**РУСЕНСКИ УНИВЕРСИТЕТ „АНГЕЛ КЪНЧЕВ“**

**КУРСОВА РАБОТА**

ПО

СОФТУЕРНО ИНЖЕНЕРСТВО

Студент:

Фак. Номер:

Група:

Специалност:

Дата: Проверил:

# Съдържание

[Съдържание 2](#_Toc373102688)

[1. Задание 3](#_Toc373102689)

[2. Диаграми 3](#_Toc373102690)

[2.1 Delphi диаграма 3](#_Toc373102693)

[2.2 Use Case 4](#_Toc373102694)

[2.3 Sequence 5](#_Toc373102695)

[2.4 Activity 6](#_Toc373102696)

[2.5 Class 7](#_Toc373102697)

[3. Работа с програмата 8](#_Toc373102698)

[4. Код на програмата 10](#_Toc373102699)

[4.1 SourceCodeMain.pas 10](#_Toc373102700)

[4.2 GenFiles.pas 11](#_Toc373102701)

[4.3 UrsProfiler.pas 13](#_Toc373102702)

# Задание

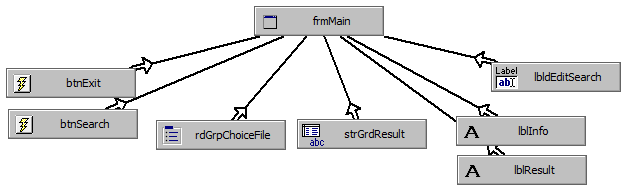
Да се проектира и реализира Windows приложение за изследване на алгоритъма за последователно търсене на един запис в последователен файл.

* Да се генерира тест от 50 записа.
* Да се генерира тест от 500 записа.
* Да се генерира тест от 5,000 записа.
* Да се извежда обема на заетата дискова памет за всеки тест.
* Да се извежда бързодействието за всеки тест.

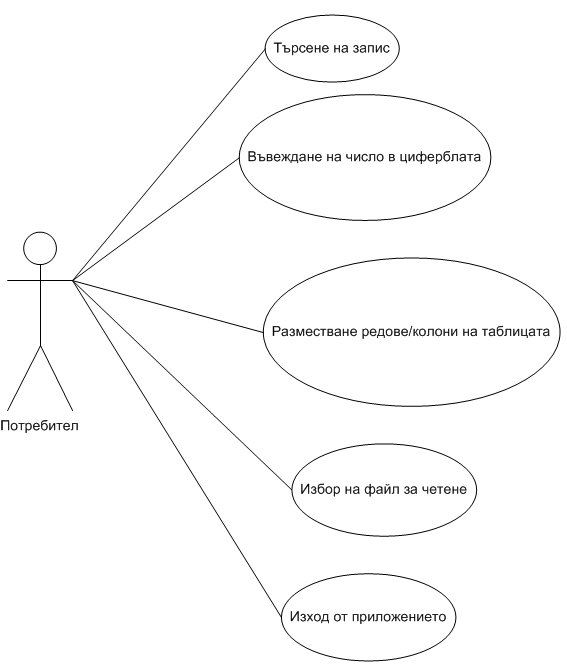
# Диаграми



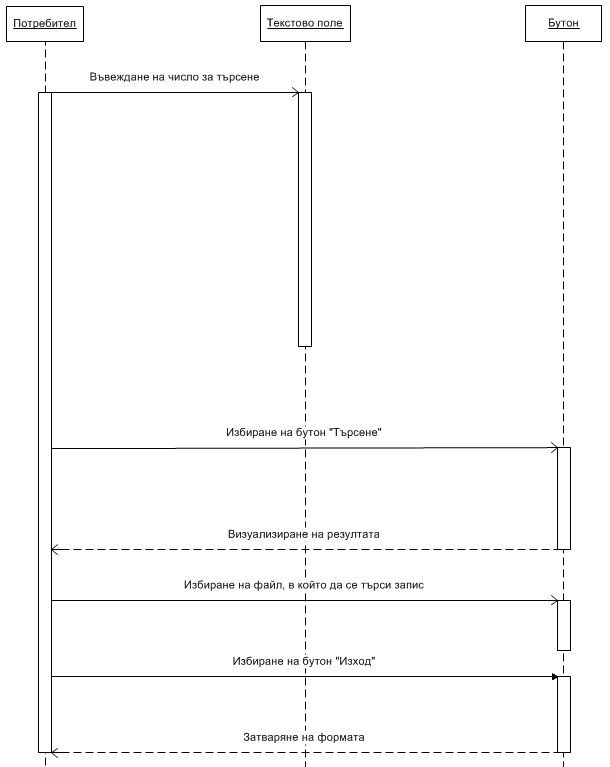
## Delphi диаграма



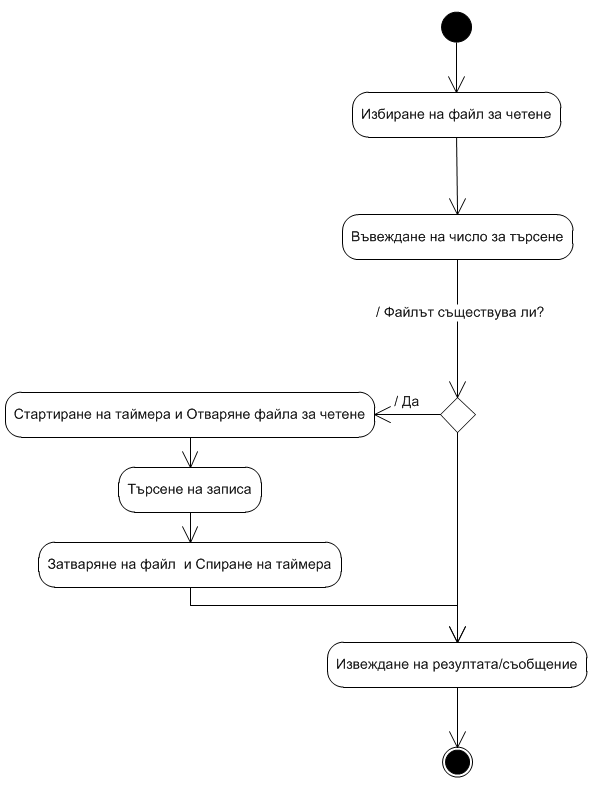
## Use Case



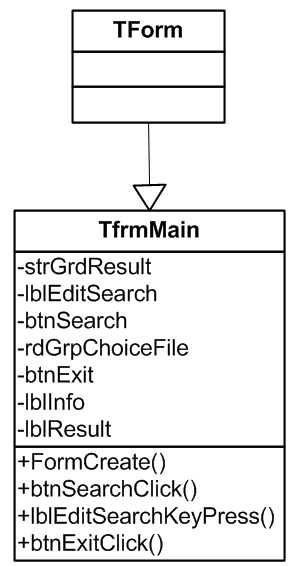
## Sequence



## Activity

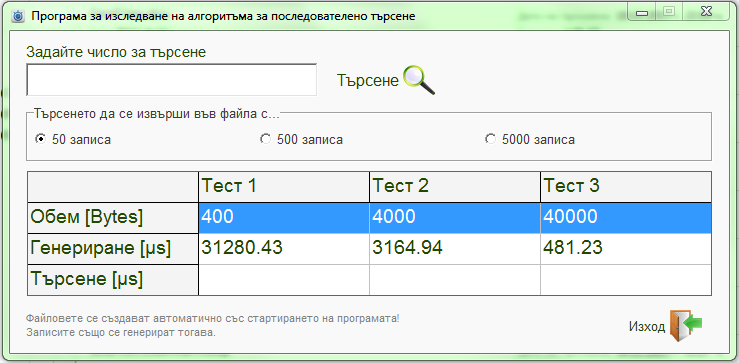


## Class



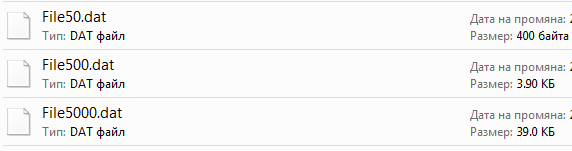
# Работа с програмата

Със стартиране на програмния файл \*.ехе на екрана на се визуализира прозорецът от фиг. 1.

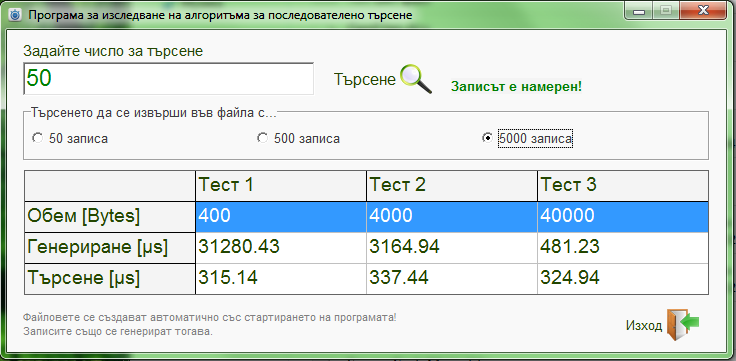


Фиг. 1 Прозорец на програмата

Трите файла се създават (в директорията, където се намира програмата) и инициализират още със създаването на програмата. Имената им са съответно: File50.dat, File500.dat и File5000.dat (фиг. 2). В тях са записани n (n = 50 | 500 | 5000) на брой записи (число и случаен символ). Обемът на заетата дискова памет и времето, за което е генериран всеки един тест се визуализира в таблицата (фиг. 3).



Фиг. 2 Генерираните файлове



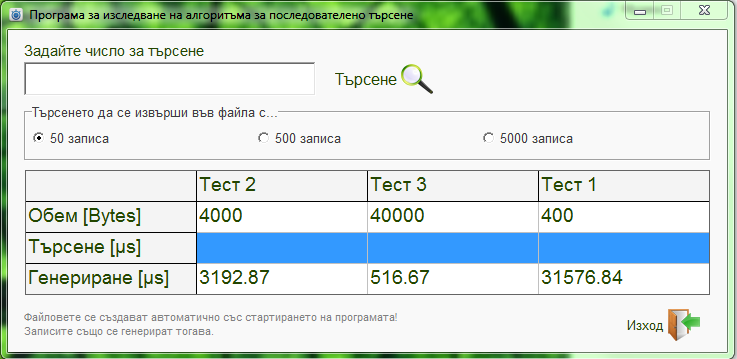
Фиг. 3 Примерен тест

Потребителят има възможност да избере един от трите файла (теста), чрез радио бутоните, в който да търси зададен запис. За целта обаче в текстовото поле трябва да се въведе цяло число за търсене.

* От 1 до 50 за тест 1;
* От 1 до 500 за тест 2;
* От 1 до 5,000 за тест 3;

След това потребителят трябва да натисне “Enter” или да избере бутона „Търсене“ и резултатът за съответният тест ще се появи в таблицата. Информацията за това дали числото е намерено или не се визуализира вдясно от бутона „Търсене“ (фиг. 3).

Потребителят има и възможност да размества редовете и колоните на таблицата (фиг. 4).



За изход от програмата е необходимо да се избере бутонът „Изход“.

# Код на програмата

## SourceCodeMain.pas

unit SourceCodeMain;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ExtCtrls, Grids, Buttons, GenFiles, UrsProfiler;

type

TfrmMain = class(TForm)

strGrdResult: TStringGrid;

lbldEditSearch: TLabeledEdit;

btnSearch: TSpeedButton;

rdGrpChoiceFile: TRadioGroup;

btnExit: TSpeedButton;

lblInfo: TLabel;

lblResult: TLabel;

procedure FormCreate(Sender: TObject);

procedure btnSearchClick(Sender: TObject);

procedure lbldEditSearchKeyPress(Sender: TObject; var Key: Char);

procedure btnExitClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmMain: TfrmMain;

implementation

{$R \*.dfm}

procedure TfrmMain.FormCreate(Sender: TObject);

begin

GenerateRecordsAndSave;

strGrdResult.Cells[1, 2] := rsProfiler[1].last;

strGrdResult.Cells[2, 2] := rsProfiler[2].last;

strGrdResult.Cells[3, 2] := rsProfiler[3].last;

strGrdResult.Cells[1, 1] := msgFileSize50;

strGrdResult.Cells[2, 1] := msgFileSize500;

strGrdResult.Cells[3, 1] := msgFileSize5000;

strGrdResult.Cells[1, 0] := 'Тест 1';

strGrdResult.Cells[2, 0] := 'Тест 2';

strGrdResult.Cells[3, 0] := 'Тест 3';

strGrdResult.Cells[0, 1] := 'Обем [Bytes]';

strGrdResult.Cells[0, 2] := 'Генериране [µs]';

strGrdResult.Cells[0, 3] := 'Търсене [µs]';

rdGrpChoiceFile.Buttons[0].Checked := true;

end;

procedure TfrmMain.btnSearchClick(Sender: TObject);

var

searchNumber: integer;

choicedFile: integer;

colNum: integer;

begin

choicedFile := 0;

colNum := 0;

if (lbldEditSearch.Text <> '') then begin

searchNumber := StrToInt(lbldEditSearch.Text);

if (rdGrpChoiceFile.Buttons[0].Checked = true) then begin

choicedFile := 50;

colNum := 1;

end

else if (rdGrpChoiceFile.Buttons[1].Checked = true) then begin

choicedFile := 500;

colNum := 2;

end

else if (rdGrpChoiceFile.Buttons[2].Checked = true) then begin

choicedFile := 5000;

colNum := 3;

end;

lblResult.Visible := true;

if (SearchRecord(searchNumber, choicedFile) = true) then begin

lblResult.Font.Color := clGreen;

lblResult.Caption := 'Записът е намерен!';

end

else begin

lblResult.Font.Color := clRed;

lblResult.Caption := 'Записът НЕ е намерен!';

end;

strGrdResult.Cells[colNum, 3] := rsProfiler[0].last;

end;

end;

procedure TfrmMain.lbldEditSearchKeyPress(Sender: TObject; var Key: Char);

begin

if Key in ['0'..'9', #8, #13] = false then

Key := #0

else if Key = #13 then

btnSearchClick(Sender);

end;

procedure TfrmMain.btnExitClick(Sender: TObject);

begin

frmMain.Close;

end;

end.

## GenFiles.pas

unit GenFiles;

interface

uses Math, UrsProfiler, Dialogs, SysUtils;

type

TRecTest = record

symbol: char;

number: integer;

end;

TestFile = File of TRecTest;

var

recTest: TRecTest;

file50: TestFile;

file500: TestFile;

file5000: TestFile;

msgGen50, msgGen500, msgGen5000: string;

msgFileSize50, msgFileSize500, msgFileSize5000: string;

procedure GenerateRecordsAndSave;

procedure NopP;

function SearchRecord(number: integer; choiceFile: integer): boolean;

implementation

procedure NopP;

asm

// nop

end;

procedure GenerateRecordsAndSave;

var

i: integer;

begin

AssignFile(file50, 'File50.dat');

AssignFile(file500, 'File500.dat');

AssignFile(file5000, 'File5000.dat');

ReWrite(file50);

ReWrite(file500);

ReWrite(file5000);

rsProfiler[1].Start;

for i := 1 to 5000 do begin

recTest.symbol := Char(RandomRange(33, 126));

recTest.number := i;

Seek(file5000, FileSize(file5000));

Write(file5000, recTest);

end;

rsProfiler[1].Stop;

rsProfiler[2].Start;

for i := 1 to 500 do begin

recTest.symbol := Char(RandomRange(33, 126));

recTest.number := i;

Seek(file500, FileSize(file500));

Write(file500, recTest);

end;

rsProfiler[2].Stop;

rsProfiler[3].Start;

for i := 1 to 50 do begin

recTest.symbol := Char(RandomRange(33, 126));

recTest.number := i;

Seek(file50, FileSize(file50));

Write(file50, recTest);

end;

rsProfiler[3].Stop;

msgFileSize50 := IntToStr( Round( FileSize(file50) \* SizeOf(recTest)));

msgFileSize500 := IntToStr( Round( FileSize(file500) \* SizeOf(recTest)));

msgFileSize5000 := IntToStr( Round( FileSize(file5000) \* SizeOf(recTest)));

CloseFile(file50);

CloseFile(file500);

CloseFile(file5000);

rsProfiler[1].AsString;

rsProfiler[2].AsString;

rsProfiler[3].AsString;

end;

function SearchRecord(number: integer; choiceFile: integer): boolean;

var

finded: boolean;

begin

finded := false;

NopP;

rsProfiler.Clear;

if (choiceFile = 50) then begin

{$I-} Reset(file50); {$I+}

if (IOResult <> 0) then begin

ShowMessage('Файлът "File50.dat" не е намерен!');

exit;

end

else begin

rsProfiler[0].Start;

while not EOF(file50) do begin

Read(file50, recTest);

if (recTest.number = number) then begin

finded := true;

break;

end;

end;

rsProfiler[0].Stop;

CloseFile(file50);

end;

end

else if (choiceFile = 500) then begin

{$I-} Reset(file500); {$I+}

if (IOResult <> 0) then begin

ShowMessage('Файлът "File500.dat" не е намерен!');

exit;

end

else begin

rsProfiler[0].Start;

while not EOF(file500) do begin

Read(file500, recTest);

if (recTest.number = number) then begin

finded := true;

break;

end;

end;

rsProfiler[0].Stop;

CloseFile(file500);

end;

end

else if (choiceFile = 5000) then begin

{$I-} Reset(file5000); {$I+}

if (IOResult <> 0) then begin

ShowMessage('Файлът "File5000.dat" не е намерен!');

exit;

end

else begin

rsProfiler[0].Start;

while not EOF(file5000) do begin

Read(file5000, recTest);

if (recTest.number = number) then begin

finded := true;

break;

end;

end;

rsProfiler[0].Stop;

CloseFile(file5000);

end;

end;

rsProfiler[0].AsString;

Result := finded;

end;

end.

## UrsProfiler.pas

unit UrsProfiler;

interface

uses Windows, Classes, SysUtils;

const

rsMaxPointCount = 100;

type

TrsProfiler = class;

TrsProfilerPoint = class

private

FProfiler: TrsProfiler;

FAvgTicks: Int64;

FLastTicks: Int64;

FMinTicks: Int64;

FMaxTicks: Int64;

FCount: integer;

FStartTicks: Int64;

function GetAvgTicks: Int64;

public

Name: string;

min, avg, max, last: string;

constructor Create(AOwner: TrsProfiler);

procedure AsString;

procedure Clear;

procedure Start;

procedure Stop;

property Count: integer read FCount;

property AvgTicks: Int64 read GetAvgTicks;

property LastTicks: Int64 read FLastTicks;

property MaxTicks: Int64 read FMaxTicks;

property MinTicks: Int64 read FMinTicks;

end;

TrsProfiler = class

private

FList: TList;

CPUClock: extended;

function GetPoint(index: integer): TrsProfilerPoint;

protected

FStartStopConstant: Int64;

public

constructor Create;

destructor Destroy; override;

function Add: integer;

function CalibrateCPU: Int64;

procedure Clear;

function Count: integer;

procedure Delete(const Index: integer);

function IndexOfName(const Value: string): integer;

function TicksToStr(const Value: Int64): string;

property Points[index: integer]: TrsProfilerPoint read GetPoint; default;

property CPUSpeed: extended read CPUClock;

end;

function GetCPUTick:Int64;

var

rsProfiler: TrsProfiler;

implementation

function GetCPUTick: Int64;

asm

DB $0F,$31

end;

function TrsProfiler.CalibrateCPU: Int64;

var

t: cardinal;

PriorityClass, Priority: Integer;

begin

PriorityClass := GetPriorityClass(GetCurrentProcess);

Priority := GetThreadPriority(GetCurrentThread);

SetPriorityClass(GetCurrentProcess, REALTIME\_PRIORITY\_CLASS);

SetThreadPriority(GetCurrentThread, THREAD\_PRIORITY\_TIME\_CRITICAL);

t := GetTickCount;

while t=GetTickCount do;

Result := GetCPUTick;

while GetTickCount<(t+400) do;

Result := GetCPUTick - result;

CPUClock := 2.5e-6\*Result;

SetThreadPriority(GetCurrentThread, Priority);

SetPriorityClass(GetCurrentProcess, PriorityClass);

end;

function TrsProfiler.TicksToStr(const Value: Int64): string;

begin

Result := FloatToStrF(Value/CPUClock,fffixed,10,2);{+ ' µs';}

end;

{ TrsProfilerPoint }

procedure TrsProfilerPoint.Clear;

begin

FMinTicks := High(Int64);

FMaxTicks := 0;

FAvgTicks := 0;

FLastTicks := 0;

FCount := 0;

end;

constructor TrsProfilerPoint.Create(AOwner: TrsProfiler);

begin

FProfiler := AOwner;

Clear;

end;

procedure TrsProfilerPoint.AsString;

begin

min := FProfiler.TicksToStr(MinTicks);

max := FProfiler.TicksToStr(MaxTicks);

avg := FProfiler.TicksToStr(AvgTicks);

last := FProfiler.TicksToStr(LastTicks);

end;

function TrsProfilerPoint.GetAvgTicks: Int64;

begin

if FCount>0 then Result := FAvgTicks div FCount

else Result:= 0;

end;

procedure TrsProfilerPoint.Start;

begin

FStartTicks := GetCPUTick;

end;

procedure TrsProfilerPoint.Stop;

begin

FLastTicks := GetCPUTick - FStartTicks - FProfiler.FStartStopConstant;

if FLastTicks<0 then FLastTicks := 0;

Inc(FCount);

if FLastTicks<FMinTicks then FMinTicks := FLastTicks;

if FLastTicks>FMaxTicks then FMaxTicks := FLastTicks;

Inc(FAvgTicks, FLastTicks);

end;

{ TrsProfiler }

function TrsProfiler.Add: integer;

var

p: TrsProfilerPoint;

begin

p := TrsProfilerPoint.Create(Self);

Result := FList.Add(p);

end;

procedure TrsProfiler.Clear;

begin

While Count>0 do Delete(0);

end;

function TrsProfiler.Count: integer;

begin

Result := FList.Count;

end;

constructor TrsProfiler.Create;

begin

FList := TList.Create;

CalibrateCPU;

Add;

Points[0].Start;

Points[0].Stop;

FStartStopConstant := Points[0].FLastTicks;

Clear;

end;

procedure TrsProfiler.Delete(const Index: integer);

begin

if (Index>=0) and (Index<FList.Count) then begin

TrsProfilerPoint(FList.Items[Index]).Free;

FList.Delete(Index);

end;

end;

destructor TrsProfiler.Destroy;

begin

Clear;

inherited Destroy;

end;

function TrsProfiler.GetPoint(index: integer): TrsProfilerPoint;

begin

if Index<0 then Result := nil

else

if Index<FList.Count then Result := FList.Items[Index]

else

if Index<rsMaxPointCount then begin

While Add<Index do ;

Result := FList.Items[Index];

end else Result := nil;

end;

function TrsProfiler.IndexOfName(const Value: string): integer;

var

i: integer;

begin

Result := -1;

for i:=Count-1 downto 0 do

if AnsiCompareText(Value, Points[i].Name)=0 then begin

Result := i;

Break;

end;

end;

initialization

rsProfiler := TrsProfiler.Create;

finalization

rsProfiler.Free;

rsProfiler := nil;

end.