Выполнил:

студент группы 19ВВ2

Павлов А.А.

Принял:

д.т.н. доцент Митрохин М.А.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма Форда-Беллмана»

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

Пенза 2020

**ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**Факультет вычислительной техники

Кафедра “Вычислительная техника”

“УТВЕРЖДАЮ”

Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_

**ЗАДАНИЕ**

на курсовое проектирование по курсу

Логика и основы алгоритмизации в инженерных задачах

**Студенту**  Павлову Алексею Александровичу **Группы**    19ВВ2

**Тема проекта:**   Реализация алгоритма Форда-Беллмана

**Исходные данные (технические требования) на проектирование**

Разработка алгоритмов и программного обеспечения в соответствии с данным заданием курсового проекта.

Пояснительная записка должна содержать:

     1. Постановку задачи;

     2. Теоретическую часть задания;

     3. Описание алгоритма поставленной задачи;

     4. Пример ручного расчёта задачи и вычислений (на небольшом участке работы алгоритма);

     5. Описание самой программы;

     6. Тесты;

     7. Список литературы;

     8. Листинг программы;

     9. Результаты работы программы;

**Расчетная часть**

Ручной расчёт работы алгоритма.

**Графическая часть**

Схема алгоритма в формате блок - схем.

**Экспериментальная часть**

Тестирование программы;                                                                                        Результаты работы программы на тестовых данных.

**Срок выполнения проекта по разделам**

В  соответствии  с  графиком  выполнения  курсового  проекта.

Дата выдачи задания “16” сентября 2020 г.

Дата защиты проекта “ ”              202 г.

**Руководитель**

**Задание получил** “16” сентября 2020 г.

**Студент** Павлов А.А.

Содержание

1. [Постановка задачи 4](#_1_Постановка_задачи)
2. [Выбор решения5](#_2_Выбор_решения)
3. [Описание разработки программы 7](#_3._Описание_разработки)
4. [Отладка и тестирование 9](#_4_Отладка_и)
5. [Описание программы. Функции 10](#_5_Описание_программы)
   1. [Иерархическая структура 11](#_5.2_Иерархическая_структура)
   2. [Разработка функций на языке Assembler 12](#_5.3_Разработка_функции)
6. [Руководство пользователя 14](#_6_Руководство_пользователя)

[Заключение 19](#_Заключение)

[Список используемых источников 20](#_Список_используемых_источников)

[Приложение А Листинг программы 21](#_Приложение_А_Листинг)

A.1 [Файл «main.cpp» 21](#Файл_main_cpp)

A.2 [Файл «std\_math.h» 23](#std_math)

A.3 [Файл «proverka.h» 24](#proverka_h)

A.4 [Файл «zastavka.h» 25](#zast_h)

A.5 [Файл «std\_math.cpp» 26](#std_math_cpp)

A.6 [Файл «proverka.cpp» 44](#prov_cpp)

A.7 [Файл «zastavka.cpp» 45](#zast_cpp)

[Приложение B Схемы программы 46](#pril_b)

# 1. Постановка задачи

Необходимо разработать программу, которая будет искать длины кратчайших путей от вершины st до всех остальных вершин, используя алгоритм Беллмана-Форда.

Исходный граф в программе должен задаваться матрицей смежности, причѐм при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности орграфа, вид орграфа и все компоненты связности орграфа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

# 2. Теоретическая часть

Граф G (рисунок 1) задается множеством вершин 1, 2, ..., 5(N). и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества А ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.

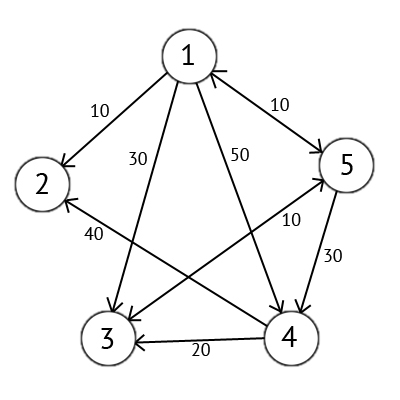


Рисунок 1 – Пример орграфа

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается весом ребра, иначе нулем.

**Алгоритм Беллмана-Форда** — алгоритм поиска кратчайшего пути во взвешенном графе. За время O(|st|\*|E|) алгоритм находит кратчайшие пути от одной вершины графа до всех остальных. В отличие от алгоритма Дейкстры, алгоритм Беллмана — Форда допускает рёбра с отрицательным весом. Предложен независимо Ричардом Беллманом и Лестером Фордом.

Дан ориентированный или неориентированный граф **G** со взвешенными рёбрами. Длиной пути назовём сумму весов рёбер, входящих в этот путь. Требуется найти кратчайшие пути от выделенной вершины **st** до всех вершин графа.

Заметим, что кратчайших путей может не существовать. Так, в графе, содержащем цикл с **отрицательным суммарным весом**, существует сколь угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути). Цикл, сумма весов рёбер которого отрицательна, называется **отрицательным циклом**.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (relax, ослабление) вдоль каждого ребра **(v,u)** веса **w**. Релаксация вдоль ребра — это попытка улучшить значение nodes[u] значением nodes[v] + w. Фактически это значит, что мы пытаемся улучшить ответ для вершины u, пользуясь ребром (v,u) и текущим ответом для вершины v.

Утверждается, что достаточно N-1 фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (при условии, что граф не содержит циклов отрицательного веса). Для недостижимых вершин расстояние nodes[] останется равным бесконечности (заведомо большим числом).

# 3. Описание алгоритма поставленной задачи

Для алгоритма Форда-Беллмана, в отличие от многих других графовых алгоритмов, более удобно представлять граф в виде одного списка всех рёбер. В приведённой реализации заводится массив структур данных **edge** для всех рёбер. Входными данными для алгоритма являются числа **N** (кол-во вершин), **e** (кол-во рёбер), список **edge** рёбер, и номер стартовой вершины **st**. Все номера вершин нумеруются с 0 по N-1. Константа **inf** обозначает число "бесконечность" — её надо подобрать таким образом, чтобы она заведомо превосходила все возможные длины путей. Массив расстояний **nodes[0…N-1]**, который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: nodes[st] = 0, а все остальные элементы nodes[] равны бесконечности (inf).

Этот алгоритм можно несколько ускорить: зачастую ответ находится уже за несколько фаз, а оставшиеся фазы никакой полезной работы не происходит, лишь впустую просматриваются все рёбра. Поэтому будем хранить флаг (**change**) того, изменилось что-то на текущей фазе или нет, и если на какой-то фазе ничего не произошло, то алгоритм можно останавливать.

Алгоритм Форда-Беллмана модифицирован, чтобы он не только находил длины кратчайших путей, но и позволял восстанавливать сами кратчайшие пути. Для этого заведён ещё один вектор **parent[0…N-1]**, в котором для каждой вершины хранится её "предкок", т.е. предпоследняя вершина в кратчайшем пути, ведущая в неё. В самом деле, кратчайший путь до какой-то вершины v является кратчайшим путём до какой-то вершины parent[v], к которому приписали в конец вершину v. Заметим, что алгоритм Форда-Беллмана работает по такой же логике: он, предполагая, что кратчайшее расстояние до одной вершины уже посчитано, пытается улучшить кратчайшее расстояние до другой вершины. Следовательно, в момент улучшения нам надо просто запоминать в parent[], из какой вершины это улучшение произошло.

В цикле восстановления путей мы сначала проходимся по предкам, начиная с вершины i, и сохраняем весь пройденный путь в векторе **path**. В этом векторе получается кратчайший путь от st до i, но в обратном порядке, поэтому мы вызываем **reverse** от него и затем выводим.

При восстановлении пути может возникнуть момент, когда после первой итерации мы попадаем в вершину, в которой уже были до этого, это свидетельствует о наличие отрицательного цикла. В этом случае останавливаем восстановление пути и выводим сообщение о некорректности результата.

Поскольку при наличии отрицательного цикла за N итераций алгоритма расстояния могли уйти далеко в минус, в коде приняты дополнительные меры против такого целочисленного переполнения:

nodes[edge[j].u] = max(-inf, nodes[edge[j].v] + edge[j].w);

Ниже представлен псевдокод функции **bellman\_ford()**

1. Открываем/создаём файл bellman\_ford.txt для сохранения результатов.

2. В случае невозможности открытия/создания файла останавливаем программу.

3. Заполняем массив **nodes[0…N-1] = inf**.

4. Инициализируем вектор **parent[0…N-1] = -1**.

5. nodes[**st**(стартовая вершина)] = 0.

6. Для i = 0; пока i < N-1; делать i = i + 1.

7. Флаг **change = 0**.

8. Для j = 0; пока j < e (кол-во рёбер); делать j = j + 1.

9. Если nodes[edge[j].v] + edge[j].w < nodes[edge[j].u].

10. nodes[edge[j].u] = наибольшее из (-inf либо nodes[edge[j].v] + edge[j].w).

11. parent[edge[j].u] = edge[j].v.

12. change = 1.

13. Если change == 0

14. Выходим из цикла.

15. Для i = 0; пока i < N; i = i + 1.

16. Если nodes[i] == inf

17. Выводим в консоль и файл, что «Путь отсутствует».

18. Иначе

19. Выводим в консоль и файл nodes[i]

20. Если 0 <= nodes[i] < 10

21. Выводим в консоль и файл Пробел.

22. Объявляем вектор **path**.

23. Для cur = i; пока cur ≠ -1; делать cur = parent[cur].

24. Для i2 = 0; пока i2 < размер path; делать i2 = i2 + 1.

25. Если cur == path[i2] и размер path > 1.

Приложение «**Kurs**» состоит из четырёх модулей:

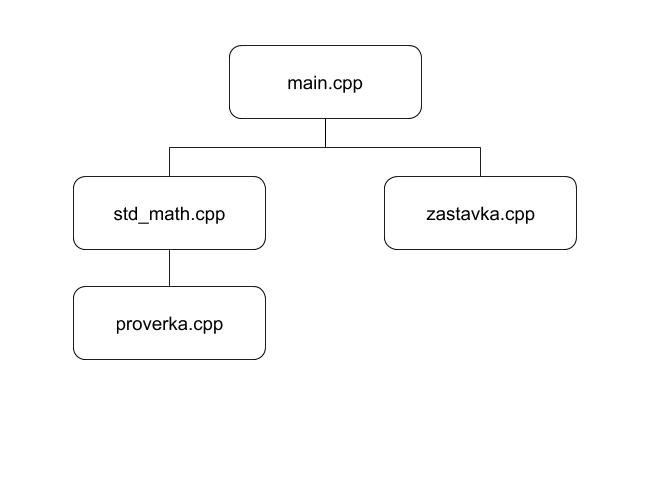
1. **main.cpp**
2. **proverka.cpp**
3. **std\_math.cpp**
4. **zastavka.cpp**

Файл **main.cpp** является основным модулем данной курсовой работы, контролирующее управляющие элементы интерфейса и вызывающем различные функции из других файлов. А также содержит функции необходимые для реализации меню калькулятора. Он связан заголовочными файлами std\_math.h, zastavka.h.

Файл **std\_math.cpp** содержит функции необходимые для реализации вычислений калькулятора. Он связан заголовочным файлами std\_math.h и proverka.h.

Файл **zastavka.cpp** используется для создания заставки.

Файл **proverka.cpp** используется для проверки вводимых данных.



**Рис.1** - **Структура модулей программы**

Все вышеперечисленное удовлетворяет требованию много модульности данной курсовой работы. Управление программой осуществляется с помощью клавиатуры. При запуске выводится заставка - приветствие, после чего программа готова к работе. Программа предназначена для работы со строкой-выражением, строка вводится при помощи клавиатуры, после ввода всех запрашиваемых переменных осуществляется запуск алгоритма проверки строки и последующего ей вычисления.

Навигация по командам реализована через клавиатуру, а именно через клавиши – **1,2,3,4,5,6,7,8,9,f,n,s,c,t,k**.

# 3. Описание разработки программы

Для написания данной программы будет использован язык программирования Си. Этот язык является распространённым языком программирования. При разработке языка Си был принят компромисс между низким уровнем языка ассемблера и высоким уровнем других языков. В языке Си предусмотрено много операций, непосредственно исполняемых большинством микропроцессоров; в то же время он дает программисту возможность выражать свои алгоритмы и данные наиболее подходящими средствами. Язык Си обеспечивает возможности структурирования данных. Он позволяет разрабатывать большие, но структурированные программы, предоставляя возможность отдельной разработки подпрограмм.

Целью создателей языка Си была разработка мобильного языка, который можно было бы использовать для разработки системного программного обеспечения. Программы, написанные на языке высокого уровня, мобильны, поскольку язык должен быть одним и тем же независимо от того, на каком компьютере и в какой операционной системе он используется. Дополнительное свойство языка Си, называемое условной компиляцией, позволяет программисту изолировать машинно-зависимые операторы и контролировать их компиляцию в другой среде. Это дополнительно повышает мобильность программного обеспечения, написанного на языке Си.

В качестве среды программирования был выбран программный продукт Visual Studio 2017.

Интерфейс калькулятора будет представлен в виде консоли. Результат будет представлен на экране, а также сохранён в документе «math\_history.doc», появляющемся в корневой папке после первого выбора действия. Перемещение по пунктам меню будет осуществляться с помощью клавиш клавиатуры.

При неправильном вводе данных или невозможности их распознавания программой пользователь увидит ошибку и предложение ввести данные заново.

Каждый раз после вывода результата на экран программа будет предлагать завершить работу **зажатием** клавиши **ESC** либо же нажатием любой другой клавиши очистить консоль и повторно запросить математическое действие.

Разработка программы началась с реализации набора функций, необходимых для работы с числами.

Основная программа **main.cpp** выполнятся функцией **main**. В функции **main** выполняется запрос на вывод заставки и запуск цикла обработки событий. Цикл обработки событий реагирует на вводимые с помощью клавиатуры данные и вызывает соответствующую функцию из модуля **std\_math.cpp** для обработки события.

Был разработан алгоритм функции main. Функция отвечает за инициализацию консоли, за запрос на вывод заставки, за передачу управления в функции обработки меню. Цикл обработки событий главного окна обеспечивает активизацию меню и выход из него. После выхода из функции обработки меню функция обработки событий главного окна получает идентификатор выбранного пункта меню. В зависимости от идентификатора функция вызывает функцию обработки выбранного пункта меню.

# 4. Отладка и тестирование

В качестве среды разработки была выбрана программа Visual Studio 2017. Программа предоставляет все средства, необходимые при разработке и отладке модулей и программ. В ходе отладки применялись такие возможности, как точка остановки, выполнение кода по шагам, анализ содержимого локальных и глобальных переменных, анализ содержимого памяти.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы.

# 5. Описание программы

## 5.1 Функции

Приложение **main.cpp** является основным модулем программы. После запуска программы выводится заставка. После программа ожидает выбор действия пользователем, затем управление переходит в функции. После нажатия указанных клавиш функции обрабатываются по таблице 1.

|  |  |  |
| --- | --- | --- |
| Клавиша | № функции в переменной «c» | Действия программы |
| 1 | (49 ASCII) | Сложение |
| 2 | (50 ASCII) | Вычитание |
| 3 | (51 ASCII) | Умножение |
| 4 | (52 ASCII) | Деление |
| 5 | (53 ASCII) | Возведение в квадрат |
| 6 | (54 ASCII) | Вычисление квадратного корня |
| 7 | (55 ASCII) | Возведение в n-ую степень |
| 8 | (56 ASCII) | Вычисление корня n-ой степени |
| 9 | (57 ASCII) | Вычисление отношения единицы к числу |
| f | (102 ASCII) | Вычисление факториала |
| n | (110 ASCII) | Вычисление натурального логарифма |
| s | (115 ASCII) | Вычисление синуса радиана |
| c | (99 ASCII) | Вычисление косинуса радиана |
| t | (116 ASCII) | Вычисление тангенса радиана |
| k | (107 ASCII) | Вычисление котангенса радиана |

**Таблица 1 - Описание состояний программы**

## 5.2 Иерархическая структура

Иерархическая структура программы представлена на рисунке 2.

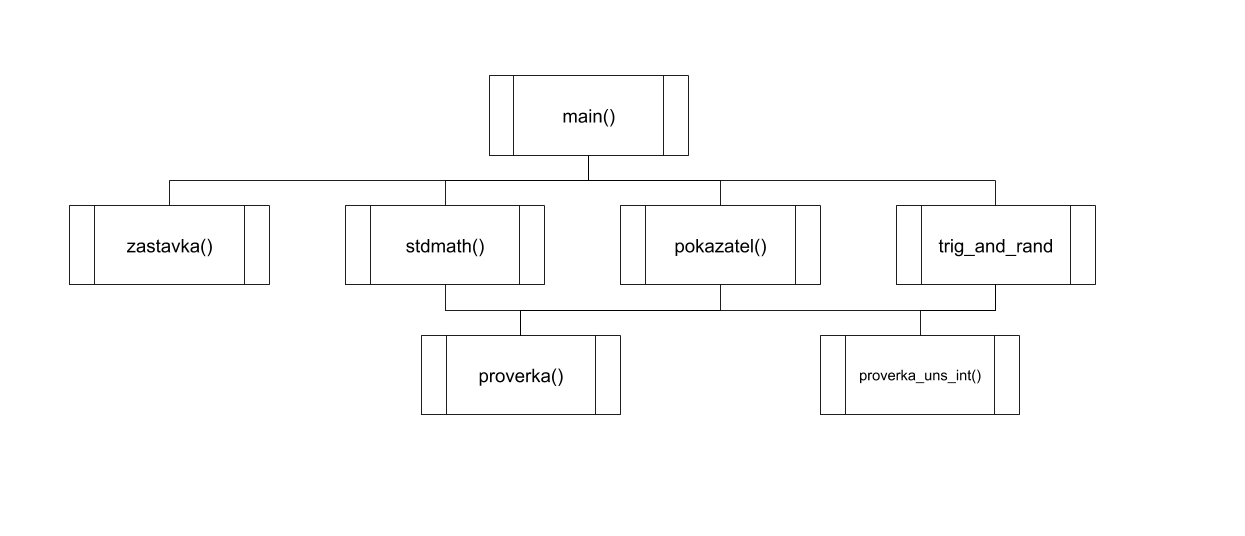


Рис. 2 – Иерархическая структура программы

## 5.3 Разработка функций на языке Assembler

Как известно, функции, реализованные на языке ассемблера, занимают меньше места и работают быстрее. В качестве разрабатываемых функций были выбраны функции для сложения, вычитания, умножения, деления 2-х чисел; отношения единицы к числу, возведения в квадрат и вычисления квадратного корня, возведения в n-ую степень, вычисления факториала, натурального логарифма, синуса, косинуса и тангенса.

Описание функций выполнено на языке C, а вычисления представляет собой ассемблерную вставку. Для реализации необходимо перед такой вставкой прописать ключевое слово «\_asm», а сам код расположить в фигурных скобках.

Были использованы следующие ассемблерные инструкции:

- команда mov используется для копирования значения из приемника в источник;

- команда call вызывает процедуру;

- команда add складывает содержимое приемника и источника и сохраняет его в последнем;

- команда cmp используется для сравнения двух операндов;

- команда je выполняет короткий переход, если первый операнд равен второму операнду при выполнении операции сравнения;

- команда jg выполняет короткий переход, если первый операнд больше второго операнда при выполнении операции сравнения;

- команда jmp выполняет безусловный переход в указанное место;

- команда dec производит вычитание "1" из указанного операнда;

- команда inc производит прибавление "1" к указаннму операнду;

- команда loop уменьшает значение в регистре сх. Если после этого значение в сх не равно нулю, то команда выполняет переход на метку;

А также система команд сопроцессора:

- команда finit сбрасывает все управляющие регистры FPU в исходное состояние;

- команда fld загружает вещественное число в стек;

- команда fild загружает целое число в стек;

- команда fstp сохраняет вещественное значение с извлечением из стека в приёмнике;

- команда fistp сохраняет целое значение с извлечением из стека в приёмнике;

- команда fmul выполняет умножение вещественных чисел источника и приёмника и помещает результат в приёмник;

- команда fimul выполняет умножение целых чисел источника и приёмника и помещает результат в приёмник;

- команда fdiv выполняет деление вещественных чисел (аналогично умножению);

- команда fsqrt извлекает квадратный корень;

- команда fldln2 загружает в стек натуральный логарифм двойки;

- команда fyl2x вычисляет значение функции ST(0)=ST(1)·log2ST(0);

- команда fsin вычисляет синус;

- команда fcos вычисляет косинус;

- команда fptan вычисляет тангенс.

# 6. Руководство пользователя

Программа Kurs предназначена для математических расчётов данных. Она имеет понятный интерфейс и поддерживает множество описанных выше операций, а также интуитивно понятное управление.

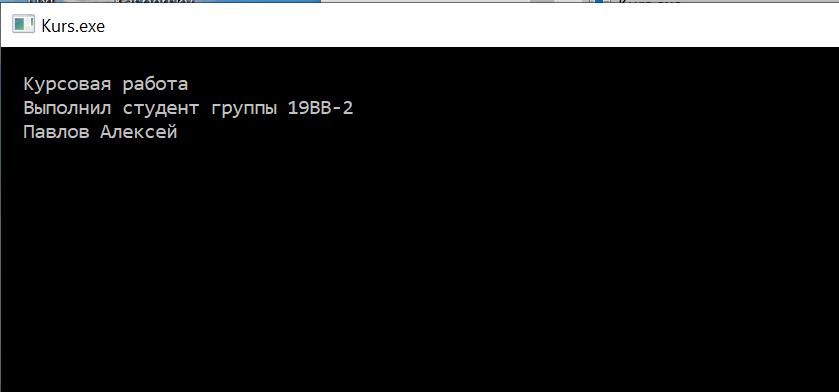


Рисунок 3.1 – заставка

Для выбора действия необходимо нажать на указанную слева клавишу:

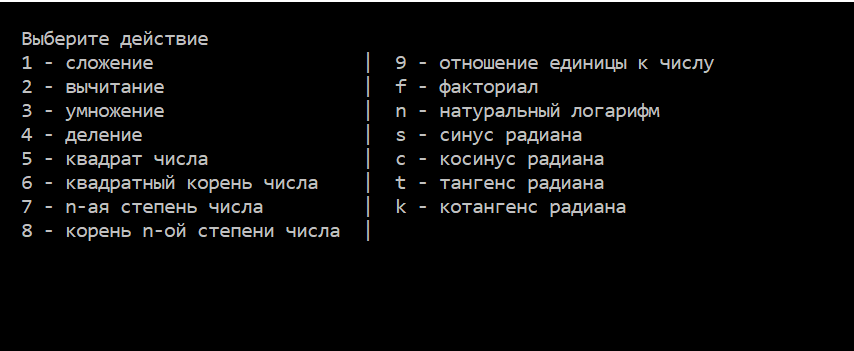


Рисунок 3.2 – окно выбора операций

Запись чисел осуществляется последовательным вводом переменных.

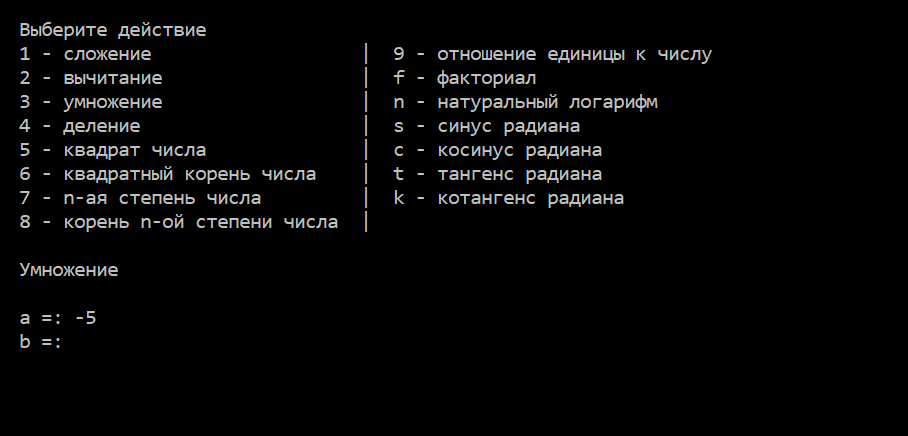


Рисунок 3.3 – ввод

Пример вычисления произведения:

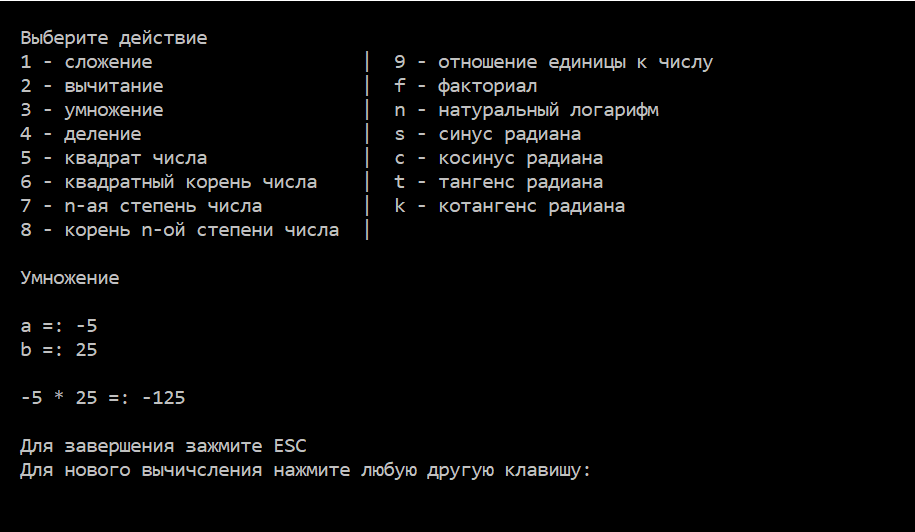


Рисунок 3.4 – вычисление произведения

В случае ввода некорректных символов программа укажет на это:

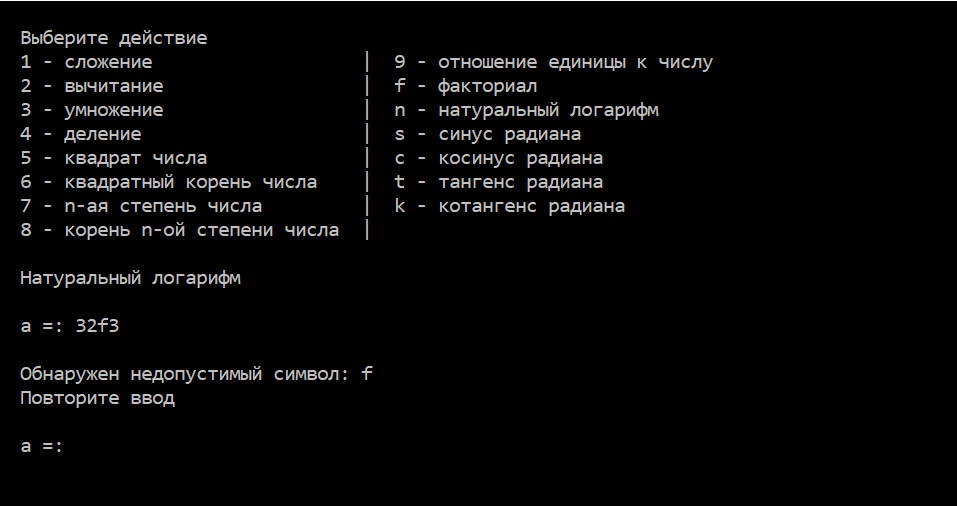


Рисунок 3.5 – некорректный ввод

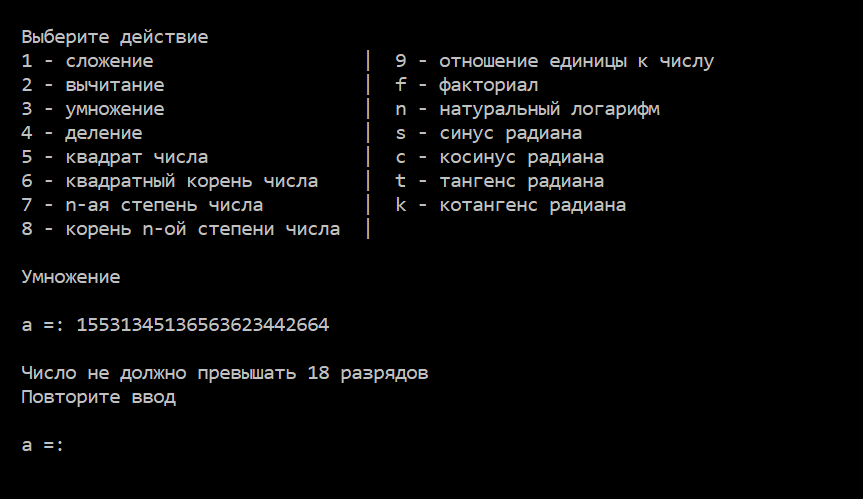


Рисунок 3.6 – предупреждение при вводе чрезмерно больших значений

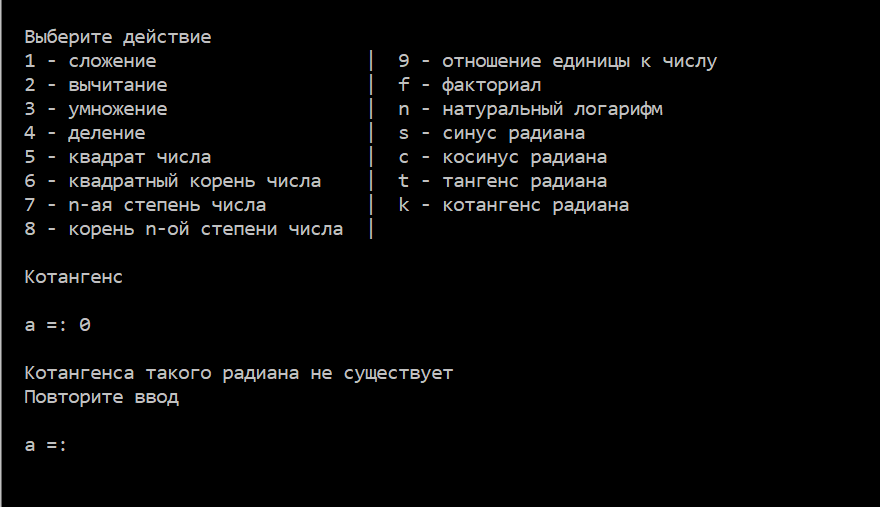


Рисунок 3.7 – предупреждение при вычислении недействительного значения

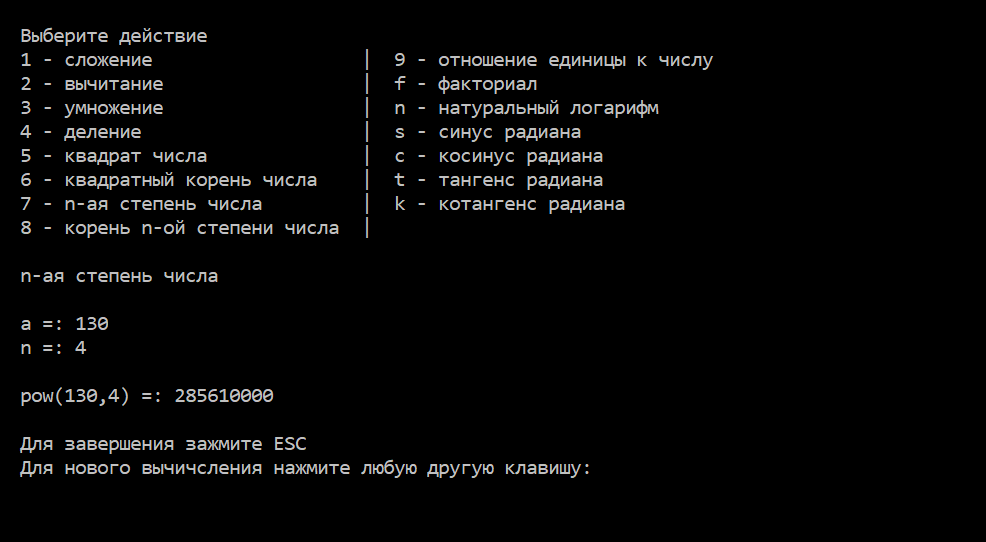
Пример корректного ввода: 

Рисунок 3.8 – вычисление значения n-ой степени числа.

Историю вычислений можно будет посмотреть в файле math\_history.doc, который появится в корневой папке.

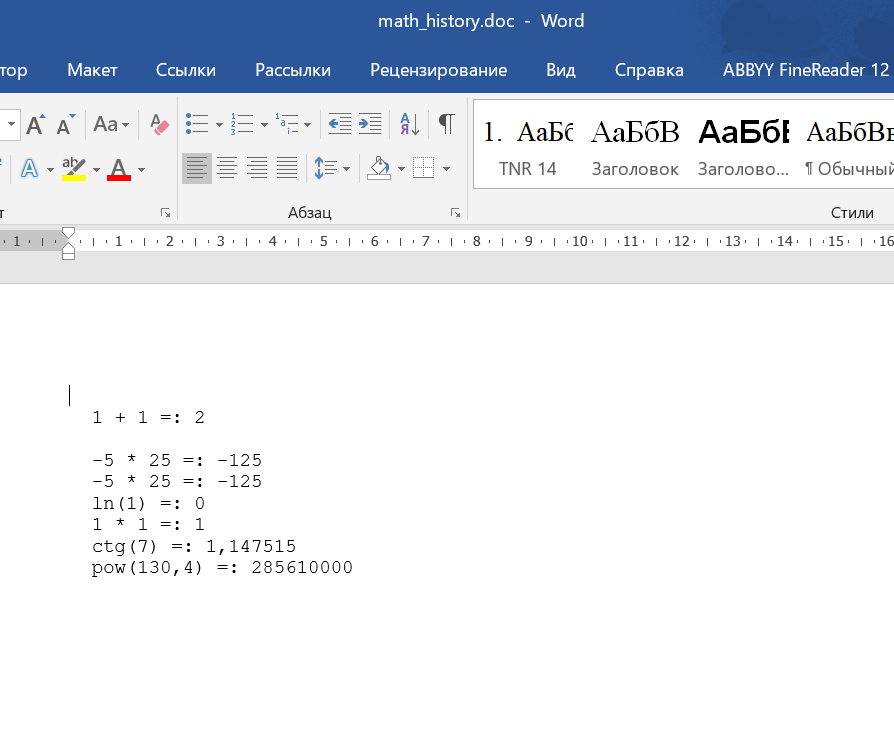


Рисунок 3.9 – история вычислений

# Заключение

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ. Были освоены способы обработки событий от клавиатуры. Усвоены принципы реализации интерфейса. Получены базовые навыки программирования на языке C и Ассемблер. Изучены возможность среды программирования Visual Studio 2017. Получены навыки отладки и тестирования программ.

В рамках выполнения курсовой работы была написана программа, выполняющая функции калькулятора. Программа не требует большого количества системных ресурсов, обеспечивает быструю обработку данных и мгновенный показ результата пользователю.

Программу можно улучшить, оптимизировав код для более быстрого выполнения функций.

# Список используемых источников

1. Н.Ф. Богаченко, Д.Н. Лавров, Ю.С. Ракицкий -Ассемблер в примерах и задачах

2. info-master.su/programming – Поляков А.В. – Основы программирования.

3. Х. Дейтел. П. Дейтел. Как программировать на C. 2007 г.

# Приложение А Листинг программы

Приложение А.1  
Файл «main.cpp»

#pragma once

#include "std\_math.h"

#include "zastavka.h"

char c = 0;

int zast = 0;

void main()

{

setlocale(LC\_ALL, "RUS");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

FILE \*file;

if(zast == 0)

zastavka();

zast = 1;

printf(" \n Выберите действие \n");

printf(" 1 - сложение | 9 - отношение единицы к числу \n");

printf(" 2 - вычитание | f - факториал \n");

printf(" 3 - умножение | n - натуральный логарифм \n");

printf(" 4 - деление | s - синус радиана \n");

printf(" 5 - квадрат числа | c - косинус радиана\n");

printf(" 6 - квадратный корень числа | t - тангенс радиана\n");

printf(" 7 - n-ая степень числа | k - котангенс радиана\n");

printf(" 8 - корень n-ой степени числа | \n");

c = \_getch(); // считывание символа из консоли без вывода символа

if (c == 49 || 50 || 51 || 52) { // ASCII символы чисел 1-9 - 49-58

\_asm {

mov bl, c

call stdmath

}

}

if (c == 53 || 54 || 55 || 56 ) {

\_asm {

mov bl, c

call pokazatel

}

}

if (c == 57 || 102 || 110 || 115 || 99 || 116 || 107 ) {

\_asm {

mov bl, c

call trig\_and\_rand

}

}

printf(" \n\n Для завершения зажмите ESC \n");

printf(" Для нового вычичсления нажмите любую другую клавишу: ");

c = \_getch();

if (c != 27) {

system("cls");

\_asm {

call main

}

}

if ((file = fopen("../../math\_history.doc", "a")) != 0) { // сдвиг в документе истории

fprintf(file, "\n");

fclose(file);

}

getch();

}

Приложение А.2  
Файл «std\_math.h»

#pragma once // защита от повторного включения

void stdmath();

void pokazatel();

void trig\_and\_rand();

#include<stdlib.h>

#include<stdio.h>

#include<conio.h>

#include<locale.h>

#include<windows.h>

#include < math.h >

Приложение А.3  
Файл «proverka.h»

#pragma once

int proverka(char vvod[999]);

int proverka\_uns\_int(char vvod[999]);

Приложение А.4  
Файл «zastavka.h»

void zastavka();

Приложение А.5  
Файл «std\_math.cpp»

#pragma once // защита от повторного включения

#include "std\_math.h"

#include "proverka.h"

void stdmath()

{

double a = 0, b = 0, y = 0;

long y\_long = 0; // проверка результата на целочисленность

long i = 0; // флаг неправильного ввода

char c = 0;

char vvod[99], str1[20], str2[20];

FILE \*file;

if((file = fopen("../../math\_history.doc", "r")) == 0 ) // если файла не существует, то создаём его

file = fopen("../../math\_history.doc", "w");

else

file = fopen("../../math\_history.doc", "a"); // иначе дописываем

\_asm {

mov c,bl

}

if (c == 49)

{

printf("\n Сложение\n ");

scan\_1: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_1;

strcpy(str1, vvod);

a = atof(vvod); //преобразование строки в число типа double

scan\_2: printf(" b =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_2;

strcpy(str2, vvod);

b = atof(vvod);

\_asm {

finit

fld a

fadd b

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n %s + %s =: %ld ", str1, str2, y\_long);

fprintf(file, "\n %s + %s =: %ld ", str1, str2, y\_long);

}

else

{

printf("\n %s + %s =: %lf ", str1, str2, y);

fprintf(file, "\n %s + %s =: %lf ", str1, str2, y);

}

}

if (c == 50) {

printf("\n Вычитание\n");

scan\_3: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_3;

strcpy(str1, vvod);

a = atof(vvod);

scan\_4: printf(" b =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_4;

strcpy(str2, vvod);

b = atof(vvod);

\_asm {

finit

fld a

fsub b

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n %s - %s =: %ld ", str1, str2, y\_long);

fprintf(file, "\n %s - %s =: %ld ", str1, str2, y\_long);

}

else

{

printf("\n %s - %s =: %lf ", str1, str2, y);

fprintf(file, "\n %s - %s =: %lf ", str1, str2, y);

}

}

if (c == 51) {

printf("\n Умножение\n ");

scan\_5: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_5;

strcpy(str1, vvod);

a = atof(vvod);

scan\_6: printf(" b =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_6;

strcpy(str2, vvod);

b = atof(vvod);

\_asm {

finit

fld a

fld b

fmul

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n %s \* %s =: %ld ", str1, str2, y\_long);

fprintf(file, "\n %s \* %s =: %ld ", str1, str2, y\_long);

}

else

{

printf("\n %s \* %s =: %lf ", str1, str2, y);

fprintf(file, "\n %s \* %s =: %lf ", str1, str2, y);

}

}

if (c == 52) {

printf("\n Деление\n ");

scan\_7: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_7;

strcpy(str1, vvod);

a = atof(vvod);

scan\_8: printf(" b =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_8;

strcpy(str2, vvod);

b = atof(vvod);

\_asm {

finit

fld a

fld b

fdiv

fst y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n %s / %s =: %ld ", str1, str2, y\_long);

fprintf(file, "\n %s / %s =: %ld ", str1, str2, y\_long);

}

else

{

printf("\n %s / %s =: %lf ", str1, str2, y);

fprintf(file, "\n %s / %s =: %lf ", str1, str2, y);

}

}

fclose(file);

}

void pokazatel() {

double a = 0, b = 0, y = 0, n2 = 0;

long y\_long = 0;

long i = 0, n = 0;

char c = 0;

char vvod[99], str1[20], str2[20];

FILE \*file;

if ((file = fopen("../../math\_history.doc", "r")) == 0) // если файла не существует, то создаём его

file = fopen("../../math\_history.doc", "w");

else

file = fopen("../../math\_history.doc", "a"); // иначе дописываем

\_asm {

mov c, bl

}

if (c == 53)

{

printf("\n Квадрат числа\n ");

scan\_1: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_1;

strcpy(str1, vvod);

a = atof(vvod);

\_asm {

finit

fld a

fst st(1) // копирует значение в st(1)

fmul

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n (%s)^2 =: %ld ", str1, y\_long);

fprintf(file, "\n (%s)^2 =: %ld ", str1, y\_long);

}

else

{

printf("\n (%s)^2 =: %lf ", str1, y);

fprintf(file, "\n (%s)^2 =: %lf ", str1, y);

}

}

if (c == 54)

{

printf("\n Квадратный корень числа\n ");

scan\_2: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_2;

strcpy(str1, vvod);

a = atof(vvod);

\_asm {

finit

fld a

fsqrt

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n sqrt(%s) =: %ld ", str1, y\_long);

fprintf(file, "\n sqrt(%s) =: %ld ", str1, y\_long);

}

else

{

printf("\n sqrt(%s) =: %lf ", str1, y);

fprintf(file, "\n sqrt(%s) =: %lf ", str1, y);

}

}

if (c == 55)

{

printf("\n n-ая степень числа\n ");

scan\_3: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_3;

strcpy(str1,vvod);

a = atof(vvod);

scan\_4: printf(" n =: ");

scanf("%s", vvod);

i = proverka\_uns\_int(vvod);

if (i == 1)

goto scan\_4;

n = atol(vvod);

\_asm {

finit

mov ecx, n

cmp ecx, 0

jne step2

fld1

jmp fin

step2:

fld a

cmp ecx, 1

je fin

dec ecx

npow:

fmul a

loop npow

fin:

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n pow(%s,%d) =: %ld ", str1, n, y\_long);

fprintf(file, "\n pow(%s,%d) =: %ld ", str1, n, y\_long);

}

else

{

printf("\n pow(%s,%d) =: %lf ", str1, n, y);

fprintf(file, "\n pow(%s,%d) =: %lf ", str1, n, y);

}

}

if (c == 56)

{

printf("\n корень n-ой степени числа\n ");

scan\_5: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_5;

strcpy(str1, vvod);

a = atof(vvod);

scan\_6: printf(" n =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_6;

strcpy(str2, vvod);

n2 = atof(vvod);

y = pow(a, 1/n2);

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n pow(%s,(1/%s)) =: %ld ", str1, str2, y\_long);

fprintf(file, "\n pow(%s,(1/%s)) =: %ld ", str1, str2, y\_long);

}

else

{

printf("\n pow(%s,(1/%s)) =: %lf ", str1, str2, y);

fprintf(file, "\n pow(%s,(1/%s)) =: %lf ", str1, str2, y);

}

}

fclose(file);

}

void trig\_and\_rand() {

double a = 0, b = 0, y = 0;

long y\_long = 0;

long i = 0, n = 0;

char c = 0;

char vvod[99], str1[20], str2[20];

FILE \*file;

int n2 = 1, n3 = 1; // для факториала счётчик

if ((file = fopen("../../math\_history.doc", "r")) == 0) // если файла не существует, то создаём его

file = fopen("../../math\_history.doc", "w");

else

file = fopen("../../math\_history.doc", "a"); // иначе дописываем

\_asm {

mov c, bl

}

if (c == 57) {

printf("\n Отношение единицы к числу\n ");

scan\_1: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_1;

strcpy(str1, vvod);

a = atof(vvod);

\_asm {

finit

fld1

fdiv a

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n 1/%s =: %ld ", str1, y\_long);

fprintf(file, "\n 1/%s =: %ld ", str1, y\_long);

}

else

{

printf("\n 1/%s =: %lf ", str1, y);

fprintf(file, "\n 1/%s =: %lf ", str1, y);

}

}

if (c == 102) {

printf("\n Факториал\n ");

scan\_2: printf("\n a =: ");

scanf("%s", vvod);

i = proverka\_uns\_int(vvod);

if (i == 1)

goto scan\_2;

strcpy(str1, vvod);

n2 = atoi(vvod);

n = 1;

y\_long = 1;

\_asm {

finit

mov ecx, n3

fild y\_long // celoe

nach:

cmp ecx, n2

jg fin

fimul n

inc n

inc ecx

jmp nach

fin:

fistp y\_long

}

if (y\_long > 0)

{

printf("\n !%s =: %ld ", str1, y\_long);

fprintf(file, "\n !%s =: %ld ", str1, y\_long);

}

else {

printf("\n Результат не помещается в разрядную сетку", vvod[i]);

printf("\n Повторите ввод\n", vvod[i]);

goto scan\_2;

}

}

if (c == 110) {

printf("\n Натуральный логарифм\n ");

scan\_3: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_3;

strcpy(str1, vvod);

a = atof(vvod);

if (a <= 0) {

printf("\n Число должно быть больше 0", vvod[i]);

printf("\n Повторите ввод\n", vvod[i]);

goto scan\_3;

}

\_asm {

finit

FLDLN2

fld a

FYL2X

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n ln(%s) =: %ld ", str1, y\_long);

fprintf(file, "\n ln(%s) =: %ld ", str1, y\_long);

}

else

{

printf("\n ln(%s) =: %lf ", str1, y);

fprintf(file, "\n ln(%s) =: %lf ", str1, y);

}

}

if (c == 115) {

printf("\n Синус\n ");

scan\_4: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_4;

strcpy(str1, vvod);

a = atof(vvod);

\_asm {

finit

fld a

fsin

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n sin(%s) =: %ld ", str1, y\_long);

fprintf(file, "\n sin(%s) =: %ld ", str1, y\_long);

}

else

{

printf("\n sin(%s) =: %lf ", str1, y);

fprintf(file, "\n sin(%s) =: %lf ", str1, y);

}

}

if (c == 99) {

printf("\n Косинус\n ");

scan\_5: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_5;

strcpy(str1, vvod);

a = atof(vvod);

\_asm {

finit

fld a

fcos

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n cos(%s) =: %ld ", str1, y\_long);

fprintf(file, "\n cos(%s) =: %ld ", str1, y\_long);

}

else

{

printf("\n cos(%s) =: %lf ", str1, y);

fprintf(file, "\n cos(%s) =: %lf ", str1, y);

}

}

if (c == 116) {

printf("\n Тангенс\n ");

scan\_6: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_6;

strcpy(str1, vvod);

a = atof(vvod);

\_asm {

finit

fld a

FPTAN

fstp y

fstp y

}

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n tg(%s) =: %ld ", str1, y\_long);

fprintf(file, "\n tg(%s) =: %ld ", str1, y\_long);

}

else

{

printf("\n tg(%s) =: %lf ", str1, y);

fprintf(file, "\n tg(%s) =: %lf ", str1, y);

}

}

if (c == 107) {

printf("\n Котангенс\n ");

scan\_7: printf("\n a =: ");

scanf("%s", vvod);

i = proverka(vvod);

if (i == 1)

goto scan\_7;

strcpy(str1, vvod);

a = atof(vvod);

if(tan(a) == 0)

{

printf("\n Котангенса такого радиана не существует", vvod[i]);

printf("\n Повторите ввод\n", vvod[i]);

goto scan\_7;

}

y = 1 / tan(a);

y\_long = y;

if ((y - y\_long) == 0)

{

printf("\n ctg(%s) =: %ld ", str1, y\_long);

fprintf(file, "\n ctg(%s) =: %ld ", str1, y\_long);

}

else

{

printf("\n ctg(%s) =: %lf ", str1, y);

fprintf(file, "\n ctg(%s) =: %lf ", str1, y);

}

}

fclose(file);

}

Приложение А.6  
Файл «proverka.cpp»

#include "proverka.h"

#include "std\_math.h"

int proverka(char vvod[999])

{

int i = 0;

int i1 = 0;

while (vvod[i] != 0)

{

if (i >= 19)

{

printf("\n Число не должно превышать 18 разрядов", vvod[i]);

printf("\n Повторите ввод\n", vvod[i]);

i1 = 1;

}

if ((vvod[i] <= 47 || vvod[i] >= 58) && vvod[i] != 44 && vvod[i] != 45)

{

printf("\n Обнаружен недопустимый символ: %c", vvod[i]);

printf("\n Повторите ввод\n", vvod[i]);

i1 = 1;

}

if (i1 == 1)

break;

i++;

}

return i1;

}

int proverka\_uns\_int(char vvod[999])

{

int i = 0;

int i1 = 0;

while (vvod[i] != 0)

{

if (i >= 19)

{

printf("\n Число не должно превышать 18 разрядов", vvod[i]);

printf("\n Повторите ввод\n", vvod[i]);

i1 = 1;

}

if ((vvod[i] <= 47 || vvod[i] >= 58))

{

printf("\n Обнаружен недопустимый символ: %c", vvod[i]);

printf("\n Повторите ввод\n", vvod[i]);

i1 = 1;

}

if (i1 == 1)

break;

i++;

}

return i1;

}

Приложение А.7  
Файл «zastavka.cpp»

#include "zastavka.h"

#include <stdio.h>

#include <conio.h>

#include <windows.h>

void zastavka()

{

char c = 0;

printf(" \n Курсовая работа \n");

printf(" Выполнил студент группы 19ВВ-2 \n");

printf(" Павлов Алексей \n");

c = \_getch();

system("cls");

}

Приложение B  
Схемы программы

