СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc448998187)

[1 ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ОСУЩЕСТВЛЯЮЩЕГО УМНОЖЕНИЕ МАТРИЦ 4](#_Toc448998188)

[1.1 Файловый ввод-вывод в С++ 4](#_Toc448998189)

[1.2 Двумерные массивы 6](#_Toc448998190)

[1.5 Анализ задания на курсовой проект 8](#_Toc448998191)

[1.6 Формулировка задания в человеко-читаемом формате 9](#_Toc448998192)

[1.7 Построение блок-схемы алгоритма задания на курсовой проект 10](#_Toc448998193)

[1.8 Выбор языка программирования и среды разработки 11](#_Toc448998194)

[2 ПРАКТИЧЕСКИЙ РАЗДЕЛ. ПРОГРАММИРОВАНИЕ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 13](#_Toc448998195)

[2.1 Перевод алгоритма в программный код 13](#_Toc448998196)

[2.2 Тестирование готового кода инструментом «Анализ кода» 14](#_Toc448998197)

[2.3 Опции компилятора при сборке программы 16](#_Toc448998198)

[2.4 Обработка ошибок и исключительных ситуаций в коде 17](#_Toc448998199)

[3 ОРГАНИЗАЦИОННО-ТЕХНИЧЕСКИЙ РАЗДЕЛ 18](#_Toc448998200)

[3.1 Организация тестирования программного обеспечения 18](#_Toc448998201)

[3.2 Лицензирование полученного программного обеспечения 19](#_Toc448998202)

[3.3 Составление файла помощи 20](#_Toc448998203)

[3.4 Расчет ресурсов, затраченных на написание программного обеспечения 21](#_Toc448998204)

[ЗАКЛЮЧЕНИЕ 23](#_Toc448998205)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24](#_Toc448998206)

[ПРИЛОЖЕНИЯ 25](#_Toc448998207)

# ВВЕДЕНИЕ

В процессе выполнения курсового проекта необходимо разработать прикладное программное обеспечение, осуществляющие умножение матриц произвольного размера с использованием динамической памяти и указателей.

Работа над курсовым проектом будет производиться в интегрированной среде разработки Microsoft Visual Studio 2013.

Кроме программного продукта, так же необходимо создать файл помощи по полученному продукту, выбрать методику его испытания и, соответственно, произвести его испытание (тестирование).

# 1 ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ОСУЩЕСТВЛЯЮЩЕГО УМНОЖЕНИЕ МАТРИЦ

# 1.1 Файловый ввод-вывод в С++

Файловый ввод-вывод позволяет считывать данные из множества различных файлов, и выводить их в другие (или те же) файлы.

На самом деле, файловый ввод-вывод ничем не отличается от консольного. За единственным исключением – если данные читаются из файла, то в любой момент можно вернуться к началу файла и считать все заново.

Для того, чтобы в C++ работать с файлами, необходимо подключить заголовочный файл *fstream*:

*#include <fstream>*

После этого можно объявлять объекты, привязанные к файлам: для чтения данных из файла используются объекты типа ifstream (аббревиатура от input file stream, для записи данных в файл используются объекты типа ofstream (output file stream). Например

*ifstream in; // Поток in будем использовать для чтения*

*ofstream out; // Поток out будем использовать для записи*

Чтобы привязать тот или иной поток к файлу (открыть файл для чтения или для записи) используется метод *open*, которому необходимо передать параметр – текстовую строку, содержащую имя открываемого файла.

*in.open("input.txt");*

*out.open("output.txt");*

После открытия файлов и привязки их к файловым потокам, работать с файлами можно так же, как со стандартными потоками ввода-вывода *cin* и *cout*. Например, чтобы вывести значение переменной x в поток *out* используются следующая операция

*out<<x;*

А чтобы считать значение переменной из потока *in*

*in>>x;*

Для закрытия ранее открытого файла используется метод close() без аргументов:

*in.close();*

*out.close();*

Закрытый файловый поток можно переоткрыть заново при помощи метода *open*, привязав его к тому же или другому файлу.

При считывании данных из файла может произойти ситуация достижения конца файла (end of file, сокращенно EOF). После достижения конца файла никакое чтение из файла невозможно. Для того, чтобы проверить состояние файла, необходимо вызвать метод eof(). Данный метод возвращает true, если достигнут конец файла или false, если не достигнут. Кроме того, состояние файлового потока можно проверить, если просто использовать идентификатор потока в качестве логического условия:

*if (in)*

*{*

*}*

Также можно использовать в качестве условия результат, возвращаемой операцией считывания. Если считывание было удачным, то результат считается истиной, а если неудачным – ложью. Например, организовать считывание последовательности целых чисел можно так:

*int d;*

*while(in>>d)*

*{*

*}*

А организовать считывание файла построчно (считая, что строка заканчивается символом перехода на новую строку) так:

*string S;*

*while ( getline(in,S))*

*{*

*}*

# 1.2 Двумерные массивы

С позволяет создавать многомерные массивы. Простейшим видом многомерного массива является двумерный массив. Двумерный массив - это массив одномерных массивов. Двумерный массив объявляется следующим образом:

*тип имя\_массива [размер второго измерения][размер первого измерения];*

Следовательно, для объявления двумерного массива целых с размером 10 на 20 следует написать:

*int d[10] [20] ;*

Двумерные массивы сохраняются в виде матрицы, где первый индекс отвечает за строку, а второй -за столбец. Это означает, что правый индекс изменяется быстрее левого, если двигаться по массиву в порядке расположения элементов в памяти.

# 1.3 Динамический массив в С++

Динамическое выделение памяти необходимо для эффективного использования памяти компьютера. Например, мы написали какую-то программу, которая обрабатывает массив. При написании данной программы необходимо было объявить массив, то есть задать ему фиксированный размер (к примеру, от 0 до 100 элементов). Тогда данная программа будет не универсальной, ведь может обрабатывать массив размером не более 100 элементов. А если нам понадобятся всего 20 элементов, но в памяти выделится место под 100 элементов, ведь объявление массива было статическим, а такое использование памяти крайне не эффективно.

В С++ операции new и delete предназначены для динамического распределения памяти компьютера. Операция new выделяет память из области свободной памяти, а операция delete высвобождает выделенную память. Выделяемая память, после её использования должна высвобождаться, поэтому операции new и delete используются парами. Даже если не высвобождать память явно, то она освободится ресурсами ОС по завершению работы программы.

# 1.4 Выделение памяти с помощью указателя.

Указатель – переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в С++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных, для того, чтобы его присвоить указателю.

Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере. Ведь программист заранее не может знать, какого размера нужен массив тому или иному пользователю, в таком случае используется динамическое выделение памяти под массив. Любой указатель необходимо объявить перед использованием, как и любую переменную.

# 1.5 Анализ задания на курсовой проект

При выполнении задания на курсовой проект должно получится прикладное программное обеспечение, выполняющее следующие функции:

1 получение с клавиатуры текстовой информации и запись ее в промежуточный буфер;

2 функцию, реализующую работу с двумерным динамическим массивом;

3 функцию умножения матриц произвольного размера;

4 функцию с использованием динамической памяти и указателей.

# 1.6 Формулировка задания в человеко-читаемом формате

Сформулируем задание на курсовой проект в человеко-читаемом формате. Для этого, опишем действия пользователя и реакцию программы на эти действия, приводящую к получению результата.

Запуск программы производится открытием исполняемого файла, следовательно, язык используемый для программирования – компилируемый язык. После запуска программа создает интерфейс пользователя – консоль и начинает принимать текст с клавиатуры. Сигналом окончания приема текста служит нажатие клавиши “Enter”.

После этого, открываются два файла для записи и производится вычисление матриц.

Программа должна иметь русскоязычный интерфейс.

# 1.7 Построение блок-схемы алгоритма задания на курсовой проект

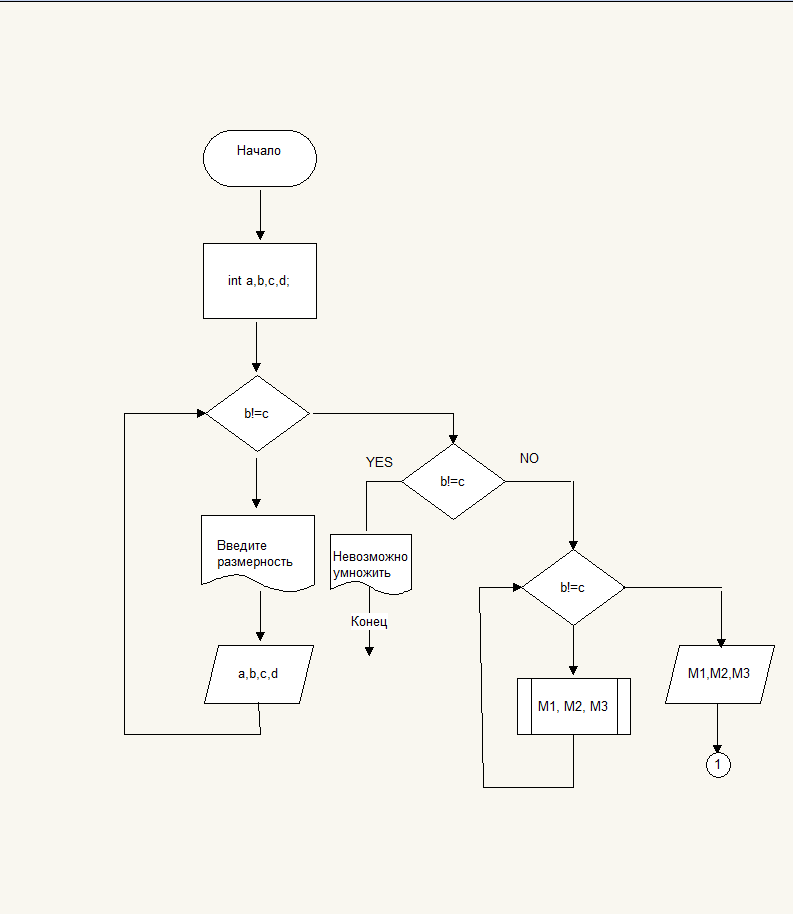


Рисунок 1 – Блок-схема алгоритма

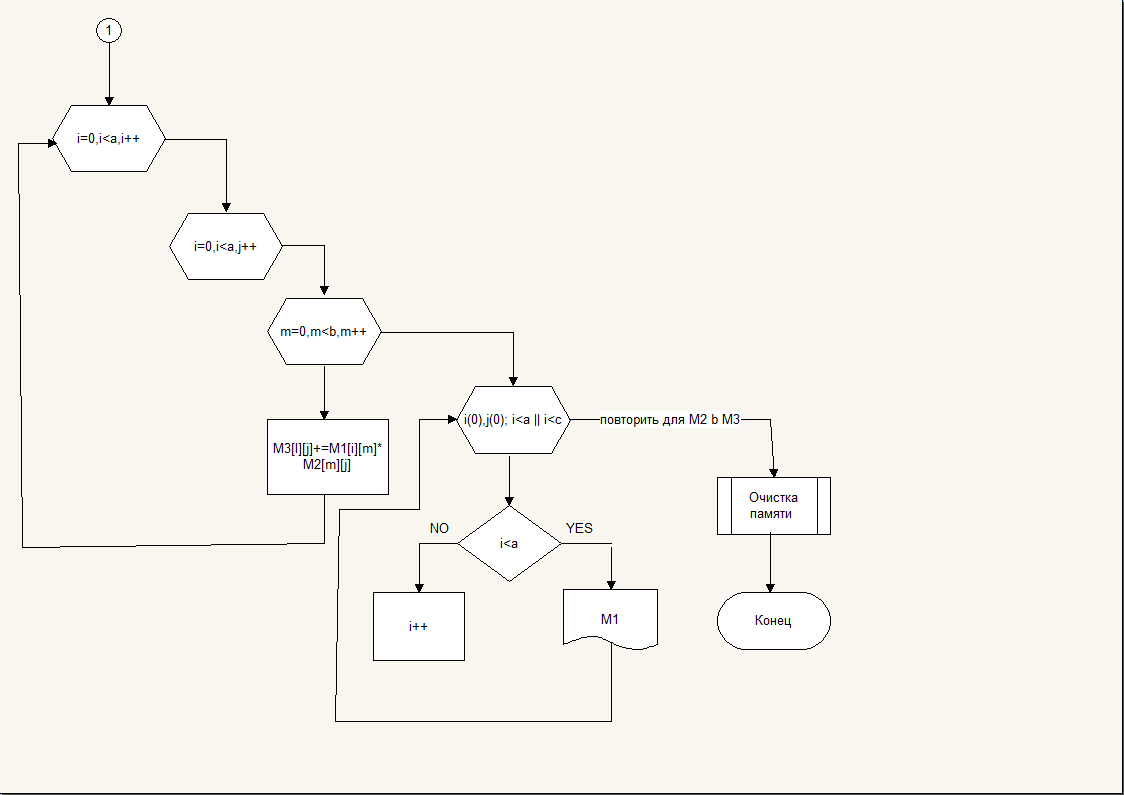


Рисунок 2 – Продолжение блок-схемы алгоритма

# 

# 1.8 Выбор языка программирования и среды разработки

Для решения поставленной задачи была выбрана интегральная среда разработки (ИСР) Microsoft Visual Studio 2013 Express for Windows 7 поскольку:

1 в ней доступны широкие возможности языка C++;

2 используется широким кругом программистов-профессионалов, разрабатывающих в ОС Windows;

3 она имеет все необходимые средства и инструменты для разработки приложений любой сложности;

4 удобна в использовании, гибка в настройке;

5 создает очень эффективный код;

6 в ней имеется возможность использования готовых библиотек классов;

7 позволяет импортировать исходный код других ОС;

Так как ИСР выбрана Microsoft Visual Studio 2013 Express for Windows 7, соответственно языком программирования выбран C++, поскольку:

1 родной язык для создания приложений;

2 подлинная объектная ориентированность (всякая языковая сущность претендует на то, чтобы быть объектом);

3 компонентно-ориентированное программирование.

Так же в процессе выполнения лабораторных работ по дисциплинам «Прикладное программирование» и «Теория алгоритмов» использовался язык С++, поэтому программировать на нем мне оказалось проще чем на чистом языке Си или C#.

В качестве языка программирования будет использоваться С++.

# 2 ПРАКТИЧЕСКИЙ РАЗДЕЛ. ПРОГРАММИРОВАНИЕ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

# 2.1 Перевод алгоритма в программный код

Воспользуемся приемами структурного программирования для создания программного кода.

Структурное программирование — одно из крупнейших достижений в технологии программирования. Хотя самое общее и довольно смутное представление о структурном программировании имеется почти у всех, общепринятого четкого его определения нет. Структурное программирование ставит своей целью писать программы минимальной сложности, заставить программиста мыслить ясно, облегчить восприятие программы. Структурное программирование основано на модульной структуре программного продукта и типовых управляющих структурах алгоритмов обработки данных различных программных модулей.

Типы управляющих структур:

– последовательность;

– альтернатива (условие выбора);

– цикл.

Распространены две методики (стратегии) разработки программ, относящиеся к структурному программированию:

– программирование «сверху вниз»;

– программирование «снизу вверх».

Программирование «сверху вниз», или нисходящее программирование – это методика разработки программ, при которой разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, заканчивающаяся детальной программой.

Сначала выделяется несколько подпрограмм, решающих самые глобальные задачи (например, инициализация данных, главная часть и завершение), потом каждый из этих модулей детализируется на более низком уровне, разбиваясь в свою очередь на небольшое число других подпрограмм, и так происходит до тех пор, пока вся задача не окажется реализованной.

В данном случае программа конструируется иерархически - сверху вниз: от главной программы к подпрограммам самого нижнего уровня, причем на каждом уровне используются только простые последовательности инструкций, циклы и условные разветвления.

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных операторов и переменных. Кроме того, появляется возможность некоторые подпрограммы не реализовывать сразу, а временно откладывать, пока не будут закончены другие части. Например, если имеется необходимость вычисления сложной математической функции, то выделяется отдельная подпрограмма такого вычисления, но реализуется она временно одним оператором, который просто присваивает заранее выбранное значение (например, 5). Когда все приложение будет написано и отлажено, тогда можно приступить к реализации этой функции.

Программирование «снизу вверх», или восходящее программирование – это методика разработки программ, начинающаяся с разработки подпрограмм (процедур, функций), в то время когда проработка общей схемы не закончилась.

Такая методика является менее предпочтительной по сравнению с нисходящим программированием так как часто приводит к нежелательным результатам, переделкам и увеличению времени разработки.

# 2.2 Тестирование готового кода инструментом «Анализ кода»

Средство анализа кода C/C++ предоставляет разработчикам сведения о возможных дефектах в исходном коде C/C++. К наиболее распространенным ошибкам кодирования, которые обнаруживает данное средство, относятся переполнение буфера, неинициализированная память, разыменования пустых указателей, а также утечки памяти и ресурсов.

Для более удобного использования разработчиками средство анализа полностью интегрировано в интерфейс IDE Visual Studio. Во время построения все предупреждения, вырабатываемые для исходного кода, отображаются в списке ошибок. При этом можно перейти к исходному коду, вызвавшему предупреждение, и просмотреть дополнительные сведений о причине и возможных способах устранения проблемы (рисунок 3).

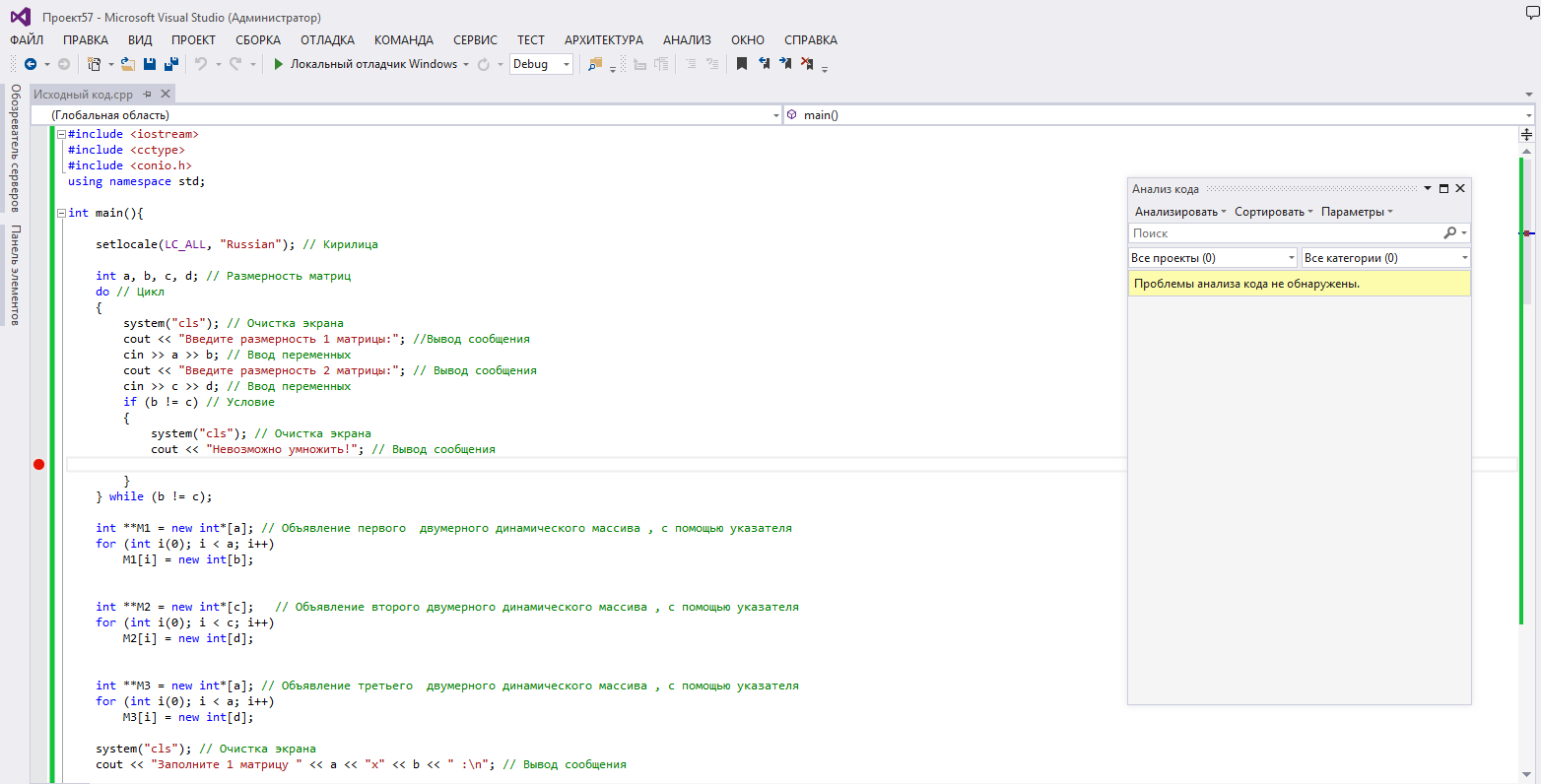


Рисунок 3 – Запуск анализатора кода

Вывод анализатора:

*1>Перестроение всех файлов начато: проект: Проект57, Конфигурация: Debug Win32*

*1> Исходный код.cpp*

*1>Проект57.vcxproj->c:\mydocuments\visualstudio 2013\Projects\Проект57\Debug\Проект57.exe*

*Перестроение всех: успешно: 1, с ошибками: 0, пропущено: 0*

*При анализе кода ошибок не обнаружено.*

# 2.3 Опции компилятора при сборке программы

Для получения работающего на всех платформах исполняемого файла, нужно произвести две операции – сборку проекта и его компиляцию. Воспользуемся встроенным в Microsoft Visual Studio 2013 компоновщиком, позволяющим собрать exe файл.

Во-первых, установит статическую линковку подключаемых библиотек (рисунок 4):

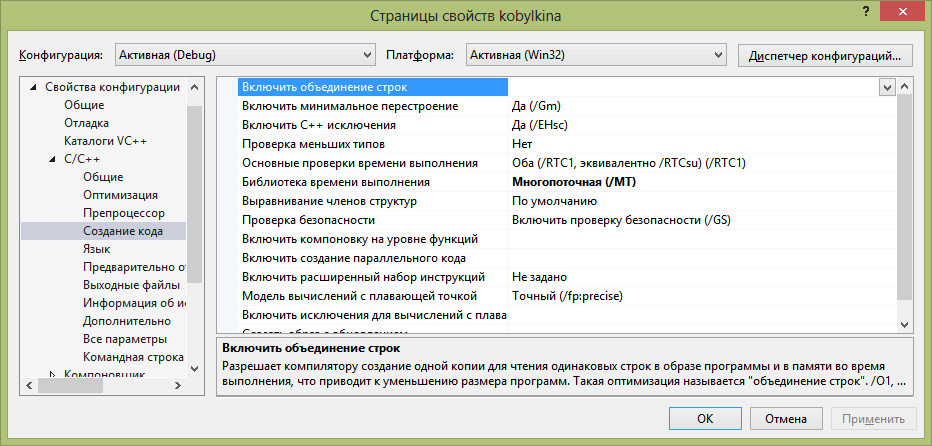


Рисунок 4 – Статическая линковка

Во-вторых, включим релизную сборку (рисунок 5):

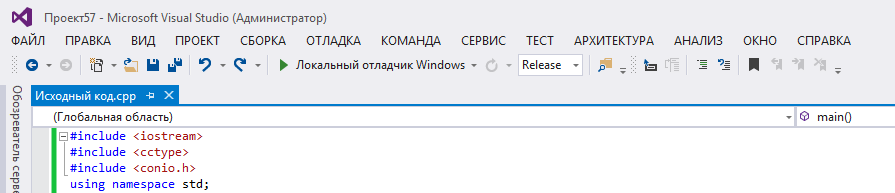


Рисунок 5 – Релизная сборка

После сборки и компиляции проекта получим исполняемый файл, работающей на любой платформе.

# 2.4 Обработка ошибок и исключительных ситуаций в коде

Рассмотрим ошибки, которые могут возникать при работе программы.

Исключение представляет собой ошибку или непредвиденную ситуацию, происходящую в процессе выполнения программы. На сегодняшний день исключения являются основным способом обработки ошибок, возникающих в программах, и в языке C++ их обработка основывается на четырех ключевых словах: *try, catch, throw и finally*.

*try*

*{ блок команд, в котором может возникнуть ошибка*

*}*

*[catch [(тип\_исключения имя\_исключения)]*

*{ блок обработки исключения*

*}]*

*[finally*

*{ команды, которые выполняются в любом случае*

*}]*

Такая конструкция выполняет команды, расположенные в блоке *try*, и, если внутри происходит исключение, передает управление на блок *catch*, которому возникшее исключение передается в качестве параметра. В любом случае (произошло исключение или нет) после выполнения блока *try* вызывается блок команд *finally*, в котором расположены команды освобождения ресурсов.

Рассмотрим ошибки, которые могут возникнуть при работе программы. Наиболее вероятной ошибкой, которая может возникнуть при работе программы, будет являться ошибка переполнения буфера при вводе данных.

Обработку этой ошибки подробно рассмотрим в третей части курсового проекта.

# 3 ОРГАНИЗАЦИОННО-ТЕХНИЧЕСКИЙ РАЗДЕЛ

# 3.1 Организация тестирования программного обеспечения

Чтобы создать качественное приложение, нужно учитывать мнение пользователей. Их отзывы лучше получать регулярно и своевременно.

Если планируется проводить альфа- и бета-тестирование, лучше всего начать с маленькой группы альфа-тестировщиков – возможно, одногруппников и руководителя курсового проекта, или других разработчиков. После этого можно приступать к бета-тестированию с участием большего количества пользователей.

Запустим программу со стандартным входным потоком (рисунок 6):

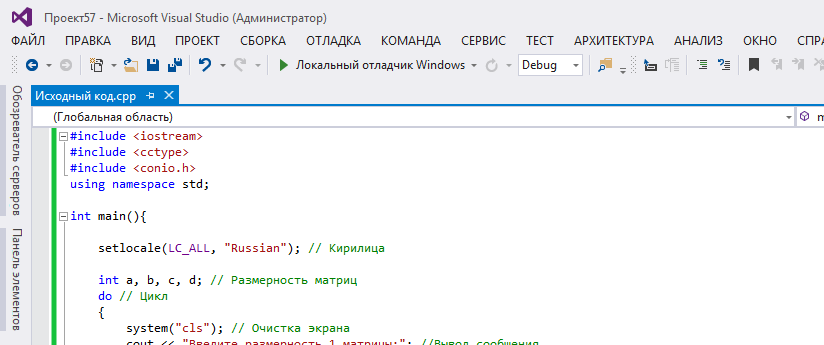


Рисунок 6 – Запуск программы со стандартными параметрами

На рисунке 7 результат работы программы.

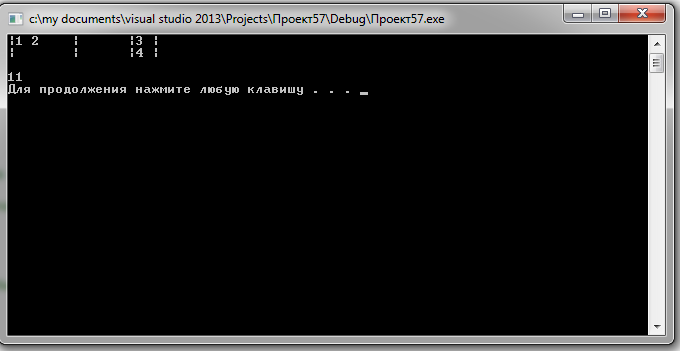


Рисунок 7 – Созданные в результате работы программы файлы

Произведем переполнение буфера при запуске программы. Для этого введем в программу строку длиной более 255 символов (рисунок 8).

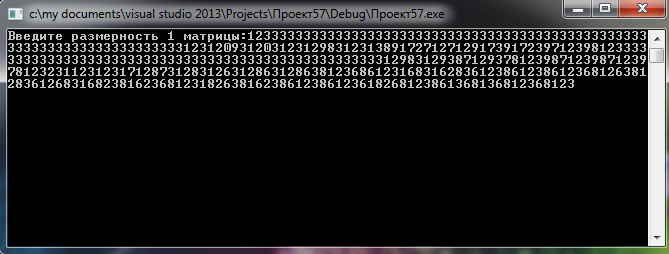


Рисунок 8 – Организация переполнения

На рисунке 9 представлен вывод ошибки переполнения буфера.

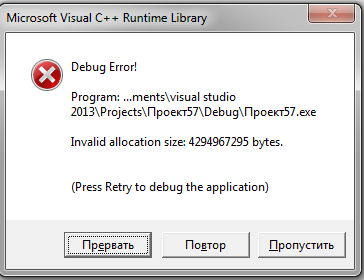


Рисунок 9 – Ошибка переполнения

Как видим, при переполнении буфера возникает необработанное исключение, приводящее к аварийному завершению программы.

# 3.2 Лицензирование полученного программного обеспечения

Так как курсовой проект состоит не только из исходного текста, но и из текстовых документов, то к нему можно применить «Лицензию для работ практического применения, которые не являются ни программами, ни документацией».

Применим к нашему проекту «Лицензию научного конструирования (Design Science License, DSL)».

Полный текст лицензии на английском языке будет приведен в приложении. Кроме этого, обязательной частью лицензии будет отказ от ответственности применения программы и исходных кодов в следующем виде:

«Программа и исходный код распространяются в виде «как есть», автор не несет ответственности ни за прямые, ни за косвенный убытки, которые могут возникнуть в результате использования программы».

# 3.3 Составление файла помощи

Для работы с файлами помощи очень удобно использовать соответствующие программы. Для курсового проекта будет использоваться программа HelpNdoc.

Основу интерфейса программы HelpNDoc составляет лента, на которой расположены все инструменты программы (рисунок 10).

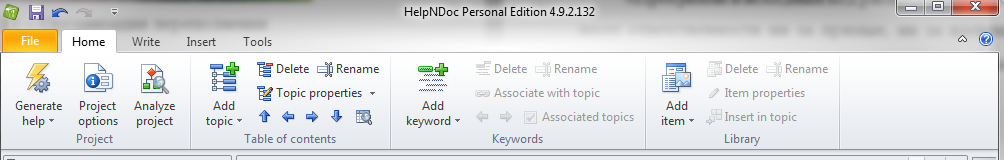


Рисунок 10 – Лента программы HelpNdoc.

Раздел project - Создание документации и настройки проекта.

Раздел table of contents - Управляют свойства темы и оглавление.

Раздел keywords - Управляют иерархией ключевых слов и ассоциацией.

Раздел library - Управляют библиотекой.

На рисунке 11 представлены скриншоты создания файла помощи.

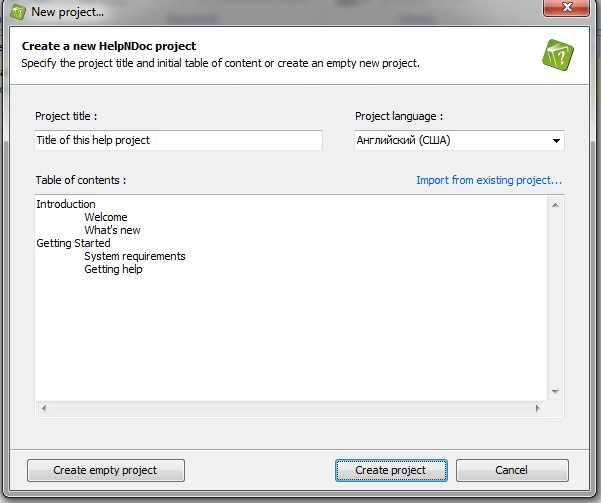


Рисунок 11 – Создание структуры справки.

# 3.4 Расчет ресурсов, затраченных на написание программного обеспечения

Рассчитаем ресурсы, потраченные на написание программного обеспечения. К примеру, произведем расчет по количеству строк исходного кода.

В ПРИЛОЖЕНИИ 1 содержится весь исходный код проекта, и он составляет 146 строк. Согласно отраслевым нормам, среднее количество отработанных и отлаженных строк кода для инженера-программиста составляет 15-20 в день. Возьмем минимальное значение – 15 строк. Соответственно, понадобиться 146/15 = 9,7 дней на написание и отладку программы.

На рисунке 12 представлена динамика изменения средней зарплаты программиста в Орле составляет 45000 рублей в месяц (по данным http://orel.trud.com/salary/893/3320.html).

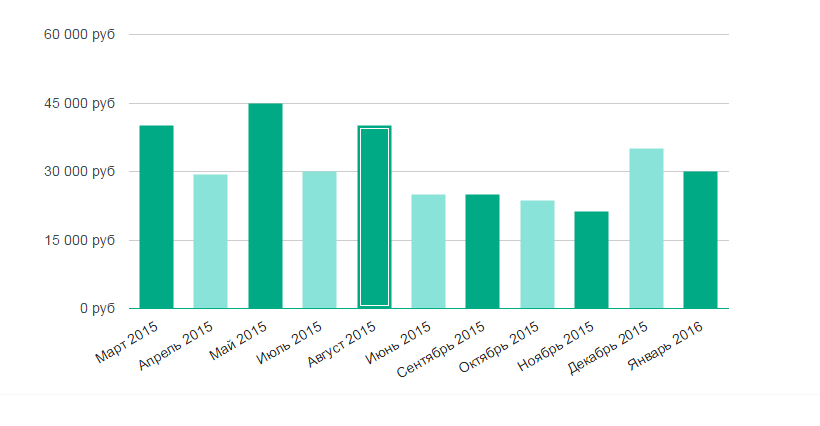


Рисунок 12 – Динамика изменения зарплат программиста в Орле

Соответственно, профессиональный программист заработал бы на написании этой программы

(1)

19840,90 рублей без учета налогов и отчислений.

# ЗАКЛЮЧЕНИЕ

При работе над курсовым проектом были освоены области математики, изучающие алгоритмы хэширования, написана функция хэширования.

Подробно изучен процесс записи данных на диск, отформатированный в файловую систему NTFS, изучены библиотеки управления потоками в С++.

Приобретен практический опыт работы с интегрированной средой разработки Microsoft Visual Studio 2013, в ней написан, проверен, проанализирован и собран исходный код приложения.

Полученное приложение протестировано, к нему составлен файл помощи.

Произведет расчёт ресурсов на написание программного продукта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Канер, С. Тестирование программного обеспечения[Текст]: Пер. с англ./С. Канер, Д. Фолк, Кек Нгуен [и др.]-Киев: ДиаСофт, 2010. – 544 с.

2 Рудаков, А.В. Технология разработки программных продуктов[Текст]: Учеб. Пособие для студ. Сред. Проф. Образования./А.В. Рудаков.-Москва, Издательский центр «Академия», 2015. – 192 с.

3 Фридман, А.Л. Основы объектно-ориентированной разработки программных систем[Текст] / Л.И. Фридман - Москва, Финансы и статистика, 2010. – 192 с.

4 Интегрированные среды разработки / Электронный ресурс http://www.studfiles.ru/dir/cat32/subj1259/file11616/view106028.html

5 Плис, А.И. Математический практикум для инженеров и программистов[Текст]: Учеб. пособие. – 2-е изд. перераб. и доп. / А.И. Плис, Н.А. Сливина. – Москва, Финансы и Статистика, 2013. – 565 с.

6 Культин, Н.Б. C/C++ в задачах и примерах[Текст]: 2-е изд., перераб. и доп. (+CD) / Н.Б. Культин - И: «ЛАНЬ»,2012 г.

7 Кузнецов, М.В. C++. Мастер-класс в задачах и примерах [Текст]: (+ CD) / М.В. Кузнецов. - И: «ЛАНЬ», 2012 г.

# **ПРИЛОЖЕНИЯ**

ПРИЛОЖЕНИЕ 1

Исходный код

1. #include <iostream>
2. #include <cctype>
3. #include <conio.h>
4. using namespace std;
5. int main(){
6. setlocale(LC\_ALL, "Russian"); // Кирилица
7. int a, b, c, d; // Размерность матриц
8. do // Цикл
9. {
10. system("cls"); // Очистка экрана
11. cout << "Введите размерность 1 матрицы:"; //Вывод сообщения
12. cin >> a >> b; // Ввод переменных
13. cout << "Введите размерность 2 матрицы:"; // Вывод сообщения
14. cin >> c >> d; // Ввод переменных
15. if (b != c) // Условие
16. {
17. system("cls"); // Очистка экрана
18. cout << "Невозможно умножить!"; // Вывод сообщения
19. \_getch();
20. }
21. } while (b != c);
22. int \*\*M1 = new int\*[a]; // Объявление первого двумерного динамического массива , с помощью указателя
23. for (int i(0); i < a; i++)
24. M1[i] = new int[b];
25. int \*\*M2 = new int\*[c]; // Объявление второго двумерного динамического массива , с помощью указателя
26. for (int i(0); i < c; i++)
27. M2[i] = new int[d];
28. int \*\*M3 = new int\*[a]; // Объявление третьего двумерного динамического массива , с помощью указателя
29. for (int i(0); i < a; i++)
30. M3[i] = new int[d];
31. system("cls"); // Очистка экрана
32. cout << "Заполните 1 матрицу " << a << "х" << b << " :\n"; // Вывод сообщения
33. for (int i(0); i < a; i++) // Заполнение первой матрицы
34. {
35. for (int j(0); j < b; j++)
36. {
37. cin >> M1[i][j]; // Ввод матриц
38. }
39. }
40. for (int i(0); i < a; i++) //
41. {
42. for (int j(0); j < d; j++)
43. {
44. M3[i][j] = 0; //обнуление третьей матрицы
45. }
46. }
47. system("cls"); // Очистка экрана
48. cout << "Заполните 2 матрицу " << c << "х" << d << " :\n"; // Вывод сообщения
49. for (int i(0); i < c; i++) // Заполнение второй матрицы
50. {
51. for (int j(0); j < d; j++)
52. {
53. cin >> M2[i][j]; // Ввод матриц
54. }
55. }
56. system("cls"); // Очистка экрана
57. for (int i = 0; i < a; i++)
58. {
59. for (int j = 0; j < d; j++)
60. {
61. for (int m = 0; m < b; m++)
62. {
63. M3[i][j] += M1[i][m] \*M2[m][j]; // Нахождение элемента каждой строки матрицы
64. }
65. }
66. }
67. for (int i(0), l(0); i < a || l < c;) // Проверка условий
68. {
69. cout << "|";
70. if (i < a) // Условие
71. {
72. for (int j(0); j < b; j++)
73. {
74. cout << M1[i][j] << " ";
75. }
76. i++; // Итерация
77. }
78. cout << "\t| |"; // Вывод сообщения
79. if (l < c) // Условие
80. {
81. for (int j(0); j < d; j++)
82. {
83. cout << M2[l][j] << " ";
84. }
85. l++; // Итерация
86. }
87. cout << "|";
88. cout << endl; // Переход на новую строку
89. }
90. cout << endl; // Переход на новую строку
91. for (int i(0); i < a; i++)
92. {
93. for (int j(0); j < d; j++) // Вывод третьей матрицы
94. {
95. cout << M3[i][j] << ' ';
96. }
97. cout << endl;
98. }
99. for (int i(0); i < a; i++) // Освобождение первой матрицы
100. delete[] M1[i];
101. delete[] M1;
102. for (int i(0); i < c; i++) // Освобождение второй матрицы
103. delete[] M2[i];
104. delete[] M2;
105. for (int i(0); i < a; i++) // Освобождение третьей матрицы
106. delete[] M3[i];
107. delete[] M3;
108. \_getch();
109. return 0;
110. }