УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.2

по предмету «Основы алгоритмизации и программирования»

Вариант 4

Выполнил:

Бражалович А. И.

Гр. 351004

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Деревья. Разработать программу работы с бинарным деревом. Программа должна содержать следующие процедуры, вызываемые из меню:

* построение пустого дерева;
* добавление нового элемента;
* удаление указанного поддерева;
* просмотр дерева в следующем порядке: левая ветвь, узел, правая ветвь.

**Код программы Delphi:**

Unit TreeUnit;

Interface

Uses

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

Type

TTree = ^TNode;

TNode = Record

Data: Integer;

Left: TTree;

Right: TTree;

End;

ERROR\_LIST = (CORRECT, ALREADY\_EXIST, NOT\_EXIST, OVER\_DEPTH);

Const

MAX\_DEPTH = 7;

Diametr = 50;

ERRORS: Array [ERROR\_LIST] Of String = ('', 'Узел уже добавлен!',

'Узел не существует', 'Слишком большая глубина!');

DEGREES\_OF\_TWO: Array [0 .. 6] Of Integer = (1, 2, 4, 8, 16, 32, 64);

Var

Depth: Integer = 0;

Procedure MakeTree(Var BinaryTree: TTree; Data: Integer);

Procedure Add(BinaryTree: TTree; Data: Integer; Var Error: ERROR\_LIST);

Procedure Delete(BinaryTree: TTree; Data: Integer; Var Error: ERROR\_LIST);

Procedure Draw(BinarySearchTree: TTree; PaintBox: TPaintBox);

Procedure Clear(Var BinaryTree: TTree);

Procedure DeleteFirstLeaf(Var BinaryTree: TTree; Var Error: ERROR\_LIST);

Procedure TraversalTree(BinaryTree: TTree; StringGrid: TStringGrid;

Var RowCount: Integer);

Implementation

Procedure MakeTree(Var BinaryTree: TTree; Data: Integer);

Begin

New(BinaryTree);

BinaryTree.Left := Nil;

BinaryTree.Right := Nil;

BinaryTree.Data := Data;

Depth := 1;

End;

Procedure TraversalTree(BinaryTree: TTree; StringGrid: TStringGrid;

Var RowCount: Integer);

Begin

If BinaryTree <> Nil Then

Begin

TraversalTree(BinaryTree.Left, StringGrid, RowCount);

StringGrid.Cells[0, RowCount] := IntToStr(BinaryTree.Data);

Inc(RowCount);

StringGrid.RowCount := StringGrid.RowCount + 1;

TraversalTree(BinaryTree.Right, StringGrid, RowCount);

End;

End;

Function FindLeaf(BinaryTree: TTree; Data: Integer): TTree;

Begin

If (BinaryTree = Nil) Or (BinaryTree.Data = Data) Then

FindLeaf := BinaryTree

Else If Data < BinaryTree.Data Then

FindLeaf := FindLeaf(BinaryTree.Left, Data)

Else

FindLeaf := FindLeaf(BinaryTree.Right, Data);

End;

Function FindDepth(BinaryTree: TTree): Integer;

Var

LeftBranchDepth, RightBranchDepth: Integer;

Begin

If BinaryTree = Nil Then

FindDepth := 0

Else

Begin

LeftBranchDepth := FindDepth(BinaryTree.Left);

RightBranchDepth := FindDepth(BinaryTree.Right);

If LeftBranchDepth > RightBranchDepth Then

FindDepth := LeftBranchDepth + 1

Else

FindDepth := RightBranchDepth + 1;

End;

End;

Procedure AddLeaf(BinaryTree: TTree; Data: Integer);

Begin

If (BinaryTree.Left = Nil) And (BinaryTree.Data > Data) Or

(BinaryTree.Right = Nil) And (BinaryTree.Data < Data) Then

Begin

If (BinaryTree.Data > Data) Then

Begin

New(BinaryTree.Left);

BinaryTree := BinaryTree.Left;

End

Else

Begin

New(BinaryTree.Right);

BinaryTree := BinaryTree.Right;

End;

BinaryTree.Data := Data;

BinaryTree.Right := Nil;

BinaryTree.Left := Nil;

End

Else If BinaryTree.Data > Data Then

AddLeaf(BinaryTree.Left, Data)

Else

AddLeaf(BinaryTree.Right, Data)

End;

Function FindMinLeaf(BinaryTree: TTree): TTree;

Begin

While (BinaryTree.Left <> Nil) Do

BinaryTree := BinaryTree.Left;

FindMinLeaf := BinaryTree;

End;

Function DeleteLeaf(BinaryTree: TTree; Data: Integer): TTree;

Begin

If BinaryTree <> Nil Then

Begin

If Data < BinaryTree.Data Then

BinaryTree.Left := DeleteLeaf(BinaryTree.Left, Data)

Else If Data > BinaryTree.Data Then

BinaryTree.Right := DeleteLeaf(BinaryTree.Right, Data)

Else If (BinaryTree.Left <> Nil) And (BinaryTree.Right <> Nil) Then

Begin

BinaryTree.Data := FindMinLeaf(BinaryTree.Right).Data;

BinaryTree.Right := DeleteLeaf(BinaryTree.Right, BinaryTree.Data);

End

Else

Begin

If BinaryTree.Left <> Nil Then

BinaryTree := BinaryTree.Left

Else If BinaryTree.Right <> Nil Then

BinaryTree := BinaryTree.Right

Else

BinaryTree := Nil;

End;

End;

DeleteLeaf := BinaryTree

End;

Procedure DeleteFirstLeaf(Var BinaryTree: TTree; Var Error: ERROR\_LIST);

Begin

Error := CORRECT;

If BinaryTree = Nil Then

Error := NOT\_EXIST

Else

Begin

If (BinaryTree.Left <> Nil) And (BinaryTree.Right <> Nil) Then

Begin

BinaryTree.Data := FindMinLeaf(BinaryTree.Right).Data;

BinaryTree.Right := DeleteLeaf(BinaryTree.Right, BinaryTree.Data);

End

Else

Begin

If BinaryTree.Left <> Nil Then

BinaryTree := BinaryTree.Left

Else If BinaryTree.Right <> Nil Then

BinaryTree := BinaryTree.Right

Else

BinaryTree := Nil;

End;

Depth := FindDepth(BinaryTree);

End;

End;

Procedure Delete(BinaryTree: TTree; Data: Integer; Var Error: ERROR\_LIST);

Begin

Error := CORRECT;

If BinaryTree = Nil Then

Error := NOT\_EXIST

Else

Begin

If FindLeaf(BinaryTree, Data) = Nil Then

Error := NOT\_EXIST

Else

Begin

DeleteLeaf(BinaryTree, Data);

Depth := FindDepth(BinaryTree)

End;

End;

End;

Procedure Add(BinaryTree: TTree; Data: Integer; Var Error: ERROR\_LIST);

Begin

Error := CORRECT;

If FindLeaf(BinaryTree, Data) = Nil Then

Begin

AddLeaf(BinaryTree, Data);

Depth := FindDepth(BinaryTree);

If Depth > MAX\_DEPTH Then

Begin

Delete(BinaryTree, Data, Error);

Error := OVER\_DEPTH;

End;

End

Else

Error := ALREADY\_EXIST;

End;

Procedure Clear(Var BinaryTree: TTree);

Begin

If BinaryTree <> Nil Then

Begin

Clear(BinaryTree.Left);

Clear(BinaryTree.Right);

BinaryTree := Nil;

Dispose(BinaryTree);

End;

End;

Procedure DrawBinarySearchTree(BinarySearchTree: TTree; PaintBox: TPaintBox;

X, Y, Depth: Integer);

Var

Offset: Integer;

Begin

If BinarySearchTree <> Nil Then

Begin

With PaintBox.Canvas Do

Begin

Dec(Depth);

Offset := 0;

If Depth <> 0 Then

Offset := DEGREES\_OF\_TWO[Depth - 1] \* Diametr;

If BinarySearchTree.Left <> Nil Then

Begin

MoveTo(X + Diametr Div 2, Y + Diametr Div 2);

LineTo(X - Offset + Diametr Div 2,

Y + Diametr + Diametr Div 2 + 20);

End;

If BinarySearchTree.Right <> Nil Then

Begin

MoveTo(X + Diametr Div 2, Y + Diametr Div 2);

LineTo(X + Offset + Diametr Div 2,

Y + Diametr + Diametr Div 2 + 20);

End;

Ellipse(X, Y, X + Diametr, Y + Diametr);

TextOut(X + (Diametr - TextWidth(IntToStr(BinarySearchTree.Data)))

Div 2, Y + (Diametr - TextHeight(IntToStr(BinarySearchTree.Data)))

Div 2, IntToStr(BinarySearchTree.Data));

Inc(Y, Diametr + 20);

DrawBinarySearchTree(BinarySearchTree.Left, PaintBox, X - Offset,

Y, Depth);

DrawBinarySearchTree(BinarySearchTree.Right, PaintBox, X + Offset,

Y, Depth);

End;

End;

End;

Procedure Draw(BinarySearchTree: TTree; PaintBox: TPaintBox);

Var

XRoot, YRoot: Integer;

Begin

With PaintBox.Canvas Do

Begin

Pen.Color := ClBlack;

FillRect(ClipRect);

XRoot := 0;

If Depth <> 0 Then

XRoot := (DEGREES\_OF\_TWO[Depth - 1] - 1) \* Diametr;

YRoot := 0;

PaintBox.Width := 2 \* XRoot + Diametr + 20;

PaintBox.Height := (Depth + 20) \* Diametr + 20;

DrawBinarySearchTree(BinarySearchTree, PaintBox, XRoot, YRoot, Depth);

End;

End;

Exports MakeTree;

Begin

End.

Unit MainForm;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus, Instruction, Developer,

Vcl.StdCtrls, Vcl.ExtDlgs, Vcl.Grids, Vcl.ExtCtrls, TreeUnit, AddLeaf,

DeleteLeaf;

Type

TArr = Array Of Integer;

ERRORS\_LIST = (CORRECT, NOT\_READABLE, NOT\_WRITEABLE, FILE\_EMPTY, LINE\_ERR,

NAME\_ERR, NUMBER\_ERR);

TMainTaskForm = Class(TForm)

MainFormMenu: TMainMenu;

FileMenu: TMenuItem;

InstructionMenu: TMenuItem;

DeveloperMenu: TMenuItem;

N1: TMenuItem;

QuitMenu: TMenuItem;

AddButton: TButton;

DeleteButton: TButton;

ScrollBox: TScrollBox;

TreePaintBox: TPaintBox;

TraversalButton: TButton;

TreeGrid: TStringGrid;

Procedure DeveloperMenuClick(Sender: TObject);

Procedure InstructionMenuClick(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Procedure FormDestroy(Sender: TObject);

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

Procedure QuitMenuClick(Sender: TObject);

Procedure TreePaintBoxPaint(Sender: TObject);

Procedure AddButtonClick(Sender: TObject);

Procedure DeleteButtonClick(Sender: TObject);

Procedure DrawGrid(Var ListGrid: TStringGrid);

Procedure TraversalButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

ERRORS: Array [ERRORS\_LIST] Of String = ('', 'Файл закрыт для чтения!',

'Файл закрыт для записи!', 'Файл пуст!', 'Неверное число строк в файле',

'Неверное имя пользователя!', 'Введён неверный номер!');

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

NO\_ZERO\_DIGITS = ['1' .. '9'];

BACKSPACE = #8;

NONE = #0;

MIN\_N = 1;

MAX\_N = 100;

MIN\_X = -100;

MAX\_X = 100;

MAX\_SIGNS = 4;

LINES = 2;

ALPHABET = ['A' .. 'Z', 'a' .. 'z'];

Var

MainTaskForm: TMainTaskForm;

IsEdited: Boolean = False;

Saved: Boolean = True;

BinaryTree: TTree;

Implementation

{$R \*.dfm}

Procedure ClearGrid(Grid: TStringGrid);

Var

I: Integer;

Begin

For I := 1 To Grid.ColCount Do

Begin

Grid.Cells[0, I] := '';

Grid.Cells[1, I] := '';

End;

Grid.RowCount := 1;

Grid.Visible := False;

End;

Procedure TMainTaskForm.AddButtonClick(Sender: TObject);

Var

AddLeafForm: TAddLeafForm;

Begin

AddLeafForm := TAddLeafForm.Create(Self);

AddLeafForm.ShowModal;

AddLeafForm.Free;

ClearGrid(TreeGrid);

End;

Procedure TMainTaskForm.DeleteButtonClick(Sender: TObject);

Var

DeleteLeafForm: TDeleteLeafForm;

Begin

DeleteLeafForm := TDeleteLeafForm.Create(Self);

DeleteLeafForm.ShowModal;

DeleteLeafForm.Free;

End;

Procedure TMainTaskForm.DeveloperMenuClick(Sender: TObject);

Var

DeveloperForm: TDeveloperForm;

Begin

DeveloperForm := TDeveloperForm.Create(Self);

DeveloperForm.ShowModal;

DeveloperForm.Free;

End;

Procedure TMainTaskForm.FormDestroy(Sender: TObject);

Begin

Clear(BinaryTree)

End;

Function TMainTaskForm.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

InstructionMenuClick(Self)

End;

Procedure TMainTaskForm.InstructionMenuClick(Sender: TObject);

Var

InstructionForm: TInstructionForm;

Begin

InstructionForm := TInstructionForm.Create(Self);

InstructionForm.ShowModal;

InstructionForm.Free;

End;

Procedure TMainTaskForm.QuitMenuClick(Sender: TObject);

Begin

Close;

End;

Procedure TMainTaskForm.DrawGrid(Var ListGrid: TStringGrid);

Begin

ListGrid.ColCount := 1;

ListGrid.ColWidths[0] := ListGrid.DefaultColWidth;

If ListGrid.RowCount > 7 Then

Begin

ListGrid.ScrollBars := TScrollStyle.SsVertical;

ListGrid.Height := 250

End

Else

Begin

ListGrid.ScrollBars := TScrollStyle.SsNone;

ListGrid.Height := (ListGrid.DefaultRowHeight) \*

(ListGrid.RowCount - 1);

End;

ListGrid.Width := ListGrid.DefaultColWidth \* 1;

ListGrid.Visible := True;

End;

Procedure TMainTaskForm.TraversalButtonClick(Sender: TObject);

Var

RowCount: Integer;

Begin

RowCount := 0;

ClearGrid(TreeGrid);

TraversalTree(BinaryTree, TreeGrid, RowCount);

DrawGrid(TreeGrid);

End;

Procedure TMainTaskForm.TreePaintBoxPaint(Sender: TObject);

Begin

Draw(BinaryTree, TreePaintBox);

ClearGrid(TreeGrid);

End;

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

Var

Confirmation: Integer;

Begin

Confirmation := Application.MessageBox('Вы действительно хотите выйти?',

'Выход', MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

CanClose := Confirmation = IDYES;

End;

End.

Unit AddLeaf;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TAddLeafForm = Class(TForm)

NameEdit: TLabel;

AddButton: TButton;

CancelButton: TButton;

EnterEdit: TEdit;

Procedure EnterEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

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

Procedure EnterEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure CancelButtonClick(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Procedure EnterEditChange(Sender: TObject);

Procedure AddButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

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

MAX = 1000;

MIN = 1;

Var

AddLeafForm: TAddLeafForm;

Implementation

Uses

MainForm, TreeUnit;

{$R \*.dfm}

Procedure CenterFormOnScreen(AddLeafForm: TAddLeafForm);

Begin

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

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

End;

Procedure CheckComboButtons(Var Key: Char; Edit: TEdit;

Const Chariki: TSysCharSet);

Begin

If (Key = #22) Or ((Key = 'v') And (GetKeyState(VK\_CONTROL) < 0)) Then

Key := #0;

If Not CharInSet(Key, Chariki) And (Key <> #8) Then

Key := #0;

End;

Procedure CheckShftAndArrows(Var Key: Word; Shift: TShiftState);

Begin

If (Key = VK\_INSERT) And (Shift = [SsShift]) Then

Key := 0;

If (Key = VK\_LEFT) Or (Key = VK\_UP) Then

Key := 0

Else If (Key = VK\_RIGHT) Or (Key = VK\_DOWN) Then

Key := 0;

End;

Procedure CheckRange(Var Key: Char; Edit: TEdit; Const MAX, MIN: Integer);

Var

BuffString: String;

Value: Integer;

Begin

Value := 0;

If TryStrToInt(Edit.Text + Key, Value) Then

Begin

If (Value > MAX) Or (Value < MIN) Then

Key := #0;

End;

End;

Procedure TAddLeafForm.AddButtonClick(Sender: TObject);

Var

Error: ERROR\_LIST;

Begin

Error := CORRECT;

If BinaryTree = Nil Then

MakeTree(BinaryTree, StrToInt(EnterEdit.Text))

Else

Add(BinaryTree, StrToInt(EnterEdit.Text), Error);

If Error = CORRECT Then

Begin

MainTaskForm.TreePaintBoxPaint(MainTaskForm.TreePaintBox);

Close;

End

Else

Application.MessageBox(PWideChar(ERRORS[Error]), 'Ошибка',

MB\_OK Or MB\_ICONERROR);

End;

Procedure TAddLeafForm.CancelButtonClick(Sender: TObject);

Begin

EnterEdit.Text := '';

Close;

End;

Procedure TAddLeafForm.EnterEditChange(Sender: TObject);

Begin

If EnterEdit.Text <> '' Then

AddButton.Enabled := True

Else

AddButton.Enabled := False;

End;

Procedure TAddLeafForm.EnterEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TAddLeafForm.EnterEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShftAndArrows(Key, Shift)

End;

Procedure TAddLeafForm.EnterEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, EnterEdit, DIGITS);

CheckRange(Key, EnterEdit, MAX, MIN);

End;

Procedure TAddLeafForm.FormCreate(Sender: TObject);

Begin

AddButton.Enabled := False;

End;

Function TAddLeafForm.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

End;

End.

Unit DeleteLeaf;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TDeleteLeafForm = Class(TForm)

NameEdit: TLabel;

DelButton: TButton;

CancelButton: TButton;

EnterEdit: TEdit;

Procedure EnterEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

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

Procedure EnterEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure CancelButtonClick(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Procedure EnterEditChange(Sender: TObject);

Procedure DelButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

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

MAX = 10000;

MIN = 1;

Var

DeleteLeafForm: TDeleteLeafForm;

Implementation

Uses

MainForm, TreeUnit;

{$R \*.dfm}

Procedure CenterFormOnScreen(DeleteLeafForm: TDeleteLeafForm);

Begin

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

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

End;

Procedure CheckComboButtons(Var Key: Char; Edit: TEdit;

Const Chariki: TSysCharSet);

Begin

If (Key = #22) Or ((Key = 'v') And (GetKeyState(VK\_CONTROL) < 0)) Then

Key := #0;

If Not CharInSet(Key, Chariki) And (Key <> #8) Then

Key := #0;

End;

Procedure CheckShftAndArrows(Var Key: Word; Shift: TShiftState);

Begin

If (Key = VK\_INSERT) And (Shift = [SsShift]) Then

Key := 0;

If (Key = VK\_LEFT) Or (Key = VK\_UP) Then

Key := 0

Else If (Key = VK\_RIGHT) Or (Key = VK\_DOWN) Then

Key := 0;

End;

Procedure CheckRange(Var Key: Char; Edit: TEdit; Const MAX, MIN: Integer);

Var

BuffString: String;

Value: Integer;

Begin

If TryStrToInt(Edit.Text + Key, Value) Then

Begin

If (Value > MAX) Or (Value < MIN) Then

Key := #0;

End;

End;

Procedure TDeleteLeafForm.DelButtonClick(Sender: TObject);

Var

Error: ERROR\_LIST;

Begin

Error := CORRECT;

If (BinaryTree <> Nil) And (BinaryTree.Data = StrToInt(EnterEdit.Text)) Then

DeleteFirstLeaf(BinaryTree, Error)

Else

Delete(BinaryTree, StrToInt(EnterEdit.Text), Error);

If Error = CORRECT Then

Begin

MainTaskForm.TreePaintBoxPaint(MainTaskForm.TreePaintBox);

Close;

End

Else

Application.MessageBox(PWideChar(ERRORS[Error]), 'Ошибка',

MB\_OK Or MB\_ICONERROR);

End;

Procedure TDeleteLeafForm.CancelButtonClick(Sender: TObject);

Begin

EnterEdit.Text := '';

Close;

End;

Procedure TDeleteLeafForm.EnterEditChange(Sender: TObject);

Begin

If EnterEdit.Text <> '' Then

DelButton.Enabled := True

Else

DelButton.Enabled := False;

End;

Procedure TDeleteLeafForm.EnterEditContextPopup(Sender: TObject;

MousePos: TPoint; Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TDeleteLeafForm.EnterEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShftAndArrows(Key, Shift)

End;

Procedure TDeleteLeafForm.EnterEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, EnterEdit, DIGITS);

CheckRange(Key, EnterEdit, MAX, MIN);

End;

Procedure TDeleteLeafForm.FormCreate(Sender: TObject);

Begin

DelButton.Enabled := False;

End;

Function TDeleteLeafForm.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

End;

End.

Unit Instruction;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TInstructionForm = Class(TForm)

InstructionLabel: TLabel;

Procedure CloseButtonClick(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

InstructionForm: TInstructionForm;

Implementation

{$R \*.dfm}

Procedure CenterFormOnScreen(InstructionForm: TInstructionForm);

Begin

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

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

End;

Procedure TInstructionForm.CloseButtonClick(Sender: TObject);

Begin

Close;

End;

Procedure TInstructionForm.FormCreate(Sender: TObject);

Begin

CenterFormOnScreen(Self);

InstructionLabel.Caption :=

'1. Для добавления нового узла дерева нужно нажать на кнопку "Добавить".'

+ #13#10 +

'2. Для удаления нужно нажать на кнопку "Удалить" и написать содержимое узла который'

+ #13#10 + ' вы хотите удалить' + #13#10 +

'3. Чтобы просмотреть отсортированное дерево нужно нажать на "Обойти дерево"'

+ #13#10

End;

End.

Unit Developer;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TDeveloperForm = Class(TForm)

DeveloperLabel: TLabel;

Procedure Button1Click(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

DeveloperForm: TDeveloperForm;

Implementation

{$R \*.dfm}

Procedure TDeveloperForm.Button1Click(Sender: TObject);

Begin

Close;

End;

Procedure CenterDeveloperFormOnScreen(DeveloperForm: TDeveloperForm);

Begin

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

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

End;

Procedure TDeveloperForm.FormCreate(Sender: TObject);

Begin

CenterDeveloperFormOnScreen(Self);

DeveloperLabel.Caption := 'Разработчик: Бражалович Александр Иванович' +

#13#10 + 'Группа: 351005' + #13#10 + 'Tg: @Sunn4es';

DeveloperLabel.Update;

End;

End.

**Код программы Java:**

import java.util.Scanner;

public class Main {

public static int depth = 0;

static final char ROOT\_CHAR = '+',

LEFT\_CHAR = 'L',

RIGHT\_CHAR = 'R';

public static final int MAX\_DEPTH = 10;

public static class TTree {

int data;

TTree left;

TTree right;

TTree(int data) {

this.data = data;

this.left = null;

this.right = null;

}

}

public static TTree binaryTree;

public static TTree findLeaf(TTree binaryTree, int data) {

if ((binaryTree == null) || (binaryTree.data == data)) {

return binaryTree;

} else if (data < binaryTree.data) {

return findLeaf(binaryTree.left, data);

} else {

return findLeaf(binaryTree.right, data);

}

}

public static int findDepth(TTree binaryTree) {

int leftBranchDepth;

int rightBranchDepth;

if (binaryTree == null) {

return 0;

} else {

leftBranchDepth = findDepth(binaryTree.left);

rightBranchDepth = findDepth(binaryTree.right);

if (leftBranchDepth > rightBranchDepth) {

return leftBranchDepth + 1;

} else {

return rightBranchDepth + 1;

}

}

}

public static void addLeaf (TTree binaryTree, int data) {

if(((binaryTree.left == null) && (binaryTree.data > data)) || ((binaryTree.right == null) && (binaryTree.data < data))) {

if (binaryTree.data > data) {

binaryTree.left = new TTree(data);

} else {

binaryTree.right = new TTree(data);

}

} else if (binaryTree.data > data) {

addLeaf(binaryTree.left, data);

} else {

addLeaf(binaryTree.right, data);

}

}

public static TTree findMinLeaf (TTree binaryTree) {

while (binaryTree.left != null) {

binaryTree = binaryTree.left;

}

return binaryTree;

}

public static TTree deleteLeaf(TTree binaryTree, int data) {

if (binaryTree != null) {

if (data < binaryTree.data) {

binaryTree.left = deleteLeaf(binaryTree.left, data);

} else if (data > binaryTree.data) {

binaryTree.right = deleteLeaf(binaryTree.right, data);

} else if (binaryTree.left != null && binaryTree.right != null) {

binaryTree.data = findMinLeaf(binaryTree.right).data;

binaryTree.right = deleteLeaf(binaryTree.right, binaryTree.data);

} else {

if (binaryTree.left != null) {

binaryTree = binaryTree.left;

} else if (binaryTree.right != null) {

binaryTree = binaryTree.right;

} else {

binaryTree = null;

}

}

}

return binaryTree;

}

public static void deleteFirstLeaf() {

if (binaryTree == null) {

printError(ErrCode.NOT\_EXIST);

} else {

if (binaryTree.left != null && binaryTree.right != null) {

binaryTree.data = findMinLeaf(binaryTree.right).data;

binaryTree.right = deleteLeaf(binaryTree.right, binaryTree.data);

} else {

if (binaryTree.left != null) {

binaryTree = binaryTree.left;

} else if (binaryTree.right != null) {

binaryTree = binaryTree.right;

} else {

binaryTree = null;

}

}

depth = findDepth(binaryTree);

}

}

public static void delete(TTree binaryTree, int data) {

if (binaryTree == null) {

printError(ErrCode.NOT\_EXIST);

} else {

if (findLeaf(binaryTree, data) == null) {

printError(ErrCode.NOT\_EXIST);

} else {

deleteLeaf(binaryTree, data);

depth = findDepth(binaryTree);

}

}

}

public static void add(TTree binaryTree, int data) {

if (findLeaf(binaryTree, data) == null) {

addLeaf(binaryTree, data);

int depth = findDepth(binaryTree);

if (depth > MAX\_DEPTH) {

delete(binaryTree, data);

printError(ErrCode.OVER\_DEPTH);

}

} else {

printError(ErrCode.ALREADY\_EXIST);

}

}

public static void makeTree(int data) {

binaryTree = new TTree(data);

binaryTree.left = null;

binaryTree.right = null;

binaryTree.data = data;

depth = 1;

}

static void printTree(TTree binaryTree, int layer, char side) {

if (binaryTree.right != null)

printTree(binaryTree.right, layer + 1, RIGHT\_CHAR);

for (int i = 0; i < layer; i++)

System.out.print(" ");

System.out.println("(" + side + ")" + binaryTree.data);

if (binaryTree.left != null)

printTree(binaryTree.left, layer + 1, LEFT\_CHAR);

}

enum ErrCode {

CORRECT,

ALREADY\_EXIST,

NOT\_EXIST,

OVER\_DEPTH,

INPUT\_ERR,

RANGE\_ERR;

}

static final String[] ERRORS = {"",

"Узел уже добавлен!",

"Узел не существует!",

"Слишком большая глубина!",

"Проверьте корректность ввода!",

"Значение не входит в диапазон!"

};

static final String INSTRUCTION = "\nДанная программа реализует бинарное дерево с возможностью прямого обхода.\n" + "\n1. Элементы в дереве не могут повторяться.\n" +

"2. Элементы в диапазоне от 1 до 1000.\n";

enum ChooseAction {

addToTree("Добавить узел"),

deleteFromTree("Удалить узел"),

printTree("Визуализация дерева"),

traversalTree("Обойти дерево в порядке \"левая ветвь, узел, правая ветвь\""),

exitProg("Заверишть программу");

private final String info;

ChooseAction (String inf) {

this.info = inf;

}

private String getInf(){

return this.ordinal() + ") " + this.info;

}

}

public static void addLeafToTree (Scanner input) {

System.out.println("Введите новый узел для дерева:");

int data = getNumConsole(input, 0, 1000);

if (binaryTree == null) {

makeTree(data);

} else {

add(binaryTree, data);

}

}

public static void deleteLeafFromTree (Scanner input) {

System.out.println("Введите номер узла который хотите удалить:");

int data = getNumConsole(input, 0, 1000);

if ((binaryTree != null) && (binaryTree.data == data)) {

deleteFirstLeaf();

} else {

delete(binaryTree,data);

}

}

public static String traversalTree (TTree binaryTree) {

String result = "";

if (binaryTree != null) {

result += traversalTree(binaryTree.left);

result += binaryTree.data + " ";

result += traversalTree(binaryTree.right);

}

return result;

}

public static boolean doMenu (Scanner input) {

boolean close = false;

System.out.print("Введите действие: ");

ChooseAction option = getChoice(input);

System.out.println();

switch (option) {

case addToTree -> {

addLeafToTree(input);

}

case deleteFromTree -> {

deleteLeafFromTree(input);

}

case printTree -> {

printTree(binaryTree, 0, ROOT\_CHAR);

}

case traversalTree -> {

System.out.print("Обход дерева: ");

System.out.println(traversalTree(binaryTree));

}

case exitProg -> {

close = true;

}

}

return close;

}

static void printMenu() {

ChooseAction[] choices = ChooseAction.values();

for (ChooseAction choice : choices) {

System.out.println(choice.getInf());

}

}

public static void printError (ErrCode error) {

System.out.println("\n" + ERRORS[error.ordinal()] + "\nПовторите попытку\n");

}

public static ErrCode readOneNum(Scanner inputScanner, int[] numberArr, final int MIN, final int MAX) {

int number = 0;

ErrCode error;

error = ErrCode.CORRECT;

try {

number = Integer.parseInt(inputScanner.nextLine());

} catch (NumberFormatException e) {

error = ErrCode.INPUT\_ERR;

}

if (error == ErrCode.CORRECT && ((number < MIN) || (number > MAX)))

error = ErrCode.RANGE\_ERR;

numberArr[0] = error == ErrCode.CORRECT ? number : 0;

return error;

}

static int getNumConsole(Scanner input, final int MIN, final int MAX) {

ErrCode err;

int[] numberArr = {0};

do {

err = readOneNum(input, numberArr, MIN, MAX);

if (err != ErrCode.CORRECT) {

System.out.printf(ERRORS[err.ordinal()], MIN, MAX);

System.out.println("\nВведите снова");

}

} while (err != ErrCode.CORRECT);

return numberArr[0];

}

static ChooseAction getChoice(Scanner input) {

int choice;

int maxChoice = ChooseAction.values().length - 1;

choice = getNumConsole(input, 0, maxChoice);

return ChooseAction.values()[choice];

}

public static void main(String[] args) {

boolean isExit;

Scanner input = new Scanner(System.in);

System.out.println(INSTRUCTION);

do {

printMenu();

System.out.println();

isExit = doMenu(input);

} while (!isExit);

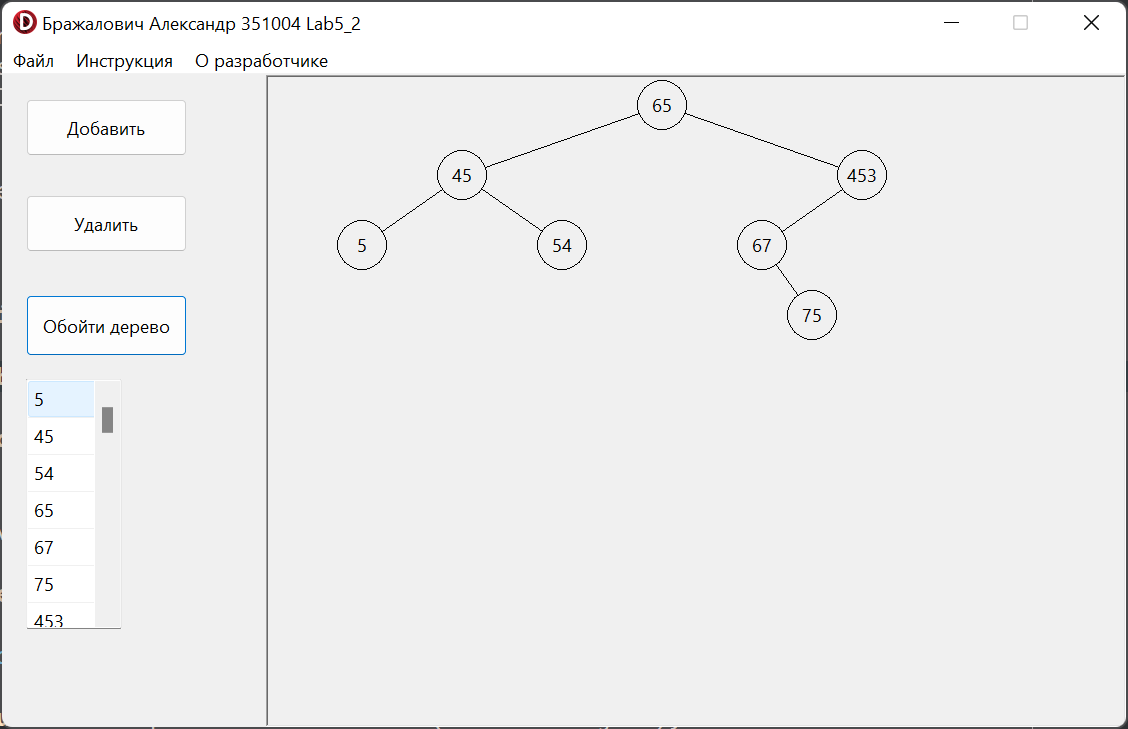
input.close();

}

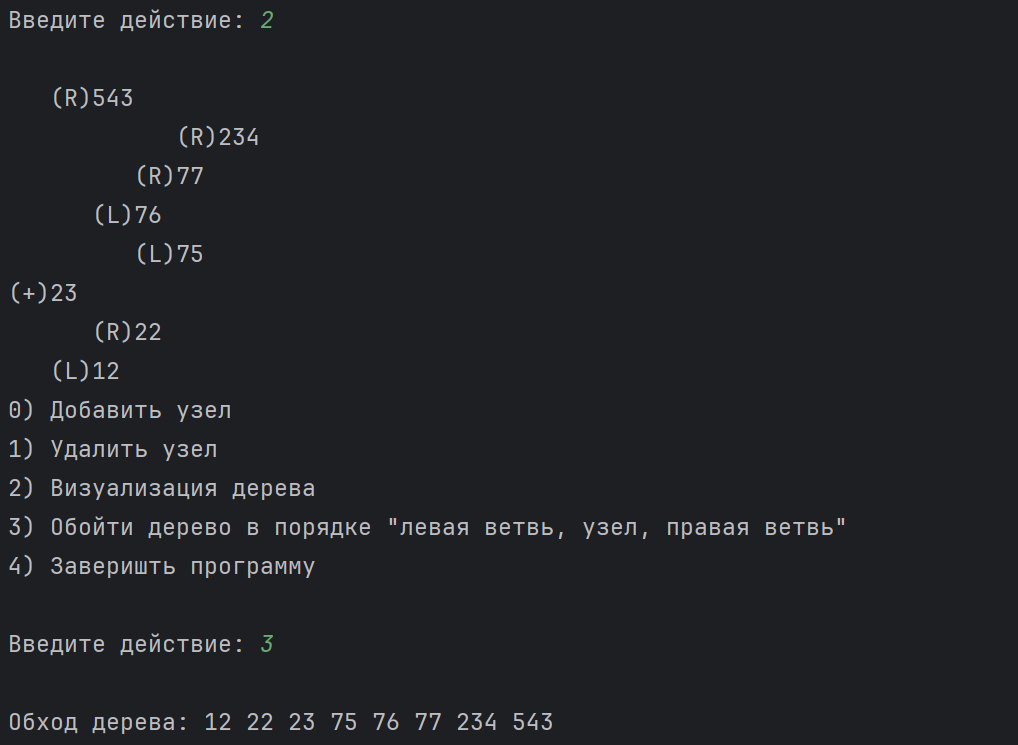
}

**Скриншоты:**

**Delphi:**

****

**Java:**

****

**Блок-схема:**



