Липецкий Государственный Технический Университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ОТЧЕТ
об учебной практике
в ЛГТУ

Студент

Группа АС-21-1 _____ Станиславчук С. М.

Подпись, дата

Руководитель

Доцент, кандидат наук _____ Муравейко А. Ю.

Подпись, дата

Липецк 2022г.

Задание кафедры:

Вариант 13

Задачей индивидуального задания является: создать структуру данных и реализовать способ их обработки в форме на C++, обеспечить заполнение данных из файла и формы, рисование и сохранение изображения (System.Drawing), создание отчета в виде html-файла, содержащего ссылки на изображения итераций процесса и отображение html-файла в форме (компонент WebBrowser) для обхода бинарного дерева в ширину.

Аннотация:

С. 31. Ил. 8. Литература 4 назв. Прил. 3;

В работе создана программа на языке C++, реализующая обход в ширину бинарного дерева. Представлено описание функций программы и контрольный пример ее работы.

Оглавление:

Ход выполнения практики:

1. Интерфейс программы:

На рисунке 1 показана форма программы. После построения дерева появляются кнопки с возможностью сделать обход дерева в ширину и сохранить результат работы.

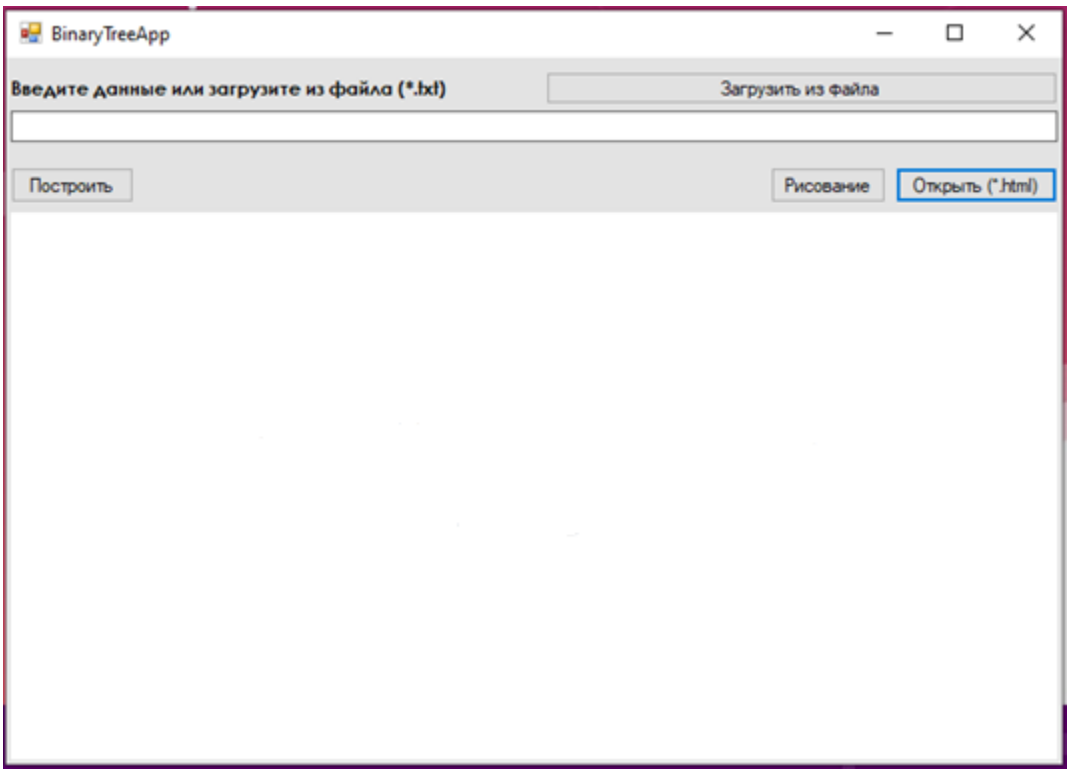

Рисунок 1 – Интерфейс программы

2. Реализация структуры бинарного дерева:

Бинарное дерево представляет собой иерархическую структуру данных, в которой каждый узел имеет не более двух дочерних узлов. Первый называется родительским узлом или корнем дерева (root), а дочерние узлы называются левым и правым наследниками.

```
struct Node {
        int key;
        int height = 1;
        bool is_rightNode;
        Node* leftNode = NULL;
        Node* rightNode = NULL;
        };
```

В структуру одного узла входит:

1) Ключевое значение;

2) Вспомогательная переменная для вывода;

3) Вспомогательная булева переменная, которая проверяет наличие правого дочернего узла;

4) Два дочерних угла;

2. Ввод данных:

Вводить данные можно с клавиатуры, а также загрузить их из файла (*.txt). Эти данные будут отображаться в текстовом поле (TextBox).

```
private: System::Void fileUpload_button_Click(System::Object^ sender, System::EventArgs^ e) {
            openFileDialog1->Title = "Открыть файл";
            openFileDialog1->Filter = "Txt files (*.txt)|*.txt|All files (*.*)|*.*";
            if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
                  this->textBox1->Text = File::ReadAllText(openFileDialog1->FileName);
            }
            else return;
}
```

На рисунке 2 показан пример ввода данных.

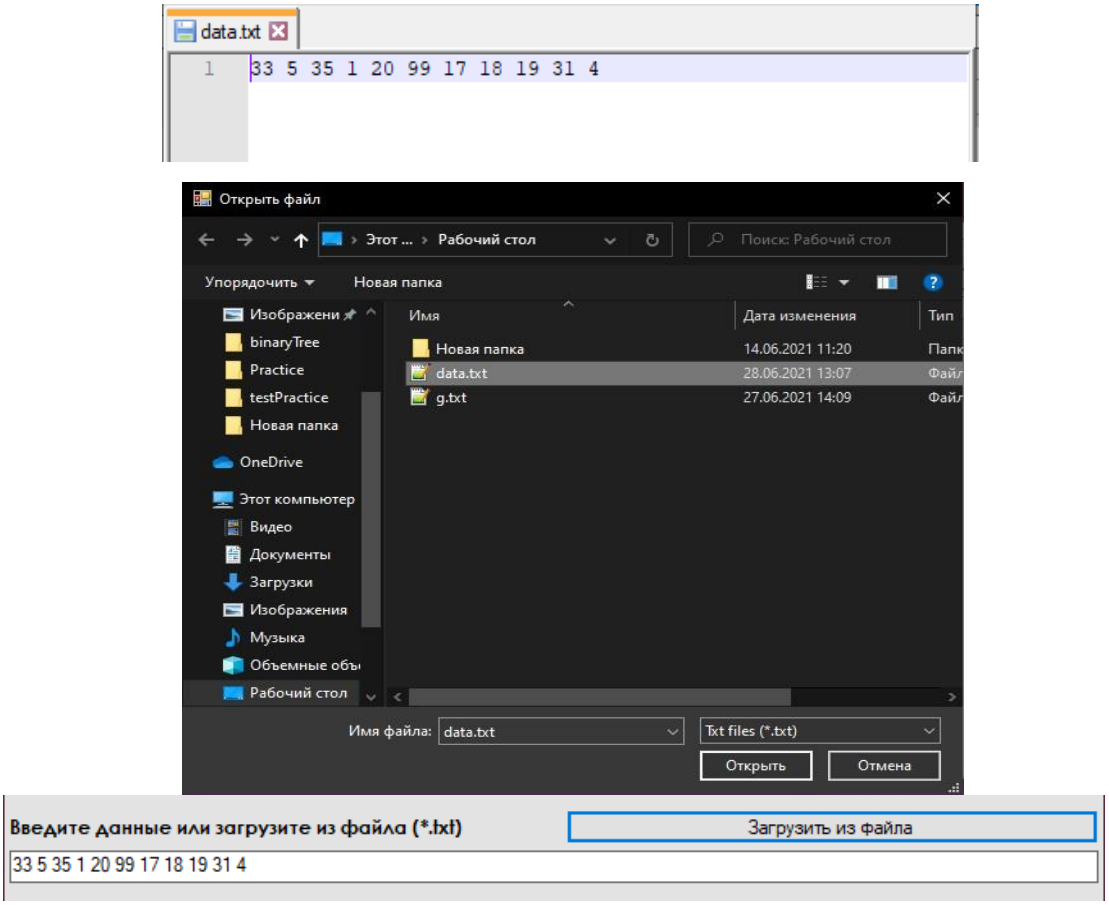

Рисунок 2 – Ввод данных из файла

3. Обработка данных:

Обработка данных начинает производиться по нажатию на кнопку «Построить». Сначала данные, которые находились в TextBox в виде строки, преобразовываются в массив. Этот массив используется для создания бинарного дерева (функция GenerateTree()). Далее, построенное дерево выводится с помощью функции PrintTree(). Код этих функций находится в приложении А. Когда дерево будет создано, станут доступны кнопки «Обход в ширину», «Сохранить (*.bmp)», «Сохранить (*.html)».

При обходе в ширину, в строку записываются ключевые значения узлов в том порядке, который подразумевает данный обход. После прохода, он выводится пользователю в элементе Label (Рисунок 3).

```
String^ strBFT = "";
	private: System::Void BreadthFirstTraversal_button_Click(System::Object^ sender, System::EventArgs^ e)
{
		int count = allKeys->Length;
		resultBFT->Text = "";
		strBFT = "";
		std::queue<Node> allKeys;
		allKeys.push(*root);
		while (!allKeys.empty()) {
			Node current = allKeys.front();
			allKeys.pop();
			count--;
			strBFT += Convert::ToString(current.key);
			if (count != 0) strBFT += " --> ";
			if (current.leftNode != nullptr)
				allKeys.push(*current.leftNode);
			if (current.rightNode != nullptr)
				allKeys.push(*current.rightNode);
		}
		resultBFT->Text = "Обход в ширину: " + strBFT;
		resultBFT->Visible = true;
		}
```

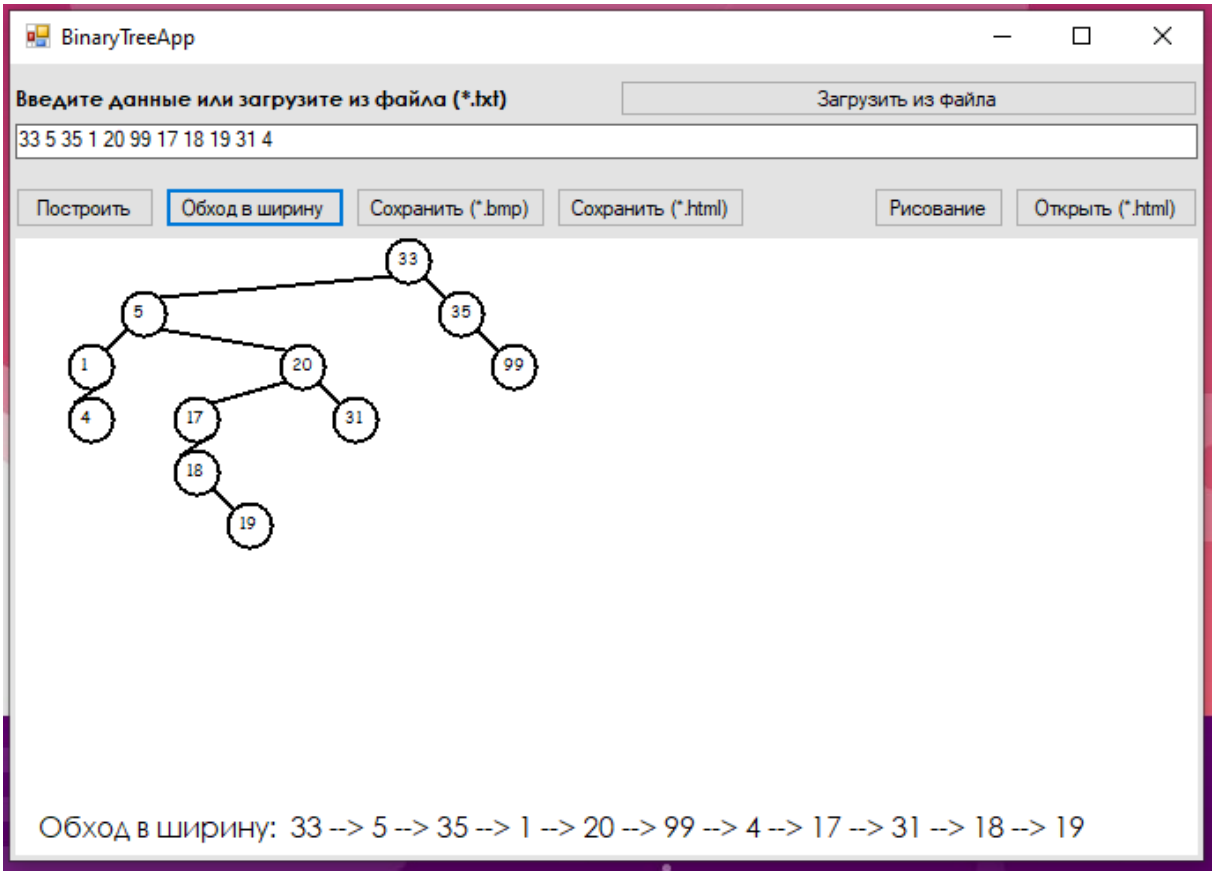

Рисунок 3 – Создание дерева и обход в ширину

4. Рисование и сохранение изображения:

Открыть форму для рисования позволяет кнопка «Рисование». В новой форме появляются следующие возможности:

1) «Взять кисть». После нажатия на кнопку пользователь может начать рисовать на PictureBox;

```
bool penTakenIs = false;

private: System::Void takenPen_button_Click(System::Object^ sender, System::EventArgs^ e) {
                if (penTakenIs == false) {
                        bmp = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);
                        pictureBox1->Image = bmp;
                        Graphics^ graph = Graphics::FromImage(pictureBox1->Image);
                        graph->Clear(Color::White);
                        penTakenIs = true;
                }
                else return;
        }
```

Теперь мы можем рисовать (Рисунок 4).

```
bool Draw = false;

private: System::Void pictureBox1_MouseDown(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
		Draw = true;
	}

	private: System::Void pictureBox1_MouseUp(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
		Draw = false;
	}
	private: System::Void takenPen_button_Click(System::Object^ sender, System::EventArgs^ e) {
		if (penTakenIs == false) {
			bmp = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);
			pictureBox1->Image = bmp;
			Graphics^ graph = Graphics::FromImage(pictureBox1->Image);
			graph->Clear(Color::White);
			penTakenIs = true;
		}
		else return;
		}
```

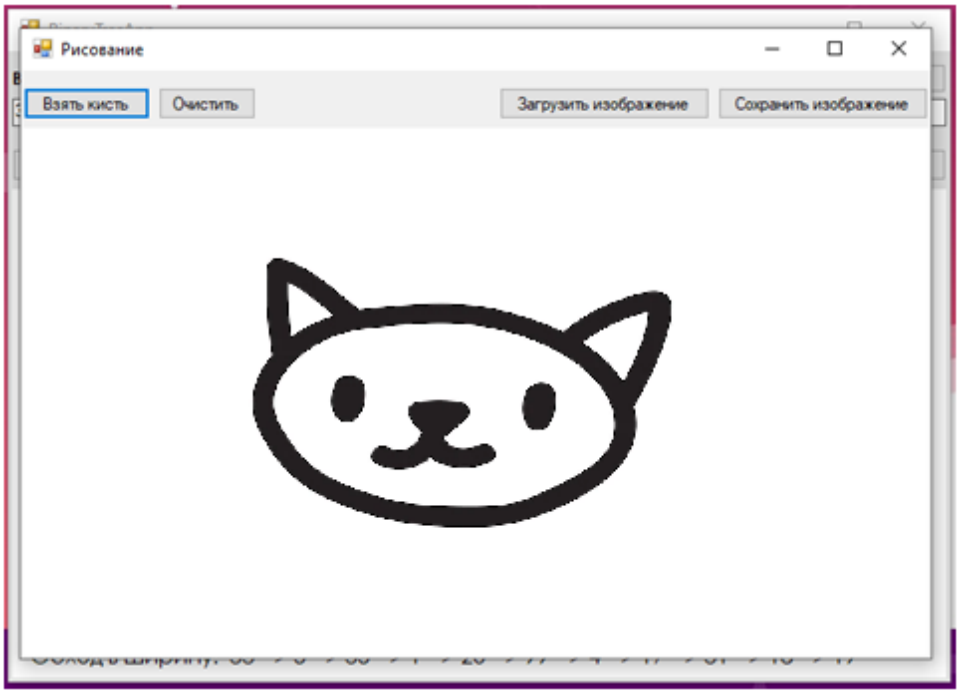

Рисунок 4 – Рисование на PictureBox

2) «Очистить». После нажатия на кнопку холст PictureBox очищается;

```
Bitmap^ bmp;
private: System::Void clear_button_Click(System::Object^ sender, System::EventArgs^ e) {
		bmp = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);
		pictureBox1->Image = bmp;
		Graphics^ graph = Graphics::FromImage(pictureBox1->Image);
		graph->Clear(Color::White);
	}
```

3) Загрузить изображение (Рисунок 5) и Сохранить изображение (Рисунок 6);

```
private: System::Void downloadImage_button_Click(System::Object^ sender, System::EventArgs^ e) {
            openFileDialog1->Title = "Открыть изображение";
            openFileDialog1->Filter = "Image Files(*.BMP;*.JPG;*.GIF|*.BMP;*.JPG;*.GIF|All files(*.*)|*.*";
            if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
                  pictureBox1->Image = Image::FromFile(openFileDialog1->FileName);
            }
      }
```

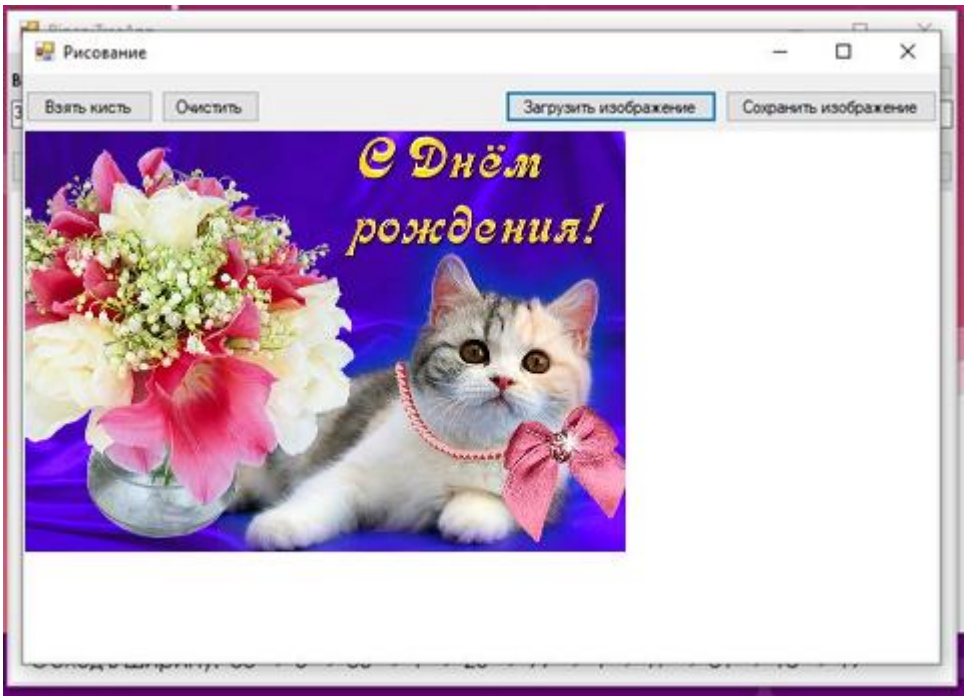

Рисунок 5 – Загрузка изображения в PictureBox

```
private: System::Void saveImage_button_Click(System::Object^ sender, System::EventArgs^ e) {
            saveFileDialog1->Title = "Сохранить изображение";
            saveFileDialog1->Filter = "Image File(*.BMP)|*.BMP";
            if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
                  bmp->Save(saveFileDialog1->FileName, System::Drawing::Imaging::ImageFormat::Bmp);
            }
      }
```

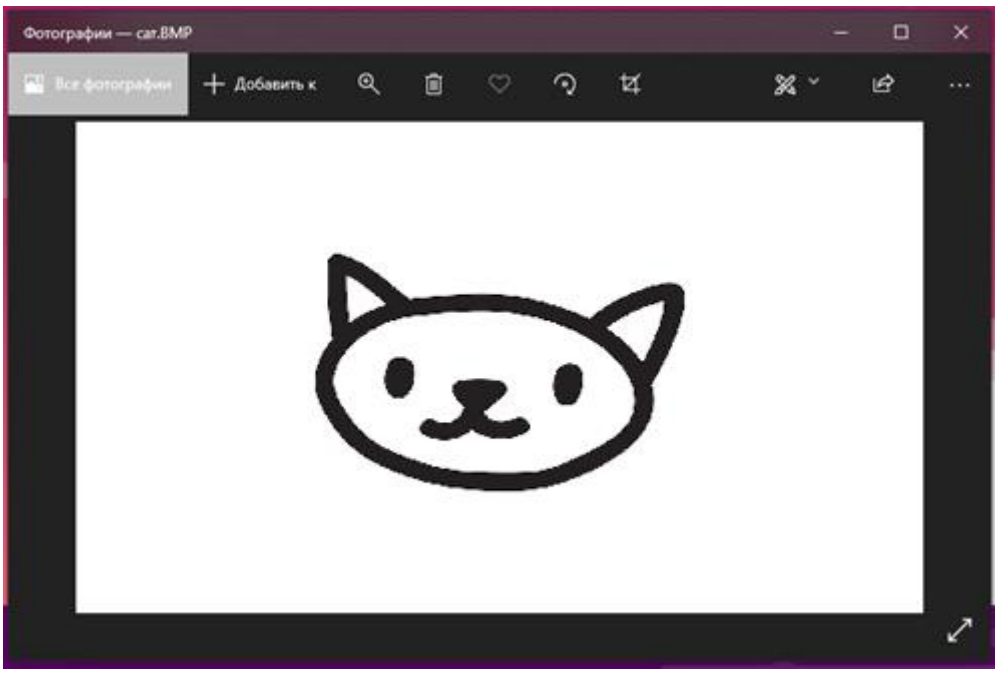

Рисунок 6 – Сохранение изображения

Полный код содержится в Приложении В.

5. Сохранение и открытие отчета HTML:

При сохранении html-отчета (Рисунок 7) создается файл, в который записывается все необходимое для успешного открытия в браузере. Форма, которая открывает html-файл с элементом WebBrowser, отображена в приложении С.

```
String^ htmlSavePath = Directory::GetCurrentDirectory();
	private: System::Void saveHTML_button_Click(System::Object^ sender, System::EventArgs^ e) {
		saveFileDialog1->Title = "Сохранить файл";
		saveFileDialog1->OverwritePrompt = true;
		saveFileDialog1->CheckPathExists = true;
		saveFileDialog1->ShowHelp = true;
		saveFileDialog1->Filter = "Html files (*.html)|*.html|All files (*.*)|*.*";
		if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
			StreamWriter^ NewHtml = gcnew StreamWriter(saveFileDialog1->FileName);
			NewHtml->WriteLine("<html>");
			NewHtml->WriteLine("<head>");
			NewHtml->WriteLine("<title>Binary tree traversal</title>");
			NewHtml->WriteLine("</head>");
			NewHtml->WriteLine("<body>");
			NewHtml->WriteLine("<p style=\"font-size: 24px\"><font face=\"Century Gothic\">" +
"Breadth traversal of a binary tree:" + "</font></p>\n");
			String^ fileName = htmlSavePath + "\\" + "resultTree.bmp";
			bmp->Save(fileName, System::Drawing::Imaging::ImageFormat::Bmp);
			NewHtml->WriteLine("<p><img src=\"" + fileName + "\" alt=\"Binary tree""\"></p>\n");
			if (strBFT != "") NewHtml->WriteLine("<p style=\"font-size: 24px\"><font face=\"Century
Gothic\">" + "Breadth-first traversal: " + strBFT + "</font></p>\n");
```

```
                NewHtml->WriteLine("</body>");
                NewHtml->WriteLine("</html>");
                NewHtml->Close();
            }
        }
```

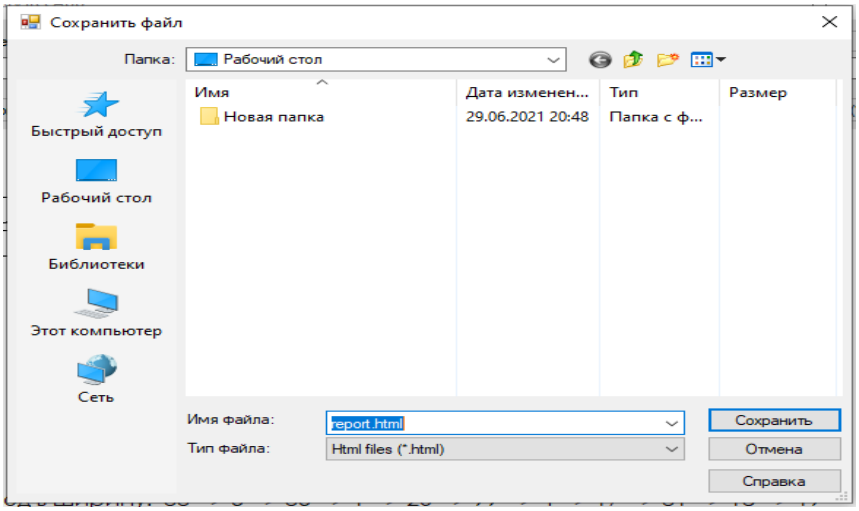

Рисунок 7 – Сохранение HTML отчета

Открытие HTML-файла (Рисунок 8).

```
private: System::Void openHTML_button_Click(System::Object^ sender, System::EventArgs^ e)
{
            Form^ web = gcnew WebForm("");
            web->ShowDialog();
            }
```

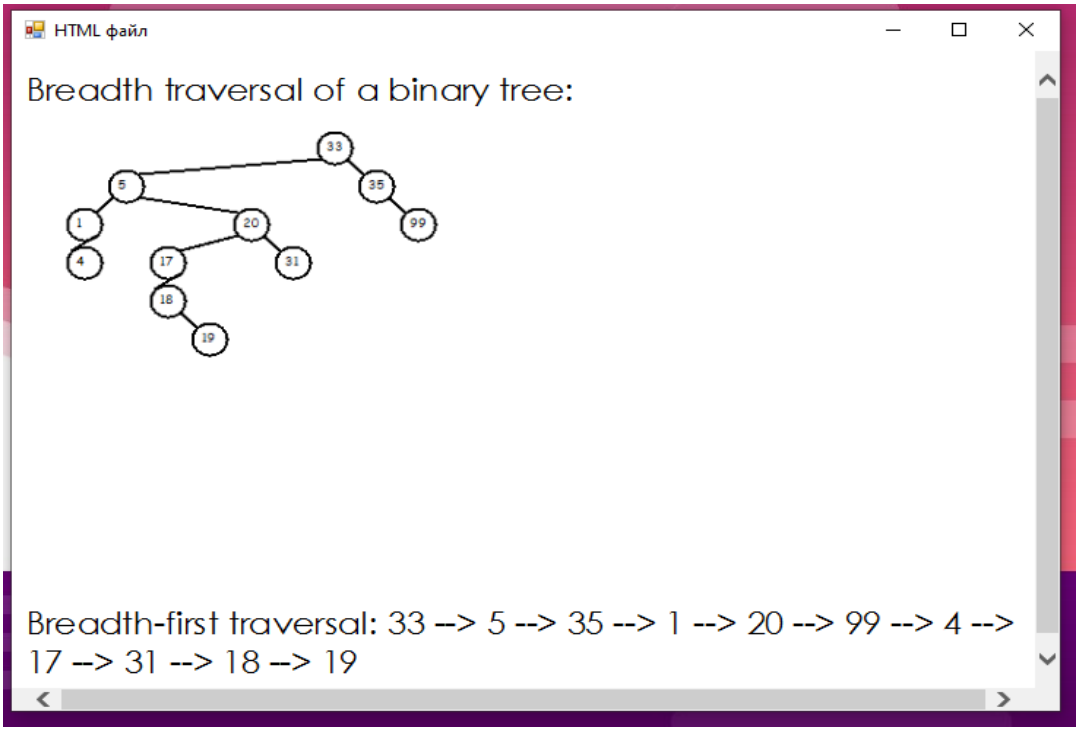

Рисунок 8 – Отчет HTML

Заключение:

В результате прохождения учебной практики, я написал программу, которая реализует обход бинарного дерева в ширину, позволяет пользователю создать, сохранить и просматривать HTML-отчет о работе программы, а также рисовать и сохранять свои рисунки на жесткий диск ПК.

Библиографический список:

1. Лаврухина, Т.В. Учебная практика для студентов 1 курса [Текст]: методические указания к проведению учебной практики для студентов 1 курса / Т.В. Лаврухина. – Липецк: Изд-во Липецкого государственного технического университета, 2016. – 16с.

2. СТО-13-2016. Студенческие работы. Общие требования к оформлению (Версия 2) [Текст]. – Введ. 2016 – 02 – 01. – Липецк: Изд-во ЛГТУ, 2016. – 36с.

3. Шилдт, Г. C++ Базовый курс [Текст] / Г. Шилдт. – 3-е изд. – Москва: Издателиский дом «Вильямс», 2015. – 624с.

4. Петцольд Ч. Программирование с использованием Microsoft Windows Forms. / Ч. Петцольд; пер. с агнл. А.Р. Врублевского. – М. : Русская Редакция; СПб. : Питер, 2006. – 432 стр.

Приложение А

Файл MainWinForm.h

```
#pragma once
#include <iostream>
#include <queue>
#include "PaintingForm.h"
#include "WebForm.h"

namespace binaryTree {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::IO;

	struct Node {
		int key;
		int height = 1;
		bool is_rightNode;
		Node* leftNode = NULL;
		Node* rightNode = NULL;
	};

	/// <summary>
	/// Сводка для MainWinForm
	/// </summary>
	public ref class MainWinForm : public System::Windows::Forms::Form
	{
	public:
		MainWinForm(void)
		{
			InitializeComponent();
			//
			//TODO: добавьте код конструктора
			//
		}

	protected:
		/// <summary>
		/// Освободить все используемые ресурсы.
		/// </summary>
		~MainWinForm()
		{
			if (components)
			{
				delete components;
			}
		}

	private: System::Windows::Forms::TextBox^ textBox1;
	private: System::Windows::Forms::Label^ label1;
```

```
		private: System::Windows::Forms::Button^ fileUpload_button;
		private: System::Windows::Forms::Button^ treeBuilding_button;
		private: System::Windows::Forms::Button^ BreadthFirstTraversal_button;
		private: System::Windows::Forms::Button^ saveBMP_button;
		private: System::Windows::Forms::Button^  painting_button;
		private: System::Windows::Forms::Button^ saveHTML_button;
		private: System::Windows::Forms::Button^ openHTML_button;
		private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;
		private: System::Windows::Forms::PictureBox^ pictureBox1;
		private: System::Windows::Forms::Label^ nameAuthor;
		private: System::Windows::Forms::Label^ group;
		private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;
		private: System::Windows::Forms::Label^ resultBFT;
		private:
			/// <summary>
			/// Обязательная переменная конструктора.
			/// </summary>
			System::ComponentModel::Container ^components;

	#pragma region Windows Form Designer generated code
			/// <summary>
			/// Требуемый метод для поддержки конструктора — не изменяйте
			/// содержимое этого метода с помощью редактора кода.
			/// </summary>
			void InitializeComponent(void)
			{
				this->textBox1 = (gcnew System::Windows::Forms::TextBox());
				this->label1 = (gcnew System::Windows::Forms::Label());
				this->fileUpload_button = (gcnew System::Windows::Forms::Button());
				this->treeBuilding_button = (gcnew System::Windows::Forms::Button());
				this->BreadthFirstTraversal_button = (gcnew
System::Windows::Forms::Button());
				this->saveBMP_button = (gcnew System::Windows::Forms::Button());
				this->painting_button = (gcnew System::Windows::Forms::Button());
				this->saveHTML_button = (gcnew System::Windows::Forms::Button());
				this->openHTML_button = (gcnew System::Windows::Forms::Button());
				this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());
				this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
				this->nameAuthor = (gcnew System::Windows::Forms::Label());
				this->group = (gcnew System::Windows::Forms::Label());
				this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());
				this->resultBFT = (gcnew System::Windows::Forms::Label());
				(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>pictureBox1))->BeginInit();
				this->SuspendLayout();
				//
				// textBox1
				//
				this->textBox1->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>(((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Left)
					| System::Windows::Forms::AnchorStyles::Right));
				this->textBox1->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
				this->textBox1->Location = System::Drawing::Point(3, 34);
				this->textBox1->Name = L"textBox1";
				this->textBox1->Size = System::Drawing::Size(671, 20);
				this->textBox1->TabIndex = 11;
```

```
                                    //
                                    // label1
                                    //
                                    this->label1->AutoSize = true;
                                    this->label1->Font = (gcnew System::Drawing::Font(L"Century Gothic", 9,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                            static_cast<System::Byte>(204)));
                                    this->label1->Location = System::Drawing::Point(0, 11);
                                    this->label1->Name = L"label1";
                                    this->label1->Size = System::Drawing::Size(287, 16);
                                    this->label1->TabIndex = 10;
                                    this->label1->Text = L"Введите данные или загрузите из файла (*.txt)";
                                    //
                                    // fileUpload_button
                                    //
                                    this->fileUpload_button->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Right));
                                    this->fileUpload_button->Location = System::Drawing::Point(346, 9);
                                    this->fileUpload_button->Name = L"fileUpload_button";
                                    this->fileUpload_button->Size = System::Drawing::Size(328, 21);
                                    this->fileUpload_button->TabIndex = 9;
                                    this->fileUpload_button->Text = L"Загрузить из файла";
                                    this->fileUpload_button->UseVisualStyleBackColor = true;
                                    this->fileUpload_button->Click += gcnew System::EventHandler(this,
&MainWinForm::fileUpload_button_Click);
                                    //
                                    // treeBuilding_button
                                    //
                                    this->treeBuilding_button->Location = System::Drawing::Point(3, 70);
                                    this->treeBuilding_button->Name = L"treeBuilding_button";
                                    this->treeBuilding_button->Size = System::Drawing::Size(79, 23);
                                    this->treeBuilding_button->TabIndex = 8;
                                    this->treeBuilding_button->Text = L"Построить";
                                    this->treeBuilding_button->UseVisualStyleBackColor = true;
                                    this->treeBuilding_button->Click += gcnew System::EventHandler(this,
&MainWinForm::treeBuilding_button_Click);
                                    //
                                    // BreadthFirstTraversal_button
                                    //
                                    this->BreadthFirstTraversal_button->Location = System::Drawing::Point(88, 70);
                                    this->BreadthFirstTraversal_button->Name = L"BreadthFirstTraversal_button";
                                    this->BreadthFirstTraversal_button->Size = System::Drawing::Size(102, 23);
                                    this->BreadthFirstTraversal_button->TabIndex = 7;
                                    this->BreadthFirstTraversal_button->Text = L"Обход в ширину";
                                    this->BreadthFirstTraversal_button->UseVisualStyleBackColor = true;
                                    this->BreadthFirstTraversal_button->Visible = false;
                                    this->BreadthFirstTraversal_button->Click += gcnew System::EventHandler(this,
&MainWinForm::BreadthFirstTraversal_button_Click);
                                    //
                                    // saveBMP_button
                                    //
                                    this->saveBMP_button->Location = System::Drawing::Point(196, 70);
                                    this->saveBMP_button->Name = L"saveBMP_button";
                                    this->saveBMP_button->Size = System::Drawing::Size(108, 23);
                                    this->saveBMP_button->TabIndex = 6;
                                    this->saveBMP_button->Text = L"Сохранить (*.bmp)";
                                    this->saveBMP_button->UseVisualStyleBackColor = true;
```

```
                this->saveBMP_button->Visible = false;
                this->saveBMP_button->Click += gcnew System::EventHandler(this,
&MainWinForm::saveBMP_button_Click);
                //
                // painting_button
                //
                this->painting_button->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Right));
                this->painting_button->Location = System::Drawing::Point(490, 70);
                this->painting_button->Name = L"painting_button";
                this->painting_button->Size = System::Drawing::Size(74, 23);
                this->painting_button->TabIndex = 5;
                this->painting_button->Text = L"Рисование";
                this->painting_button->UseVisualStyleBackColor = true;
                this->painting_button->Click += gcnew System::EventHandler(this,
&MainWinForm::painting_button_Click);
                //
                // saveHTML_button
                //
                this->saveHTML_button->Location = System::Drawing::Point(310, 70);
                this->saveHTML_button->Name = L"saveHTML_button";
                this->saveHTML_button->Size = System::Drawing::Size(107, 23);
                this->saveHTML_button->TabIndex = 4;
                this->saveHTML_button->Text = L"Сохранить (*.html)";
                this->saveHTML_button->UseVisualStyleBackColor = true;
                this->saveHTML_button->Visible = false;
                this->saveHTML_button->Click += gcnew System::EventHandler(this,
&MainWinForm::saveHTML_button_Click);
                //
                // openHTML_button
                //
                this->openHTML_button->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Right));
                this->openHTML_button->Location = System::Drawing::Point(570, 70);
                this->openHTML_button->Name = L"openHTML_button";
                this->openHTML_button->Size = System::Drawing::Size(104, 23);
                this->openHTML_button->TabIndex = 3;
                this->openHTML_button->Text = L"Открыть (*.html)";
                this->openHTML_button->UseVisualStyleBackColor = true;
                this->openHTML_button->Click += gcnew System::EventHandler(this,
&MainWinForm::openHTML_button_Click);
                //
                // openFileDialog1
                //
                this->openFileDialog1->FileName = L"openFileDialog1";
                //
                // pictureBox1
                //
                this->pictureBox1->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Bottom)
                        | System::Windows::Forms::AnchorStyles::Left)
                        | System::Windows::Forms::AnchorStyles::Right));
                this->pictureBox1->BackColor =
System::Drawing::SystemColors::ControlLightLight;
                this->pictureBox1->Location = System::Drawing::Point(3, 99);
```

```
                                this->pictureBox1->Name = L"pictureBox1";
                                this->pictureBox1->Size = System::Drawing::Size(671, 350);
                                this->pictureBox1->TabIndex = 2;
                                this->pictureBox1->TabStop = false;
                                //
                                // nameAuthor
                                //
                                this->nameAuthor->AutoSize = true;
                                this->nameAuthor->BackColor =
System::Drawing::SystemColors::ControlLightLight;
                                this->nameAuthor->Font = (gcnew System::Drawing::Font(L"Century Gothic",
36, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                        static_cast<System::Byte>(204)));
                                this->nameAuthor->ForeColor = System::Drawing::SystemColors::MenuBar;
                                this->nameAuthor->Location = System::Drawing::Point(132, 199);
                                this->nameAuthor->Name = L"nameAuthor";
                                this->nameAuthor->Size = System::Drawing::Size(415, 58);
                                this->nameAuthor->TabIndex = 1;
                                this->nameAuthor->Text = L"Dmitriy Selivanov";
                                this->nameAuthor->TextAlign =
System::Drawing::ContentAlignment::TopCenter;
                                //
                                // group
                                //
                                this->group->AutoSize = true;
                                this->group->BackColor = System::Drawing::SystemColors::ControlLightLight;
                                this->group->Font = (gcnew System::Drawing::Font(L"Century Gothic", 36,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                        static_cast<System::Byte>(204)));
                                this->group->ForeColor = System::Drawing::SystemColors::MenuBar;
                                this->group->Location = System::Drawing::Point(249, 257);
                                this->group->Name = L"group";
                                this->group->Size = System::Drawing::Size(184, 58);
                                this->group->TabIndex = 0;
                                this->group->Text = L"AI-20-1";
                                //
                                // resultBFT
                                //
                                this->resultBFT->AutoSize = true;
                                this->resultBFT->BackColor = System::Drawing::SystemColors::ControlLightLight;
                                this->resultBFT->Font = (gcnew System::Drawing::Font(L"Century Gothic", 12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                        static_cast<System::Byte>(204)));
                                this->resultBFT->Location = System::Drawing::Point(12, 422);
                                this->resultBFT->Name = L"resultBFT";
                                this->resultBFT->Size = System::Drawing::Size(155, 21);
                                this->resultBFT->TabIndex = 12;
                                this->resultBFT->Text = L"Результат обхода: ";
                                this->resultBFT->Visible = false;
                                //
                                // MainWinForm
                                //
                                this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                                this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                                this->BackColor = System::Drawing::SystemColors::ControlLight;
                                this->ClientSize = System::Drawing::Size(677, 452);
                                this->Controls->Add(this->resultBFT);
                                this->Controls->Add(this->group);
```

```
                this->Controls->Add(this->nameAuthor);
                this->Controls->Add(this->pictureBox1);
                this->Controls->Add(this->openHTML_button);
                this->Controls->Add(this->saveHTML_button);
                this->Controls->Add(this->painting_button);
                this->Controls->Add(this->saveBMP_button);
                this->Controls->Add(this->BreadthFirstTraversal_button);
                this->Controls->Add(this->treeBuilding_button);
                this->Controls->Add(this->fileUpload_button);
                this->Controls->Add(this->label1);
                this->Controls->Add(this->textBox1);
                this->MinimumSize = System::Drawing::Size(630, 407);
                this->Name = L"MainWinForm";
                this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
                this->Text = L"BinaryTreeApp";
                this->Resize += gcnew System::EventHandler(this,
&MainWinForm::MainWinForm_Resize);
                (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>pictureBox1))->EndInit();
                this->ResumeLayout(false);
                this->PerformLayout();

            }
        #pragma endregion
            private: System::Void fileUpload_button_Click(System::Object^ sender, System::EventArgs^ e) {
                openFileDialog1->Title = "Открыть файл";
                openFileDialog1->Filter = "Txt files (*.txt)|*.txt|All files (*.*)|*.*";
                if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
                    this->textBox1->Text = File::ReadAllText(openFileDialog1->FileName);
                }
                else return;
            }

            private: System::Void painting_button_Click(System::Object^ sender, System::EventArgs^ e) {
                Form^ paintingForm = gcnew PaintingForm();
                paintingForm->ShowDialog();
            }

            private: System::Void openHTML_button_Click(System::Object^ sender, System::EventArgs^ e) {
                Form^ web = gcnew WebForm("");
                web->ShowDialog();
            }

            private: System::Void MainWinForm_Resize(System::Object^ sender, System::EventArgs^ e) {
                nameAuthor->Visible = false;
                group->Visible = false;
            }

              array<int>^ allKeys;
              Bitmap^ bmp;
              Node* root;

            private: System::Void treeBuilding_button_Click(System::Object^ sender, System::EventArgs^ e) {
                //Включение кнопок и выключение водяного знака
                BreadthFirstTraversal_button->Visible = true;
                saveBMP_button->Visible = true;
                saveHTML_button->Visible = true;
```

```
nameAuthor->Visible = false;
group->Visible = false;
resultBFT->Visible = false;

//Создание и заполнение массива
String^ str = textBox1->Text;
array<String^>^ tmp = str->Split(' ');
int count = 0;
for (int i = 0; i < tmp->Length; i++) {
	if (tmp[i] != "" && tmp[i] != " ") count++;
}
allKeys = gcnew array<int>(count);
for (int i = 0, j = 0; i < tmp->Length && j <= count; i++)
{
	if (tmp[i] != "")
	{
		try
		{
			allKeys[j] = Int32::Parse(tmp[i]);
		}
		catch (...)
		{
			MessageBox::Show("Некорректные данные в поле");
			return;
		}
		j++;
	}
}

//Генерация дерева
try
{
	int counterer = 0;
	root = new Node;
	root->key = allKeys[counterer];
	counterer++;
	GenerateTree(root, root, &counterer, allKeys);
}
catch (...)
{
	MessageBox::Show("Некорректные данные в поле");
	return;
}

//Вывод дерева
int newWidth = height(root);
int newHeight = 0;
Point a(0, 0);
if (pictureBox1->Height < (30 * newWidth + 50) * (height(root->rightNode) + 1) && (90 *
newWidth > 350) && (60 * newWidth > 652)) {
	this->Width = 90 * newWidth;
	this->Height = 60 * newWidth;
	pictureBox1->Width = 90 * newWidth;
	pictureBox1->Height = 60 * newWidth;
	pictureBox1->Refresh();
}
bmp = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);
Graphics^ G = Graphics::FromImage(bmp);
```

```
		G->Clear(SystemColors::Window);
		PrintTree(root, newWidth + 1, newHeight, a);
		pictureBox1->Image = bmp;
	}

	private: void GenerateTree(Node* root, Node* currentNode, int* counter, array<int>^ allKeys)
	{
		if (*counter >= allKeys->Length) return;
		if (currentNode->key > allKeys[*counter]) {
			if (currentNode->leftNode != NULL) {
				GenerateTree(root, currentNode->leftNode, counter, allKeys);
				if (currentNode->leftNode != NULL) changeHeight(currentNode);
			}
			else {
				if (height(currentNode->leftNode) >= height(currentNode->rightNode))
currentNode->height++;
				currentNode->leftNode = new Node;
				currentNode->leftNode->key = allKeys[*counter];
				currentNode->leftNode->is_rightNode = false;
				*counter += 1;
				GenerateTree(root, root, counter, allKeys);
			}
		}
		else if (currentNode->key <= allKeys[*counter]) {
			if (currentNode->rightNode != NULL) {
				GenerateTree(root, currentNode->rightNode, counter, allKeys);
				if (currentNode->rightNode != NULL) changeHeight(currentNode);
			}
			else {
				if (height(currentNode->leftNode) <= height(currentNode->rightNode))
currentNode->height++;
				currentNode->rightNode = new Node;
				currentNode->rightNode->key = allKeys[*counter];
				currentNode->rightNode->is_rightNode = true;
				*counter += 1;
				GenerateTree(root, root, counter, allKeys);
			}
		}
	}
	private: int height(Node* node) {
		if (node) {
			return node->height;
		}
		else return 0;
	}
	private: void changeHeight(Node* node) {
		int left = height(node->leftNode);
		int right = height(node->rightNode);
		if (left > right) {
			node->height = left + 1;
		}
		else {
			node->height = right + 1;
		}
	}

	private: void PrintTree(Node* root, int newWidth, int newHeight, Point^ parent) {
		if (root == NULL) return;
```

```
                        Graphics^ graph = Graphics::FromImage(bmp);
                        Graphics^ output = pictureBox1->CreateGraphics();
                        Pen^ pen = gcnew Pen(Brushes::Black);
                        graph->DrawEllipse(pen, 30 * newWidth, 30 * newHeight, 25, 25);
                        output->DrawEllipse(pen, 30 * newWidth, 30 * newHeight, 25, 25);
                        String^ str = Convert::ToString(root->key);
                        graph->DrawString(str, gcnew System::Drawing::Font("Arial", 8), gcnew
SolidBrush(Color::Black), Point(30 * newWidth + 5, 30 * newHeight + 5));
                        output->DrawString(str, gcnew System::Drawing::Font("Arial", 8), gcnew
SolidBrush(Color::Black), Point(30 * newWidth + 5, 30 * newHeight + 5));
                        if (parent->X != 0 || parent->Y != 0) {
                                if (root->is_rightNode == true) {
                                        graph->DrawLine(pen, Point(25 * 0.875 + parent->X, 25 * 0.875 +
parent->Y), Point(30 * newWidth + 25 * 0.125, 30 * newHeight + 25 * 0.125));
                                        output->DrawLine(pen, Point(25 * 0.875 + parent->X, 25 * 0.875 +
parent->Y), Point(30 * newWidth + 25 * 0.125, 30 * newHeight + 25 * 0.125));
                                }
                                else {
                                        graph->DrawLine(pen, Point(25 * 0.125 + parent->X, 25 * 0.875 +
parent->Y), Point(30 * newWidth + 25 * 0.875, 30 * newHeight + 25 * 0.125));
                                        output->DrawLine(pen, Point(25 * 0.125 + parent->X, 25 * 0.875 +
parent->Y), Point(30 * newWidth + 25 * 0.875, 30 * newHeight + 25 * 0.125));
                                }
                                delete graph;
                                delete output;
                        }
                        else {
                                delete graph;
                                delete output;
                                if (root->leftNode != NULL) {
                                        PrintTree(root->leftNode, newWidth - 1 * height(root->leftNode-
>rightNode) - 1, newHeight + 1, Point(30 * newWidth, 30 * newHeight));
                                }
                                if (root->rightNode != NULL) {
                                        PrintTree(root->rightNode, newWidth + 1 * height(root->rightNode-
>leftNode) + 1, newHeight + 1, Point(30 * newWidth, 30 * newHeight));
                                }
                                return;
                        }
                        if (root->is_rightNode == false) {
                                if (height(root->leftNode) >= height(root->rightNode)) {
                                        PrintTree(root->leftNode, newWidth - 1, newHeight + 1, Point(30 *
newWidth, 30 * newHeight));
                                        if (root->leftNode != NULL) {
                                                PrintTree(root->rightNode, newWidth + 1 * height(root-
>leftNode->rightNode), newHeight + 1, Point(30 * newWidth, 30 * newHeight));
                                        }
                                }
                                else if (height(root->rightNode) >= height(root->leftNode)) {
                                        PrintTree(root->leftNode, newWidth - 1, newHeight + 1, Point(30 *
newWidth, 30 * newHeight));
                                        if (root->rightNode != NULL) {
                                                PrintTree(root->rightNode, newWidth + 1 * height(root-
>rightNode->leftNode), newHeight + 1, Point(30 * newWidth, 30 * newHeight));
                                        }
                                }
                        }
                        else {
```

```
                        if (height(root->leftNode) >= height(root->rightNode)) {
                                if (root->leftNode != NULL) {
                                        PrintTree(root->leftNode, newWidth - 1 * height(root-
>leftNode->rightNode), newHeight + 1, Point(30 * newWidth, 30 * newHeight));
                                }
                                PrintTree(root->rightNode, newWidth + 1, newHeight + 1, Point(30 *
newWidth, 30 * newHeight));
                        }
                        else if (height(root->rightNode) >= height(root->leftNode)) {
                                if (root->rightNode != NULL) {
                                        PrintTree(root->leftNode, newWidth - 1 * height(root-
>rightNode->leftNode), newHeight + 1, Point(30 * newWidth, 30 * newHeight));
                                }
                                PrintTree(root->rightNode, newWidth + 1, newHeight + 1, Point(30 *
newWidth, 30 * newHeight));
                        }
                }
        }

        //Обход в ширину
                String^ strBFT = "";
        private: System::Void BreadthFirstTraversal_button_Click(System::Object^ sender,
System::EventArgs^ e) {
                int count = allKeys->Length;
                resultBFT->Text = "";
                strBFT = "";
                std::queue<Node> allKeys;
                allKeys.push(*root);
                while (!allKeys.empty()) {
                        Node current = allKeys.front();
                        allKeys.pop();
                        count--;
                        strBFT += Convert::ToString(current.key);
                        if (count != 0) strBFT += " --> ";
                        if (current.leftNode != nullptr)
                                allKeys.push(*current.leftNode);
                        if (current.rightNode != nullptr)
                                allKeys.push(*current.rightNode);
                }
                resultBFT->Text = "Обход в ширину: " + strBFT;
                resultBFT->Visible = true;
        }

        private: System::Void saveBMP_button_Click(System::Object^ sender, System::EventArgs^ e) {
                saveFileDialog1->Title = "Сохранить изображение";
                saveFileDialog1->Filter = "Image File(*.BMP)|*.BMP";
                if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
                        bmp->Save(saveFileDialog1->FileName,
System::Drawing::Imaging::ImageFormat::Bmp);
                }
        }

                String^ htmlSavePath = Directory::GetCurrentDirectory();
        private: System::Void saveHTML_button_Click(System::Object^ sender, System::EventArgs^ e) {
                saveFileDialog1->Title = "Сохранить файл";
                saveFileDialog1->OverwritePrompt = true;
                saveFileDialog1->CheckPathExists = true;
                saveFileDialog1->ShowHelp = true;
```

```
                saveFileDialog1->Filter = "Html files (*.html)|*.html|All files (*.*)|*.*";
                if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
                        StreamWriter^ NewHtml = gcnew StreamWriter(saveFileDialog1->FileName);
                        NewHtml->WriteLine("<html>");
                        NewHtml->WriteLine("<head>");
                        NewHtml->WriteLine("<title>Binary tree traversal</title>");
                        NewHtml->WriteLine("</head>");
                        NewHtml->WriteLine("<body>");
                        NewHtml->WriteLine("<p style=\"font-size: 24px\"><font face=\"Century
Gothic\">" + "Breadth traversal of a binary tree:" + "</font></p>\n");
                        String^ fileName = htmlSavePath + "\\" + "resultTree.bmp";
                        bmp->Save(fileName, System::Drawing::Imaging::ImageFormat::Bmp);
                        NewHtml->WriteLine("<p><img src=\"" + fileName + "\" alt=\"Binary
tree""\"></p>\n");
                        if (strBFT != "") NewHtml->WriteLine("<p style=\"font-size: 24px\"><font
face=\"Century Gothic\">" + "Breadth-first traversal: " + strBFT + "</font></p>\n");
                        NewHtml->WriteLine("</body>");
                        NewHtml->WriteLine("</html>");
                        NewHtml->Close();
                }
        }
};
}
```

Приложение В

Файл PaintingForm.h

```
#pragma once

namespace binaryTree {

        using namespace System;
        using namespace System::ComponentModel;
        using namespace System::Collections;
        using namespace System::Windows::Forms;
        using namespace System::Data;
        using namespace System::Drawing;

        /// <summary>
        /// Сводка для PaintingForm
        /// </summary>
        public ref class PaintingForm : public System::Windows::Forms::Form
        {
        public:
                PaintingForm(void)
                {
                        InitializeComponent();
                        //
                        //TODO: добавьте код конструктора
                        //
                }

        protected:
                /// <summary>
                /// Освободить все используемые ресурсы.
```

```
		/// </summary>
		~PaintingForm()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::PictureBox^ pictureBox1;
	private: System::Windows::Forms::Button^ takenPen_button;
	private: System::Windows::Forms::Button^ clear_button;
	private: System::Windows::Forms::Button^ downloadImage_button;
	private: System::Windows::Forms::Button^ saveImage_button;
	private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;
	private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;




	protected:

	private:
		/// <summary>
		/// Обязательная переменная конструктора.
		/// </summary>
		System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
		/// <summary>
		/// Требуемый метод для поддержки конструктора — не изменяйте
		/// содержимое этого метода с помощью редактора кода.
		/// </summary>
		void InitializeComponent(void)
		{
			this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
			this->takenPen_button = (gcnew System::Windows::Forms::Button());
			this->clear_button = (gcnew System::Windows::Forms::Button());
			this->downloadImage_button = (gcnew System::Windows::Forms::Button());
			this->saveImage_button = (gcnew System::Windows::Forms::Button());
			this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());
			this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))-
>BeginInit();
			this->SuspendLayout();
			//
			// pictureBox1
			//
			this->pictureBox1->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Bottom)
				| System::Windows::Forms::AnchorStyles::Left)
				| System::Windows::Forms::AnchorStyles::Right));
			this->pictureBox1->BackColor = System::Drawing::SystemColors::ControlLightLight;
			this->pictureBox1->Location = System::Drawing::Point(2, 41);
			this->pictureBox1->Name = L"pictureBox1";
			this->pictureBox1->Size = System::Drawing::Size(650, 378);
			this->pictureBox1->TabIndex = 0;
```

```
                this->pictureBox1->TabStop = false;
                this->pictureBox1->MouseDown += gcnew
System::Windows::Forms::MouseEventHandler(this, &PaintingForm::pictureBox1_MouseDown);
                this->pictureBox1->MouseMove += gcnew
System::Windows::Forms::MouseEventHandler(this, &PaintingForm::pictureBox1_MouseMove);
                this->pictureBox1->MouseUp += gcnew
System::Windows::Forms::MouseEventHandler(this, &PaintingForm::pictureBox1_MouseUp);
                //
                // takenPen_button
                //
                this->takenPen_button->Location = System::Drawing::Point(2, 12);
                this->takenPen_button->Name = L"takenPen_button";
                this->takenPen_button->Size = System::Drawing::Size(91, 23);
                this->takenPen_button->TabIndex = 1;
                this->takenPen_button->Text = L"Взять кисть";
                this->takenPen_button->UseVisualStyleBackColor = true;
                this->takenPen_button->Click += gcnew System::EventHandler(this,
&PaintingForm::takenPen_button_Click);
                //
                // clear_button
                //
                this->clear_button->Location = System::Drawing::Point(99, 12);
                this->clear_button->Name = L"clear_button";
                this->clear_button->Size = System::Drawing::Size(70, 23);
                this->clear_button->TabIndex = 2;
                this->clear_button->Text = L"Очистить";
                this->clear_button->UseVisualStyleBackColor = true;
                this->clear_button->Click += gcnew System::EventHandler(this,
&PaintingForm::clear_button_Click);
                //
                // downloadImage_button
                //
                this->downloadImage_button->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Right));
                this->downloadImage_button->Location = System::Drawing::Point(344, 12);
                this->downloadImage_button->Name = L"downloadImage_button";
                this->downloadImage_button->Size = System::Drawing::Size(151, 23);
                this->downloadImage_button->TabIndex = 3;
                this->downloadImage_button->Text = L"Загрузить изображение";
                this->downloadImage_button->UseVisualStyleBackColor = true;
                this->downloadImage_button->Click += gcnew System::EventHandler(this,
&PaintingForm::downloadImage_button_Click);
                //
                // saveImage_button
                //
                this->saveImage_button->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Right));
                this->saveImage_button->Location = System::Drawing::Point(501, 12);
                this->saveImage_button->Name = L"saveImage_button";
                this->saveImage_button->Size = System::Drawing::Size(151, 23);
                this->saveImage_button->TabIndex = 4;
                this->saveImage_button->Text = L"Сохранить изображение";
                this->saveImage_button->UseVisualStyleBackColor = true;
                this->saveImage_button->Click += gcnew System::EventHandler(this,
&PaintingForm::saveImage_button_Click);
                //
```

```
			// openFileDialog1
			//
			this->openFileDialog1->FileName = L"openFileDialog1";
			//
			// PaintingForm
			//
			this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
			this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
			this->ClientSize = System::Drawing::Size(654, 421);
			this->Controls->Add(this->saveImage_button);
			this->Controls->Add(this->downloadImage_button);
			this->Controls->Add(this->clear_button);
			this->Controls->Add(this->takenPen_button);
			this->Controls->Add(this->pictureBox1);
			this->MinimumSize = System::Drawing::Size(502, 316);
			this->Name = L"PaintingForm";
			this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;
			this->Text = L"Рисование";
			this->Resize += gcnew System::EventHandler(this, &PaintingForm::PaintingForm_Resize);
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))-
>EndInit();
			this->ResumeLayout(false);

		}
#pragma endregion
		bool Draw = false;
		Bitmap^ bmp;
		bool penTakenIs = false;
	private: System::Void pictureBox1_MouseMove(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
		if (penTakenIs == true) {
			Graphics^ graph = Graphics::FromImage(pictureBox1->Image);
			if (Draw == true) {
				graph->FillEllipse(Brushes::DarkCyan, e->X, e->Y, 10, 10);
				pictureBox1->Refresh();
			}
		}
		else return;
	}

	private: System::Void pictureBox1_MouseDown(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
		Draw = true;
	}

	private: System::Void pictureBox1_MouseUp(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
		Draw = false;
	}

	private: System::Void takenPen_button_Click(System::Object^ sender, System::EventArgs^ e) {
		if (penTakenIs == false) {
			bmp = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);
			pictureBox1->Image = bmp;
			Graphics^ graph = Graphics::FromImage(pictureBox1->Image);
			graph->Clear(Color::White);
			penTakenIs = true;
		}
```

```
		else return;
	}

	private: System::Void clear_button_Click(System::Object^ sender, System::EventArgs^ e) {
		bmp = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);
		pictureBox1->Image = bmp;
		Graphics^ graph = Graphics::FromImage(pictureBox1->Image);
		graph->Clear(Color::White);
	}

	private: System::Void downloadImage_button_Click(System::Object^ sender, System::EventArgs^ e) {
		openFileDialog1->Title = "Открыть изображение";
		openFileDialog1->Filter = "Image Files(*.BMP;*.JPG;*.GIF|*.BMP;*.JPG;*.GIF|All files(*.*)|*.*";
		if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
			pictureBox1->Image = Image::FromFile(openFileDialog1->FileName);
		}
	}

	private: System::Void saveImage_button_Click(System::Object^ sender, System::EventArgs^ e) {
		saveFileDialog1->Title = "Сохранить изображение";
		saveFileDialog1->Filter = "Image File(*.BMP)|*.BMP";
		if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
			bmp->Save(saveFileDialog1->FileName, System::Drawing::Imaging::ImageFormat::Bmp);
		}
	}

	private: System::Void PaintingForm_Resize(System::Object^ sender, System::EventArgs^ e) {
		bmp = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);
		pictureBox1->Image = bmp;
	}

};
}
```

Приложение С

Файл WebForm.h

```
#pragma once
namespace binaryTree {
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	/// <summary>
	/// Сводка для WebForm
	/// </summary>
	public ref class WebForm : public System::Windows::Forms::Form
	{
	public:
		String^ path;
		WebForm(String^ str)
		{
```

```
			InitializeComponent();
			path = str;
		}

	protected:
		/// <summary>
		/// Освободить все используемые ресурсы.
		/// </summary>
		~WebForm()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;
	private: System::Windows::Forms::OpenFileDialog^ openFileDialog2;
	private: System::Windows::Forms::WebBrowser^ webBrowser1;
	protected:

	private:
		/// <summary>
		/// Обязательная переменная конструктора.
		/// </summary>
		System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
		/// <summary>
		/// Требуемый метод для поддержки конструктора — не изменяйте
		/// содержимое этого метода с помощью редактора кода.
		/// </summary>
		void InitializeComponent(void)
		{
			this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());
			this->openFileDialog2 = (gcnew System::Windows::Forms::OpenFileDialog());
			this->webBrowser1 = (gcnew System::Windows::Forms::WebBrowser());
			this->SuspendLayout();
			//
			// openFileDialog1
			//
			this->openFileDialog1->FileName = L"openFileDialog1";
			//
			// openFileDialog2
			//
			this->openFileDialog2->FileName = L"openFileDialog2";
			//
			// webBrowser1
			//
			this->webBrowser1->Dock = System::Windows::Forms::DockStyle::Fill;
			this->webBrowser1->Location = System::Drawing::Point(0, 0);
			this->webBrowser1->MinimumSize = System::Drawing::Size(20, 20);
			this->webBrowser1->Name = L"webBrowser1";
			this->webBrowser1->Size = System::Drawing::Size(654, 421);
			this->webBrowser1->TabIndex = 0;
			//
			// WebForm
			//
			this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
```

```
			this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
			this->ClientSize = System::Drawing::Size(654, 421);
			this->Controls->Add(this->webBrowser1);
			this->MinimumSize = System::Drawing::Size(502, 316);
			this->Name = L"WebForm";
			this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;
			this->Text = L"HTML файл";
			this->Shown += gcnew System::EventHandler(this, &WebForm::WebForm_Shown);
			this->ResumeLayout(false);
		}
#pragma endregion
	private: System::Void WebForm_Shown(System::Object^ sender, System::EventArgs^ e) {
		if (path == "")
		{
			openFileDialog1->Filter = "Html files (*.html) | *.html|All files (*.*)|*.*";
			if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
				webBrowser1->Navigate(openFileDialog1->FileName);
			}
			else Close();
		}
		else webBrowser1->Navigate(path);
	}
	};
}
```