» разработать API;
2. На языке «C++» написать конструктор и деструктор класса разрабатываемой структуры для типа «int»;
3. Разработать метод вывода структуры в строку;
4. Реализовать основные функции для работы.
5. Разработать тесты.

СТРУКТУРА ПРОЕКТА

* Task\_2
  + source
    - Binary\_tree.cpp
    - main.cpp
    - Node.cpp
  + tests
    - test\_tree.cpp
  + Makefile

# ЛИСТИНГ ПРОГРАММЫ

## Файл Node.cpp»

1. #pragma once
2. #include <iostream>
3. #include <string>
4. #include <sstream>

7. /\*
8. \* \brief Узел дерева
9. \*/
10. struct node;
12. /\*
13. \* \brief Создание нового узла дерева
14. \* \param Данные узла
15. \*/
16. struct node\* newNode(int key);
18. struct node {
19. int          key;
20. struct node\* left = NULL;
21. struct node\* right = NULL;

24. /\*
25. \* \brief Конструктор без параметров
26. \*/
27. node();
29. /\*
30. \* \brief Конструктор с параметрами
31. \* \param Данные узла
32. \*/
33. node(const int key);
35. /\*
36. \* \brief Деструктор
37. \*/
38. ~node();
40. /\*
41. \* \brief Оператор вывода в стандартный поток данных
42. \* \param Стандартный поток данных
43. \* \param Узел дерева
44. \*/
45. friend std::ostream& operator<<(std::ostream& out, const node& node);
47. /\*
48. \* \brief Оператор вывода в стандартный поток данных
49. \* \param Стандартный поток данных
50. \* \param Узел дерева
51. \*/
52. friend std::wostream& operator<<(std::wostream& out, const node& node);
54. /\*
55. \* \brief Оператор ввода из стандартного потока данных
56. \* \param Стандартный поток данных
57. \* \param Узел дерева
58. \*/
59. friend std::istream& operator>>(std::istream& in, node& node);
61. /\*
62. \* \brief Оператор равенства двух узлов
63. \* \param Левый узел
64. \* \param Правый узел
65. \* \return true, если равны
66. \*/
67. friend bool operator==(const node& left, const node& right);
69. /\*
70. \* \brief Оператор неравенства двух узлов
71. \* \param Левый узел
72. \* \param Правый узел
73. \* \return true, если не равны
74. \*/
75. friend bool operator!=(const node& left, const node& right);
77. /\*
78. \* \brief Оператор больше
79. \* \param Левый узел
80. \* \param Правый узел
81. \* \return true, если левый узел больше
82. \*/
83. friend bool operator>(const node& left, const node& right);
85. /\*
86. \* \brief Оператор больше или равно
87. \* \param Левый узел
88. \* \param Правый узел
89. \* \return true, если левый узел больше
90. \*/
91. friend bool operator>=(const node& left, const node& right);
93. /\*
94. \* \brief Оператор меньше
95. \* \param Левый узел
96. \* \param Правый узел
97. \* \return true, если левый узел меньше
98. \*/
99. friend bool operator<(const node& left, const node& right);
101. /\*
102. \* \brief Оператор меньше
103. \* \param Левый узел
104. \* \param Правый узел
105. \* \return true, если левый узел меньше
106. \*/
107. friend bool operator<=(const node& left, const node& right);
108. };
110. struct node\* newNode(int key) {
111. struct node\* temp = new struct node;
112. temp->key = key;
113. return temp;
114. }
116. node::node() {}
118. node::node(const int key) {
119. this->key = key;
120. }
122. node::~node() { left = NULL; right = NULL; }
124. std::ostream& operator<<(std::ostream& out, const node& root)
125. {
126. out << root.key;
127. return out;
128. }
130. std::wostream& operator<<(std::wostream& out, const node& root)
131. {
132. out << root.key;
133. return out;
134. }
136. std::istream& operator>>(std::istream& in, node& node)
137. {
138. if (in.peek() == '\n' || in.eof())
139. {
140. return in;
141. }
143. in >> node.key;
144. return in;
145. }
147. bool operator==(const node& left, const node& right)
148. {
149. return left.key == right.key;
150. }
152. bool operator!=(const node& left, const node& right)
153. {
154. return left.key != right.key;
155. }
157. bool operator>(const node& left, const node& right)
158. {
159. return left.key > right.key;
160. }
162. bool operator>=(const node& left, const node& right)
163. {
164. return left.key >= right.key;
165. }
167. bool operator<(const node& left, const node& right)
168. {
169. return left.key < right.key;
170. }
172. bool operator<=(const node& left, const node& right)
173. {
174. return left.key <= right.key;
175. }

## Файл «Binary\_tree.cpp»

1. #pragma once
2. #include <iostream>
3. #include <string>
4. #include <sstream>
5. #include "Node.cpp"

8. /\*
9. \* \brief Вывод дерева
10. \* \param Корень дерева
11. \*/
12. std::string ToString(struct node\* root);

15. /\*
16. \* \brief Добавить ноду в дерево
17. \* \param Значение для вставки
18. \*/
19. struct node\* insert(struct node\* node, int key);
21. /\*
22. \* \brief Поиск inorder-преемника
23. \* \param Корень дерева
24. \*/
25. struct node\* minValueNode(struct node\* node);
27. /\*
28. \* \brief Удалить ноду из дерева
29. \* \param Корень дерева
30. \* \param Значение для удаления
31. \*/
32. struct node\* deleteNode(struct node\* root, int key);
34. /\*
35. \* \brief Заменить одну ноду на другую в дереве
36. \* \param Корень дерева
37. \* \param Текущая нода
38. \* \param Старое значение
39. \* \param Новое значение
40. \*/
41. void changeNode(struct node\* tree\_root, struct node\* root, int old\_key, int new\_key);
43. /\*
44. \* \brief Проверка дерева на пустоту
45. \* \param Корень дерева
46. \*/
47. bool IsEmpty(struct node\* root);
49. /\*
50. \* \brief Узел дерева
51. \* \param Корень дерева
52. \*/
53. void deleteTree(struct node\* root);

56. std::string ToString(struct node\* root) {
57. if (root == nullptr) {
58. return std::string("");
59. }
60. std::stringstream ss;
61. ss << ToString(root->left) << " " << root->key << " " << ToString(root->right);
62. return ss.str();
63. }
65. struct node\* insert(struct node\* node, int key) {
66. // Возвращаем новый узел, если дерево пустое
67. if (node == NULL) {
68. return newNode(key);
69. }
71. // Проходим в нужное место и вставляет узел
72. if (key <= node->key) {
73. node->left = insert(node->left, key);
74. }
75. else {
76. node->right = insert(node->right, key);
77. }
78. return node;
79. }
81. struct node\* minValueNode(struct node\* node) {
82. struct node\* current = node;
84. // Ищем крайний левый лист — он и будет inorder-преемником
85. while (current && current->left != NULL) {
86. current = current->left;
87. }
88. return current;
89. }
91. struct node\* deleteNode(struct node\* root, int key) {
92. // Возвращаем, если дерево пустое
93. if (root == NULL) {
94. return root;
95. }
97. // Ищем узел, который хотим удалить
98. if (key < root->key) {
99. root->left = deleteNode(root->left, key);
100. }
101. else if (key > root->key) {
102. root->right = deleteNode(root->right, key);
103. }
104. else {
105. // Если у узла один дочерний элемент или их нет
106. if (root->left == NULL) {
107. struct node\* temp = root->right;
108. free(root);
109. return temp;
110. }
111. else if (root->right == NULL) {
112. struct node\* temp = root->left;
113. free(root);
114. return temp;
115. }
117. // Если у узла два дочерних элемента
118. struct node\* temp = minValueNode(root->right);
120. // Помещаем inorder-преемника на место узла, который хотим удалить
121. root->key = temp->key;
123. // Удаляем inorder-преемника
124. root->right = deleteNode(root->right, temp->key);
125. }
126. return root;
127. }
129. void changeNode(struct node\* tree\_root, struct node\* root, int old\_key, int new\_key) {
130. if (root == NULL) {
131. return;
132. }
133. if (root->key == old\_key) {
134. deleteNode(tree\_root, old\_key);
135. insert(tree\_root, new\_key);
136. return;
137. }
138. if (old\_key <= root->key) {
139. changeNode(tree\_root, root->left, old\_key, new\_key);
140. }
141. else {
142. changeNode(tree\_root, root->right, old\_key, new\_key);
143. }
144. }
146. bool IsEmpty(struct node\* root) {
147. return (root == NULL);
148. }
150. void deleteTree(struct node\* root)
151. {
152. if (root != nullptr)
153. {
154. deleteTree(root->left);
155. deleteTree(root->right);
156. delete root;
157. }
158. }

## Файл «test\_tree.cpp»

1. #include <iostream>
2. #include <gtest/gtest.h>
3. #include <gmock/gmock.h>
5. #include <../source/main.cpp>

8. TEST(AvlTreeTestGroup, AvlTreeInsert) {
9. // Arange
10. const int expected\_data = 10;
11. struct node\* root = NULL;
13. // Act
14. root = insert(root, expected\_data);
16. // Assert
17. ASSERT\_TRUE(root->key == expected\_data);
18. }
20. TEST(AvlTreeTestGroup, AvlTreeIsEmpty) {
21. // Arange
22. const int expected\_data = 10;
23. struct node\* root = NULL;
25. // Act
26. root = insert(root, expected\_data);
27. root = deleteNode(root, expected\_data);
29. // Assert
30. ASSERT\_TRUE(root == nullptr);
31. }
33. TEST(AvlTreeTestGroup, AvlTreeRemove) {
34. // Arange
35. struct node\* root = NULL;
37. // Act
38. bool empty\_flag = IsEmpty(root);
40. // Assert
41. ASSERT\_TRUE(empty\_flag);
42. }
44. TEST(AvlTreeTestGroup, AvlTreeToString) {
45. // Arange
46. struct node\* root = NULL;
47. std::string expected = " 1  3  4  6  7  8  10  14 ";
49. // Act
50. root = insert(root, 8);
51. root = insert(root, 3);
52. root = insert(root, 1);
53. root = insert(root, 6);
54. root = insert(root, 7);
55. root = insert(root, 10);
56. root = insert(root, 14);
57. root = insert(root, 4);
59. // Assert
60. ASSERT\_TRUE(ToString(root) == expected);
61. }
63. int main(int argc, char \*\*argv)
64. {
65. ::testing::InitGoogleTest(&argc, argv);
67. return RUN\_ALL\_TESTS();
68. }

## Файл «Makefile»

**install**:

mkdir -p build

g++ -g source/main.cpp -o build/main) -I source/

g++ -g tests/test\_tree.cpp -o build/test\_tree -I source/ -lgtest -lpthread

build/test\_tree

# ДИАГРАММЫ структур

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 1. Диаграмма методов структуры Node

Text

Description automatically generated

Рисунок 2. Диаграмма методов структуры Binary\_tree

# РЕЗУЛЬТАТЫ ПРОХОЖДЕНИЙ ТЕСТОВ

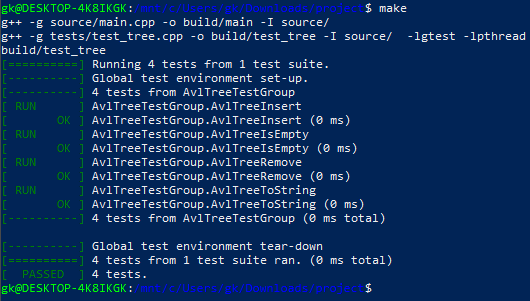


Рисунок 3. Результаты прохождения тестов