Министерство образования и науки Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
_____Г.И. Радченко
«___»_____2018 г.

Разработка генератора схем алгоритмов на основе исходного кода на языке C#

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2018.484 ПЗ ВКР

Руководитель работы,
доцент каф. к.т.н. «Электронные вычислительные машины» к.т.н.
_____ И.Л. Надточий
«___»_____2018 г.

Автор работы
студент группы КЭ-484
_____В.В. Игнатьев
«___»_____2018 г.

Нормоконтролёр, ст. преп. каф. «Электронные вычислительные машины»
_____В.В. Лурье
«___»_____2018 г.

Челябинск 2018

Аннотация

Игнатьев В. В. Разработка генератора схем алгоритмов на основе исходного кода на языке C#. – Челябинск: ФГБОУ ВПО «ЮУрГУ» (НИУ) ВШЭКН; 2017, 59с., 25 ил. Библиографический список – 20 наименований.

Работа посвящена разработке приложения под Windows, позволяющее генерировать удобное графическое представление кода на основе исходного кода проекта на языке C# (Си Шарп).

Основной целью данного проекта является: Разработка приложения генератор схем алгоритмов на основе исходного кода на языке C# для построения графического представления исходного кода.

Получившееся приложение будет полезно студентам, начинающим и опытным разработчикам C# проектов:

1. Для студентов оно будет полезно тем, что можно будет сразу узнать верно ли они пишут код для своего проекта, отталкиваясь от графического представления, которое сможет построить данное приложение.
2. Начинающие и опытный разработчики смогут не только получить графического представление своих проектов, но так же смогут узнать в каких частях кода, у них происходят просадки по времени. Иными словами, смогут узнать где их проект еще нуждается в оптимизации.

В результате выпускной квалификационной работы было создано приложение "Генератор схем алгоритмов на основе исходного кода программы на языке C#" особенностью которого является использование Microsoft Visio для генерации блок схемы. С помощью определенных

функций управления:

1. настройка размера блока, что поможет решить проблему масштабирования;
2. настройка цвета блоков, для удобной подцветки элементов, которые принадлежат одному классу;
3. создание связей между элементами, что поможет приложению в правильном расположении элементов в блок - схеме;
4. вставка необходимого фрагмента кода в созданный элемент блок - схемы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В наше время становятся все более востребованы специальности связанные с информационными технологиями. Это все связано с проникновением информационных технологий во всё новые сферы деятельности, что в свою очередь приводит к тому, что всё больше людей начинают интересоваться специальностями, связанные с IT индустрией, изучать всё новые языки программирования, следовательно начинают сталкиваться с множеством препятствий, такими как: не работающий код, ошибки в написании, плохая оптимизация кода и так далее, которые их поджидают при изучении новых языков программирования.

С решением некоторых из этих проблем сможет помочь разрабатываемое мной приложение, которое поможет не только построить графическое представление исходного кода, но так же, в дальнейшем, поможет лучше оптимизировать его или показать фрагменты кода, которые требуют доработки из-за просадок по времени выполнения или из-за слишком больших затрат ресурсов компьютера.

Цель и задачи

Целью данной работы является разработка генератора схем алгоритмов на основе исходного кода на языке C# на платформе персонального компьютера под управлением ОС Windows.

Чтобы достичь поставленной цели, необходимо выполнить ряд следующих функциональных требований:

1. произвести анализ предметной области;
2. ознакомиться с существующими аналогами;
3. изучить техническую литературу;

4. спроектировать логику приложения;
5. спроектировать интерфейс управления приложением;
6. провести необходимые тесты.

Структура и объем поставленных работ

Данная работа состоит из следующих компонентов: введение, четырех глав, заключения, библиографического списка и приложения. Объем поставленной работы **N** страниц, объем библиографического списка – **N** источников.

В первой главе содержится анализ и обоснования выбора языка программирования и платформы для разработки, а так же операционной среды.

Во второй главе содержится обзор, некоторых существующих аналогов. Проводится, некоторый анализ и небольшое сравнение.

В третьей главе представлено описание логики приложения, функциональные требования, группы требования и требования групп к разрабатываемому приложению. Так же представлено краткое описание реализации основных технических составляющих приложения.

В четвертой главе содержится небольшая демонстрация разработанного приложения. И небольшое руководство по использованию

В заключении подводятся итоги по проделанной работе, что уже реализовано, а что требует доработки.

Приложение содержит в себе код программы, а именно: исходный, заголовочный и файлы формы.

Основные понятия предметной области:

ООП – объектно-ориентированное программирование;

ПК – Персональный компьютер;

ОС – операционная система;

ПО – программное обеспечение;

C# – Си Шарп.

1 ОБЗОР АНАЛАГОВ ПРИЛОЖЕНИЯ

1.1 Визуализатор Python-кода

Данный визуализатор является один из первых аналогов, которые я смог найти с подробным описанием его работы. Изначально данный визуализатор мог строить только простые блок схемы на Python-коде, но при дальнейшей его разработке было добавлено очень много полезных функций. С общим видом среды можно ознакомиться на рисунке 2.1.1

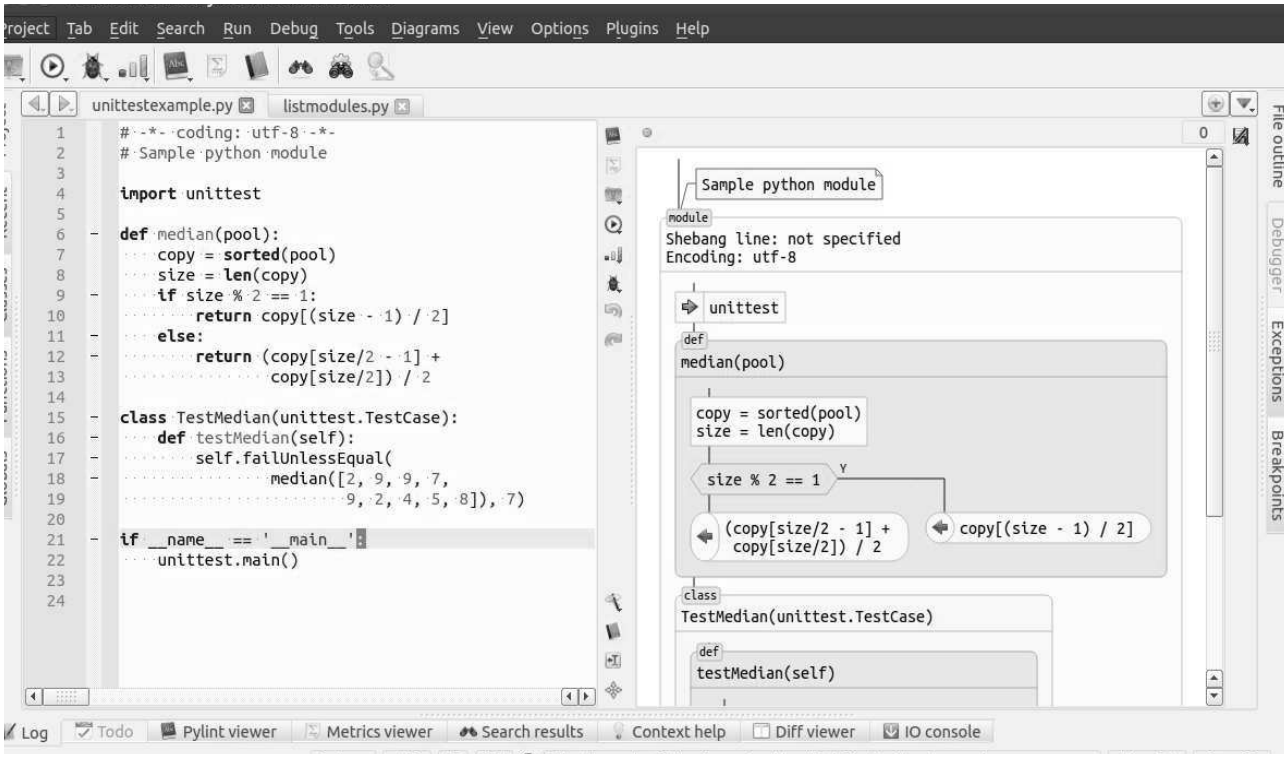

Рисунок 2.1.1 – Общий вид среды с альтернативными представлениями кода

Описание функционала данного визуализатора:

Комментарии

Ближайшее рассмотрение показывает, что можно выделить как минимум три типа комментариев, в зависимости от того, каким образом разработчик разместил их в коде. Пустые строки, аналогично тому, как это было сделано для блоков кода, также должны быть учтены, так как они формируют смысловые фрагменты. Итак, выделенные типы комментариев такие:

1. независимые;
2. лидирующие;
3. боковые.

```
a = 154

# Line 1
# Line 2

b = a
```

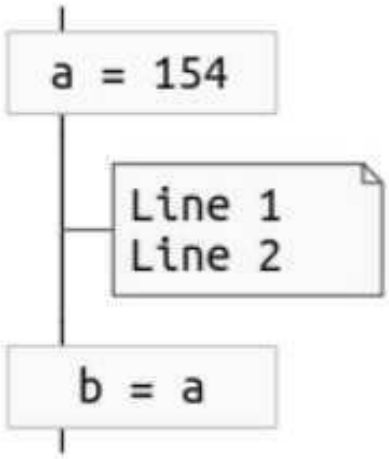

Рисунок 1.1.2 – Независимые комментарии (код и блок-схема)

В этом примере независимый комментарий состоит из двух строк. Очевидно, что комментарий находится между двумя блоками кода, а это положение на диаграмме соответствует линии, соединяющей блоки. Подходящей графикой для этого случая кажется прямоугольник комментария, напоминающий заметку, с горизонтальным соединителем, ведущим к линии между блоками.

```
# Line 1
# Line 2
a = 154
b = a
```

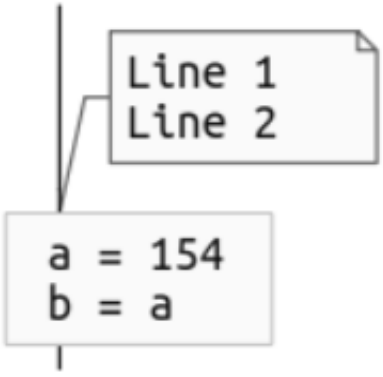

Рисунок 1.1.3 – Лидирующий комментарий (код и блок-схема)

Этот лидирующий комментарий предназначен для блока, поэтому можно нарисовать прямоугольник комментария выше блока и добавить соединитель, ведущий к нужному блоку.

```
a = 154
b = a        # No comment for the first line
c = b + 1    # Comment for c
             # A tail -----^
```

Рисунок 1.1.4 – Боковые комментарии (код)

Рисунок 1.1.4 – Боковые комментарии (блок-схема)

В случае с боковым комментарием требуется внимание к паре моментов. Во-первых, есть визуальное соответствие между строчками кода и комментариями. В примере выше нет комментария к первой строчке, и такие ситуации должны быть учтены в графике. Другими словами строки в прямоугольнике комментария должны быть по вертикали выровнены точно так же, как они выровнены в тексте. Во-вторых, иногда комментарий к последней строчке кода занимает больше чем одну строчку. Признаки такого «хвоста» взяты такими:

1. строка хвостового комментария следует непосредственно за предыдущей (нет пустых строк);
2. символ # находится в той же позиции, что и в предыдущей строке.

Импорты

Импорты приносят с собой зависимости. А управление зависимостями в сложных проектах зачастую становится трудной задачей. Поэтому графика для импортов, желательно, должна привлекать внимание даже при беглом просмотре диаграммы. Исходя из этого выбран прямоугольник с размещенной в левой части иконкой, как показано на примере ниже.

```
import sys

# Leading comment for import
from os.path import sep, \
                    isdir       # Side comment

from x import ( y,          # side for y
                name )      # side for name
```

Рисунок 1.1.5 – Импорты (код)

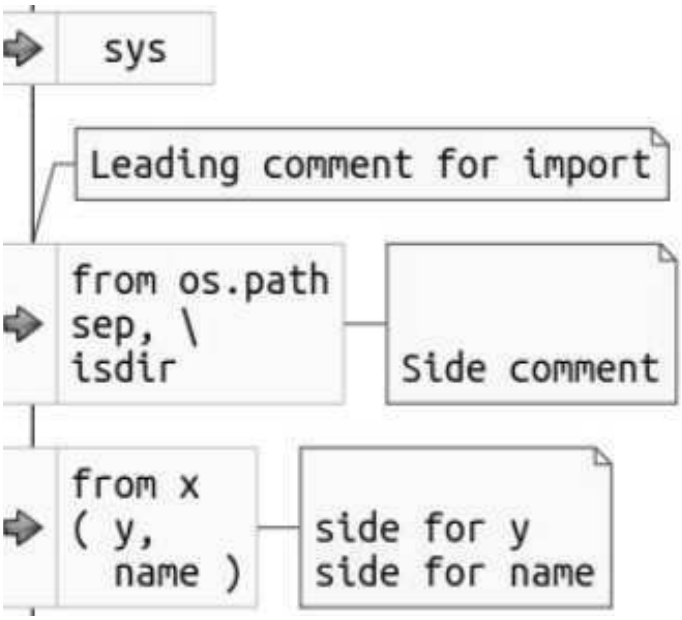

Рисунок 1.1.6 – Импорты (блок-схема)

Более подробно про данный редактор можно узнать в электронном источнике: https://habr.com/post/320184/ [6]

1.2 Quick Diagram

Следующим из проектов рассмотрим визуализатор "Quick Diagram" — это инструмент визуализации кода для C#:

1. для быстрого изучения языка программирования;
2. навигации по коду проекта;
3. документирования структуры исходного кода проекта.

Достоинства данного приложения:

1. интеграция с Visual Studio 2015 и Visual Studio 2017;
2. использование полученной диаграммы для поиска связанных типов;
3. быстрый переход от диаграммы к нужному фрагменту кода.

На основе уже построенной диаграммы можно обнаружить объекты, который связаны с нужным узлом. Точки, который находяться на сторонах прямоугольников диаграммы указывают на существование связных объектов.

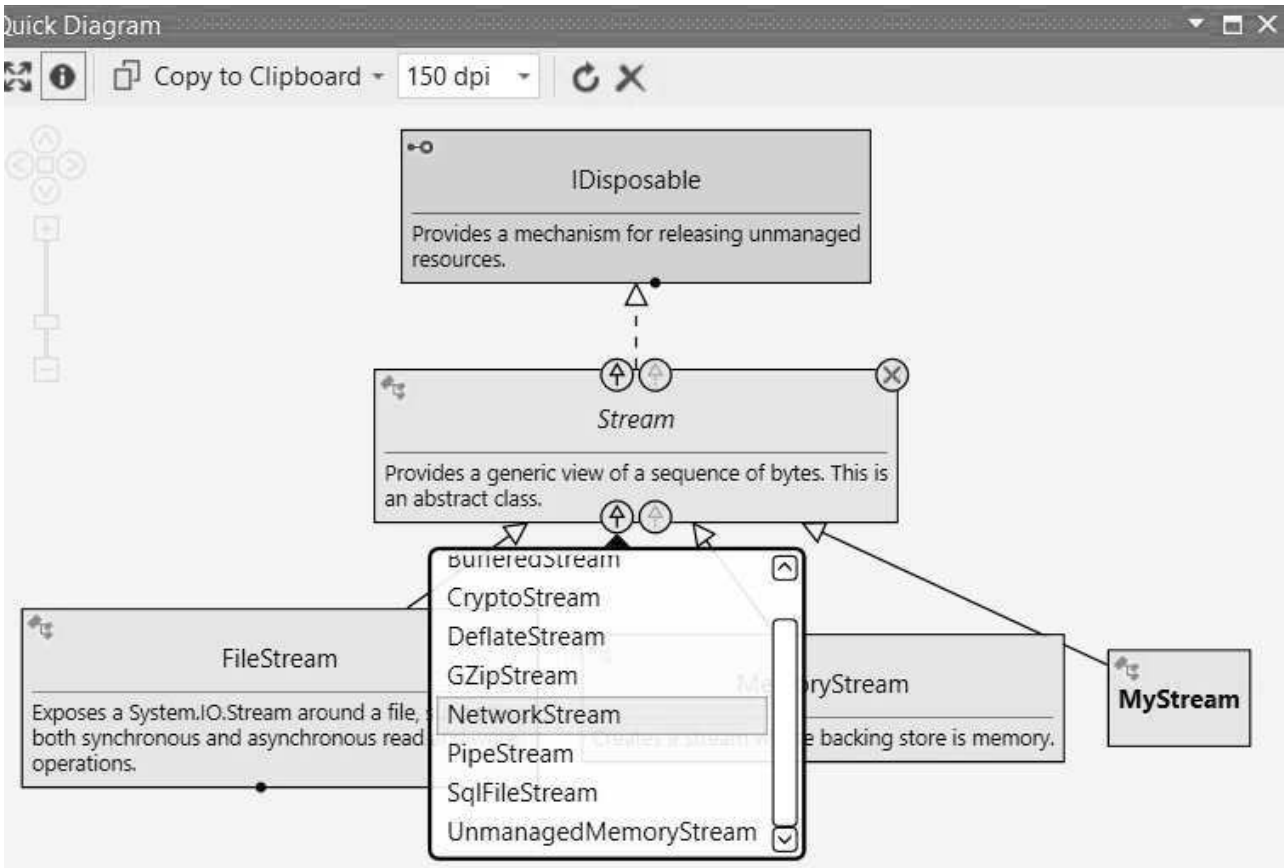

Рисунок 1.2.1 – Управление диаграммой

На рисунке 1.2.1 представлена версия инструмента "Quick Diagram", которая является экспериментальной и поддерживает только несколько типов отношений:

1. наследование;
2. реализация интерфейса
3. вызов методов

В более следующих версиях будут отображаться намного больше типов отношений: пространство имен, члены типа, чтение и запись свойств.

Более подробно про данный редактор можно узнать в электронном источнике: https://github.com/realvizu/QuickDiagram [12]

1.3 Flowchart

Flowchart – это приложение для автоматической генерации блок-схем алгоритмов по исходным кодам на языке программирования Pascal в среде разработки Delphi.

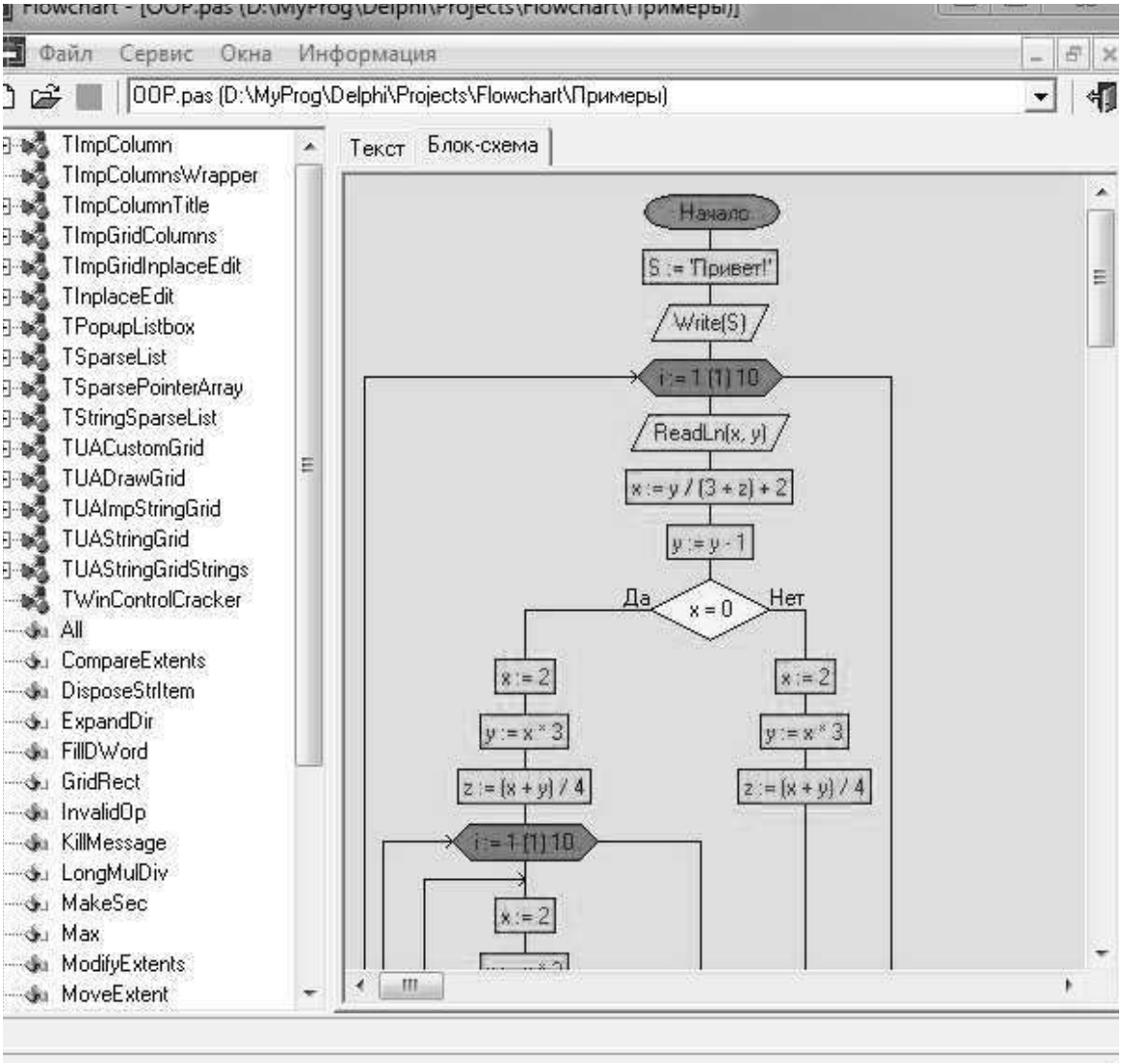

Рисунок 1.3.1 – Flowchart – автоматический генератор блок–схем на языке Pascal

Данная программа выполнена в виде многооконного приложения, то есть можно открыть сразу несколько файлов. Слева вы увидите проводник (дерево элементов), как в Delphi, где показываются все классы, процедуры и функции. Справа - две вкладки: в первой текст модуля, во второй - блок-схема алгоритма, которую программа рисует автоматически. При изменении текста автоматически изменяется и дерево элементов, и блок-схема.

Предусмотрена подсветка ключевых слов и прочих элементов текста программы, опять же, как в Delphi. При помощи данной программы можно также создавать и наглядные блок-схемы на обычном «человеческом» языке. Для этого надо просто подготовить соответствующие прототипы алгоритмов с использованием синтаксиса Pascal. Примеры таких файлов смотрите в папке «Прототипы алгоритмов». Данная программа нетребовательна к ресурсам компьютера, не требует установки и работает на всех версиях Windows.

Более подробно про данный редактор можно узнать в электронном источнике: https://almiur.ru/show_prog_9.html [10]

1.4 AFCE РЕДАКТОР БЛОК-СХЕМ

AFCE – это приложение, которое предназначено для создания, редактирования и экспорта блок-схем алгоритмов. Пользователю не нужно заботиться о размещении и выравнивании объектов, программа автоматически разместит все блоки. Редактор позволит экспортировать блок-схему в исходный текст программы для разных языков программирования. Данный редактор блок– схем работает с несколькими языками программирования:

1. Pascal;
2. C/C++;
3. Алгоритмический язык.

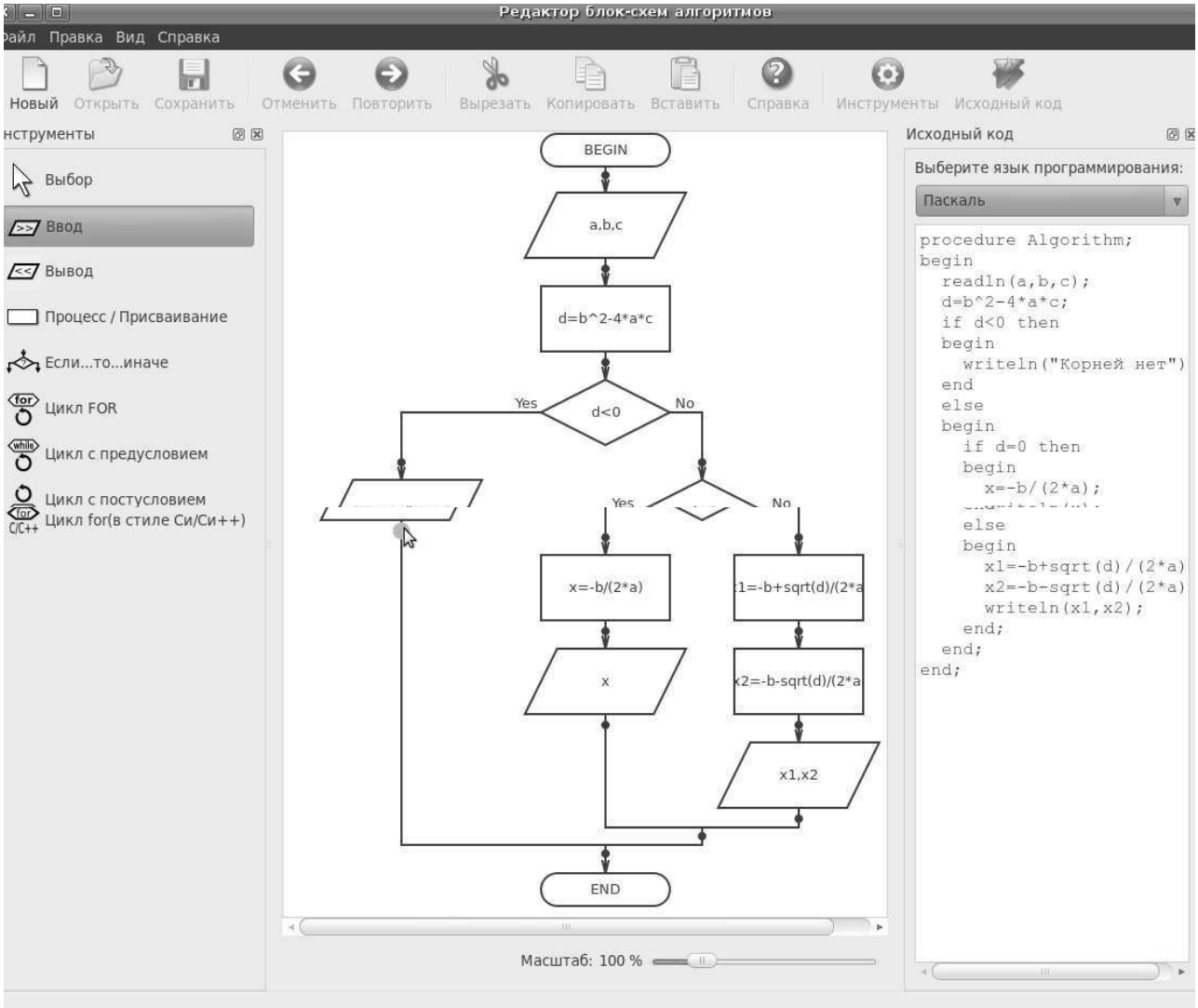

Рисунок 1.4.1 – AFCE Редактор блок-схем

Редактор блок-схем позволяет экспортировать изображение схемы в различные графические форматы:

1. BMP;
2. JPEG;
3. PNG;
4. TIFF;
5. ICO;
6. PPM;
7. XBM;
8. XPM;
9. SVG;

Данное приложение распространяется на условиях лицензии GNU General Public License (GPL). Программа написана на языке C++ на основе библиотеки Qt.

Более подробно про данный редактор можно узнать в электронном источнике: www.twirpx.com/file/574148/ [11]

1.5 Visustin v8 Flow chart generator

Visustin – Visustin - это программное обеспечение для блок-схемы, которое преобразует исходный код в блок-схемы - автоматически.

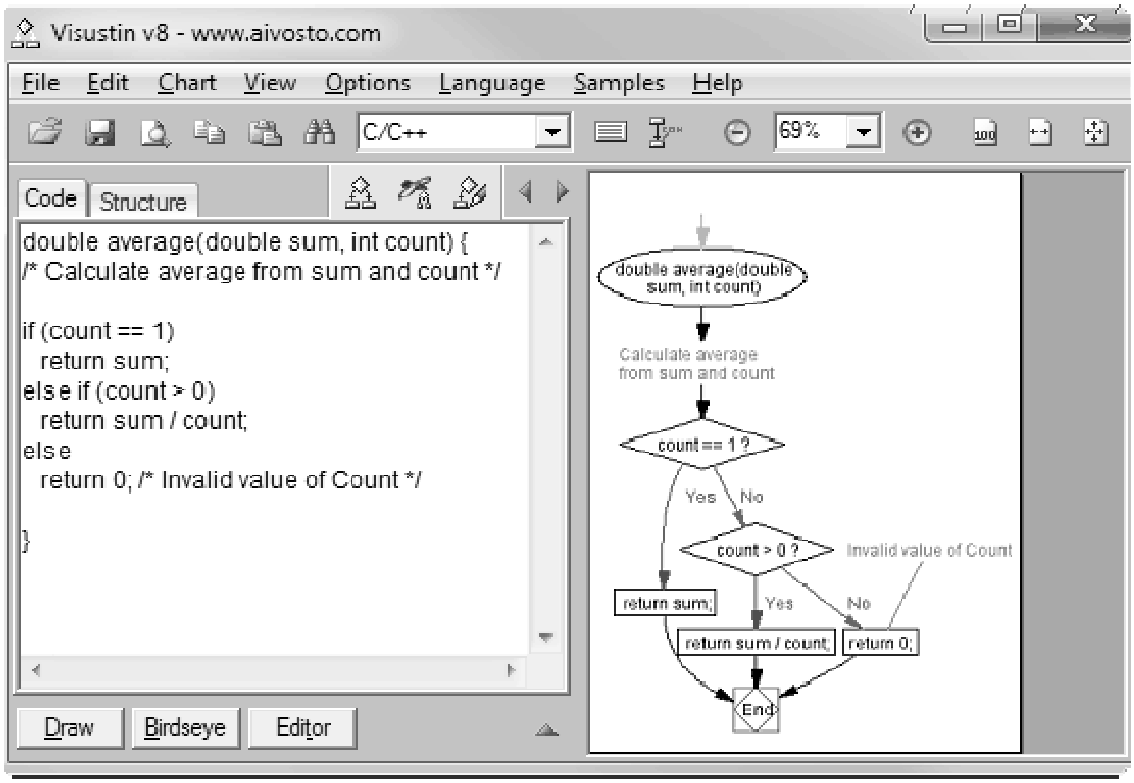

Рисунок 1.5.1 – AFCE Редактор блок-схем

Visustin для разработчиков программного обеспечения. Visustin - это программа для автоматизированного создания блок-схемы для разработчиков программного обеспечения и авторов документов. Сохраните усилия по документированию с автоматической визуализацией кода. Visustin обращает инженеров ваш исходный код в блок-схемы или диаграммы активности *UML* . Visustin читает if и else утверждения, циклы и прыжки и строит диаграмму - полностью автоматически.

Спомощью данной программы можно:

1. Проверьте логику программы с блок-схемами.
2. Найти ошибки до пользователей.

Особенности данного генератора лагоритмов:

1. поддержка более чем 10 языков, таких как : ABAP, ActionScript, Ada, ASP, несколько языков ассемблера, файлы AutoIt, BASIC, .bat, C, C ++, C #, Clipper, COBOL, ColdFusion, Delphi, Fortran, GW-BASIC, HTML, Java, JavaScript, JCL (MVS), JavaServer Pages, LotusScript, MATLAB, MXML, Pascal, Perl, PHP, PL / I, PL / SQL, PowerBASIC, PowerBuilder PowerScript, PureBasic, Python, QuickBASIC, REALbasic, Rexx, RPG, Ruby, SAS, Tcl, TSQL , Сценарий оболочки Unix (bash, csh, tcsh, ksh, sh), VB, VBA, VBScript, VB.Net, Visual FoxPro, XHTML, XML и XSLT.
2. многостраничная печать, иными словами печать больших блок-схем в виде многостраничной мозаики;
3. поддержка как блок-схемы, так и диаграммы активности UML.

Более подробно про данный редактор можно узнать в электронном источнике: http://www.aivosto.com/visustin.html [9]

2 ОБОСНОВАНИЯ ВЫБОРА ЯЗЫКА ПРОГРАММИРОВАНИЯ И ОПЕРАЦИОННОЙ СРЕДЫ

2.1 Требования к операционной системе

Разработанное приложение "Генератор схем алгоритмов на основе исходного кода на языке C#" в данное время может быть запущено только на ОС Windows.

На ПК должна быть установлена лицензионная локализованная версия ОС Windows 7 или выше разрядностью 32 или 64 битной версии. Так же должно быть установлено приложение Microsoft Visio так как именно в это приложении будет происходить основная генерация графического представления кода. И беря во внимание, что компания Microsoft в скором времени прекратит поддержку операционных систем версий Windows 7 и Windows 8, то рекомендуется перейти на операционную систему Windows 10, что бы избежать конфликтов приложения с ОС.

2.2 Обоснование выбора платформы ОС Windows

Данная ОС была выбрана из тех соображений, что большая часть пользователей использует ОС Windows, а именно на 81.8% персональных компьютеров стоит данная операционная система по данным Stat Counter Global Stats.

Stat Counter Global Stats – это инструмент для получения онлайн статистики пользователей ОС и не только.

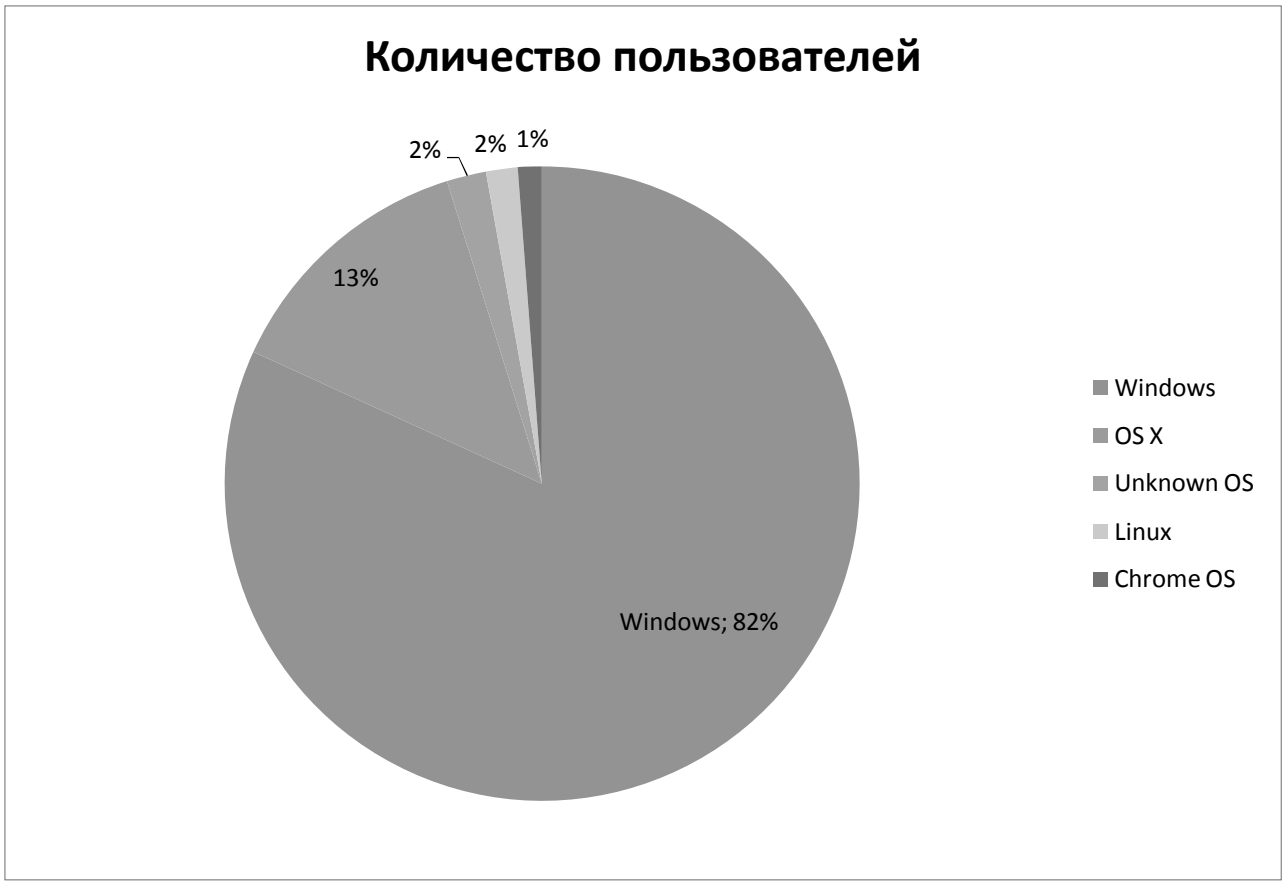

Рисунок 2.2.1 – Статистика распространенности ОС на ПК

Более подробно про данный инструмент можно узнать в электронном источнике: http://gs.statcounter.com/os-market-share

2.3 Обоснование выбора сред разработки

Для написания приложения использовалась среда разработки Microsoft Visual Studio 2017.

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ,

поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения.

Более подробно про данный редактор можно узнат ь в электронном источнике: https://visualstudio.microsoft.com/ru/vs/features/. [18]

2.4 Обоснование выбора языка программирования

Для реализации данного проекта был выбран объектно ориентированный язык программирования C#. Данный язык программирования выбран с точки зрения эффективности. Иными словами, чтобы тратить меньше времени на реализацию и получать хороший результат на выходе.

C# – простой, современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C. C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов — это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также

встроенные элементы документации. C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# отлично подходит для создания и применения программных компонентов.

Более подробно про данный язык программирования можно узнать в электронном источнике: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/index [17]

3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

3.1 Проектирование архитектуры приложения

Для проектирования приложения "Генератор схем алгоритмов на основе исходного кода на языке C#" был использован язык графического описания объектных моделей UML. Данный язык был выбран с целью быстрого составления схемы отталкиваясь только от алгоритма работы программы и без написания программного кода проекта.

Для описания проекта были выбраны и построены UML диаграммы, которые представлены на рисунке 3.1.1

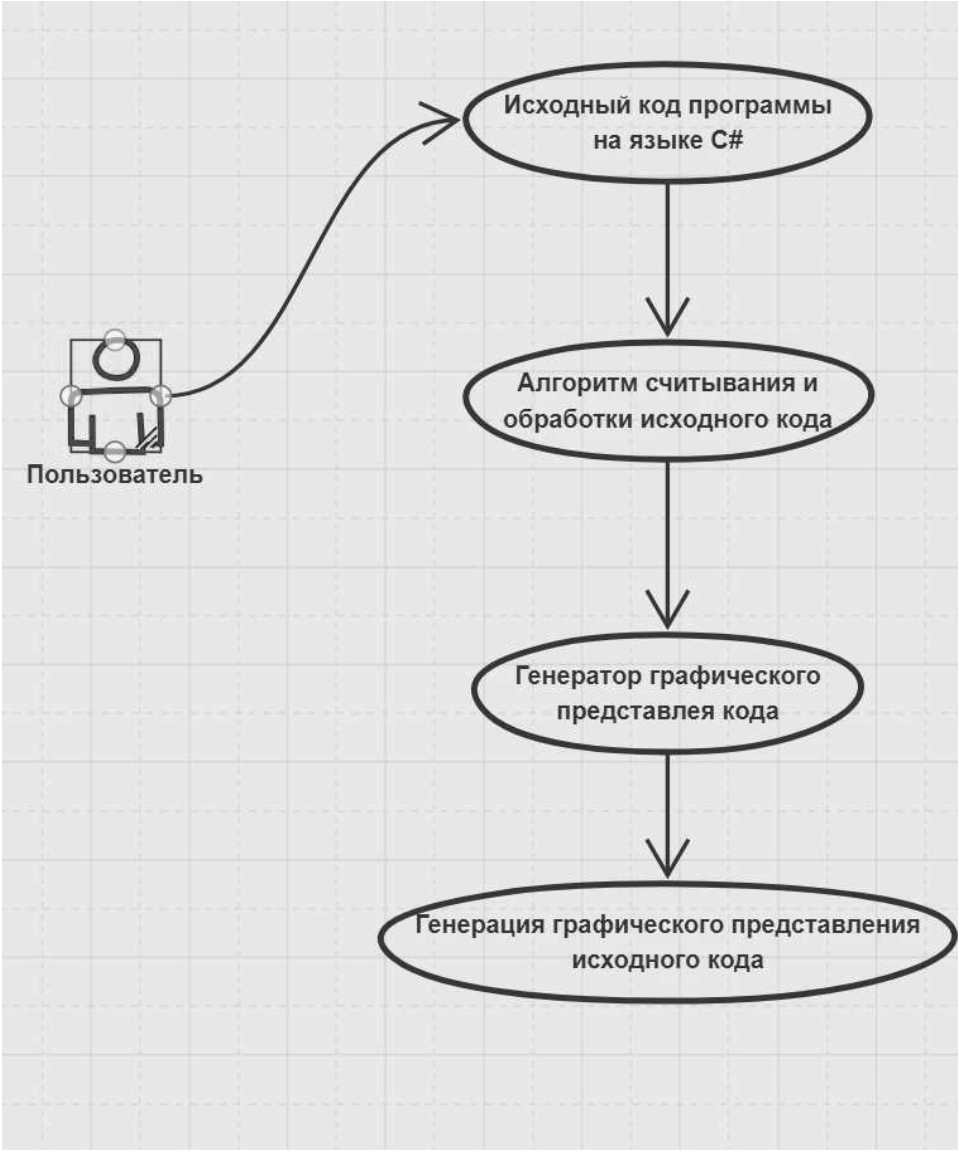

Рисунок 3.1.1 – UML Диаграмма прецедентов

Рассмотрим все варианты диаграммы прецедентов более подробно:

Исходный код программы

На данном этапе пользователь создает новый проект и вводит исходный код своего алгоритма программы в специально отведенное для этого поле. Дальше все, что требуется от пользователя это нажать кнопку старта программы.

Алгоритм считывания и обработки

Это алгоритм, который отвечает за правильное считывание исходного кода проекта и начинает начальные этапы его обработки. Иными словами он разделяет весь поток поступающего кода на отдельно взятые фрагменты для дальнейшей генерации графического представления программного проекта. Алгоритм генерации представлен на рисунке 3.1.2.

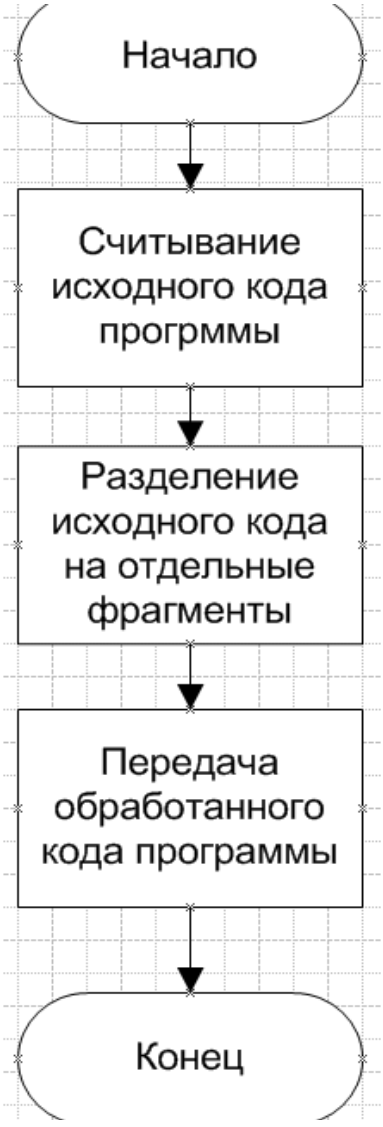

Рисунок 3.1.2 – Алгоритм считывания и обработки

Генератор графического представления

В данном блоке происходит следующее. Сначала "Генератор графического представления" принимает обработанный код алгоритма программного проекта. Далее он начинает определять, какой фрагмент кода относится к тому или иному блоку схемы.

Алгоритм генерации представлен на рисунке 3.1.3.

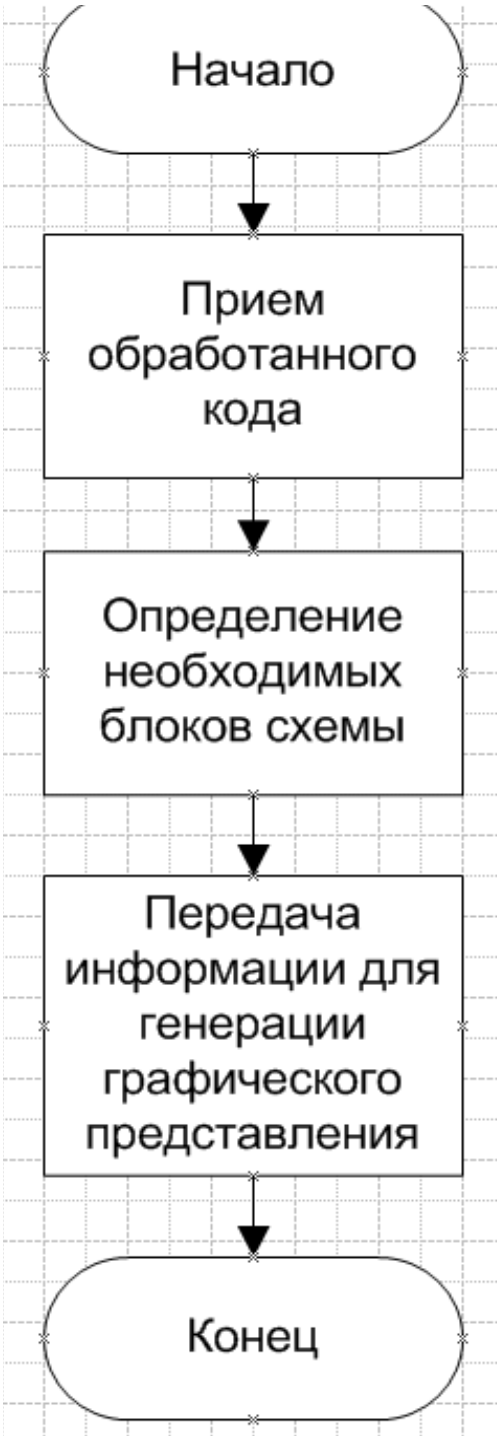

Рисунок 3.1.3 – Генератор графического представления

Генерация графического представления

В финальном этапе формирования полноценного графического представления исходного кода проекта, создается проект в другой программе, которая называется Microsoft Visio, в ней происходит финальная стадия создания графического представления. Иными словами мое приложение создает необходимые блоки программы далее происходит распределение обработанных фрагментов кода по блокам и создание связей между ними.

Алгоритм генерации представлен на рисунке 3.1.4.

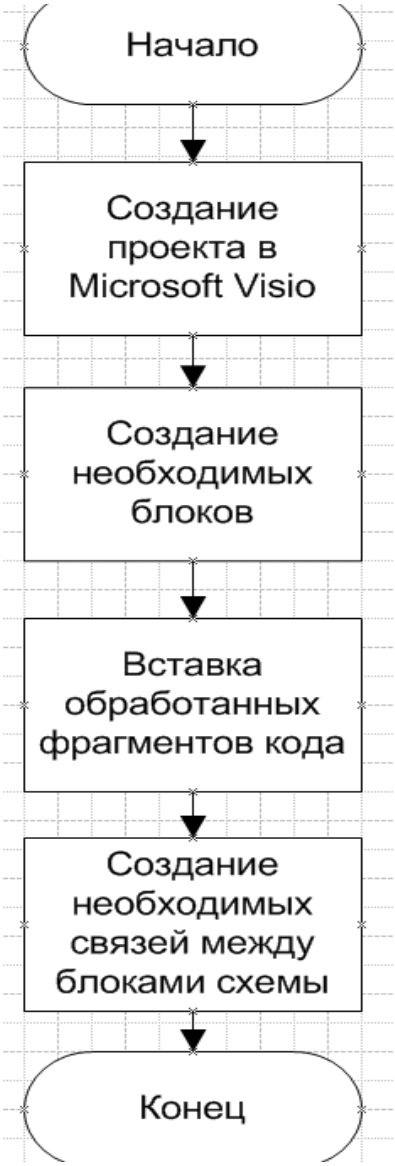

Рисунок 3.1.4 – Финальная генерация графического представления

Более подробно про UML диаграммы можно узнать в электронном источнике: https://docs.microsoft.com/ru-ru/visualstudio/vsto/visio-object-model-overview [1]

3.2 Проектирование интерфейса

Так как интерфейс приложения должен быть понятен даже тем, кто недавно начал работать с языками программирования, то интерфейс был сделан как можно проще. Это сделано для того, что бы пользователь не запутался среди всех этих настроек. В целя удобства все элементы управления интерфесом представлены в виде 3 вкладок:

1. главное меню визуализатора;
2. создание схемы;
3. о программе;

Все необходимые вкладки меню приложения указаны на рисунках 3.2.1, 3.2.2, 3.2.3.

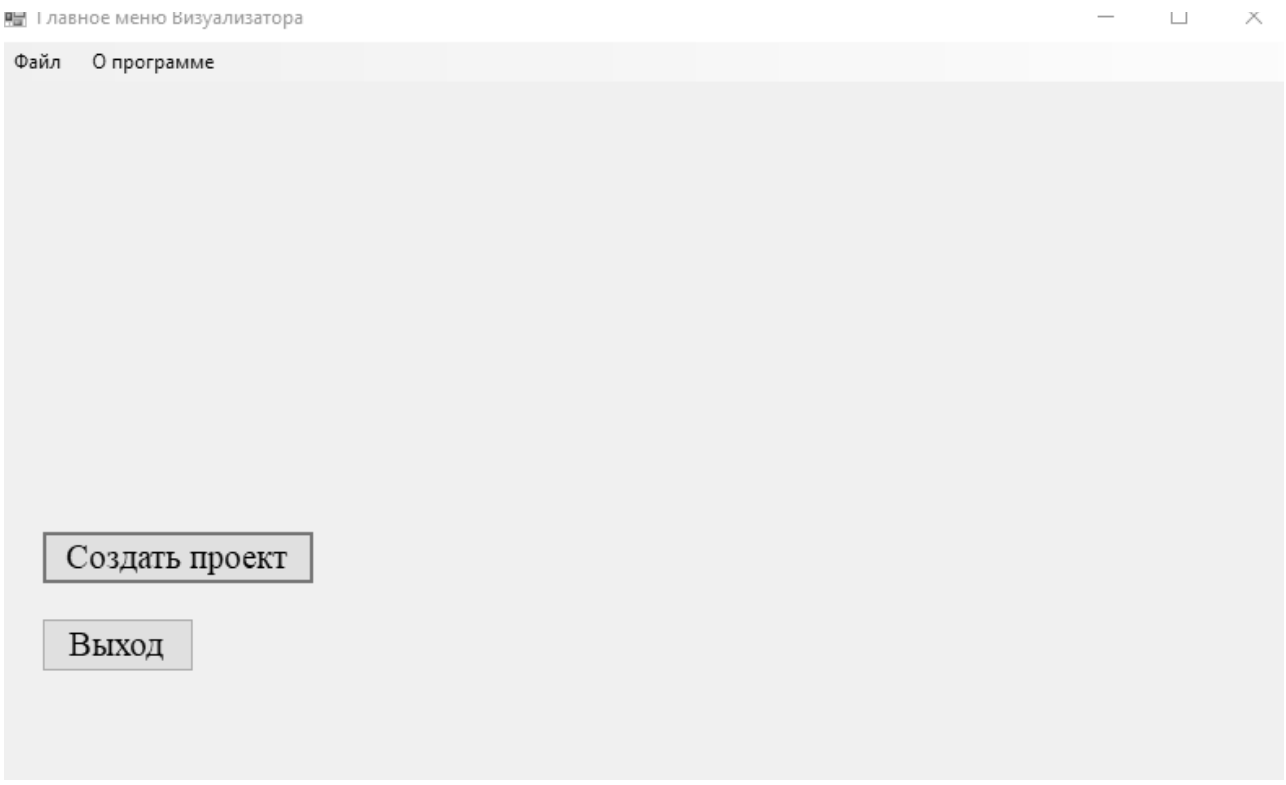

Рисунок 3.2.1 – Главное меню визуализатора

Главное меню было спроектировано как можно проще, что бы любой пользователь мог, без особых сложностей создать графического представление своего исходного кода.

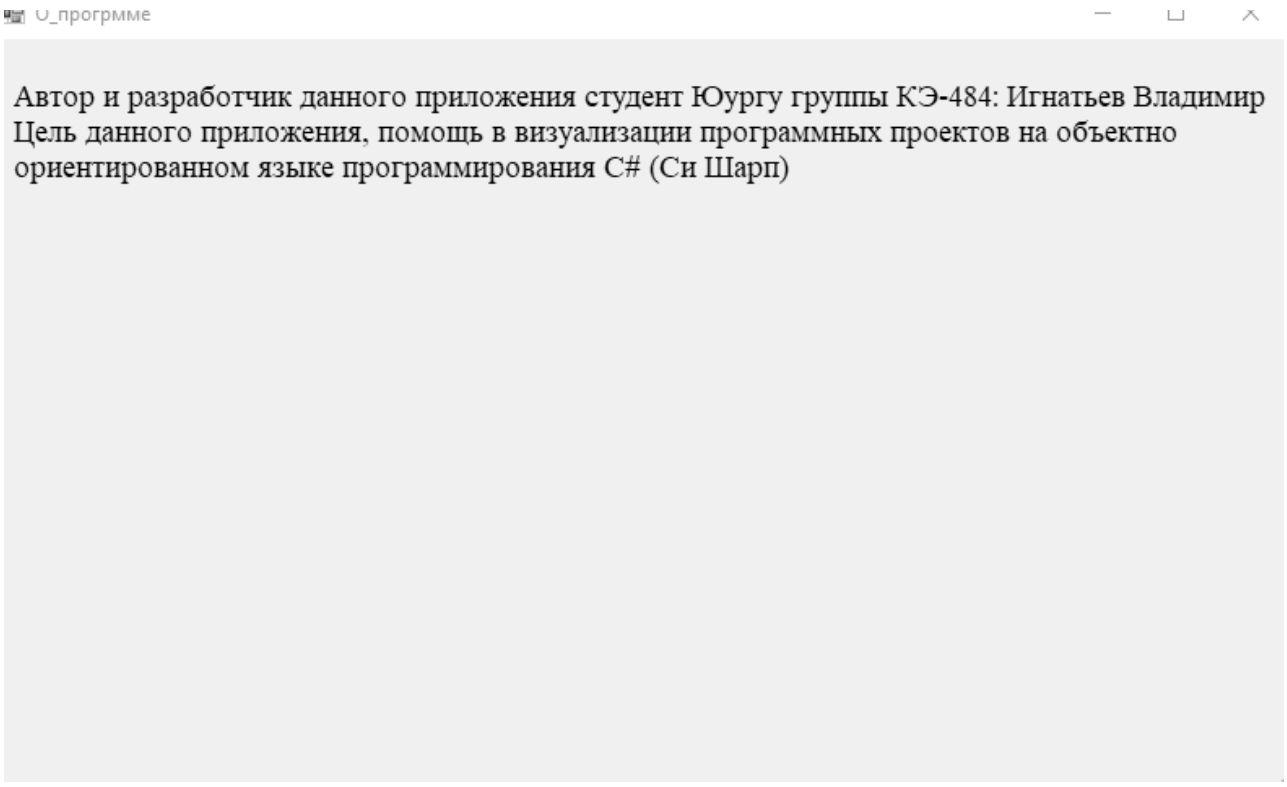

Рисунок 3.2.2 – О программе

Окно "О программе" содержит информацию о цели создания данного приложения и информацию об авторе данного программного продукта.

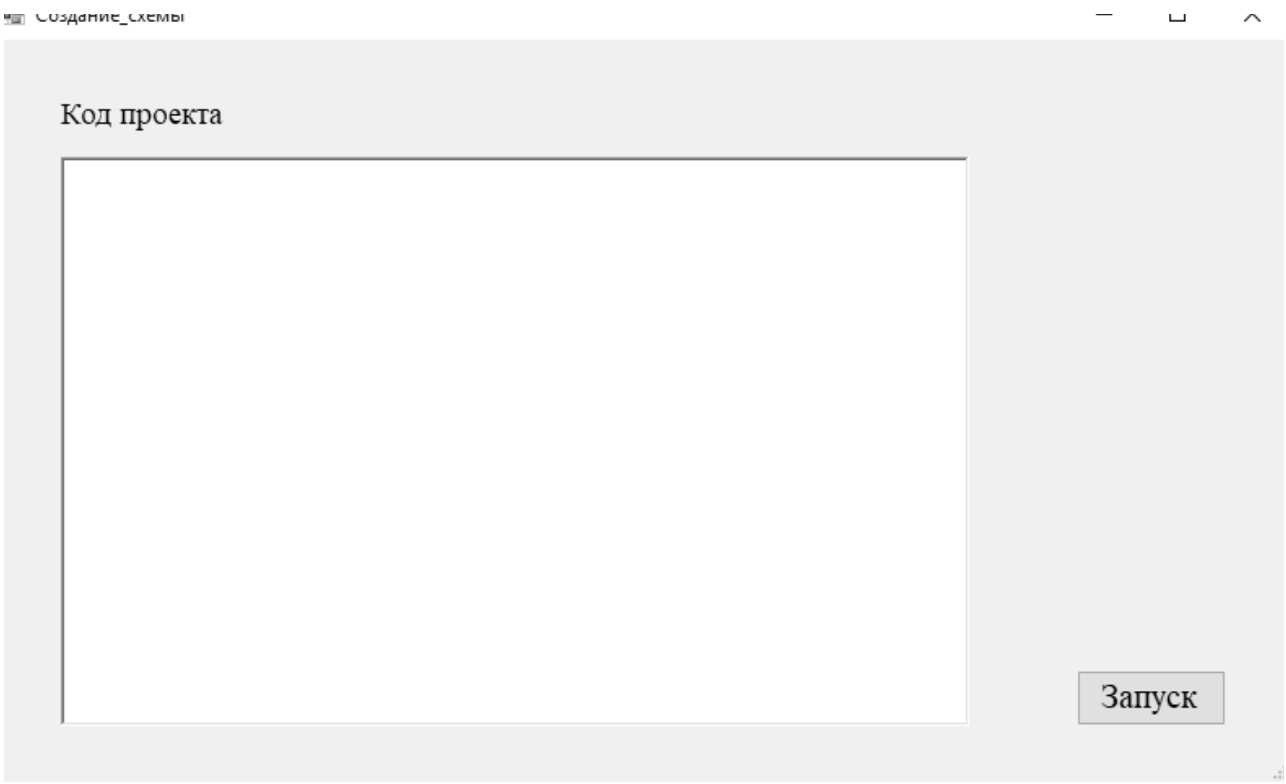

Рисунок 3.3.3 – Генератор схем алгоритмов

Окно "Генератор схем алгоритмов" содержит основное поле для ввода исходного кода от проекта на языке C# и кнопку, которая запускает алгоритм обработки кода и генерации графического представления проекта на языке C#.

4 РЕАЛИЗАЦИЯ

4.1 Реализация логики приложения "Генератор схем алгоритмов на основе исходного кода на языке C#"

Основой логики данного приложения является класс CodeVisualizer, который и осуществляет преобразование исходного кода проекта на языке C# в графическое представление этого кода. Иными словами, при получении определенного фрагмента кода, данный класс анализирует его следующим образом. Прежде всего этот класс смотрит, есть ли внутри данного фрагмента еще другие, будь то матричный массив или набор условий, который должны выполняться поочередно. И если в этом фрагменте кода нет других, которые требовали бы обработка, тогда он устанавливает, какой блок использовать для работы с этим фрагментом кода, далее он определяет объем данного фрагмента, для подбора правильного размера блока, в который данный код будет помещен.

Далее у нас идет класс, который отвечает как раз за построчное считывание исходного кода. Этой задачей занимается класс CodeReader, в задачу которого входит считывание исходного кода проекта и передача его классу CodeVisualizer. Суть этого класса в том, что он так же как и класс для генерации графического представление ищет ключевые слова, когда он находит один из них, он отправляет запрос классу CodeVisualizer для проверки, эта проверка необходима для устранения ошибок, на случай если в алгоритме считывателя произошел сбой. После того как он получил положительный сигнал от класса CodeVisualizer он начинает передачу уже считанного фрагмента кода для дальнейшей обработки.

Следующий класс, который требует рассмотрения является класс GeneratorCH. Это основной класс для окончательной визуализации схемы. В его функции входит, при получении данных от класса CodeVisualizer он

сначала определяет координаты x и y каждого из блоков, а так как код считывается построчно, класс GeneratorCH выставляет их по очереди на лист в уже созданном проекте в Microsoft Visio

За перемещение и установку связи между блоками отвечает класс ControlMoverOrConnect. Данный класс в самом конце работы алгоритма проверяет расположение всех блоков графического представления. Проверка проходит путем установки связи между блоками, после этого он передвигает все блоки ближе к тем, с кем у них установлена связь, после он перемещает их так, что бы они не наезжали друг на друга. В финале свой работы данный класс выстраивает окончательный графического представления.

4.2 Реализация работы с проектом в Microsoft Visio.

4.2.1 Создание документа в Microsoft Visio.

Чтобы создать графическое представление кода, необходимо ввесте исходный код в специальное поле и после нажать кнопку "Запуск" В приложении при нажатии кнопки запуска, происходит создание нового документа в Microsoft visio. Так же проходит настрока листа, на котором будет размещаться графическое представление

```
        // Подготовка документа
Microsoft.Office.Interop.Visio.Application application =
          new Microsoft.Office.Interop.Visio.Application();
application.Visible = false;
Microsoft.Office.Interop.Visio.Document doc = application.Documents.Add(templatePath);
Microsoft.Office.Interop.Visio.Page page = application.Documents[1].Pages[1];

double xPosition = page.PageSheet.get_CellsU("PageWidth").ResultIU;
double yPosition = page.PageSheet.get_CellsU("PageHeight").ResultIU
```

Листинг 4.2.1.1 – Создание документа в Microsoft Visio

4.2.2 Установка ширины и высоты блоков

Мы используем эту информацию о ширине и высоте листа, чтобы знать, где разместить блоки. Мы помещаем корневые блоки в середину листа, разделяя ширину листа на количество корней. Также мы вычитаем из yPosition номера уровня, чтобы поля с увеличением номера уровня получили более низкую позицию на графике.

При считывании исходного кода проекта приложение анализирует какой размер задать конкретно взятому блоку.

```
//установить ширину формы
shape.get_CellsSRC(
                (short) Microsoft.Office.Interop.Visio.VisSectionIndices.
                 visSectionObject,
                (short) Microsoft.Office.Interop.Visio.VisRowIndices.
                visRowXFormIn,
                (short) Microsoft.Office.Interop.Visio.VisCellIndices.
                visXFormWidth).ResultIU = box.Width;

//установить высоту формы
shape.get_CellsSRC(
               (short) Microsoft.Office.Interop.Visio.VisSectionIndices.
                visSectionObject,
               (short) Microsoft.Office.Interop.Visio.VisRowIndices.
               visRowXFormIn,
               (short) Microsoft.Office.Interop.Visio.VisCellIndices.
               visXFormHeight).ResultIU = box.Height;
```

Листинг 4.2.2.1 – Настройка размеров блоков в Microsoft Visio

4.2.3. Установка связи между блоками

Для создания полноценной блок схемы необходимо создать линии связи между блоками. Такая связь помогает сразу понять какие действия выполняются следующими.

```
//Установка связи между двумя разными блоками
connectWithDynamicGlueAndConnector(shape, childShape);
```

Листинг 4.2.3.1 – установка связи между двумя элементами блок-схемы

4.2.4 Вставка текста в блок

Чтобы вставить текст в определенный блок, необходимо задать название данного блока

```
//установить текст
shape.Text = box.Name;

//установить гиперссылку
if (!String.IsNullOrEmpty(box.HyperLink.Trim()))
{
     Hyperlink link = shape.Hyperlinks.Add();
link.Address = box.HyperLink;
}
```

Листинг 4.2.4.1 – Процесс вставки текста в Microsoft Visio

4.2.5 Редактирование цвета блоков

В блок–схеме два отдельных блока можно делать разных размеров, все зависит от кода, который содержится в том или ином блоке кода. Логично можно предположить, что если есть изменение размера блока, то и есть изменение его цвета, которое может быть полезно Например использование одного цвета для отдельно взятого класса было бы очень удобно, при рассмотрении масштабной схемы алгоритма.

```
//установить цвет переднего плана
shape.Characters.set_CharProps(
                (short) Microsoft.Office.Interop.Visio.
                     VisCellIndices.visCharacterColor,
                (short) Utilities.GetVisioColor(box.ForeColor));
//установить цвет задней формы
shape.get_CellsSRC((short) VisSectionIndices.visSectionObject,
         (short) VisRowIndices.visRowFill,
    (short) VisCellIndices.visFillForegnd).FormulaU =
       "RGB(" + box.BackColor.R.ToString() + "," + box.BackColor.G.ToString() + ","
       + box.BackColor.B.ToString() + ")";
```

Листинг 4.2.5.1 – Процесс смены цвета в Microsoft Visio

4.3 Тестирование приложения "Генератор схем алгоритмов на основе исходного кода на языке C#"

4.3.1 Тест генерации блока с условием

Результат работы приложения "Генератор схем алгоритмов на основе исходного кода на языке C#" на правильность генерации блок-схемы.

Приложение запущено, пользователь ввел код алгоритма, нажимает кнопку запуска. После начинается считывание, обработка и генерация графического представления исходного кода.

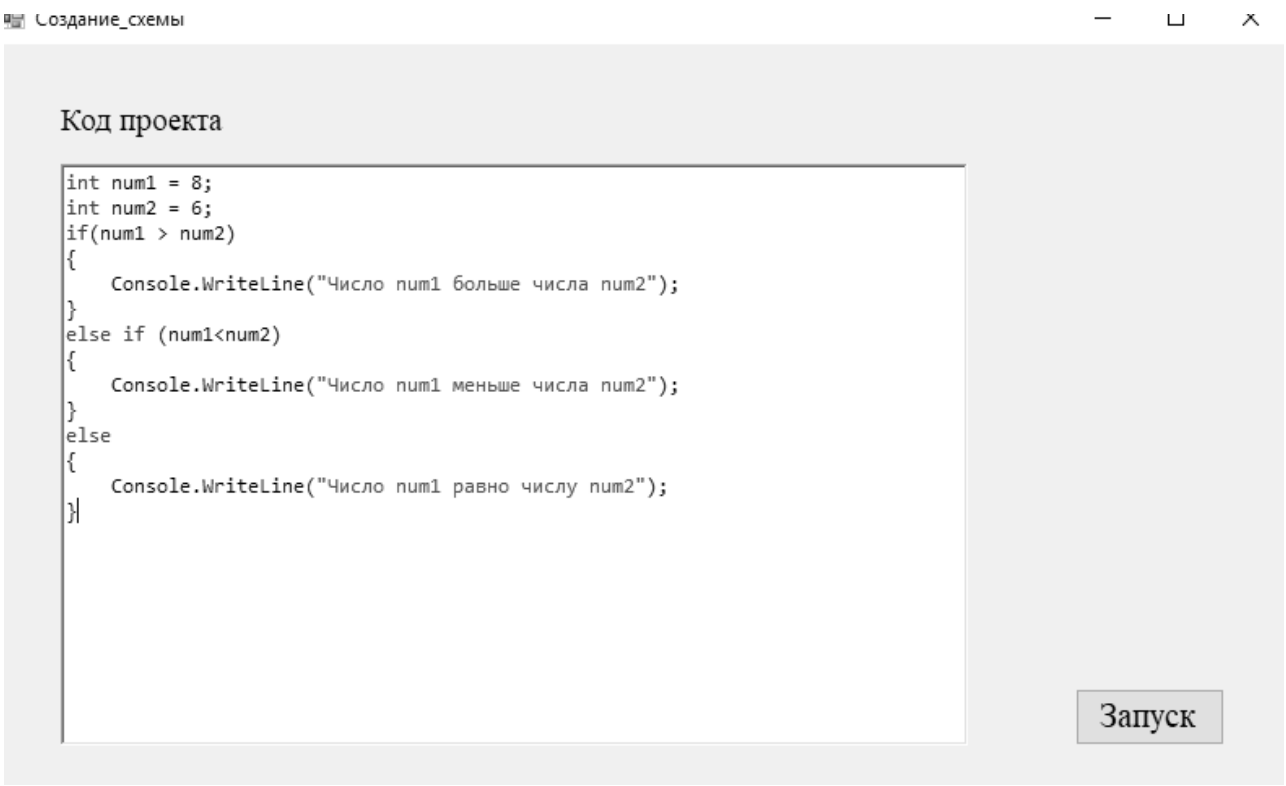

Рисунок 4.3.1.1 – код условия, введенный для теста генерации графического представления

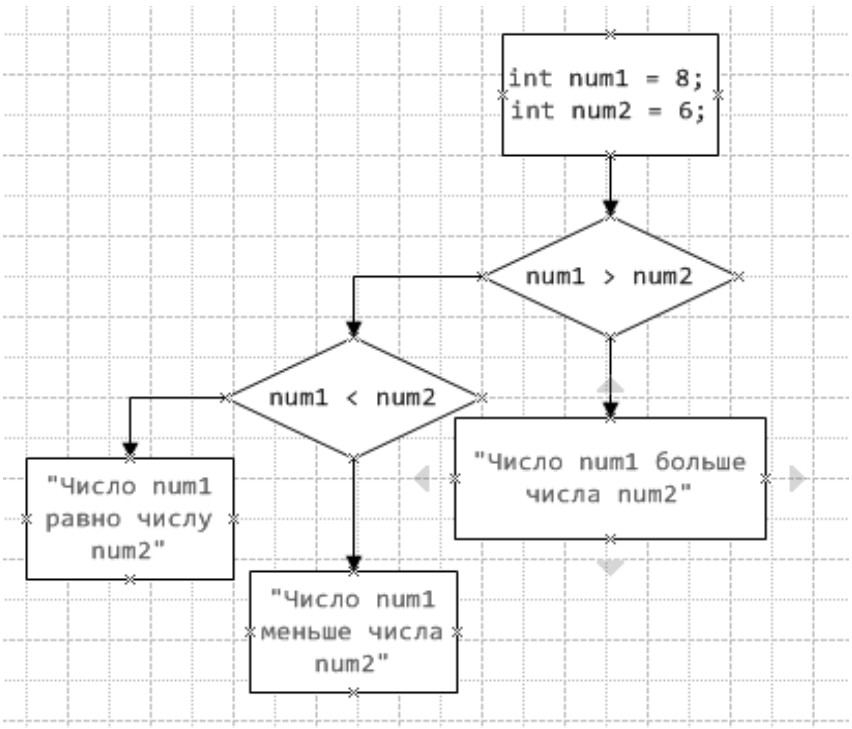

Рисунок 4.3.1.1 – полученная блок-схема алгоритма на выходе

4.3.2 Тест генерации цикла

Результат работы приложения "Генератор схем алгоритмов на основе исходного кода на языке C#" на правильность генерации блок-схемы.

Приложение запущено, пользователь ввел код алгоритма, нажимает кнопку запуска. После начинается считывание, обработка и генерация графического представления исходного кода.

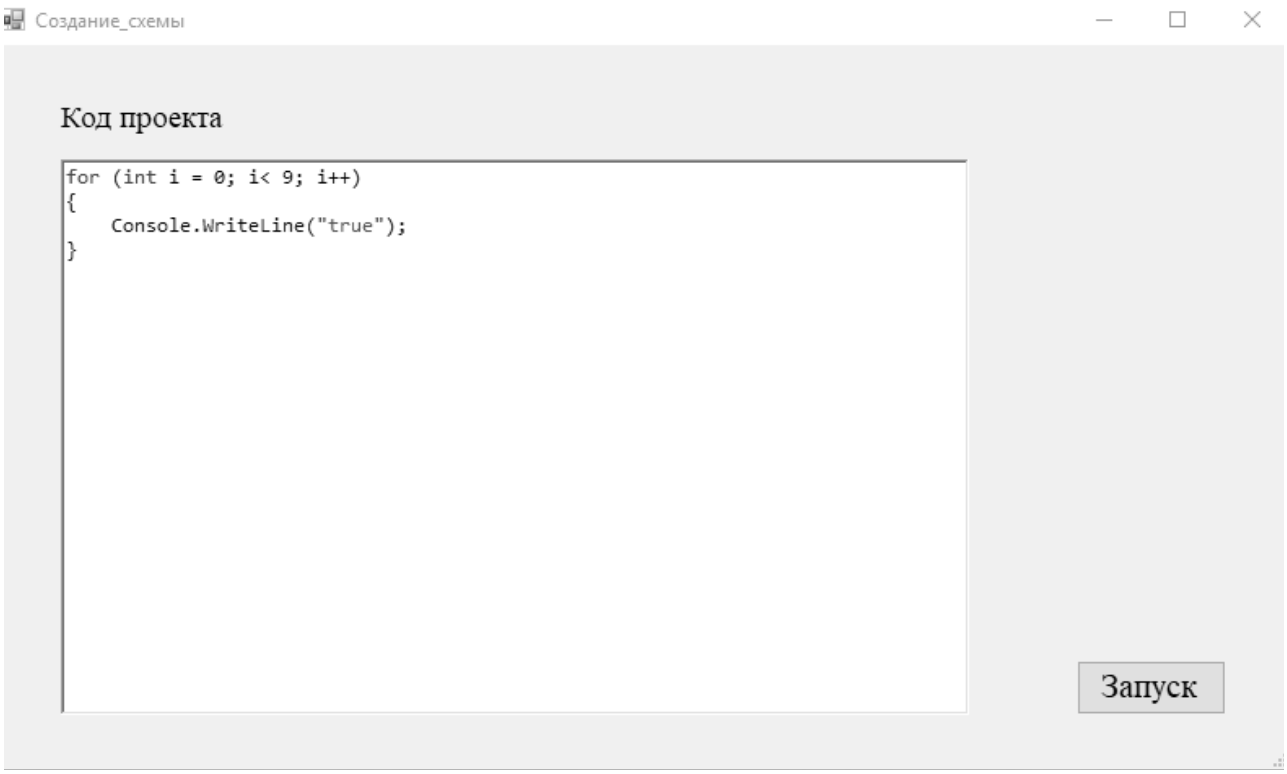

Рисунок 4.3.2.1 – код условия, введенный для теста генерации графического представления

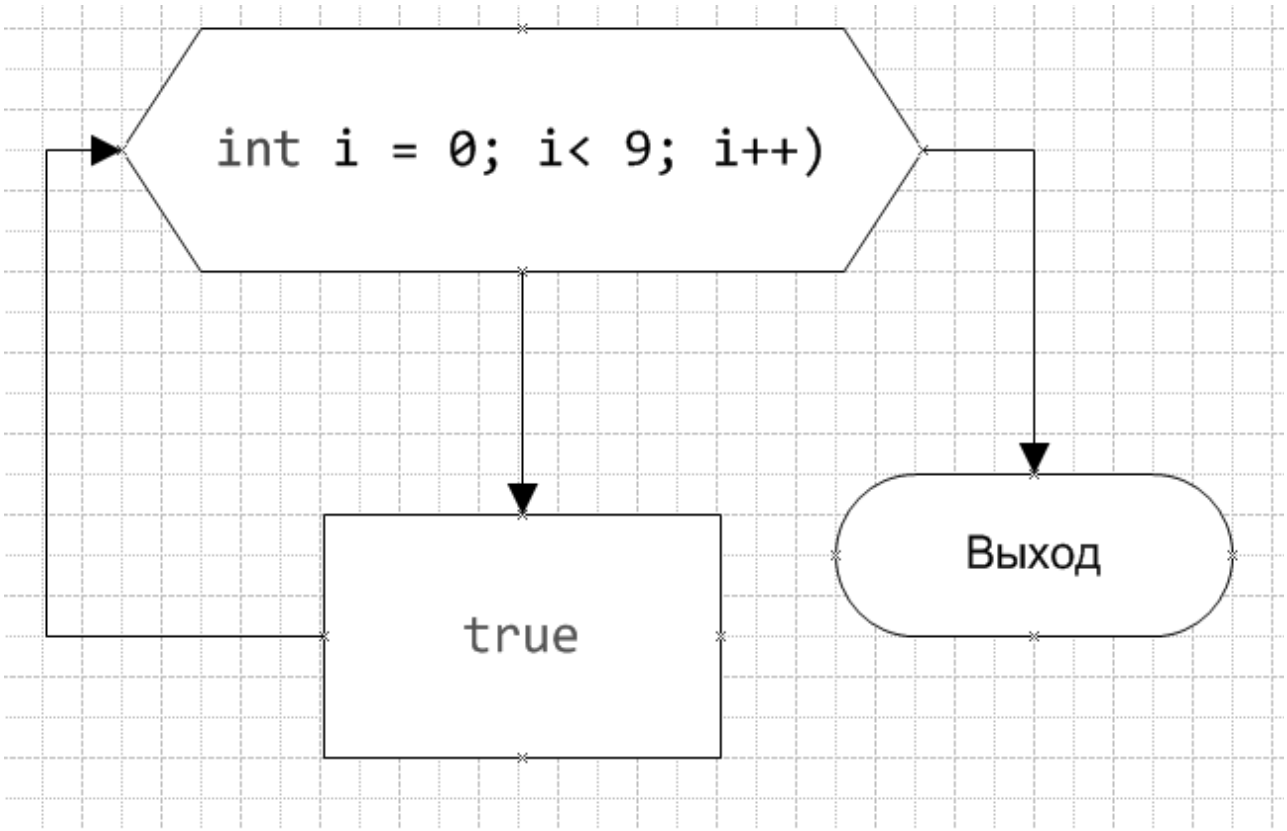

Рисунок 4.3.2.1 – полученная блок-схема алгоритма на выходе

ЗАКЛЮЧЕНИЕ

Целью данного проекта была разработка приложения для генерации графического представления на основе исходного кода на языке C#. В ходе работы над программным проектом "Генератор схем алгоритмов на основе исходного кода на языке C#" все функциональные требования, которые были выявлены в ходе постановки задачи, были полностью реализованы:

1. генерация графического представления;
2. масштабирование блоков;
3. создание простого и удобного интерфейса.

В результате было разработано приложение, которое предназначено для генерации схем алгоритмов по исходному кода на языке C#. Которое будет полезно, как для студентов, так и для IT-разработчиков.

В ходе выполнения данного проекта были решены следующие проблемы:

1. проблема правильной генерации;
2. определение блоков;
3. верное построение графического определения;
4. проблема масштабирование блоков.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Общие сведения о модели объектов Microsoft Visio. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/vsto/visio-object-model-overview – Заглавие с экрана. – (Дата обращения 15.04.2018).

2 ШилдтГ. C# 4.0 полное руководство/ Г. Шилдт– М.: Бином, 2011. – 754 с.

3 Интерактивный учебник по Visual C# - MSDN - Microsoft. [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/bb383962(v=vs.90).aspx. –Загл. с экрана.– (Дата обращения 04.04.2018).

4 Руководство по программированию на C#. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/index – Заглавие с экрана. – (Дата обращения 09.04.2018).

5 Документация по Visual Studio [Электронныйресурс].– Режимдоступа: https://docs.microsoft.com/ru-ru/visualstudio/#pivot=workloads&panel=windows – Заглавие с экрана. – (Дата обращения 01.03.2018).

6 Автоматическая визуализации python-кода с использованием блок-схем. [Электронный ресурс]. – Режим доступа: https://habr.com/post/320184/ – Заглавие с экрана. – (Дата обращения 01.03.2018).

7 Microsoft Visio 2013 Business Process Diagramming and Validation. [Электронный ресурс]. – Режим доступа: https://recommendedforyou.xyz/br/r1?ts=22844&pct=bookrd&q=Microsoft+Visio+2013+Business+Process+Diagramming+and+Validation – Заглавие с экрана. – (Дата обращения 20.04.2018).

8 Разработка Windows приложений на C#/ Н.А. Осипов [Электронный ресурс]. – Режим доступа: http://books.ifmo.ru/file/pdf/905.pdf – Заглавие с экрана. – (Дата обращения 14.03.2018).

9 Visustin Генератор блок-схемы. [Электронный ресурс]. – Режим доступа: http://www.aivosto.com/visustin.html – Заглавие с экрана. – (Дата обращения 14.03.2018).

10 Flowchart автозарисовщик блок-схем. [Электронный ресурс]. – Режим доступа: https://almiur.ru/show_prog_9.html – Заглавие с экрана. – (Дата обращения 14.03.2018).

11 AFCE редактор блок-схем. [Электронный ресурс]. – Режим доступа: https://www.twirpx.com/file/574148/ – Заглавие с экрана. – (Дата обращения 20.05.2018).

12 Разработка визуализатора блок-схем на C#. [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/ms164759.aspx?f=255&MSPPError=-2147217396 – Заглавие с экрана. – (Дата обращения 01.06.2018).

13 Разработка алгоритма действий и создание блок-схемы [Электронный ресурс]. – Режим доступа: https://kopilkasovetov.com/programmyi-servisi-prilogeniya/razrabatyivaem-algoritmyi-deystviy-i-sozdaem-blok-shemyi – Заглавие с экрана. – (Дата обращения 13.05.2018).

14 ГОСТ 19.701-90 ЕСПД Схемы алгоритмов [Электронный ресурс]. – Режим доступа: http://www.pntd.ru/19.701.htm – Заглавие с экрана. – (Дата обращения 03.05.2018).

15 СкитД. C# для профессионалов: тонкости программирования / Д. Скит– М.: Вильямс, 2014. – 608 с

16 Технология разработки генераторов [Электронный ресурс]. – Режим доступа: http://studbooks.net/2110900/informatika/tehnologiya_razrabotki_generatorov – Заглавие с экрана. – (Дата обращения 08.05.2018).

17 Технология разработки визуализаторов алгоритмов [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/_a_process.pdf – Заглавие с экрана. – (Дата обращения 04.05.2018).

18 Краткий обзор языка C#. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/index – Заглавие с экрана. – (Дата обращения 08.05.2018).

19 Обзор среды разработки microsoft visual studio. [Электронный ресурс]. – Режим доступа: https://visualstudio.microsoft.com/ruru/vs/features – Заглавие с экрана. – (Дата обращения 10.05.2018).

20 Stat Counter Global Stats. [Электронный ресурс]. – Режим доступа: http://gs.statcounter.com/os-market-share – Заглавие с экрана. – (Дата обращения 10.06.2018).

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

В данном приложении представлен листинг ядра приложения, код главного меню программы:

- VisualizerCode.h – заголовочный файл главного меню;
- VisualizerCode.cs – исходный код главного меню;
- Core.cs – исходный код ядра.

« VisualizerCode.h» – заголовочный файл главного меню

```
namespace visualizercode
{
    partial class Form1
    {
        /// <summary>
        /// Обязательная переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален;
иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows

        /// <summary>
        /// Требуемый метод для поддержки конструктора — не изменяйте
        /// содержимое этого метода с помощью редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
            this.файлToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.создатьПроектToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.сохранитьToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.выходToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.оПрограммеToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.button1 = new System.Windows.Forms.Button();
            this.button2 = new System.Windows.Forms.Button();
```

```
            this.menuStrip1.SuspendLayout();
            this.SuspendLayout();
            //
            // menuStrip1
            //
            this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.файлToolStripMenuItem,
            this.оПрограммеToolStripMenuItem});
            this.menuStrip1.Location = new System.Drawing.Point(0, 0);
            this.menuStrip1.Name = "menuStrip1";
            this.menuStrip1.Size = new System.Drawing.Size(800, 24);
            this.menuStrip1.TabIndex = 0;
            this.menuStrip1.Text = "menuStrip1";
            //
            // файлToolStripMenuItem
            //
            this.файлToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.создатьПроектToolStripMenuItem,
            this.сохранитьToolStripMenuItem,
            this.выходToolStripMenuItem});
            this.файлToolStripMenuItem.Name = "файлToolStripMenuItem";
            this.файлToolStripMenuItem.Size = new System.Drawing.Size(48, 20);
            this.файлToolStripMenuItem.Text = "Файл";
            //
            // создатьПроектToolStripMenuItem
            //
            this.создатьПроектToolStripMenuItem.Name = "создатьПроектToolStripMenuItem";
            this.создатьПроектToolStripMenuItem.Size = new System.Drawing.Size(158, 22);
            this.создатьПроектToolStripMenuItem.Text = "Создать проект";
            this.создатьПроектToolStripMenuItem.Click += new
System.EventHandler(this.создатьПроектToolStripMenuItem_Click);
            //
            // сохранитьToolStripMenuItem
            //
            this.сохранитьToolStripMenuItem.Name = "сохранитьToolStripMenuItem";
            this.сохранитьToolStripMenuItem.Size = new System.Drawing.Size(158, 22);
            this.сохранитьToolStripMenuItem.Text = "Сохранить";
            //
            // выходToolStripMenuItem
            //
            this.выходToolStripMenuItem.Name = "выходToolStripMenuItem";
            this.выходToolStripMenuItem.Size = new System.Drawing.Size(158, 22);
```

```
            this.выходToolStripMenuItem.Text = "Выход";
            this.выходToolStripMenuItem.Click += new
System.EventHandler(this.выходToolStripMenuItem_Click);
            //
            // оПрограммеToolStripMenuItem
            //
            this.оПрограммеToolStripMenuItem.Name = "оПрограммеToolStripMenuItem";
            this.оПрограммеToolStripMenuItem.Size = new System.Drawing.Size(94, 20);
            this.оПрограммеToolStripMenuItem.Text = "О программе";
            this.оПрограммеToolStripMenuItem.Click += new
System.EventHandler(this.оПрограммеToolStripMenuItem_Click);
            //
            // button1
            //
            this.button1.Font = new System.Drawing.Font("Times New Roman", 15.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.button1.Location = new System.Drawing.Point(32, 296);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(168, 33);
            this.button1.TabIndex = 1;
            this.button1.Text = "Создать проект";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // button2
            //
            this.button2.Font = new System.Drawing.Font("Times New Roman", 15.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.button2.Location = new System.Drawing.Point(32, 349);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(94, 33);
            this.button2.TabIndex = 2;
            this.button2.Text = "Выход";
            this.button2.UseVisualStyleBackColor = true;
            this.button2.Click += new System.EventHandler(this.button2_Click);
            //
            // Form1
            //
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Inherit;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.menuStrip1);
```

```
            this.MainMenuStrip = this.menuStrip1;
            this.Name = "Form1";
            this.Text = "Главное меню Визуализатора";
            this.Load += new System.EventHandler(this.Form1_Load);
            this.menuStrip1.ResumeLayout(false);
            this.menuStrip1.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.MenuStrip menuStrip1;
        private System.Windows.Forms.ToolStripMenuItem файлToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem создатьПроектToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem сохранитьToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem выходToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem оПрограммеToolStripMenuItem;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
    }
}
```

«VisualizerCode.cs» - исходный код главного меню

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Генератор_схем_алгоритмов
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void выходToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void создатьПроектToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Создание_схемы visual = new Создание_схемы();
            visual.ShowDialog();
        }

        private void оПрограммеToolStripMenuItem_Click(object sender, EventArgs e)
        {
            О_прогрмме program = new О_прогрмме();
            program.ShowDialog();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void button1_Click(object sender, EventArgs e)
```

```
        {
            Создание_схемы visual = new Создание_схемы();
            visual.ShowDialog();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}
```

«Core.cs» - ядро программы

```
namespace Core.BasicCore
{
    public class SourseCore
    {
        public string CodeOutput { get; private set; }
        public int braces = 0; //фигурные скобки

        public SourseCore(string code)
        {
            CodeOutput = "";
            CodeBasicCore(code);

        }
        private void CodeBasicCore(string codeInput)
        {
            for (int i = 0; i < codeInput.Length; i++)
            {
                if (codeInput[i] == '(')
                {
                    while (true)
                    {
                        if (codeInput[i] == ')')
                            break;

                        CodeOutput += codeInput[i];
                        i++;
                    }
                }

                if (codeInput[i] == ')' || codeInput[i] == ';' || codeInput[i] == '{')
                {
                    CodeOutput += codeInput[i];
                    CodeOutput += "\n"; //добавить новую строку

                    if (codeInput[i] == '{')
                    {
                        braces++; //увеличить количество скобок
                    }

                    for (int j = 1; j <= braces; j++)
                    {
```

```
                        CodeOutput += "\t"; //добавить вкладку
                    }

private void CreateConditionObject(Condition condition, string text)
        {
            Match op = _regex.BooleanOperator.Match(text);
            condition.BooleanOperator = op.Groups[0].ToString();

            Match conditionString = _regex.Condition.Match(text);
            Match param =
_regex.InstructionRegex.Match(conditionString.Groups[0].ToString());

            byte count = 0;
            while (param.Success)
            {
                if (count == 0)
                {
                    condition.LeftParameter = param.Groups[0].ToString();
                }
                else
                {
                    condition.RightParameter = param.Groups[0].ToString();
                }
                count++;
                param = param.NextMatch();
            }
        }

        private int CropScope(Scope scope, int start, string text)
        {
            Stack<char> stack = new Stack<char>();
            int end = start;
            while (text[start] != '{')
            {
                start = ++end;
            }
            stack.Push('{');

            while (stack.Count != 0)
            {
                end++;
                if (text[end] == '{')
                {
```

```
                stack.Push('{');
            }
            else if (text[end] == '}')
            {
                stack.Pop();
            }
        }
        CreateScopeObject(scope, text.Substring(start, end - start + 1));
        return ++end;
    }

    private void CreateScopeObject(Scope scope, string currentScope)
    {
        int start = 1, end = 1;
        while (end < currentScope.Length - 1)
        {
            string selectedText = currentScope.Substring(start, end - start + 1);
            if (_regex.FunctionRegex.IsMatch(selectedText))
            {
                Function funcObject = new Function();
                funcObject.IsBody = true;
                scope.Items.Enqueue(funcObject);
                CreateFunctionObject(funcObject, selectedText);
                start = ++end;
                start = end = CropScope(funcObject.Scope, start, currentScope);
            }
            else if (_regex.FunctionCall.IsMatch(selectedText))
            {
                Function funcObject = new Function();
                scope.Items.Enqueue(funcObject);
                CreateFunctionCallObject(funcObject, selectedText);
                start = ++end;
            }
            else if (_regex.VarDeclaration.IsMatch(selectedText))
            {
                CreateVariableObject(scope, selectedText, false);
                start = ++end;
            }
            else if (_regex.ArrayRegex.IsMatch(selectedText))
            {
                CreateVariableObject(scope, selectedText, true);
                start = ++end;
            }
```

```
                else if (_regex.IfRegex.IsMatch(selectedText))
                {
                    If ifObject = new If();
                    scope.Items.Enqueue(ifObject);
                    CreateConditionObject(ifObject.Condition, selectedText);
                    start = ++end;
                    start = end = CropScope(ifObject.Scope, start, currentScope);
                }
                else if (_regex.WhileRegex.IsMatch(selectedText))
                {
                    While whileObject = new While();
                    scope.Items.Enqueue(whileObject);
                    CreateConditionObject(whileObject.Condition, selectedText);
                    start = ++end;
                    start = end = CropScope(whileObject.Scope, start, currentScope);
                }
                else if(_regex.VarAssignmentRegex.IsMatch(selectedText) ||
_regex.ArrayAssignment.IsMatch(selectedText))
                {
                    CreateAssignmentObject(scope,selectedText);
                    start = ++end;
                }
                else if (currentScope[start] == ' ')
                {
                    start = ++end;
                }
                else if (selectedText == "\n")
                {
                    start = ++end;
                }
                else
                {
                    end++;
                }
            }
        }

        private void CreateFunctionCallObject(Function funcObject, string text)
        {
            if (_regex.ArgumentRegex.IsMatch(text))
            {
                Match parameters = _regex.ArgumentRegex.Match(text);
```

```
                Match m =
_regex.SingleInstructionRegex.Match(parameters.Groups[0].ToString());

                while (m.Success)
                {
                    funcObject.Parameters.Add(new Parameter(m.Groups[0].ToString()));
                    m = m.NextMatch();
                }
                text = _regex.ParameterRegex.Replace(text, "");
            }
            if (_regex.Variable.IsMatch(text))
            {
                Match m = _regex.Variable.Match(text);
                funcObject.Name = m.Groups[0].ToString();
            }
        }

        //будет объявлен глобальный объект объекта функции
        private void CreateFunctionObject(Function funcObject, string text)
        {
            if (_regex.ParameterRegex.IsMatch(text))
            {
                Match parameters = _regex.ParameterRegex.Match(text);
                Regex param = new Regex("(" + _regex.DataType + ")[\\s]+(" +
_regex.Variable + ")");
                Match m = param.Match(parameters.Groups[0].ToString());
                while (m.Success)
                {
                    Parameter p = new Parameter();
                    string[] parameterContent = m.ToString().Split(' ');

                    switch (parameterContent[0])
                    {
                        case "int":
                            p.Type = Enums.Type.Int;
                            break;
                        case "float":
                            p.Type = Enums.Type.Float;
                            break;
                        case "double":
                            p.Type = Enums.Type.Double;
                            break;
                        case "string":
```

```
                            p.Type = Enums.Type.String;
                            break;
                        case "char":
                            p.Type = Enums.Type.Char;
                            break;
                        case "bool":
                            p.Type = Enums.Type.Bool;
                            break;
                    }
                    p.Name = parameterContent[1];

                    funcObject.Parameters.Add(p);

                    m = m.NextMatch();
                }
                text = _regex.ParameterRegex.Replace(text, "");
            }
                }
                else if (codeInput[i] == '}')
                {
                    braces--; //уменьшить количество скобок
                    CodeOutput += "\n"; //добавить новую строку

                    for (int j = 1; j <= braces; j++)
                    {
                        CodeOutput += "\t"; //добавить вкладку
                    }
                    CodeOutput += codeInput[i];
                    CodeOutput += "\n"; //добавить новую строку

                    for (int j = 1; j <= braces; j++)
                    {
                        CodeOutput += "\t"; //добавить вкладку
                    }
                }
                else
                {
                    CodeOutput += codeInput[i];
                }
            }
        }
    }
}
```