# Глава 3. Программная реализация выбранного алгоритма

## 3.1 Описание платформы .NET и языка C#

Платформа Microsoft .NET(и связанный с ней язык программирования C#) впервые была представлена примерно в 2002г. и быстро стала одной из основных современных сред разработки программного обеспечения.

До того, как компания Microsoft выпустила язык C# и платформу .NET, разработчики программного обеспечения, создававшие приложения для операционных систем семейства Windows, часто применяли модель программирования COM. Технология COM (Component Object Model – модель компонентных объектов) позволяла строить библиотеки, которые можно было использовать в различных языках программирования.

Когда говорят C#, нередко имеют в виду технологии платформы .NET (WPF, ASP.NET). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире.

Синтаксис языка C# выглядит очень похожим на язык Java. На самом деле и Java, и C# являются членами семейства языков программирования, основанного на C (куда также входят C, Objective C, C++) и поэтому они разделяют схожий синтаксис.

Вследствие того, что C# представляет собой гибрид из нескольких языков, он является таким же синтаксически чистым, как и Java, почти столько же простым, как и VB, и практически таким же мощным и гибким, как C++.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

**Основные преимущества платформы .NET**

Целью создания языка C# и платформы .NET было обеспечение более мощной, гибкой и простой модели программирования по сравнению с COM. .NET Framework – это программная платформа для построения приложений на базе семейства операционных систем Windows, а также многочисленных операционных систем производства не Microsoft, таких как MacOS X и различные дистрибутивы Unix и Linux. Перечень некоторых ключевых средств, поддерживаемых .NET:

* *Возможность взаимодействовать с существующим кодом*. Эта возможность, несомненно, является очень полезной, поскольку позволяет комбинировать существующие двоичные компоненты COM (т.е. обеспечивать взаимодействие с ними) с более новыми программными компонентами .NET и наоборот. С выходом .NET 4.0 и последующих версий возможность взаимодействовать дополнительно упростилась благодаря добавлению ключевого слова dynamic;
* *Поддержка многочисленных языков программирования*. Приложение .NET можно создавать с использованием любого числа языков программирования (C#, VB, F# и т.д.);
* *Общий дополнительный механизм*, разделяемый всеми поддерживающими .NET языками. Одним из аспектов этого механизма является наличие хорошо определённого набора типов, которые способен понимать каждый поддерживающий .NET язык;
* *Языковая интеграция*. В .NET поддерживается межъязыковая наследование, межъязыковая обработка исключений и межъязыковая отладка кода. Например, базовый класс может быть определён на C#, а затем расширен в VB;
* *Обширная библиотека базовых классов*. Эта библиотека позволяет избегать сложностей, связанных с выполнением низкоуровневых обращений к API– интерфейсам, и предполагает согласованную объектную модель, используемую всеми поддерживающими .NET языками;
* *Упрощённая модель развертывания*. В отличие от COM, библиотеки .NET не регистрируются в системном регистре. Более того, платформа .NET позволяет сосуществовать на одном и том же компьютере несколькими версиями одной и той же сборки \*.dll.

**JIT-компиляция**

Код на C# компилируется в приложения или сборки с расширениями exe или dll на языке CIL. Далее при запуске на выполнение подобного приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. При этом, поскольку наше приложение может быть большим и содержать кучу инструкций, в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. При том уже скомпилированная часть приложения сохраняется до завершения работы программы. В итоге это повышает производительность.

**Сравнение управляемого и неуправляемого кода**

Возможно, наиболее важный аспект, который следует знать о языке C#, заключается в том, что он порождает код, который может выполняться только в рамках исполняемой среды .NET (использовать C# для построения COM-сервера или неуправляемого приложения C/C++ не допускается). Выражаясь официально, для обозначения кода, ориентированного на исполняющую среду .NET, применяется термин, управляемый код. Двоичный модуль, который содержит управляемый код, называется сборкой. В противоположность этому, код, который не может обслуживаться непосредственно исполняющей средой .NET, называется неуправляемым кодом.

## 3.2 Алгоритм работы программы

1. Анализируем текст программы, выбранный пользователем;
2. Разбиваем по шаблону весь текст и находим необходимые блоки кода;
3. Определяем, возможно, неоптимальные участки кода, путём проверки: если в найденных блоках имеется поиск элементов в контейнерах, то, считаем, что это неоптимальный код, и мы будем его рассматривать.
4. Определяем тип рассматриваемых контейнеров;
5. Методом Монте-Карло определяем оптимальное решение, т.е. решение, где выигрыш будет максимальным, и удовлетворяющее введённому ограничению;

## 3.3 Описание работы программы

Для реализации алгоритма, описанного выше, было разработано приложение на языке C#, платформы .NET 3.5 Windows Forms. Интерфейс программы представляет собой окно, содержащее элементы управления (кнопки) для загрузки пользователем файла с исходным кодом C++(рис. 3.1), а так же его анализом. Помимо них имеются иные элементы для вывода полученных данных анализа кода, а так же результат работы алгоритма.

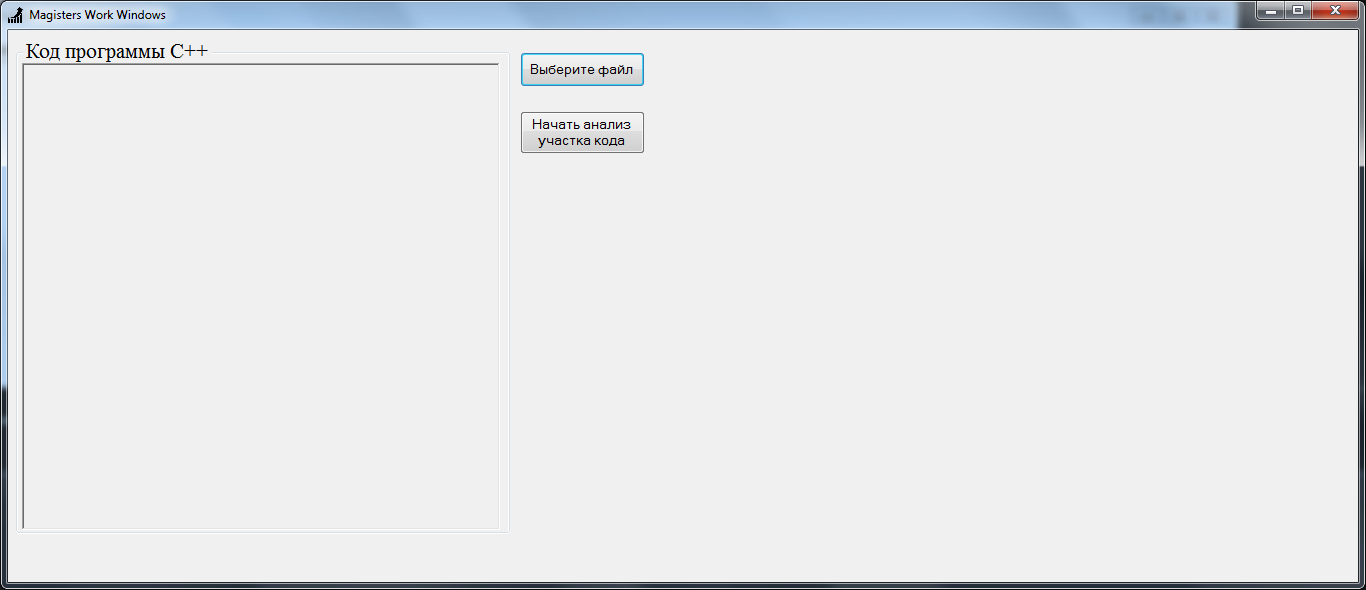


рис.3.1 – Внешний вид программы

Для загрузки файла с исходным кодом программы, написанной на C++, нажимаем на кнопку «Выберите файл». Откроется диалоговое окно, в котором нужно будет выбрать файл (рис. 3.2);

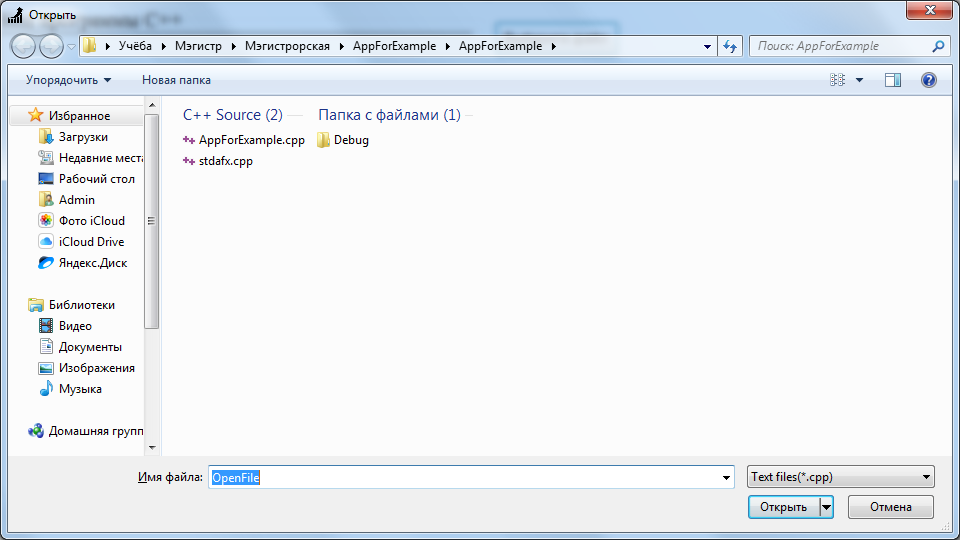


рис. 3.2 – Выбор файла

Когда пользователь выберет файл, то его содержимое, будет выведено на экран (3.3):

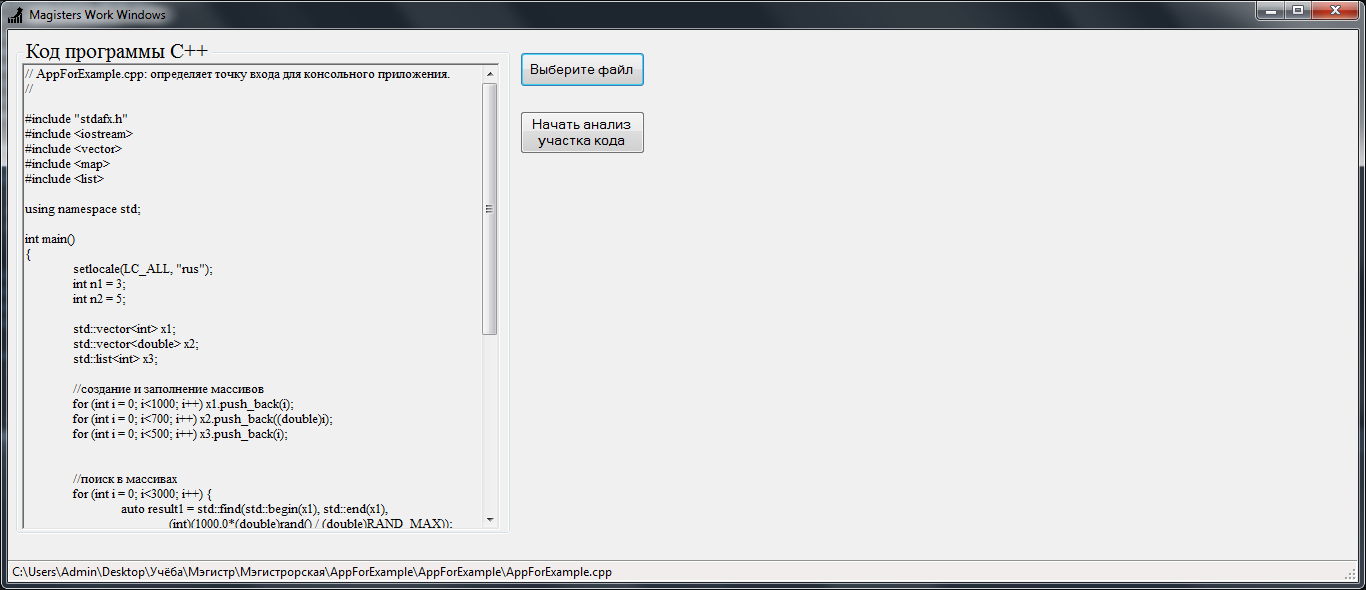


рис. 3.3 – Вывод текста из файла

После выбора файла, пользователь должен нажать на кнопку «Начать анализ участка кода», т.е. программа должна проанализировать код и вывести соответствующие результаты на экран (рис. 3.4);

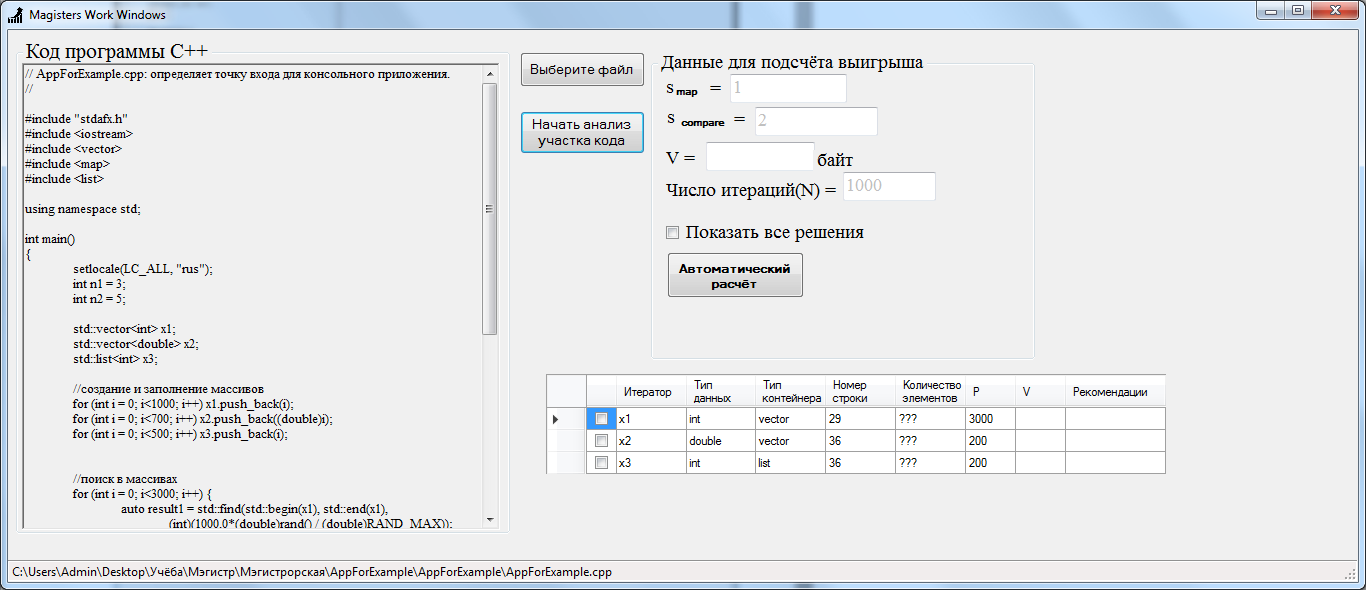
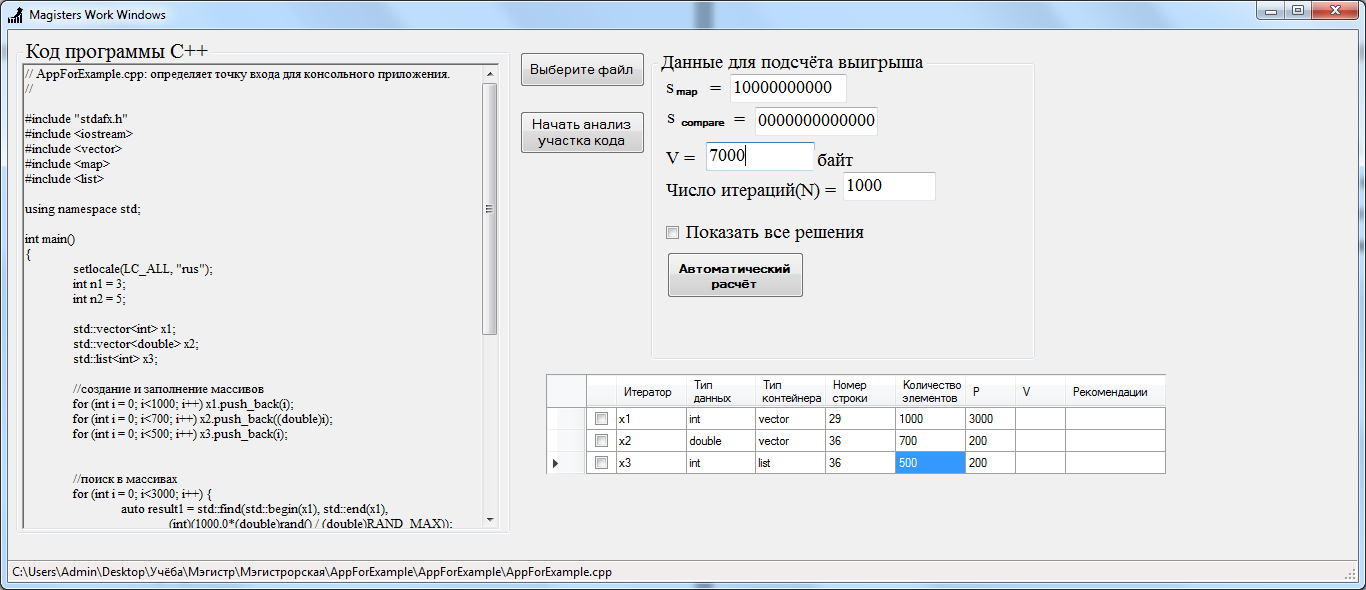
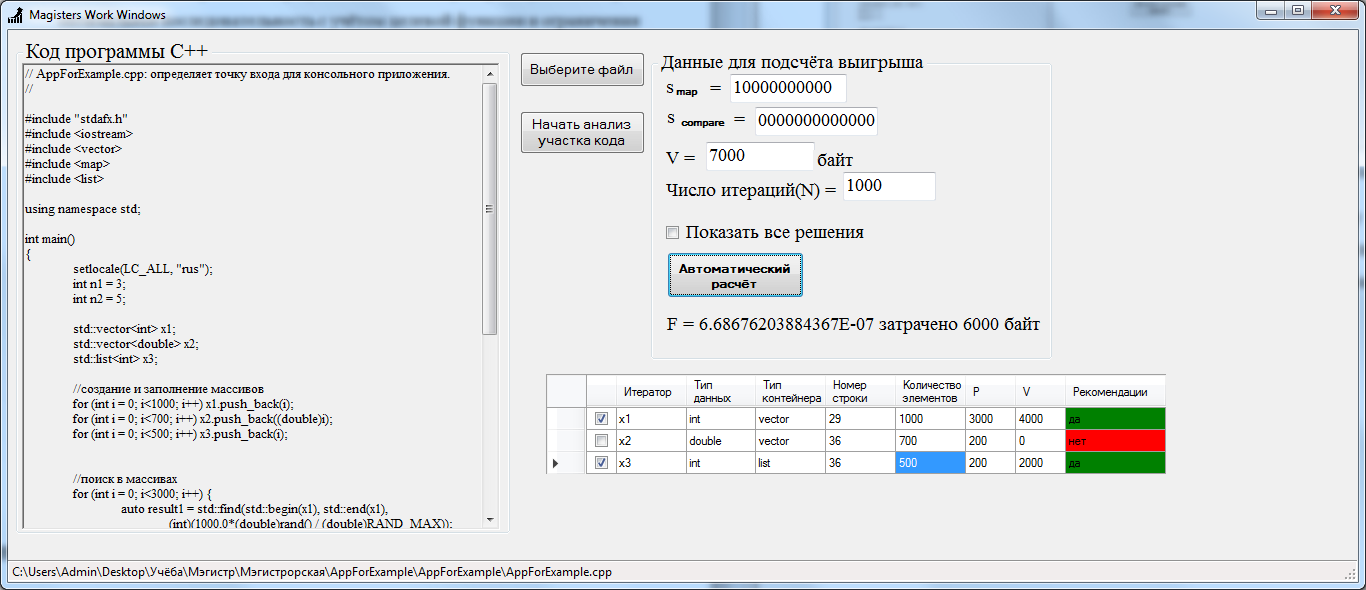


рис. 3.4 – Результат анализа участка кода

После анализа кода, пользователь должен ввести некоторые данные в специальном окне «Данные для подсчёта выигрыша», такие, как: (по умолчанию 1), (по умолчанию 2), объём доступной ОП(V), число итераций для метода Монте-Карло(N = 10000 по умолчанию) и по желанию пользователя можно сделать активным флаг для вывода подробного решения. Так же в таблице необходимо заполнить столбец «Количество элементов» у каждого контейнера (рис. 3.5);

 рис. 3.5 – Заполнение необходимых полей для подсчёта

Когда все данные введены, можно нажимать на кнопку «Автоматический расчёт». В результате полным перебором будет подобрана оптимальная последовательность с учётом целевой функции и ограничения (рис. 3.6);

рис. 3.6 – Результат работы программы, при автоматическом расчёте

Так же пользователь сам может выбирать элементы, вопреки тому, что вывела программа, но колонка «Рекомендации» остаётся неизменной, в случае если пользователь захочет использовать стратегию, предложенную программой.

Листинг программы, которая использовалась в качестве примера (Листинг 3.1):

// AppForExample.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <map>

#include <list>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

int n1 = 3;

int n2 = 5;

std::vector<int> x1;

std::vector<double> x2;

std::list<int> x3;

//создание и заполнение массивов

for (int i = 0; i<1000; i++) x1.push\_back(i);

for (int i = 0; i<700; i++) x2.push\_back((double)i);

for (int i = 0; i<500; i++) x3.push\_back(i);

//поиск в массивах

for (int i = 0; i<3000; i++) {

auto result1 = std::find(std::begin(x1), std::end(x1),

(int)(1000.0\*(double)rand() / (double)RAND\_MAX));

std::cout << (result1 != x1.end() ? "Значение найдено!" :

"Значение не найдено!");

}

for (int i = 0; i < 200; i++) {

auto result2 = std::find(std::begin(x2), std::end(x2),

700.0\*(double)rand() / (double)RAND\_MAX);

std::cout << (result2 != x2.end() ? "Значение найдено!" :

"Значение не найдено!");

auto result3 = std::find(std::begin(x3), std::end(x3),

(int)(500.0\*(double)rand() / (double)RAND\_MAX));

std::cout << (result3 != x3.end() ? "Значение найдено!" :

"Значение не найдено!");

}

system("pause");

return 0;

}

Листинг 3.1 – Программа на C++ для примера решения