УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №6.2

по предмету «Основы алгоритмизации и программирования»

Вариант 6

Выполнил:

Галуха П. А.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

**Код программы Delphi:**

Unit MainUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls,

Vcl.Imaging.pngimage, FrontEndUnit, BackEndUnit, Vcl.ComCtrls;

Const

BOARD\_CELLS\_AMOUNT = 8;

Type

TMainForm = Class(TForm)

BoardImage: TImage;

HorseFigureImage: TImage;

StartButton: TButton;

HorseFigureTimer: TTimer;

SpeedTrackBar: TTrackBar;

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

Procedure FormResize(Sender: TObject);

Procedure HorseFigureImageMouseDown(Sender: TObject; Button: TMouseButton; Shift:

TShiftState; X, Y: Integer);

Procedure HorseFigureImageMouseMove(Sender: TObject; Shift: TShiftState; X, Y:

Integer);

Procedure HorseFigureImageMouseUp(Sender: TObject; Button: TMouseButton; Shift:

TShiftState; X, Y: Integer);

Procedure StartButtonClick(Sender: TObject);

Procedure HorseFigureTimerTimer(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

MainForm: TMainForm;

CellSize, BorderSize: Integer;

CurrentCol: Integer = 0;

CurrentRow: Integer = 0;

Board: Array [0..BOARD\_CELLS\_AMOUNT - 1, 0..BOARD\_CELLS\_AMOUNT - 1] Of Integer;

MoveCount: Integer;

Implementation

{$R \*.dfm}

Var

IsDragging: Boolean;

StartX, StartY: Integer;

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

Begin

CallHelp := False;

FormHelp := False;

End;

Procedure TMainForm.FormResize(Sender: TObject);

Begin

DrawBoard(BoardImage);

DrawHorse(BoardImage, HorseFigureImage, CurrentCol, CurrentRow);

DrawButton(BoardImage, StartButton);

DrawSpeedBar(BoardImage, SpeedTrackBar);

HorseFigureTimer.Enabled := False;

End;

Procedure TMainForm.HorseFigureImageMouseDown(Sender: TObject; Button: TMouseButton; Shift:

TShiftState; X, Y: Integer);

Begin

If Button = mbLeft Then

Begin

DrawBoard(BoardImage);

HorseFigureTimer.Enabled := False;

IsDragging := True;

StartX := X;

StartY := Y;

End;

End;

Procedure TMainForm.HorseFigureImageMouseMove(Sender: TObject; Shift: TShiftState; X, Y:

Integer);

begin

If IsDragging Then

Begin

HorseFigureImage.Left := HorseFigureImage.Left + (X - StartX);

HorseFigureImage.Top := HorseFigureImage.Top + (Y - StartY);

End;

End;

Procedure TMainForm.HorseFigureImageMouseUp(Sender: TObject; Button: TMouseButton; Shift:

TShiftState; X, Y: Integer);

Begin

If Button = mbLeft Then

Begin

IsDragging := False;

CurrentCol := Round((HorseFigureImage.Left - BoardImage.Left - BorderSize) / CellSize);

CurrentRow := Round((HorseFigureImage.Top - BoardImage.Top - BorderSize) / CellSize);

If CurrentCol < 0 Then

CurrentCol := 0

Else If CurrentCol > BOARD\_CELLS\_AMOUNT - 1 Then

CurrentCol := BOARD\_CELLS\_AMOUNT - 1;

If CurrentRow < 0 Then

CurrentRow := 0

Else If CurrentRow > BOARD\_CELLS\_AMOUNT - 1 Then

CurrentRow := BOARD\_CELLS\_AMOUNT - 1;

DrawHorse(BoardImage, HorseFigureImage, CurrentCol, CurrentRow);

End;

End;

Procedure TMainForm.StartButtonClick(Sender: TObject);

Begin

HorseFigureTimer.Enabled := False;

DrawBoard(BoardImage);

MoveCount := 1;

CalcMoves();

HorseFigureTimer.Enabled := True;

End;

Procedure TMainForm.HorseFigureTimerTimer(Sender: TObject);

Begin

HorseFigureTimer.Interval := SpeedTrackBar.Position \* 100;

DrawHorseStep(BoardImage, HorseFigureImage);

CurrentCol := Round((HorseFigureImage.Left - BoardImage.Left - BorderSize) / CellSize);

CurrentRow := Round((HorseFigureImage.Top - BoardImage.Top - BorderSize) / CellSize);

If MoveCount = BOARD\_CELLS\_AMOUNT \* BOARD\_CELLS\_AMOUNT + 1 Then

HorseFigureTimer.Enabled := False;

End;

End.

Unit FrontEndUnit;

Interface

Uses

System.SysUtils, Vcl.Graphics, Vcl.ExtCtrls, Vcl.StdCtrls, Vcl.Imaging.pngimage,

Vcl.ComCtrls;

Procedure DrawBoard(BoardImage: TImage);

Procedure DrawHorse(BoardImage, HorseFigureImage: TImage; Col, Row: Integer);

Procedure DrawButton(BoardImage: TImage; StartButton: TButton);

Procedure DrawSpeedBar(BoardImage: TImage; SpeedTrackBar: TTrackBar);

Procedure DrawHorseStep(BoardImage, HorseFigureImage: TImage);

Implementation

Uses MainUnit;

Const

BASED\_CLIENT\_WIDTH : Integer = 800;

BASED\_CLIENT\_HEIGHT : Integer = 600;

BASED\_FORM\_MARGINS\_TOP\_BOTTOM : Integer = 25;

BASED\_BOARD\_SIZE : Integer = 450;

BASED\_BOARD\_FONT\_SIZE : Integer = 10;

BASED\_BOARD\_BORDER\_SIZE : Integer = 10;

BASED\_START\_BUTTON\_WIDTH : Integer = 200;

BASED\_START\_BUTTON\_HEIGHT : Integer = 50;

BASED\_START\_BUTTON\_FONT\_SIZE : Integer = 20;

BOARD\_BRUSH\_BORDER\_COLOR: TColor = $1B60A5;

BOARD\_BRUSH\_LIGHT\_COLOR: TColor = $9ECEFF;

BOARD\_BRUSH\_DARK\_COLOR: TColor = $478CD2;

Procedure ChangeBrushColor(BoardImage: TImage);

Begin

With BoardImage.Canvas.Brush Do

Begin

If Color = BOARD\_BRUSH\_LIGHT\_COLOR Then

Color := BOARD\_BRUSH\_DARK\_COLOR

Else

Color := BOARD\_BRUSH\_LIGHT\_COLOR;

End;

End;

Procedure DrawBoard(BoardImage: TImage);

Var

BoardSize, Col, Row: Integer;

X, Y: Integer;

Begin

With BoardImage Do

Begin

If (MainForm.ClientWidth / MainForm.ClientHeight) < (BASED\_CLIENT\_WIDTH /

BASED\_CLIENT\_HEIGHT) Then

Begin

BoardSize := BASED\_BOARD\_SIZE \* MainForm.ClientWidth Div BASED\_CLIENT\_WIDTH;

BorderSize := BASED\_BOARD\_BORDER\_SIZE \* MainForm.ClientWidth Div

BASED\_CLIENT\_WIDTH;

End

Else

Begin

BoardSize := BASED\_BOARD\_SIZE \* MainForm.ClientHeight Div BASED\_CLIENT\_HEIGHT;

BorderSize := BASED\_BOARD\_BORDER\_SIZE \* MainForm.ClientHeight Div

BASED\_CLIENT\_HEIGHT;

End;

CellSize := (BoardSize - 2 \* BorderSize) Div BOARD\_CELLS\_AMOUNT;

Width := BoardSize;

Height := BoardSize;

Left := (MainForm.ClientWidth - Width) Div 2;

Top := BASED\_FORM\_MARGINS\_TOP\_BOTTOM \* MainForm.ClientHeight Div BASED\_CLIENT\_HEIGHT;

End;

With BoardImage.Canvas Do

Begin

Brush.Color := BOARD\_BRUSH\_BORDER\_COLOR;

Rectangle(0, 0, BoardSize, BoardSize);

Brush.Color := BOARD\_BRUSH\_LIGHT\_COLOR;

For Col := 0 To BOARD\_CELLS\_AMOUNT - 1 Do

Begin

X := Col \* CellSize + BorderSize;

For Row := 0 To BOARD\_CELLS\_AMOUNT - 1 Do

Begin

Y := Row \* CellSize + BorderSize;

Rectangle(X, Y, X + CellSize, Y + CellSize);

ChangeBrushColor(BoardImage);

End;

ChangeBrushColor(BoardImage);

End;

Brush.Style := bsClear;

Font.Size := BASED\_BOARD\_FONT\_SIZE \* MainForm.ClientWidth Div BASED\_CLIENT\_WIDTH;

End;

End;

Procedure DrawHorse(BoardImage, HorseFigureImage: TImage; Col, Row: Integer);

Begin

With HorseFigureImage Do

Begin

Width := CellSize;

Height := CellSize;

Left := BoardImage.Left + BorderSize + Col \* CellSize;

Top := BoardImage.Top + BorderSize + Row \* CellSize;

End;

End;

Procedure DrawButton(BoardImage: TImage; StartButton: TButton);

Begin

With StartButton Do

Begin

Width := BASED\_START\_BUTTON\_WIDTH \* MainForm.ClientWidth Div BASED\_CLIENT\_WIDTH;

Height := BASED\_START\_BUTTON\_HEIGHT \* MainForm.ClientHeight Div BASED\_CLIENT\_HEIGHT;

Left := (MainForm.ClientWidth - Width) Div 2;

Top := MainForm.ClientHeight - BASED\_FORM\_MARGINS\_TOP\_BOTTOM \* MainForm.ClientHeight

Div BASED\_CLIENT\_HEIGHT - Height;

If (MainForm.ClientWidth / MainForm.ClientHeight) < (BASED\_CLIENT\_WIDTH /

BASED\_CLIENT\_HEIGHT) Then

Font.Size := BASED\_START\_BUTTON\_FONT\_SIZE \* MainForm.ClientWidth Div

BASED\_CLIENT\_WIDTH

Else

Font.Size := BASED\_START\_BUTTON\_FONT\_SIZE \* MainForm.ClientHeight Div

BASED\_CLIENT\_HEIGHT;

End;

End;

Procedure DrawSpeedBar(BoardImage: TImage; SpeedTrackBar: TTrackBar);

Begin

With SpeedTrackBar Do

Begin

Height := BoardImage.Height;

Left := BoardImage.Left + BoardImage.Width;

Top := BoardImage.Top;

End;

End;

Procedure DrawHorseStep(BoardImage, HorseFigureImage: TImage);

Var

Col, Row: Integer;

Begin

For Col := 0 To BOARD\_CELLS\_AMOUNT - 1 Do

For Row := 0 To BOARD\_CELLS\_AMOUNT - 1 Do

If Board[Col, Row] = MoveCount Then

Begin

DrawHorse(BoardImage, HorseFigureImage, Col, Row);

With BoardImage.Canvas Do

TextOut(Col \* CellSize + (CellSize - TextWidth(IntToStr(MoveCount))) Div 2

+ BorderSize, Row \* CellSize + (CellSize –

TextHeight(IntToStr(MoveCount))) Div 2 + BorderSize,

IntToStr(MoveCount));

Inc(MoveCount);

Exit;

End;

End;

End.

Unit BackEndUnit;

Interface

Procedure CalcMoves();

Implementation

Uses MainUnit;

Type

TPosition = Record

Col, Row: Integer;

End;

Const

Moves: Array [1 .. 8] Of TPosition = ((Col: 2; Row: 1), (Col: 1; Row: 2),

(Col: -1; Row: 2), (Col: -2; Row: 1),

(Col: -2; Row: -1), (Col: -1; Row: -2),

(Col: 1; Row: -2), (Col: 2; Row: -1));

Procedure InitializeBoard();

Var

Col, Row: Integer;

Begin

For Col := 0 To BOARD\_CELLS\_AMOUNT - 1 Do

For Row := 0 To BOARD\_CELLS\_AMOUNT - 1 Do

Board[Col, Row] := 0;

End;

Function IsInsideBoard(Col, Row: Integer): Boolean;

Begin

Result := (Col >= 0) And (Col < BOARD\_CELLS\_AMOUNT) And (Row >= 0) And (Row <

BOARD\_CELLS\_AMOUNT);

End;

Function IsNotVisited(Col, Row: Integer): Boolean;

Begin

Result := Board[Col, Row] = 0;

End;

Function CountAvailableMoves(Col, Row: Integer): Integer;

Var

MoveIndex, NextCol, NextRow: Integer;

Begin

Result := 0;

For MoveIndex := Low(Moves) To High(Moves) Do

Begin

NextCol := Col + Moves[MoveIndex].Col;

NextRow := Row + Moves[MoveIndex].Row;

If IsInsideBoard(NextCol, NextRow) And IsNotVisited(NextCol, NextRow) Then

Inc(Result);

End;

End;

Function FindNextOptimalMove(Col, Row: Integer): TPosition;

Var

MoveIndex, NextCol, NextRow, MinMoves, MinMovesIndex: Integer;

Begin

MinMoves := Length(Moves) + 1;

MinMovesIndex := -1;

For MoveIndex := Low(Moves) To High(Moves) Do

Begin

NextCol := Col + Moves[MoveIndex].Col;

NextRow := Row + Moves[MoveIndex].Row;

If IsInsideBoard(NextCol, NextRow) And IsNotVisited(NextCol, NextRow) Then

Begin

If CountAvailableMoves(NextCol, NextRow) < MinMoves Then

Begin

MinMoves := CountAvailableMoves(NextCol, NextRow);

MinMovesIndex := MoveIndex;

End;

End;

End;

NextCol := Col + Moves[MinMovesIndex].Col;

NextRow := Row + Moves[MinMovesIndex].Row;

Result.Col := NextCol;

Result.Row := NextRow;

End;

Procedure KnightTour(Col, Row, MoveCount: Integer);

Var

NextPosition: TPosition;

NextCol, NextRow: Integer;

Begin

Board[Col, Row] := MoveCount;

If MoveCount = BOARD\_CELLS\_AMOUNT \* BOARD\_CELLS\_AMOUNT Then

Exit;

NextPosition := FindNextOptimalMove(Col, Row);

NextCol := NextPosition.Col;

NextRow := NextPosition.Row;

KnightTour(NextCol, NextRow, MoveCount + 1);

End;

Procedure CalcMoves();

Begin

InitializeBoard();

KnightTour(CurrentCol, CurrentRow, 1);

End;

End.

**Код программы C#:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace Lab62

{

internal class Program

{

internal const int BOARD\_SIZE = 8;

internal struct Position

{

internal int col, row;

}

internal static readonly Position[] MOVES = {

new Position {col = -2, row = -1},

new Position {col = -2, row = 1},

new Position {col = -1, row = -2},

new Position {col = -1, row = 2},

new Position {col = 1, row = -2},

new Position {col = 1, row = 2},

new Position {col = 2, row = -1},

new Position {col = 2, row = 1}

};

internal static int[,] board = new int[BOARD\_SIZE, BOARD\_SIZE];

public enum ERRORS\_CODE

{

CORRECT,

INCORRECT\_NUM

}

static readonly string[] ERRORS = new string[]

{

"",

"Некорректное число!",

};

static void InitializeBoard()

{

for (int col = 0; col < BOARD\_SIZE; col++)

for (int row = 0; row < BOARD\_SIZE; row++)

board[col, row] = 0;

}

static bool isInsideBoard(int col, int row)

{

return col >= 0 && col < BOARD\_SIZE && row >= 0 && row < BOARD\_SIZE;

}

static bool isNotVisited(int col, int row)

{

return board[col, row] == 0;

}

static int CountAvailableMoves(int col, int row)

{

int nextCol, nextRow, count = 0;

for (int moveIndex = 0; moveIndex < MOVES.Length; moveIndex++)

{

nextCol = col + MOVES[moveIndex].col;

nextRow = row + MOVES[moveIndex].row;

if (isInsideBoard(nextCol, nextRow) && isNotVisited(nextCol, nextRow))

count++;

}

return count;

}

static Position FindNextOptimalMove(int col, int row)

{

int nextCol, nextRow, minMoves = MOVES.Length + 1, minMovesIndex = -1;

Position optimalMove;

for (int moveIndex = 0; moveIndex < MOVES.Length; moveIndex++)

{

nextCol = col + MOVES[moveIndex].col;

nextRow = row + MOVES[moveIndex].row;

if (isInsideBoard(nextCol, nextRow) && isNotVisited(nextCol, nextRow))

if (CountAvailableMoves(nextCol, nextRow) < minMoves)

{

minMoves = CountAvailableMoves(nextCol, nextRow);

minMovesIndex = moveIndex;

}

}

nextCol = col + MOVES[minMovesIndex].col;

nextRow = row + MOVES[minMovesIndex].row;

optimalMove.col = nextCol;

optimalMove.row = nextRow;

return optimalMove;

}

static void PrintBoard()

{

Console.Clear();

for (int i = 0; i < 41; i++)

Console.Write("-");

Console.WriteLine();

for (int col = 0; col < BOARD\_SIZE; col++)

{

for (int row = 0; row < BOARD\_SIZE; row++)

if (board[row, col] == 0) Console.Write("| ");

else Console.Write("| {0, 2} ", board[row, col]);

Console.WriteLine("|");

for (int i = 0; i < 41; i++)

Console.Write("-");

Console.WriteLine();

}

Thread.Sleep(500);

}

static void KnightTour(int col, int row, int moveCount)

{

Position nextPosition;

int nextCol, nextRow;

board[col, row] = moveCount;

if (moveCount != BOARD\_SIZE \* BOARD\_SIZE)

{

nextPosition = FindNextOptimalMove(col, row);

nextCol = nextPosition.col;

nextRow = nextPosition.row;

PrintBoard();

KnightTour(nextCol, nextRow, moveCount + 1);

}

else

PrintBoard();

}

static void PrintTask()

{

Console.WriteLine("Данная программа обходит конём все клетки шахматной доски 8x8.");

}

static int ReadNumWithinRange(int borderBottom, int borderTop, ERRORS\_CODE patentialError)

{

ERRORS\_CODE error;

int option = 1;

do

{

error = ERRORS\_CODE.CORRECT;

try

{

option = int.Parse(Console.ReadLine());

}

catch

{

error = patentialError;

}

if ((error == ERRORS\_CODE.CORRECT) && ((option < borderBottom) || (option > borderTop)))

error = patentialError;

if (error != ERRORS\_CODE.CORRECT)

{

Console.Error.WriteLine(ERRORS[(int)error]);

Console.Write("Попробуйте снова: ");

}

} while (error != ERRORS\_CODE.CORRECT);

return option;

}

static void Main(string[] args)

{

PrintTask();

Console.Write("Введите колонку, в которой вы хотите разместить коня[1..8]: ");

int currentCol = ReadNumWithinRange(1, 8, ERRORS\_CODE.INCORRECT\_NUM);

Console.Write("Введите ряд, в который вы хотите разместить коня[1..8]: ");

int currentRow = ReadNumWithinRange(1, 8, ERRORS\_CODE.INCORRECT\_NUM);

InitializeBoard();

KnightTour(currentCol - 1, currentRow - 1, 1);

Console.ReadLine();

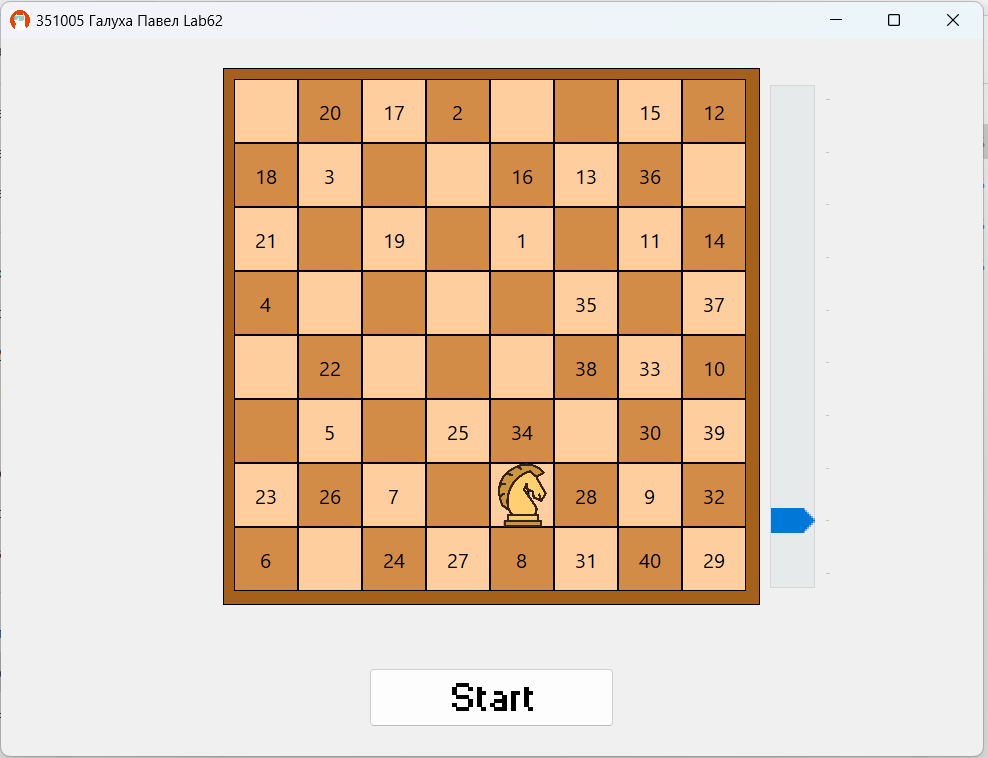
}

}

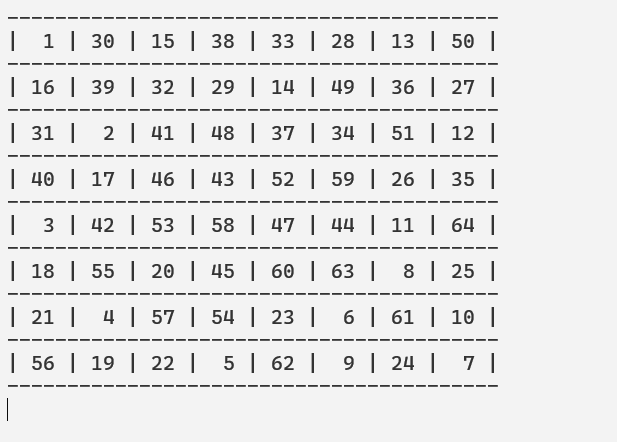
}

**Скриншоты**

**Delphi:**



**C#:**



**Блок-схема**

