УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.2

по предмету «Основы алгоритмизации и программирования»

Вариант 15

Выполнила:

Городко К. Е.

Гр. 351005

Проверила:

Данилова Г. В.

Минск 2024

**ЗАДАНИЕ:**

Деревья. Вывести номера вершин, у которых количество потомков в левом поддереве не равно количеству потомков в правом поддереве.

Дерево визуализировать!

**КОД ПРОГРАММЫ DELPHI:**

**Library LIBRARY5\_2;**

Uses

System.SysUtils, Vcl.ExtCtrls, Vcl.Graphics, Vcl.ComCtrls;

Type

PNode = ^TNode;

TNode = Record

Value: Integer;

Right, Left: PNode;

End;

Var

Root: PNode = Nil;

Procedure CreateTree(RootValue: Integer);

Begin

New(Root);

Root.Value := RootValue;

Root.Right := Nil;

Root.Left := Nil;

End;

Procedure FindPlace(Node, NewNode: PNode);

Begin

If NewNode.Value > Node.Value Then

If Node.Right <> Nil Then

FindPlace(Node.Right, NewNode)

Else

Node.Right := NewNode

Else If Node.Left <> Nil Then

FindPlace(Node.Left, NewNode)

Else

Node.Left := NewNode

End;

Procedure AddNode(Value: Integer);

Var

NewNode: PNode;

Begin

New(NewNode);

NewNode.Value := Value;

NewNode.Right := Nil;

NewNode.Left := Nil;

FindPlace(Root, NewNode);

End;

Procedure DestroyNode(Var Node: PNode);

Begin

If Node.Right <> Nil Then

DestroyNode(Node.Right);

If Node.Right = Nil Then

Begin

If Node.Left <> Nil Then

DestroyNode(Node.Left);

If Node.Left = Nil Then

Begin

Dispose(Node);

Node := Nil;

End

End;

End;

Procedure DestroyTree();

Begin

DestroyNode(Root);

End;

Function IsValueInTree(Value: Integer): Boolean;

Var

Temp: PNode;

IsInTree: Boolean;

Begin

IsInTree := False;

If Root <> Nil Then

Begin

Temp := Root;

While (Temp <> Nil) And Not IsInTree Do

Begin

If Value = Temp.Value Then

IsInTree := True

Else If Value > Temp.Value Then

Temp := Temp.Right

Else

Temp := Temp.Left;

End;

End;

IsValueInTree := IsInTree;

End;

Function FindTreeHeight(Node: PNode): Integer;

Var

LeftHeight, RightHeight: Integer;

Begin

If Node = Nil Then

FindTreeHeight := 0

Else

Begin

LeftHeight := FindTreeHeight(Node.Left);

RightHeight := FindTreeHeight(Node.Right);

If RightHeight < LeftHeight Then

FindTreeHeight := LeftHeight + 1

Else

FindTreeHeight := RightHeight + 1;

End;

End;

Function FindNodeHeight(Value: Integer): Integer;

Var

NodeHeight: Integer;

Temp: PNode;

Begin

NodeHeight := 1;

Temp := Root;

While Temp <> Nil Do

Begin

If Value > Temp.Value Then

Temp := Temp.Right

Else

Temp := Temp.Left;

Inc(NodeHeight);

End;

FindNodeHeight := NodeHeight;

End;

///////////////////////////////VISUALISATION/////////////////////////////////////

Type

TCoordinates = Record

X, Y: Integer;

End;

Const

Offset: Array [1 .. 7] Of Integer = (1, 2, 3, 6, 12, 24, 48);

Radius = 20;

Var

RootCoord: TCoordinates = (X: Radius + 5; Y: Radius + 5);

Procedure DrawNode(Value: Integer; Image: TImage; Coord: TCoordinates);

Begin

With Image.Canvas Do

With Coord Do

Begin

Ellipse(X - 20, Y - 20, X + 20, Y + 20);

TextOut((X - 4) - (Length(IntToStr(Value)) - 1) \* 4, Y - 10,

IntToStr(Value));

Draw(0, 0, Image.Picture.Bitmap);

End;

End;

Procedure DrawTree(Node: PNode; ParentCoord: TCoordinates; TreeHeight: Integer;

Image: TImage);

Var

Coord: TCoordinates;

Begin

If Node.Left <> Nil Then

With Coord Do

Begin

X := ParentCoord.X - Radius \* Offset[TreeHeight];

Y := ParentCoord.Y + 60;

With Image.Canvas Do

Begin

MoveTo(ParentCoord.X - Radius, ParentCoord.Y);

LineTo(X, Y - Radius);

DrawNode(Node.Left.Value, Image, Coord);

End;

DrawTree(Node.Left, Coord, TreeHeight - 1, Image);

End;

If Node.Right <> Nil Then

With Coord Do

Begin

X := ParentCoord.X + Radius \* Offset[TreeHeight];

Y := ParentCoord.Y + 60;

With Image.Canvas Do

Begin

MoveTo(ParentCoord.X + Radius, ParentCoord.Y);

LineTo(X, Y - Radius);

DrawNode(Node.Right.Value, Image, Coord);

End;

DrawTree(Node.Right, Coord, TreeHeight - 1, Image);

End;

End;

Procedure VisualizeTree(Image: TImage);

Var

TreeHeight: Integer;

Begin

With Image Do

Begin

Height := 420;

Width := 3850;

Canvas.Brush.Color := Clwhite;

Canvas.Rectangle(0, 0, Width - 1, Height - 1);

End;

TreeHeight := FindTreeHeight(Root);

If TreeHeight > 1 Then

RootCoord.X := Offset[TreeHeight] \* Radius \* 2 + 5;

DrawNode(Root.Value, Image, RootCoord);

DrawTree(Root, RootCoord, TreeHeight, Image);

End;

Procedure WriteNodeToFile(Node: PNode; Offset: String; NodeType: Char;

Var FOut: TextFile);

Begin

Writeln(FOut, Offset, '[', NodeType, '] ', Node.Value);

If Node.Right <> Nil Then

WriteNodeToFile(Node.Right, Offset + ' ', 'R', FOut);

If Node.Left <> Nil Then

WriteNodeToFile(Node.Left, Offset + ' ', 'L', FOut);

End;

Procedure WriteTreeToFile(Var FOut: TextFile);

Begin

WriteNodeToFile(Root, '', '-', FOut);

End;

///////////////////////////////////TASK////////////////////////////////////////

Procedure AddNodeToListView(ListView: TListView;

NodeValue, RightChildren, LeftChildren: Integer);

Var

NewItem: TListItem;

Begin

NewItem := ListView.Items.Add;

NewItem.Caption := IntToStr(NodeValue);

With NewItem.SubItems Do

Begin

Add(IntToStr(LeftChildren));

Add(IntToStr(RightChildren));

End;

End;

Procedure CountChildren(Node: PNode; Var CurCount: Integer);

Begin

If Node = Nil Then

CurCount := 0

Else

Begin

If Node.Right <> Nil Then

Begin

Inc(CurCount);

CountChildren(Node.Right, CurCount);

End;

If Node.Left <> Nil Then

Begin

Inc(CurCount);

CountChildren(Node.Left, CurCount);

End;

End;

End;

Procedure FindUnequalNodes(Node: PNode; ListView: TListView);

Var

RightChildren, LeftChildren: Integer;

Begin

RightChildren := 1;

LeftChildren := 1;

CountChildren(Node.Right, RightChildren);

CountChildren(Node.Left, LeftChildren);

If RightChildren <> LeftChildren Then

AddNodeToListView(ListView, Node.Value, RightChildren, LeftChildren);

If Node.Right <> Nil Then

FindUnequalNodes(Node.Right, ListView);

If Node.Left <> Nil Then

FindUnequalNodes(Node.Left, ListView);

End;

Procedure ShowUnequalNodes(ListView: TListView);

Var

I: Integer;

Begin

For I := ListView.Items.Count - 1 DownTo 0 Do

ListView.Items[I].Delete;

If Root <> Nil Then

FindUnequalNodes(Root, ListView);

End;

{$R \*.res}

Exports CreateTree, AddNode, DestroyTree, IsValueInTree, FindNodeHeight,

VisualizeTree, WriteTreeToFile, ShowUnequalNodes;

Begin

End.

**Unit MainUnit;**

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus,

Vcl.ExtCtrls, Vcl.StdCtrls, Vcl.ComCtrls, InstructionUnit, DevInfoUnit,

AddNodeUnit, Vcl.Imaging.Pngimage, TreeUnit, TaskUnit;

Const

MAXHEIGHT = 7;

MINVALUE = -999;

MAXVALUE = 999;

Type

TError = (Correct, ErrNodeInTree, ErrMaxHeight, ErrIncorrectFileExt,

ErrCantOpenFile, ErrCantSaveFile, ErrIncorrectData);

TMainForm = Class(TForm)

ButtonAddNode: TButton;

ButtonShowTree: TButton;

ButtonTask: TButton;

LabelProgramInfo: TLabel;

MainMenu: TMainMenu;

MMFile: TMenuItem;

MMOpenFile: TMenuItem;

MMSaveFile: TMenuItem;

MMSeparator: TMenuItem;

MMExit: TMenuItem;

MMInstruction: TMenuItem;

MMDevInfo: TMenuItem;

OpenDialog: TOpenDialog;

SaveDialog: TSaveDialog;

ImageTree: TImage;

Procedure ShowErrorMessage(Error: TError);

Function CheckNodeValue(Value: Integer): TError;

Procedure ButtonAddNodeClick(Sender: TObject);

Procedure ButtonShowTreeClick(Sender: TObject);

Procedure ButtonTaskClick(Sender: TObject);

Procedure FormClose(Sender: TObject; Var Action: TCloseAction);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Function FormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean):

Boolean;

Procedure FormResize(Sender: TObject);

Procedure MMDevInfoClick(Sender: TObject);

Procedure MMExitClick(Sender: TObject);

Procedure MMInstructionClick(Sender: TObject);

Procedure MMOpenFileClick(Sender: TObject);

Procedure MMSaveFileClick(Sender: TObject);

End;

Const

ERRORMESSAGES: Array [Low(TError) .. High(TError)] Of PWideChar =

('', 'Узел с таким значением уже есть в дереве!', 'Достигнута максимальная

высота дерева!', 'Файл должен иметь разрешение .txt!', 'Произошла ошибка при

открытии файла!', 'Произошла ошибка при записи в файл!', 'Проверьте

корректность данных!');

Var

MainForm: TMainForm;

IsSaved: Boolean;

IsTreeCreated: Boolean = False;

Implementation

{$R \*.dfm}

Function IsValueInTree(Value: Integer): Boolean; External 'LIBRARY5\_2.dll';

Function FindNodeHeight(Value: Integer): Integer; External 'LIBRARY5\_2.dll';

Procedure DestroyTree(); External 'LIBRARY5\_2.dll';

Procedure WriteTreeToFile(Var FOut: TextFile); External 'LIBRARY5\_2.dll';

Procedure TMainForm.ShowErrorMessage(Error: TError);

Begin

Application.MessageBox(ERRORMESSAGES[Error], 'Ошибка', MB\_ICONERROR);

End;

Function TMainForm.CheckNodeValue(Value: Integer): TError;

Var

Error: TError;

Begin

Error := Correct;

If IsValueInTree(Value) Then

Error := ErrNodeInTree

Else If FindNodeHeight(Value) > MAXHEIGHT Then

Error := ErrMaxHeight;

CheckNodeValue := Error;

End;

Procedure TMainForm.ButtonAddNodeClick(Sender: TObject);

Begin

AddNodeForm.ShowModal;

End;

Procedure TMainForm.ButtonShowTreeClick(Sender: TObject);

Begin

TreeForm.ShowModal;

End;

Procedure TMainForm.ButtonTaskClick(Sender: TObject);

Begin

TaskForm.ShowModal;

End;

Procedure TMainForm.FormClose(Sender: TObject; Var Action: TCloseAction);

Begin

If IsTreeCreated Then

DestroyTree();

End;

Procedure TMainForm.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Var

ButtonSelected: Integer;

Begin

If MMSaveFile.Enabled And Not IsSaved Then

Begin

ButtonSelected := Application.MessageBox('Желаете сохранить дерево в файл?',

'Выход', MB\_YESNOCANCEL + MB\_ICONQUESTION);

If ButtonSelected = MrYes Then

Begin

MMSaveFile.Click;

If Not IsSaved Then

Close;

End

Else If ButtonSelected = MrNo Then

CanClose := True

Else

CanClose := False;

End

Else

Begin

ButtonSelected := Application.MessageBox('Вы уверены, что хотите выйти?',

'Выход', MB\_YESNO + MB\_ICONQUESTION);

If ButtonSelected = MrYes Then

CanClose := True

Else

CanClose := False;

End;

End;

Function TMainForm.FormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean):

Boolean;

Begin

InstructionForm.ShowModal;

CallHelp := False;

End;

Procedure TMainForm.FormResize(Sender: TObject);

Begin

Left := (Screen.Width - Width) Div 2;

Top := (Screen.Height - Height) Div 2;

End;

Procedure TMainForm.MMDevInfoClick(Sender: TObject);

Begin

DevInfoForm.ShowModal;

End;

Procedure TMainForm.MMExitClick(Sender: TObject);

Begin

Close;

End;

Procedure TMainForm.MMInstructionClick(Sender: TObject);

Begin

InstructionForm.ShowModal;

End;

Function CheckFileExtension(Path: String): TError;

Var

Error: TError;

Begin

Error := Correct;

If ExtractFileExt(Path) <> '.txt' Then

Error := ErrIncorrectFileExt;

CheckFileExtension := Error;

End;

Procedure TMainForm.MMOpenFileClick(Sender: TObject);

Var

Value: Integer;

Error: TError;

FIn: TextFile;

Begin

If OpenDialog.Execute Then

Begin

Error := CheckFileExtension(OpenDialog.FileName);

If Error = Correct Then

Try

Try

AssignFile(FIn, OpenDialog.FileName);

Reset(FIn);

If Eof(FIn) Then

Error := ErrCantOpenFile;

If Error = Correct Then

Begin

Read(FIn, Value);

Error := CheckNodeValue(Value);

If Not Eof(FIn) Then

Error := ErrCantOpenFile;

End;

Finally

CloseFile(FIn);

End;

Except

Error := ErrCantOpenFile;

End;

If Error <> Correct Then

ShowErrorMessage(Error)

Else

AddNodeForm.AddValueToTree(Value);

End;

End;

Procedure TMainForm.MMSaveFileClick(Sender: TObject);

Var

Error: TError;

FOut: TextFile;

Begin

If SaveDialog.Execute Then

Begin

Error := CheckFileExtension(SaveDialog.FileName);

If Error = Correct Then

Try

Try

AssignFile(FOut, SaveDialog.FileName);

Rewrite(FOut);

WriteTreeToFile(FOut);

Finally

CloseFile(FOut);

End;

Except

Error := ErrCantSaveFile;

End;

If Error <> Correct Then

ShowErrorMessage(Error)

Else

IsSaved := True;

End;

End;

End.

**Unit AddNodeUnit;**

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus,

Vcl.StdCtrls, InstructionUnit;

Type

TAddNodeForm = Class(TForm)

MainMenu: TMainMenu;

MMInstruction: TMenuItem;

EditValue: TEdit;

LabelValue: TLabel;

ButtonAdd: TButton;

ButtonCancel: TButton;

PopupMenu: TPopupMenu;

Procedure AddValueToTree(Value: Integer);

Procedure ButtonAddClick(Sender: TObject);

Procedure ButtonCancelClick(Sender: TObject);

Procedure EditValueChange(Sender: TObject);

Procedure EditValueKeyPress(Sender: TObject; Var Key: Char);

Procedure FormClose(Sender: TObject; Var Action: TCloseAction);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Procedure FormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Procedure FormShow(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean):

Boolean;

Procedure MMInstructionClick(Sender: TObject);

End;

Var

AddNodeForm: TAddNodeForm;

Implementation

{$R \*.dfm}

Uses MainUnit;

Const

NUMBERS = ['0' .. '9'];

MINUS = '-';

NULL = #0;

BACKSPACE = #8;

Var

IsEditChanged: Boolean;

Procedure AddNode(Value: Integer); External 'LIBRARY5\_2.dll';

Procedure CreateTree(RootValue: Integer); External 'LIBRARY5\_2.dll';

Procedure TAddNodeForm.AddValueToTree(Value: Integer);

Begin

If Not IsTreeCreated Then

Begin

CreateTree(Value);

IsTreeCreated := True;

MainForm.ButtonShowTree.Enabled := True;

MainForm.MMSaveFile.Enabled := True;

End

Else

AddNode(Value);

IsSaved := False;

End;

Procedure TAddNodeForm.ButtonAddClick(Sender: TObject);

Var

Value: Integer;

Error: TError;

Begin

Value := StrToInt(EditValue.Text);

Error := MainForm.CheckNodeValue(Value);

If Error = Correct Then

AddValueToTree(Value)

Else

MainForm.ShowErrorMessage(Error);

IsEditChanged := False;

Close;

End;

Procedure TAddNodeForm.ButtonCancelClick(Sender: TObject);

Begin

Close;

End;

Procedure TAddNodeForm.EditValueChange(Sender: TObject);

Begin

ButtonAdd.Enabled := (EditValue.Text <> '') And (EditValue.Text <> MINUS);

IsEditChanged := ButtonAdd.Enabled;

End;

Procedure TAddNodeForm.EditValueKeyPress(Sender: TObject; Var Key: Char);

Var

SelStart, Len: Integer;

EditText: String;

Begin

SelStart := EditValue.SelStart;

EditText := EditValue.Text;

Len := Length(EditText);

If Not CharInSet(Key, NUMBERS) And Not CharInSet(Key, [BACKSPACE, MINUS]) Then

Key := NULL

Else If (Selstart > 0) And (Key = MINUS) Then

Key := NULL

Else If (SelStart = 0) And (Len > 0) And (EditText[1] = MINUS) And

(Key = MINUS) Then

Key := NULL

Else If (Len > 2) And ((Abs(StrToInt(EditText)) > MAXVALUE Div 10) Or

(Abs(StrToInt(EditText)) = MAXVALUE)) And Not CharInSet(Key,

[BACKSPACE, MINUS]) Then

Key := NULL

Else If (Key = '0') And (Len > 0) And ((SelStart = 0) Or (SelStart = 1) And

(EditText[1] = MINUS)) Then

Key := NULL

Else If (EditText = '0') Then

If (SelStart = Len) And (Key <> BACKSPACE) Then

Key := NULL

Else If (SelStart = 0) And (Key = MINUS) Then

Key := NULL;

End;

Procedure TAddNodeForm.FormClose(Sender: TObject; Var Action: TCloseAction);

Begin

EditValue.Text := '';

End;

Procedure TAddNodeForm.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Var

ButtonSelected: Integer;

Begin

If IsEditChanged Then

Begin

ButtonSelected := Application.MessageBox('Вы уверены, что хотите отменить

добавление узла?', 'Отмена', MB\_YESNO + MB\_ICONQUESTION);

If ButtonSelected = MrYes Then

CanClose := True

Else

CanClose := False;

End

Else

CanClose := True;

End;

Function TAddNodeForm.FormHelp(Command: Word; Data: NativeInt; Var CallHelp:

Boolean): Boolean;

Begin

InstructionForm.ShowModal;

CallHelp := False;

End;

Procedure TAddNodeForm.FormKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close

Else If (Key = 13) And ButtonAdd.Enabled Then

ButtonAdd.Click

Else If Key = VK\_INSERT Then

Key := Ord(NULL);

End;

Procedure TAddNodeForm.FormShow(Sender: TObject);

Begin

IsEditChanged := False;

End;

Procedure TAddNodeForm.MMInstructionClick(Sender: TObject);

Begin

InstructionForm.ShowModal;

End;

End.

**Unit TaskUnit;**

Interface

Uses

Winapi.Windows, System.Classes, Vcl.Forms, Vcl.ComCtrls, Vcl.StdCtrls,

Vcl.Controls;

Type

TTaskForm = Class(TForm)

LabelTask: TLabel;

ListViewNodes: TListView;

ButtonClose: TButton;

Procedure ButtonCloseClick(Sender: TObject);

Procedure FormShow(Sender: TObject);

Procedure FormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

End;

Var

TaskForm: TTaskForm;

Implementation

{$R \*.dfm}

Procedure ShowUnequalNodes(ListView: TListView); External 'LIBRARY5\_2.dll';

Procedure TTaskForm.ButtonCloseClick(Sender: TObject);

Begin

Close;

End;

Procedure TTaskForm.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Procedure TTaskForm.FormShow(Sender: TObject);

Begin

ShowUnequalNodes(ListViewNodes);

End;

End.

**Unit TreeUnit;**

Interface

Uses

Winapi.Windows, System.SysUtils, System.Classes, Vcl.Controls, Vcl.Forms,

Vcl.ExtCtrls, Vcl.StdCtrls, AddNodeUnit, TaskUnit;

Type

TTreeForm = Class(TForm)

ScrollBox: TScrollBox;

ImageTree: TImage;

ButtonAddNode: TButton;

ButtonTask: TButton;

Procedure FormShow(Sender: TObject);

Procedure ButtonAddNodeClick(Sender: TObject);

Procedure ButtonTaskClick(Sender: TObject);

Procedure FormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

End;

Var

TreeForm: TTreeForm;

Implementation

{$R \*.dfm}

Procedure VisualizeTree(Image: TImage); External 'LIBRARY5\_2.dll';

Procedure TTreeForm.ButtonAddNodeClick(Sender: TObject);

Begin

AddNodeForm.ShowModal;

VisualizeTree(ImageTree);

End;

Procedure TTreeForm.ButtonTaskClick(Sender: TObject);

Begin

TaskForm.ShowModal;

End;

Procedure TTreeForm.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Procedure TTreeForm.FormShow(Sender: TObject);

Begin

VisualizeTree(ImageTree);

End;

End.

**КОД ПРОГРАММЫ С++:**

#include <iostream>

#include <fstream>

#include <locale>

#include <Windows.h>

#include <string>

using namespace std;

struct Node {

int value;

Node\* left;

Node\* right;

};

enum Error {correct, errNodeInTree, errMaxHeight, errIncorrectFileExt,

errCantOpenFile, errCantSaveFile, errIncorrectValue};

const string ERRORMESSAGES [] = {"", "Узел с таким значением уже есть в дереве!",

"Достигнута максимальная высота дерева!",

"Файл должен иметь разрешение.txt!", "Произошла ошибка при открытии файла!", "Произошла ошибка при записи в файл!", "Проверьте корректность данных!"};

const int MINVALUE = -999;

const int MAXVALUE = 999;

const int MAXHEIGHT = 7;

void showErrorMessage(Error error) {

cout << ERRORMESSAGES[error] << " Повторите попытку.\n";

}

Node\* createTree(bool& isCreated, int rootValue) {

Node\* root;

root = new Node;

root->value = rootValue;

root->left = nullptr;

root->right = nullptr;

isCreated = true;

return root;

}

void findPlace(Node\* node, Node\* newNode) {

if (newNode->value > node->value)

if (node->right != nullptr)

findPlace(node->right, newNode);

else

node->right = newNode;

else if (node->left != nullptr)

findPlace(node->left, newNode);

else

node->left = newNode;

}

void addNode(int value, Node\* root) {

Node\* newNode;

newNode = new Node;

newNode->value = value;

newNode->left = nullptr;

newNode->right = nullptr;

findPlace(root, newNode);

}

void destroyTree(Node\*& node) {

if (node->right != nullptr)

destroyTree(node->right);

if (node->right == nullptr) {

if (node->left != nullptr)

destroyTree(node->left);

if (node->left == nullptr) {

delete node;

node = nullptr;

}

}

}

bool isValueInTree(int value, Node\* root) {

bool isInTree;

isInTree = false;

if (root != nullptr) {

Node\* temp;

temp = root;

while ((temp != nullptr) && !isInTree) {

if (value == temp->value)

isInTree = true;

else if (value > temp->value)

temp = temp->right;

else

temp = temp->left;

}

}

return isInTree;

}

int findNodeHeight(int value, Node\* root) {

int nodeHeight;

Node\* temp;

nodeHeight = 1;

temp = root;

while (temp != nullptr) {

if (value > temp->value)

temp = temp->right;

else

temp = temp->left;

nodeHeight++;

}

return nodeHeight;

}

void countChildren(Node\* node, int& curCount) {

if (node == nullptr)

curCount = 0;

else {

if (node->right != nullptr) {

curCount++;

countChildren(node->right, curCount);

}

if (node->left != nullptr) {

curCount++;

countChildren(node->left, curCount);

}

}

}

void findUnequalNodes(Node\* node) {

int rightChildren;

int leftChildren;

rightChildren = 1;

leftChildren = 1;

countChildren(node->right, rightChildren);

countChildren(node->left, leftChildren);

if (rightChildren != leftChildren)

cout << node->value << " " << leftChildren << " "

<< rightChildren << "\n";

if (node->right != nullptr)

findUnequalNodes(node->right);

if (node->left != nullptr)

findUnequalNodes(node->left);

}

int inputNum(string outputMessage, const int MIN, const int MAX) {

int num;

Error error;

do {

error = correct;

cout << outputMessage;

cin >> num;

if (cin.get() != '\n') {

cin.clear();

while (cin.get() != '\n');

error = errIncorrectValue;

}

if ((error == correct) && ((num < MIN) || (num > MAX)))

error = errIncorrectValue;

if (error != correct)

showErrorMessage(error);

} while (error != correct);

return num;

}

Error checkNodeValue(int value, Node\* root) {

Error error;

error = correct;

if (isValueInTree(value, root))

error = errNodeInTree;

else if (findNodeHeight(value, root) > MAXHEIGHT)

error = errMaxHeight;

return error;

}

void addValueToTree(Node\*& root, bool& isCreated) {

int value;

Error error;

do {

value = inputNum("Введите значение для добавления в дерево (-999..999):

", MINVALUE, MAXVALUE);

error = checkNodeValue(value, root);

if (error != correct)

showErrorMessage(error);

} while (error != correct);

if (!isCreated)

root = createTree(isCreated, value);

else

addNode(value, root);

cout << "Вершина успешно добавлена!\n";

}

void outputTree(Node\* node, string offset, char nodeType, ostream& out) {

out << offset << "[" << nodeType << "] " << node->value << "\n";

if (node->right != nullptr)

outputTree(node->right, offset + " ", 'R', out);

if (node->left != nullptr)

outputTree(node->left, offset + " ", 'L', out);

}

Error checkFileExtension(string path, Error error) {

if ((error == correct) && (path.substr(path.size() - 4) != ".txt"))

error = errIncorrectFileExt;

return error;

}

string getFileInputPath() {

string path;

Error error;

do {

error = correct;

cout << "Введите путь к файлу:\n";

getline(cin, path, '\n');

ifstream fin(path);

fin.open(path);

if (!fin.is\_open())

error = errCantOpenFile;

error = checkFileExtension(path, error);

fin.close();

if (error != correct)

showErrorMessage(error);

} while (error != correct);

return path;

}

void getNumFromFile(Node\*& root, bool& isCreated) {

int value;

Error error;

string path;

path = getFileInputPath();

do {

error = correct;

ifstream fin(path);

try {

fin >> value;

if (!fin.eof())

error = errCantOpenFile;

else if ((value > MAXVALUE) || (value < MINVALUE))

error = errIncorrectValue;

else if (isCreated)

error = checkNodeValue(value, root);

}

catch (string errorMessage) {

error = errCantOpenFile;

}

fin.close();

if (error != correct) {

showErrorMessage(error);

path = getFileInputPath();

}

} while (error != correct);

if (!isCreated)

root = createTree(isCreated, value);

else

addNode(value, root);

cout << "Данные из файла успешно считаны.\n";

}

string getFileOutputPath() {

string path;

Error error;

do {

error = correct;

cout << "Введите путь к файлу, в который нужно записать результат:\n";

getline(cin, path, '\n');

error = checkFileExtension(path, error);

if (error == correct) {

ofstream fout(path);

fout.open(path);

if (!fout.is\_open())

error = errCantSaveFile;

fout.close();

}

if (error != correct)

showErrorMessage(error);

} while (error != correct);

return path;

}

void writeTreeIntoFile(Node\* root) {

Error error;

string path;

path = getFileOutputPath();

do {

error = correct;

ofstream fout(path);

try {

outputTree(root, "", '-', fout);

}

catch (string errorMessage) {

error = errCantSaveFile;

}

fout.close();

if (error != correct) {

showErrorMessage(error);

path = getFileOutputPath();

}

} while (error != correct);

cout << "Результат записан.\n";

}

void outputInstruction() {

cout << "Инструкция\n\n" << "ИЗМЕНЕНИЕ ДЕРЕВА:\n1. Элементы дерева - целые

числа в диапазоне - 999..999.\n2. Значения у вершин не могут

повторяться.\n3. Максимальная высота дерева - 7.\n\n"

<< "ФАЙЛЫ:\n1. Открываемый / сохраняемый файл должен быть формата

\* .txt.\n2. В открываемом файле должно быть записано только 1 число,

которое добавится в дерево.\n3. При сохранении данные указанного

файла перезаписываются.\n\n"

<< "Разработчик: Городко Ксения\nЛабораторная №5.2\n";

}

void outputProgramInfo() {

cout << "Добро пожаловать в программу для работы с бинарным деревом!\n";

}

void outputMenu() {

cout << "Меню:\n";

cout << "1) Добавить вершину;\n" << "2) Вывести дерево;\n" << "3) Вывести

информацию по заданию;\n" << "4) Открыть файл;\n" << "5) Сохранить

файл;\n" << "6) Вывести инструкцию;\n" << "7) Выход из программы.\n";

cout << "Для выбора пункта меню введите его номер.\n";

}

void returnToMenu() {

cout << "\nНажмите Enter для возвращения в меню...\n";

getchar();

}

void performMenuOption() {

const int ADD = 1;

const int SHOW = 2;

const int DOTASK = 3;

const int OPENFILE = 4;

const int SAVEFILE = 5;

const int SHOWINSTRUCTION = 6;

const int EXIT = 7;

int choice;

bool isCreated;

Node\* root;

choice = 0;

isCreated = false;

root = nullptr;

while (choice != EXIT) {

outputMenu();

choice = inputNum("", ADD, EXIT);

cout << "\n";

if ((choice == ADD) || (choice == OPENFILE)) || (choice > SAVEFILE)

|| isCreated) {

switch (choice) {

case ADD: {

addValueToTree(root, isCreated);

break;

}

case SHOW: {

outputTree(root, "", '-', cout);

break;

}

case DOTASK: {

cout << "Вывести номера вершин, у которых количество

потомков в левом поддереве не равно количеству

потомков в правом поддереве.\nВершина || Левые

потомки || Правые потомки\n";

findUnequalNodes(root);

break;

}

case OPENFILE: {

getNumFromFile(root, isCreated);

break;

}

case SAVEFILE: {

writeTreeIntoFile(root);

break;

}

case SHOWINSTRUCTION: {

outputInstruction();

break;

}

}

}

else

cout << "На данный момент дерева не существует.\n";

if (choice != EXIT)

returnToMenu();

}

if (isCreated)

destroyTree(root);

cout << "Ждем Вашего возвращения...";

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

outputProgramInfo();

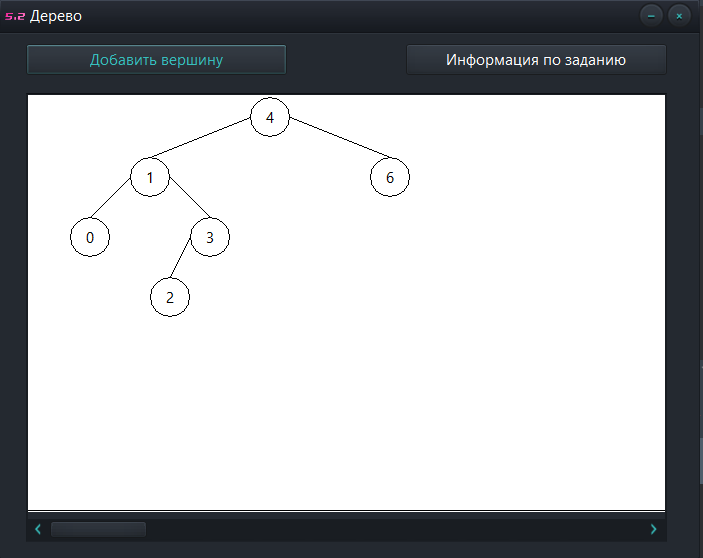
performMenuOption();

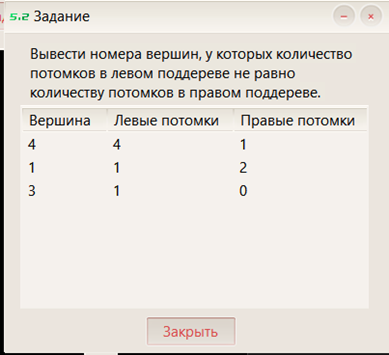
return 0;

}

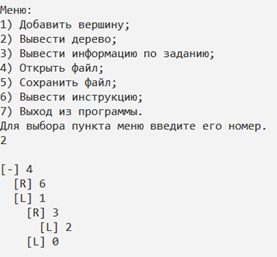
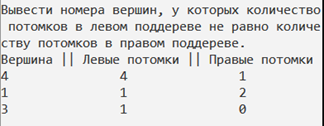
**СКРИНШОТЫ:**

**Delphi:**





**C++****:**

****

**БЛОК-СХЕМА:**

