Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение

высшего образования

«**Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные

системы»

**ОТЧЁТ**

**по лабораторной работе №20**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: “Бинарные деревья.”

Вариант 13

Выполнил:

Студент группы ИВТ-20-2б

Сафронов Владислав Владиславович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

**Пермь, 2021**

**Постановка задачи**

1. Сформировать идеально сбалансированное бинарное дерево.

2. Тип информационного поля double. Найти количество лепестков в дереве.

3. Распечатать полученное дерево.

4. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.

5. Преобразовать идеально сбалансированное дерево в дерево поиска.

Распечатать полученное дерево.

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Организовать класс Tree с полями под указатель на правый, левый и родительский узлы, а также на хранящееся значение в узле, координаты узла и уровень, на котором узел находится.
   2. Подключить библиотеку отрисовки glut.h для печати дерева.
   3. Организовать отдельные функции в главном файле для работы с библиотекой OpenGL.
2. В ходе работы были использованы следующие типы данных:
   1. Класс Tree с полями под указатель на правый, левый и родительский узлы, а также на хранящееся значение в узле, координаты узла и уровень, на котором узел находится.
3. class Tree
4. {
5. double data;
6. Tree \*parent, \*left, \*right;
7. public:
8. GLfloat x = 0, y = 4;
9. int state, level = 1;
11. Tree();
12. Tree(double tmp);
13. ~Tree();
15. void insert\_r(double tmp);
16. void insert\_l(double tmp);
17. void insert\_node(Tree\*& rt, double tmp);
18. void in\_order(Tree\* node);
20. int get\_h();
22. Tree\* find\_node(double tmp);
24. friend void LevelCounter(Tree\* root);
25. friend void CountLevels(Tree\* root, void (\*LevelCounter)(Tree\* root));
26. friend void Coords(Tree\* node);
27. friend void CoordsCalculate(Tree\* node, void (\*Coords)(Tree\* node));
28. friend void DrawOneNode(Tree\* root);
29. friend void DrawNodes(Tree\* root, void (\*DrawOneNode)(Tree\* root));
30. friend void DrawOneLine(Tree\* root);
31. friend void DrawLines(Tree\* root, void (\*DrawOneLine)(Tree\* root));
32. void drawTree(int argc, char\*\* argv, int win\_height, int win\_width);
33. };
    1. Переменные типа int: width – максимальная ширина дерева (кол-во узлов на нижнем уровне), depth – глубина дерева(кол-во уровней).
34. int depth = 0, width = 0;
    1. Указатель tree класса Tree для работы с бинарным деревом. Значение, которое передаётся в качестве параметра – значение корня бинарного дерева.
35. Tree\* tree = new Tree(5);
    1. Вектор arr типа double для хранения данных узлов дерева.
36. vector<double> arr = { 8.1, 5.5, 11.0, 4.1, 7.2, 3.5, 15.1, 8.3, 3.0, 3.6, 4.6, 4.3, 4.8, 5.3 };
37. Для операций ввода и вывода использовались следующие операторы и функции:

4.1. Функция drawTree() для вывода дерева в отдельное окно.

1. void Tree::drawTree(int argc, char\*\* argv, int win\_height, int win\_width)
2. {
3. glutInit(&argc, argv);
4. glutInitWindowPosition(0, 0);
5. glutInitWindowSize(win\_height, win\_width);
6. glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);
7. glutCreateWindow("Binary Tree");
8. glutReshapeFunc(reshape);
9. glutDisplayFunc(display);
10. glutMainLoop();
11. }

 4.2. Функция display для отрисовки узлов и связей между ними.

1. void display()
2. {
3. glClearColor(1, 1, 1, 1);
4. glClear(GL\_COLOR\_BUFFER\_BIT);
5. glColor3f(0.0, 0.0, 0.0);
6. glLineWidth(2);
7. DrawLines(tree, DrawOneLine);
8. DrawNodes(tree, DrawOneNode);
9. glutSwapBuffers();
10. }

 4.3. Функция PrintingInfo для печати в консоль данных о глубине, ширине и количестве лепестков в дереве.

1. void PrintingInfo(int& count, vector<double>& arr)
2. {
3. depth = tree->get\_h();
4. width = pow(2, depth - 1);
5. cout << "Глубина дерева: " << depth << endl
6. << "Ширина дерева: " << width << endl;
7. cout << "Количество листьев в дереве: " << count << endl;
8. }

**Блок-схема программы**

**Код программы**

**Файл Three.h:**

1. #pragma once
2. #include <iostream>
3. #include "GL/glut.h"
4. #include <string>
5. #include <cmath>
6. #include <windows.h>
7. #include <stdio.h>
8. using namespace std;
10. class Tree
11. {
12. double data;
13. Tree \*parent, \*left, \*right;
14. public:
15. GLfloat x = 0, y = 4;
16. int state, level = 1;
18. Tree();
19. Tree(double tmp);
20. ~Tree();
22. void insert\_r(double tmp);
23. void insert\_l(double tmp);
24. void insert\_node(Tree\*& rt, double tmp);
25. void in\_order(Tree\* node);
27. int get\_h();
29. Tree\* find\_node(double tmp);
31. friend void LevelCounter(Tree\* root);
32. friend void CountLevels(Tree\* root, void (\*LevelCounter)(Tree\* root));
33. friend void Coords(Tree\* node);
34. friend void CoordsCalculate(Tree\* node, void (\*Coords)(Tree\* node));
35. friend void DrawOneNode(Tree\* root);
36. friend void DrawNodes(Tree\* root, void (\*DrawOneNode)(Tree\* root));
37. friend void DrawOneLine(Tree\* root);
38. friend void DrawLines(Tree\* root, void (\*DrawOneLine)(Tree\* root));
39. void drawTree(int argc, char\*\* argv, int win\_height, int win\_width);
40. };

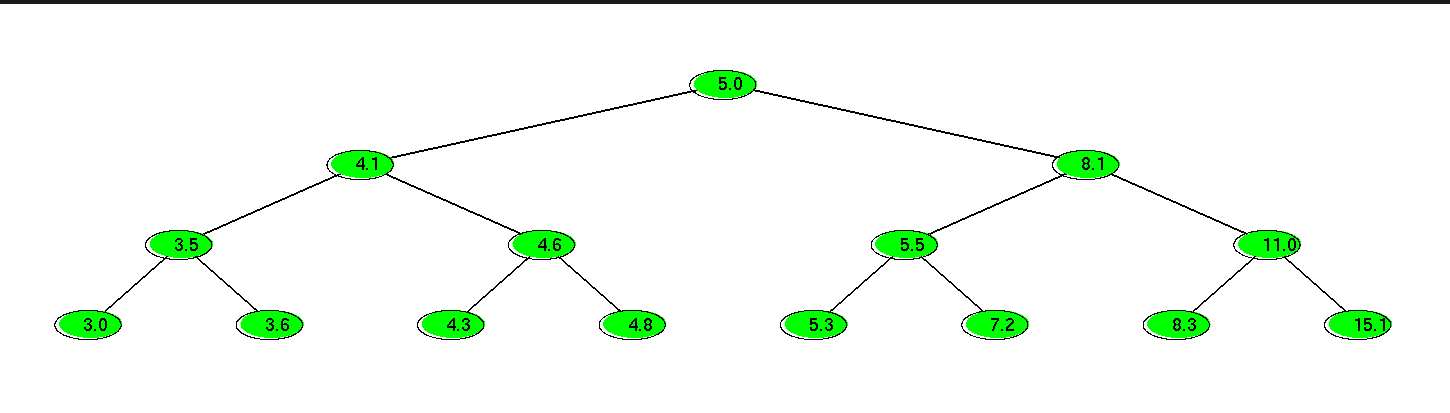
**Файл Three.cpp:**

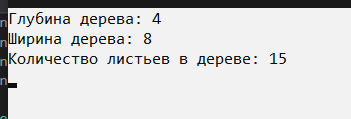
1. #include "Tree.h"
3. Tree::Tree()
4. {
5. parent = NULL;
6. right = NULL;
7. left = NULL;
8. }
9. Tree::Tree(double tmp)
10. {
11. data = tmp;
12. parent = NULL;
13. right = NULL;
14. left = NULL;
15. }
16. Tree::~Tree()
17. {
18. if (this != NULL) delete this;
19. }
21. void Tree::insert\_r(double tmp)
22. {
23. Tree\* node = new Tree(tmp);
24. if (this->right != NULL)
25. {
26. this->right->parent = node;
27. node->right = this->left;
28. this->right = node;
29. node->parent = this;
30. }
31. else
32. {
33. Tree\* node = new Tree(tmp);
34. this->right = node;
35. node->parent = this;
36. }
37. }
38. void Tree::insert\_l(double tmp)
39. {
40. Tree\* node = new Tree(tmp);
41. if (this->left != NULL)
42. {
43. this->left->parent = node;
44. node->left = this->left;
45. this->left = node;
46. node->parent = this;
47. }
48. else
49. {
50. Tree\* node = new Tree(tmp);
51. this->left = node;
52. node->parent = this;
53. }
54. }
55. void Tree::insert\_node(Tree\*& rt, double tmp)
56. {
57. if (rt == NULL)
58. {
59. rt = new Tree(tmp);
60. return;
61. }
62. Tree\* node = rt;
63. while (node != NULL)
64. {
65. if (tmp >= node->data)
66. {
67. if (node->right != NULL) node = node->right;
68. else
69. {
70. node->insert\_r(tmp);
71. return;
72. }
73. }
74. else
75. {
76. if (node->left != NULL) node = node->left;
77. else
78. {
79. node->insert\_l(tmp);
80. return;
81. }
82. }
83. }
84. }
86. void Tree::in\_order(Tree\* node)
87. {
88. if (node == NULL) return;
89. in\_order(node->left);
90. cout << node->data << "|";
91. in\_order(node->right);
92. }
94. int Tree::get\_h()
95. {
96. int h1 = 0, h2 = 0;
97. if (this == NULL) return 0;
98. if (this->left != NULL) h1 = this->left->get\_h();
99. if (this->right != NULL) h2 = this->right->get\_h();
100. if (h1 >= h2) return (h1 + 1);
101. else return (h2 + 1);
102. }
104. Tree\* Tree::find\_node(double tmp)
105. {
106. Tree\* node = this;
107. while (node != NULL)
108. {
109. if (node->data == tmp) return node;
110. else
111. {
112. if (node->data > tmp) node = node->left;
113. else node = node->right;
114. }
115. }
116. return NULL;
117. }

**Файл main.cpp:**

1. #include "Tree.h"
2. #include <vector>
3. #include <algorithm>
4. #include <sstream>
6. Tree\* tree = new Tree(5);
7. float radiusA = 0.35; //радиус окружности
8. int depth = 0, width = 0, count\_nodes = 1;
10. template <typename T>
11. string to\_string\_with\_precision(const T a\_value, const int n = 1)
12. {
13. std::ostringstream out;
14. out.precision(n);
15. out << std::fixed << a\_value;
16. return out.str();
17. }
19. void DrawCircle(char colour, GLfloat x, GLfloat y, float radiusB, int count)
20. {
21. glColor3f(0.0, 250.0, 0.0);
22. glBegin(GL\_TRIANGLE\_FAN);
23. glVertex2f(x, y);
24. for (int i = 0; i <= count; i++) {
25. glVertex2f(
26. (x + (radiusA \* cos(i \* count))),
27. (y + (radiusB \* sin(i \* count))));
28. }
29. glEnd();
30. }
32. void DrawOutline(float tmp\_x, float tmp\_y, float radiusB)
33. {
34. glColor3f(0.0, 0.0, 0.0);
35. glBegin(GL\_POINTS);
36. for (int i = 0; i <= radiusA; i++) {
37. for (int j = 0; j <= 540; j++) {
38. tmp\_x = radiusA \* sin(j) + tmp\_x;
39. tmp\_y = radiusB \* cos(j) + tmp\_y;
40. glVertex2f(tmp\_x - 0.35, tmp\_y - 0.1);
41. }
42. }
43. glEnd();
44. }
46. void drawNode(const char\* str, GLfloat x, GLfloat y, char colour)
47. {
48. double coefficient = 0;
49. coefficient = (4 + depth) / pow(2, depth);
50. float radiusB = coefficient \* radiusA;
51. int count = 50;
52. DrawCircle(colour, x, y, radiusB, count);
53. DrawOutline(x, y, radiusB);
54. glColor3f(0.0, 0.0, 0.0);
55. glRasterPos2f(x - 0.075, y - 0.075);
56. const char\* p;
57. for (p = str; \*p != '\0'; p++) {
58. glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*p);
59. }
60. }
62. void reshape(int height, int width)
63. {
64. glMatrixMode(GL\_PROJECTION);
65. glLoadIdentity();
66. glViewport(0, 0, height, width);
67. gluOrtho2D(-pow(2, depth - 1), pow(2, depth - 1), -depth, 5);
68. }
70. void DrawOneLine(Tree\* root)
71. {
72. if (root->parent != NULL) {
73. glBegin(GL\_LINES);
74. glVertex2d(root->parent->x, root->parent->y);
75. glVertex2d(root->x, root->y);
76. glEnd();
77. }
78. }
79. void DrawOneNode(Tree\* root)
80. {
81. char colour;
82. if (root->parent != NULL) {
83. colour = 'g';
84. drawNode(to\_string\_with\_precision(root->data).c\_str(), root->x, root->y, colour);
85. }
86. }
87. void display()
88. {
89. glClearColor(1, 1, 1, 1);
90. glClear(GL\_COLOR\_BUFFER\_BIT);
91. glColor3f(0.0, 0.0, 0.0);
92. glLineWidth(2);
93. DrawLines(tree, DrawOneLine);
94. DrawNodes(tree, DrawOneNode);
95. glutSwapBuffers();
96. }
98. void LevelCounter(Tree\* root)
99. {
100. if (root->parent != NULL)
101. root->level = root->parent->level + 1; //если предок существует, то +1
102. if (depth < root->level) //глубина дерева
103. depth = root->level;
104. }
106. void CountLevels(Tree\* root, void (\*LevelCounter)(Tree\* root))
107. {
108. if (root == NULL)
109. return;
110. (\*LevelCounter)(root);
111. CountLevels(root->left, LevelCounter);
112. CountLevels(root->right, LevelCounter);
113. }
115. void Coords(Tree\* node)
116. {
117. if (node->parent != NULL) {
118. if (node->level == 2)
119. node->x = node->parent->x + node->state \* (pow(2, depth - 1) / 2);
120. else
121. node->x = node->parent->x + node->state \*
122. (pow(2, depth - 1) / pow(2, node->level - 1));
123. node->y = node->parent->y - 1;
124. }
125. }
127. void CoordsCalculate(Tree\* node, void (\*Coords)(Tree\* node))
128. {
129. if (node == NULL)
130. return;
131. (\*Coords)(node);
132. if (node->left != NULL) {
133. node->left->state = -1;
134. CoordsCalculate(node->left, Coords);
135. }
136. if (node->right != NULL) {
137. node->right->state = 1;
138. CoordsCalculate(node->right, Coords);
139. }
140. return;
141. }
143. void DrawNodes(Tree\* root, void (\*DrawOneNode)(Tree\* root))
144. {
145. char colour;
146. if (root == NULL)
147. return;
148. (\*DrawOneNode)(root);
149. DrawNodes(root->left, DrawOneNode);
150. DrawNodes(root->right, DrawOneNode);
151. colour = 'g';
152. drawNode(to\_string\_with\_precision(root->data).c\_str(), tree->x, tree->y, colour);
153. }
155. void DrawLines(Tree\* root, void (\*DrawOneLine)(Tree\* root))
156. {
157. if (root == NULL)
158. return;
159. (\*DrawOneLine)(root);
160. DrawLines(root->left, DrawOneLine);
161. DrawLines(root->right, DrawOneLine);
162. }
164. void Tree::drawTree(int argc, char\*\* argv, int win\_height, int win\_width)
165. {
166. glutInit(&argc, argv);
167. glutInitWindowPosition(0, 0);
168. glutInitWindowSize(win\_height, win\_width);
169. glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);
170. glutCreateWindow("Binary Tree");
171. glutReshapeFunc(reshape);
172. glutDisplayFunc(display);
173. glutMainLoop();
174. }
176. void PrintingInfo(int& count, vector<double>& arr)
177. {
178. depth = tree->get\_h();
179. width = pow(2, depth - 1);
180. cout << "Глубина дерева: " << depth << endl
181. << "Ширина дерева: " << width << endl;
182. cout << "Количество листьев в дереве: " << count << endl;
183. }
185. int main(int argc, char\*\* argv)
186. {
187. system("chcp 1251>nul");
188. vector<double> arr = { 8.1, 5.5, 11.0, 4.1, 7.2, 3.5, 15.1, 8.3, 3.0, 3.6, 4.6, 4.3, 4.8, 5.3 };
189. for (int i = 0; i < arr.size(); i++) {
190. tree->insert\_node(tree, arr[i]);
191. count\_nodes++;
192. }
193. PrintingInfo(count\_nodes, arr);
194. CountLevels(tree, LevelCounter);
195. CoordsCalculate(tree, Coords);
196. tree->drawTree(argc, argv, 1451, 720);
197. return 0;
198. }

**Скриншоты результатов работы программы**

****

****