Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по курсовой работе

Тема работы: Инженерный калькулятор

Выполнил

студент: гр. 151004 Иванов И.И.

Проверил: Фадеева Е.П.

Минск 2022

содержание

[1 Постановка задачи 3](#_Toc97906624)

[2 Описание алгоритмов решения задачи 4](#_Toc97906625)

[3 Структура данных 5](#_Toc97906626)

[3.1 Структура данных программы 5](#_Toc97906627)

[3.2 Структура данных алгоритма Swap 5](#_Toc97906628)

[3.3 Структура данных алгоритма Fill 5](#_Toc97906629)

[3.4 Структура данных алгоритма BubbleSort 6](#_Toc97906630)

[3.5 Структура данных алгоритма HeapSort 6](#_Toc97906631)

[3.6 Структура данных алгоритма SiftDown 7](#_Toc97906632)

[4 Схема алгоритма решения задачи по ГОСТ 19.701-90 8](#_Toc97906633)

[4.1 Схема алгоритма решения задачи 8](#_Toc97906634)

[4.2 Схема алгоритма Swap 9](#_Toc97906635)

[4.3 Схема алгоритма Fill 10](#_Toc97906636)

[4.4 Схема алгоритма BubbleSort 11](#_Toc97906637)

[4.5 Схема алгоритма HeapSort 12](#_Toc97906638)

[4.6 Схема алгоритма SiftDown 13](#_Toc97906639)

[5 Результаты расчетов 14](#_Toc97906640)

[Приложение А 15](#_Toc97906641)

[Приложение Б 19](#_Toc97906642)

# ВВЕДЕНИЕ

Данный курсовой проект посвящен изучению объектно-ориентированного языка Delphi.

Практической целью проекта является разработка приложения «Инженерный калькулятор» с применением объектно-ориентированного программирования. Для решения поставленной задачи были использованы стандартные библиотеки языка Delphi.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить теоретический материал, связанный с языком Delphi;
2. На основе изученного материала написать приложение;
3. Протестировать приложение.

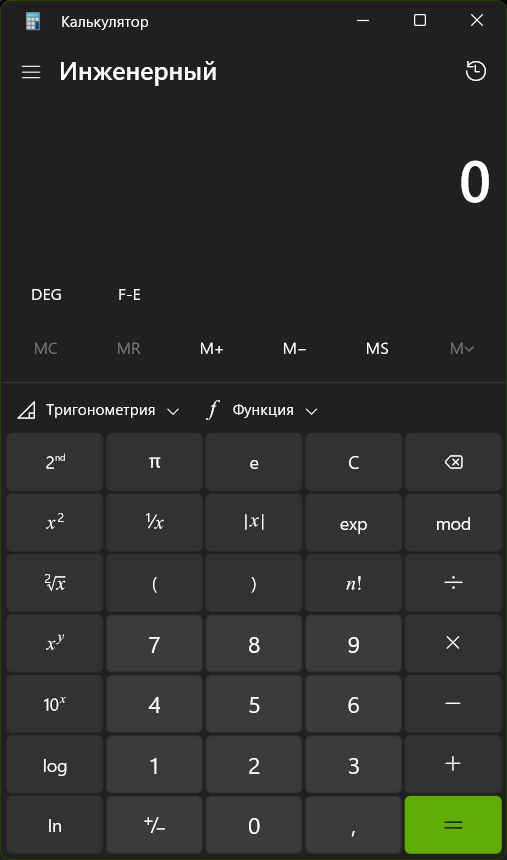
Выполнение данных задач позволит создать приложение «Инженерный калькулятор».

Курсовой проект актуален, так как полученный результат приложения можно будет в дальнейшем совершенствовать.

# Аналитический обзор литературы

## Обзор существующих аналогов

### Калькулятор Windows



Встроенный калькулятор Windows имеет несколько режимов: обычный, инженерный, дат.

[тригонометрические](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D0%B3%D0%BE%D0%BD%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8) и [гиперболические](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D0%B1%D0%BE%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8) (флажок «Hyp») функции, обратные им функции, которые доступны через флажок «Inv» (сбрасывается автоматически), натуральный и десятичный [логарифмы](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B0%D1%80%D0%B8%D1%84%D0%BC), натуральная (тоже доступна через «Inv») и десятичная экспоненты, возведение в степень и извлечение корня (отдельно реализованы вторая, третья и произвольная степени), а также число {\displaystyle \pi }.

перевод долей градуса в минуты и секунды (обратно через флажок «Inv»), вычисление [факториала](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B0%D0%BB), а для нецелого аргумента — пи-функции, которая является обобщением факториала и определяется через [гамма-функцию](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D0%BC%D0%BC%D0%B0-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) как {\displaystyle \Pi (x)=\Gamma (x+1)}.

группировка операций (кнопки со скобками, есть индикатор уровня вложенности), переключение режимов отображения (фиксированная/плавающая точка).

вычисление остатка от деления

по[битовые операции](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8): AND, OR, NOT, XOR. Перед вычислением дробная часть отбрасывается.

сдвиг влево (сдвиг вправо через флажок «Inv»)

# Структура данных

## Структура данных программы

Таблица 2 – Структура данных

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| I, J | Integer | Параметры цикла |
| Comp | Integer | Число сравнений |
| Arr1, Arr2 | Array[1..3000] Of Integer | Массивы для сортировки |
| N | Array[1..6] Of Integer | Массив из размерностей массивов |
| S | Array[1..3] Of String | Массив из имён массивов |

## Структура данных алгоритма Swap

Таблица 3 – Структура данных алгоритма Swap(A, B)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| A | Integer | Обмениваемое число | Формальный |
| B | Integer | Обмениваемое число | Формальный |
| T | Integer | Сохранённое число | Локальный |

## Структура данных алгоритма Fill

Таблица 4 – Структура данных алгоритма Fill(Arr, N, Opt)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| N | Integer | Количество элементов массива | Формальный |
| Opt | Integer | Выбор условия, по которому заполняется массив | Формальный |
| I | Integer | Параметр цикла | Локальный |

## Структура данных алгоритма BubbleSort

Таблица 5 – Структура данных алгоритма BubbleSort(Arr, N, Comp)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| N | Integer | Количество элементов массива | Формальный |
| Comp | Integer | Число сравнений | Формальный |
| I, J | Integer | Параметры цикла | Локальный |
| Sorted | Boolean | «Флажок» сортировки | Локальный |

## Структура данных алгоритма HeapSort

Таблица 6 – Структура данных алгоритма HeapSort (Arr, N, Comp)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| N | Integer | Количество элементов массива | Формальный |
| Comp | Integer | Число сравнений | Формальный |
| NodeCurr | Integer | Текущий индекс | Локальный |
| NodeLast | Integer | Сохранённый индекс | Локальный |

## Структура данных алгоритма SiftDown

Таблица 7 – Структура данных алгоритма SiftDown(Arr, Node, N, Comp)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| Node | Integer | Текущий индекс | Формальный |
| N | Integer | Количество элементов массива | Формальный |
| Comp | Integer | Число сравнений | Формальный |
| Root | Integer | Текущий индекс | Локальный |
| Child | Integer | Дочерний индекс | Локальный |
| Sifted | Boolean | Закончена ли сборка дерева | Локальный |

# Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема алгоритма решения задачи

|  |
| --- |
|  |
| Рисунок – Схема алгоритма решения задачи |

## Схема алгоритма Swap

|  |
| --- |
|  |
| Рисунок 2 – Схема алгоритма Swap |

## Схема алгоритма Fill

|  |
| --- |
|  |
| Рисунок 3 – Схема алгоритма Fill |

## Схема алгоритма BubbleSort

|  |
| --- |
|  |
| Рисунок 4 – Схема алгоритма BubbleSort |

## Схема алгоритма HeapSort

|  |
| --- |
|  |
| Рисунок 5 – Схема алгоритма HeapSort |

## Схема алгоритма SiftDown

|  |
| --- |
|  |
| Рисунок 6 – Схема алгоритма SiftDown |

# Результаты расчетов

Вследствие результатов программы мы получаем следующие результаты:

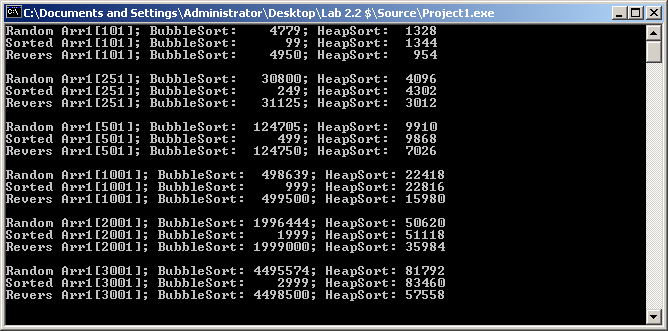


Рисунок 7 – Результаты расчетов

Приложение А

(обязательное)

Исходный код программы (постановка 1)

Program Lab2;

{Comp sorts}

//Use app

{$APPTYPE CONSOLE}

//Declare modules

Uses

SysUtils;

//Declare types

Type

TArr = Array[1..3000] Of Integer;

//TArr - our array

//Declare Vars

Var

I, J: Integer;

Comp: Integer;

Arr1, Arr2: TArr;

N: Array[1..6] Of Integer = (100, 250, 500, 1000, 2000, 3000);

S: Array[1..3] Of String = ('Random', 'Sorted', 'Revers');

//I,J - loop params

//N - array sizes

//Comp - quantity of comparisons

//Arr1, Arr2 - our arrays

//N - num of elements

//S - name of array

//Swaps 2 elements

//A, B - elements

Procedure Swap(Var A, B: Integer);

Var

T: Integer;

//T - temp

Begin

T:= A;

A:= B;

B:= T;

End;

//Fills array with elements

//Arr - array, N - array size, Opt - filler type

Procedure Fill(Var Arr: TArr; Const N, Opt: Integer);

Var

I: Integer;

//I - select type of filling

Begin

Randomize;

Case Opt Of

1:

For I:= Low(Arr) To N Do

Arr[I]:= Random(N);

2:

For I:= Low(Arr) To N Do

Arr[I]:= I;

3:

For I:= Low(Arr) To N Do

Arr[I]:= N - I;

End;

End;

//Sorts an array and calculates the number of comparisons

//Arr - array, N - array size, Comp - comparisons

Procedure BubbleSort(Var Arr: TArr; Const N: Integer; Var Comp: Integer);

Var

I, J: Integer;

Sorted: Boolean;

//I, J - loop params

//Sorted - condition to exit

Begin

Sorted:= False;

I:= 1;

While (I <= N - 1) And Not Sorted Do

Begin

Sorted:= True;

For J:= 1 To N - I Do

Begin

If Arr[J] > Arr[J + 1] Then

Begin

Sorted:= False;

Swap(Arr[J], Arr[J + 1]);

End;

Inc(Comp);

End;

Inc(I);

End;

End;

//Sorts an array and calculates the number of comparisons

//Arr - array, Node - current index, N - array size, Comp - comparisons

Procedure SiftDown(Var Arr: TArr; Node: Integer; Const N: Integer; Var Comp: Integer);

Var

Root, Child: Integer;

Sifted: Boolean;

//Root, Child - indexes

//Sifted - condition to exit

Begin

Root:= Node;

Sifted:= False;

While (Not Sifted) And (Root \* 2 - Node + 1 <= N) Do

Begin

Comp:= Comp + 2;

Child:= Root \* 2 - Node + 1;

If (Child + 1 <= N) And (Arr[Child] < Arr[Child + 1]) Then

Inc(Child);

If Arr[Root] < Arr[Child] Then

Begin

Swap(Arr[Root], Arr[Child]);

Root:= Child;

End

Else

Sifted:= True;

End;

End;

//Sorts an array and calculates the number of comparisons

//Arr - array, N - array size, Comp - comparisons

Procedure HeapSort(Var Arr: TArr; Const N: Integer; Var Comp: Integer);

Var

NodeLast, NodeCurr: Integer;

//NodeLast, NodeCurr - indexes

//Comps - counter

Begin

NodeCurr:= N Div 2 - 1;

While NodeCurr >= Low(Arr) Do

Begin

SiftDown(Arr, NodeCurr, N, Comp);

Dec(NodeCurr);

End;

NodeLast:= N;

While NodeLast > Low(Arr) Do

Begin

Swap(Arr[Low(Arr)], Arr[NodeLast]);

Dec(NodeLast);

SiftDown(Arr, Low(Arr), NodeLast, Comp);

End;

End;

Begin

For I:= 1 To 6 Do

Begin

For J:= 1 To 3 Do

Begin

Fill(Arr1, N[I], J);

//Show arr size

Write(S[J], ' Arr[', N[I], ']; ');

//Copy array

Arr2:= Arr1;

//Enzero to show true value

Comp:= 0;

BubbleSort(Arr1, N[I], Comp);

Write('BubbleSort: ', Comp: 7, '; ');

//Enzero to show true value

Comp:= 0;

HeapSort(Arr2, N[I], Comp);

Write('HeapSort: ', Comp: 5);

WriteLn;

End;

WriteLn;

End;

ReadLn;

End.

Приложение Б

(обязательное)

Тестовые наборы

Формула числа сравнений для BubbleSort: C = (N2-N)/2

Поскольку в постановке задачи применялся не простой BubbleSort, а BubbleSort с флажком, то отсортированный массив будет проверен всего 1 раз, а количество сравнений будет высчитываться не по обычной формуле, а по формуле C = N – 1.

Формула среднего числа сравнений для HeapSort: C = 2 \* N \* log2N

Результаты на практике подтверждают теоретические результаты, рассчитанные в Microsoft Excel.

|  |  |
| --- | --- |
|  |  |