МБОУ “Школа №38” г. Рязани

**«ОБРАБОТЧИК РЕЗУЛЬТАТОВ ЛАБОРАТОРНЫХ РАБОТ ПО ФИЗИКЕ (ДЛЯ 9 – 11 КЛАССОВ) – “PhysicsLabs”»**

Автор:

ученик 10 класса Асташкин Максим.

Научные руководители и консультанты работы:

Воробьев Юрий Николаевич, учитель физики;

Рагозина Ирина Ивановна, учитель информатики;

Конев Владимир Алексеевич, старший преподаватель кафедры

прикладной математики и вычислительной техники РГРТУ.

Рязань, 2018 год.

***Содержание:***

ВВЕДЕНИЕ………………………………………………………………..3

1.Программирование.…………………………………………………......4

* 1. Краткая история языков программирования………..…………. 4
  2. **О**бъектно – **О**риентированное Программирование…..………...4

1. Платформа Microsoft .NET 4.0……………….………………………..5
   1. Преимущества и недостатки платформы Microsoft .NET...……5
2. Концепция **«**PhysicsLabs»…………………….………………………..6
   1. Программирование – помощник в учебе………………………..6
   2. Концепция **«**PhysicsLabs» …………………….......6
   3. WPF(**W**indows **P**resentation **F**oundation**)**….……………………...7
3. Алгоритм «PhysicsLabs»……….………………………………………8
   1. Структура программы………..…………………………………...8
   2. Визуальное представление структуры………...............................9
4. Процесс написания программы………………………………….……10
   1. Использование системы контроля версий GIT + GitHub…..…..10
   2. Написание основной части кода и создание логотипа…………10
   3. Проблемы и ошибки, с которыми пришлось столкнуться..……11

ЗАКЛЮЧЕНИЕ ………..……………………………...………………….12

ПРИЛОЖЕНИЯ......……………………………………………………….13

**Введение.**

**Актуальность работы:**

**В процессе учебы нам, ученикам, приходится обрабатывать большое количество информации. Одним из видов обработки является оперирование полученными в результате выполнения лабораторных работ данными. Этот процесс скучен и отнимает достаточно много времени. Ранее физики использовали лаборантов, но мы живем в 21 веке, и компьютерная программа может заменить человека, хоть и не полностью.**

**Цель работы:**

**Получить компьютерную программу способную производить вычисления с данными, полученными в результате выполнения лабораторных работ и облегчающую процесс обучения для учеников, процесс проверки лабораторных работ для учителей.**

**Задачи:**

1. **Составить концепцию программы.**
2. **Написать алгоритм.**
3. **Написать программу.**
   1. **Разработать пользовательский интерфейс.**
   2. **Организовать ввод и вывод данных.**
   3. **Написать основную часть кода.**
4. **Разработать и нарисовать логотип.**
5. **Улучшить свои навыки программирования на платформе Microsoft .NET.**
6. **Улучшить своё владение языком C#**

**Программирование.**

**История языков программирования.**

**Машинные коды и Ассемблер**

Принципы работы электронно – вычислительных машин таков, что компьютер может распознавать и использовать команды, которые состоят лишь из единиц и нулей (Есть ток - нет тока; Намагничен диск – не намагничен). И вначале, человек составлял программы только на языке понятном компьютеру – в машинных кодах. Все команды состояли из операндов, которые в свою очередь состояли из единиц и нулей. Ученые пришли к тому, что этот язык громоздкий и неудобный. И действительно, можно было сделать ошибку в одной единичке или нуле, и вся команда бы не работала.

Уже в 1950 году программисты начали применять *мнемонический язык – язык assembly.* (См. Приложение 1, пункт «Ассемблер»)

**Первые языки программирования высокого уровня**

Уже с середины 50 – х годов начали появляться первые **“high – level”** языки. Главным их плюсом было то, что они не “привязывались” к определенному типу процессора. Для каждого из языков появлялись собственные компиляторы. ***Компиляция*** *- трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду.* Также несомненным плюсом является то, что языки высокого уровня имитируют естественные языки. Первым языком высокого уровня стал Фортан (1957), далее появились COBOL (1959), Паскаль (1970), Си (1972)

**Появление структурного программирования**

Одной из причин появления структурного программирования является необходимость увеличить производительность труда программистов. Эдсгер Дейстра – основоположник принципов структурного программирования (1968).

Следующими нововведениями ученых были процедуры и функции. То есть, при наличии повторяющейся задачи её можно объявить в функцию и далее в любом месте программы обращаться к ней.

Продуктом дальнейшего развития стали структуры (См. Приложение 1, пункт «Структуры»).

**Объектно – Ориентированное Программирование.**

В широком смысле **О**бъектно – **О**риентированное **П**рограммирование можно представлять, как моделирование информационных объектов. Это как никогда важно при реализации больших проектов. Главным преимуществом **ООП** является дальнейшее управление и поддержка проекта. Если говорить конкретнее, то можно выделять *полиморфизм*, *инкапсуляцию* и *наследование*. Основным объектом **ООП** является *класс* Главным отличием класса от структуры является наличие методов и более сложных тпов.

**Платформа Microsoft .NET 4.0 и язык C# 5.1**

.NET Framework –программная платформа, которая была выпущена Microsoft в 2002 году (Ходит слух, что в ответ на популярную платформу того времени Java). Ядром платформы является среда исполнения Common Language Runtime(**CLR**). Одним из преимуществ **CLR** является то, что она не “заточена” под определенный язык программирования: подходит для разных языков программирования. Такими языками стали: C#, Visual Basic, F# и другие (См. Приложение 1, пункт «.NET»).

**Язык C#**

**C#** (произносится си шарп) — объектно-ориентированный язык программирования, разработанный группой инженеров Microsoft под руководством Андерса Хейлсберга в 1998 – 2001 годах. (Cм. Приложение 1, пункт «C#»).

**Преимущества и недостатки платформы Microsoft .NET**

Технология .NET, которая появилась сравнительно недавно, имеет множество достоинств по сравнению с более старыми технологиями. Но, естественно, любой продукт имеет как преимущества, так и недостатки.

**Преимущества**:

1. Единые средства API (набор готовых классов, процедур, функций, структур и констант) для разработки программ на разных языках.
2. Простота соединения разноязыковых модулей.
3. Множество, упрощающих и ускоряющих процесс разработки приложения готовых классов и методов, которые реализуют различные алгоритмы.
4. Установка программ под .NET не требует программ-инсталляторов, делается простое копирование программы в нужную папку. Как следствие, при установке не вносятся ни какие записи в реестр Windows

**Недостатки**:

1. Заметно снижается скорость работы программ. (на 40 – 50 процентов). Но это не смертельно в связи с постоянным совершенствованием техники.
2. Требуется больше оперативной памяти. .NET имеет встроенный сборщик мусора. За него следует “платить” оперативной памятью.
3. На компьютере должна быть установлена среда выполнения программ Microsoft .NET.

**Анализируя выше упомянутые аргументы, я пришел к выводу: платформа .NET в связке с языком C# отлично подойдет для реализации моей идеи.**

**Концепция «PhysicsLabs»**

**Программирование – помощник в учебе.**

Ещё до того, как мне в голову пришла идея программы PhysicsLabs (с подачи Воробьева Юрия Николаевича), мною было написано множество консольных и оконных программ, помогающих мне в учебе. Одной из таких программ был решатель квадратных уравнений (См. Приложение2, Рис.1). В первой версии мой решатель мог всего лишь вычислять корни квадратного трехчлена. Но в более новых версиях появилось множество функций (См. Приложение 2, Рис.2). Например, построение графика(параболы) с точками пересечениями оси X.

На самом деле, этот решатель упростил мне жизнь. Я частенько пользуюсь решателем собственного “производства”, когда в составе какой-либо задачи мне приходится решать несколько квадратных уравнений.

И, забегая вперед, скажу, что в «PhysicsLabs» достаточно удобный и понятный интерфейс (См. Приложение 2, Рис.3). Именно благодаря ему обучение станет интересным даже для человека, который не любит физику. Так же программа не просто производит вычисления и выводит итоговые значения на экран, но и наглядно объясняет какие формулы были задействованы и их практическое применение в дальнейшем обучении.

**Концепция «PhysicsLabs»**

Как я уже упоминал раньше идея «PhysicsLabs» пришла мне на ум с подачи Воробьева Юрия Николаевича. На одном из уроков физики Юрий Николаевич сказал: “Какой смысл от этих расчетов? Красота физики не в этом. Этим должны заниматься математики.” После этих слов я подумал:” Почему бы не автоматизировать всю эту возню с числами?” Ведь я могу это сделать, тем самым облегчить обучение учеников хотя бы нашей школы. Да и работу учителей физики при проверке лабораторных работ. Действительно, обучение в школах благодаря новым технологиям можно сделать доступнее и проще.

Придя домой, я бросился рисовать концепты будущей программы (См. Приложение 2, Рис.4, Рис.5). Концепты предполагали: меню с выбором класса и лабораторной работы, окна лабораторных работ с вводом и выводом данных.

После того как было нарисовано мое виденье этой программы, я нашел список лабораторных работ по физике для 9-11 классов. Их было около 30 штук. Я понял, что писать все 30 отдельно - не вариант. Проконсультировавшись с Коневым Владимиром Алексеевичем, старшим преподавателем кафедры вычислительной техники и прикладной математики РГРТУ, и Рагозиной Ириной Ивановной, моим учителем информатики, я решил, что нужно находить рациональный способ написания программы. В следующем разделе и будет описан этот рациональный способ.

**WPF (Windows Presentation Foundation)**

Для написания пользовательского интерфейса программы мной была выбрана технология от Microsoft - **W**indows **P**resentation **F**oundation. Эта технология новее и удобнее более ранней Windows Forms: она позволяет сделать удобный и понятный интерфейс с минимальными затратами энергии программиста.

Так как для **WPF** требуется любой **.NET** совместимый язык, **C#** идеально подходит для выполнения этой роли.

В основе **WPF** лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. **WPF** предоставляет средства для создания визуального интерфейса, включая язык **XAML** (e**X**tensible **A**pplication **M**arkup **L**anguage), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

Графической технологией, лежащей в основе **WPF**, является **DirectX**, в отличие от **Windows Forms**, где используется GDI/GDI+. И именно благодаря **Directx**'у производительность приложения больше.

XAML представляет собой язык декларативного описания интерфейса, основанный на XML. Также реализована модель разделения кода и дизайна, позволяющая кооперироваться программисту и дизайнеру. Очень удобная функция, но не актуальная для моей работы.

Кроме того, есть встроенная поддержка стилей элементов, а сами элементы легко разделить на элементы управления второго уровня, которые, в свою очередь, разделяются до уровня векторных фигур и свойств/действий. Это позволяет легко задать стиль для любого элемента, например, для Button (кнопка).

**Алгоритм «PhysicsLabs»**

**Структура программы.**

В C# основными понятиями организационной структуры являются программы, пространства имен, типы, члены и сборки. Программа на языке C# состоит из одного или нескольких файлов. В программе объявляются типы, которые содержат члены. Эти типы можно организовать в пространства имен. Примерами типов являются классы и интерфейсы. К членам относятся поля, методы, свойства и события. При компиляции программы на C# упаковываются в сборки. Сборка — это файл, обычно с расширением .exe или .dll, если она реализует приложение или библиотеку, соответственно. Мой проект достаточно большой, и, естественно, он состоит из нескольких файлов.

Проект состоит из пространства имён PhysicsLabs, в котором находится класс главного окна (MainWindow.xaml.cs), файл пользовательского интерфейса главного окна на языке XAML (MainWindow.xaml), статический класс (Form.cs), который позволил мне не писать один и тот же код для разных лабораторных. (Использование этого статического класса - рациональное решение, о котором я упоминал в предыдущем разделе.) И пространство имён (папка) с ресурсами – картинками (Pictures).

Пространство имён PhysicsLabs включает в себя пространство имён Labs, которое включает в себя подпространства nine, ten, eleven (подпространства для классов лабораторных). Каждое подпространство имён включает в себя 3 объекта: основной класс лабораторной (tenOne.xaml.cs), в котором заключена основная логика лабораторной (ввод входных данных и вывод выходных, преобразованных из вида **LaTex** в нормальный, человеческий вид) и связь логики и пользовательского интерфейса; файл пользовательского интерфейса на языке **XAML**(tenOne.xaml) и статический класс (Calc.cs), в котором выполняется привязка входных данных к формулам, сами вычисления и вывод выходных данных в виде **LaTex** формул. Например, для того чтобы в коде программы программист мог обратиться первой лабораторной 10 класса, нужно использовать команду PhysicsLabs.ten.tenOne, где tenOne название лабораторной.

Если говорить конкретнее о структуре каждого класса можно выделять отдельные элементы. Все они взаимодействуют друг с другом.

* объявление пространства имен (своего рода контейнеры);
* объявление класса (основная сущность программы);
* методы класса (подпрограммы);
* операторы и выражения;
* комментарии.

Здесь уже можно говорить о концепциях **ООП.** О *полиморфизме, наследовании и инкапсуляции.*

**Визуальное представление структуры программы**

Естественно, визуальное представление структуры лучше рассматривать на рисунке.

На первой схеме (См. Приложение 2, Рис.6) рассмотрен частный случай, который наглядно показывает программный (скрытый) путь пользователя при взаимодействии с программой. В данном случае пользователь выбрал 10 класс, первую лабораторную работу, и алгоритм ”привёл” пользователя к окну ввода входных данных для первой лабораторной работы 10 класса.

На второй схеме (См. Приложение 2, Рис.7) отражена структура программы, которая была описана в предыдущем пункте.

Как можно заметить на второй схеме (Приложение 2, Рис.7), каждая лабораторная имеет одинаковый набор классов. Можно подумать и смутиться, потому что в каждой лабораторной работе имеется класс с именем “Calc.cs”. И возникает вопрос: как программисту обратиться к нужному ему классу? Ответ очень прост. Ранее я упоминал о пространствах имён. Именно в этой ситуации они играют ключевую роль. Calc является статическим классом (его объявление используется ключевое слово ***static*** (Приложение 2, Рис.8,)), и для того чтобы программисту обратиться к статическому классу первой лабораторной работы 10 класса нужно использовать команду – PhysicsLabs.Labs.ten.tenOne.Calc, а для того чтобы к статическому классу второй лабораторной работы - PhysicsLabs.Labs.ten.tenTwo.Calc. Не буду углубляться и говорить о модификаторах доступа, и что, на самом деле, не из любого места программы можно обратиться к какому-либо классу. Но в моем случае это и не требуется. Банально просто – прелесть ООП.

Пришло время рассказать о взаимодействии конкретных классов. После того как пользователь, выбрав класс и лабораторную, в главном меню нажимает на кнопку “ОК”, создается экземпляр класса выбранной пользователем лабораторной, который в свою очередь обращается к статическому классу Form.cs, который передает команды в файл пользовательского интерфейса, а тот уже в свою очередь создает (рисует) этот пользовательский интерфейс. Далее пользователю предлагается ввести входные данные, описанные в справке по лабораторной. Эти входные данные записываются в массив чисел с плавающей точкой “data”. После данные посылаются в ранее упомянутый статический класс Form.cs. Там они проверяются на правильность ввода. Если при вводе данных была допущена ошибка, пользователю предлагается вводить некорректные данные заново до тех пор, пока они не будет корректны. Далее проверенные данные отправляются в статический класс Calc.cs (От Calculator). В этом классе данные “прикручиваются” к физическим формулам, и с ними выполняются алгебраические операции. После того, как выходные данные посчитаны, формируется их вывод в красивом, человеческом виде. И они отправляются в основной класс лабораторной работы, где и осуществляется присваивание их к ранее созданному компоненту пользовательского интерфейса.

**Процесс написания программы.**

**Использование системы контроля версий.**

С самого начала разработки мне стало понятно, что если я буду хаотично вести учет версий программы и саму разработку, то ничего хорошего из этого не выйдет. Я принял решение: использовать **Git**. В двух словах **Git –** система контроля версий, созданная Линусом Торвальдсом. Изначально Линус не предполагал, что его проект будут использовать миллионы человек. Он создал её для управления разработкой ядра Linux.

Система **Git** состоит из нескольких программ, специально разработанных для создания специализированных (созданных для конкретных проектов) систем контроля версий. Благодаря командам в командной строке **Git Bash** можно легко и просто вести контроль над разработкой своей программы: создавать слепки патчей, вести различные ветви программы, сливать версии между собой. Так же для наглядности я использовал сервис **GitHub** с удаленным репозитирием **GitHub** (См. Приложение 2, Рис.8).

**Написание основной части кода и создание логотипа.**

Разработку программы я вёл с помощью нескольких программ под управлением операционной системы Microsoft Windows: Adobe Photoshop CS6 (для разработки логотипа и SplashScreen'a); Visual Studio 2017(VS 2017) Professional и Blend for Visual Studio 2017.

Отдельное место хочется выделить для VS 2017 (См. Приложение 2, Рис.9) , так как это моя основная рабочая программа. Именно с помощью неё я писал код программы и компилировал её. Отмечу, что уже к 2017 году Microsoft довели Visual Studio практически для совершенства. Думаю, что скоро VS будет сама писать код, нужно будет только успевать нажимать волшебные кнопки. На данный момент она может исправлять ошибки за программистом, предлагать различные способы рефакторинга (перепроектирование или переработка) кода, самостоятельно реализовывать программные интерфейсы (не стоит путать с пользовательскими).

При создании логотипа я советовался с людьми, чьи художественные способности намного лучше, чем мои. Логотип я представлял лаконичным и аккуратным. Первый вариант логотипа был просто текстовый (См. Приложение 2, Рис.10) . Но он не подходил по ряду причин. (товарный знак был размещен в качестве шутки). Новый логотип вместил в себя как знаковый стиль: очень узнаваемый силуэт Альберта Эйнштейна; так и текстовый: надпись “PhysicsLabs” (См. Приложение 2, Рис.11) .

Так же хочется отметить, что процесс написания кода, конечно, не сравнится с работой в шахте или на стройке, но тоже достаточно затратный. Сильно устают глаза, да и вообще сидячая работа пагубно может сказываться на человеческом здоровье. Так же это достаточно длительный процесс, и в связи с моим обучением в школе времени на программирование практические не остается. Приходится уделять ночное время.

Я разделяю написание кода «PhysicsLabs» на следующие этапы:

1. Разработка концептов. Разработка главного окна (меню).
   1. Разработка пользовательского интерфейса на языке **XAML**.
   2. Разработка логики.
2. Методом проб и ошибок нахождение рационального способа написания кода для лабораторных работ.
   1. Разработка пользовательского интерфейса на языке **XAML**.
   2. Разработка логики и расчетов.
   3. Переработка пользовательского интерфейса и его реализация с помощью статического класса (**C#**)**.**
   4. Создание статических классов в лабораторных для проведения расчетов и вывода данных в виде **LaTex**.
3. Исправление в программе.
   1. Рефакторинг кода.
   2. Debug программы - тестирование и нахождение уязвимостей в приложении.
4. Создание логотипа.
   1. Концепт логотипа.
   2. Создание иконок, логотипа для приложения.

**Проблемы и ошибки, с которыми пришлось столкнуться.**

Однозначно, главной проблемой, с которой пришлось справляться на протяжении всего проекта, было то, что нужно было находить рациональный способ разработки.

Основная моя ошибка заключалась в том, что я продолжал писать программу, осознавая то, что долго я так не протяну. Почему? Да потому что уже после написания первой лабораторной кода было очень много, и он был ужасно организован. Только лишь к концу написания второй лабораторной был окончательно написан рациональный способ. И, естественно, мне пришлось переписывать тот весь ужасный код.

Второй и последней проблемой в проекте была ошибка использования делегатов при присваивании обработчика “Click” для управляющих кнопок на окнах лабораторных. Но разрешить эту проблему мне помогли: официальная документация по платформе **.NET** и языку **C#** на сайте docs.microsoft.com, книга *Джеффри Рихтера «CLR via C# 4-е издание»* - глава 17 “Делегаты”.

Заключение.

В результате разработки проекта «PhysicsLabs» (10.11.2017 – 05.01.2018), поставленные мной задачи были выполнены сполна. Результатом выполнения работы стал готовый продукт - компьютерная программа, работающая под управлением платформы Microsoft .NET, которая прошла стадию бета – тестирования, и готова к использованию учениками 10 классов. Но я подразумеваю дальнейшую поддержку проекта, потому что в программе, на данный момент, доступны только лишь лабораторные 10 класса.

В нашем мире осталось ещё множество процессов, которые можно автоматизировать благодаря компьютерам и роботам. Думаю, это станет плодом моих дальнейших научно – практических работ.

Приложение 1

**Ассемблер**

Последовательностям из единиц и нулей присвоили определенные буквенные команды (отдельные буквы или сокращенные слова, которые отражали суть команды). Тем самым ученые добились более удобного представления машинного кода для человека. Например, в языке **ассемблера** команда, которая складывает два числа, обозначается словом “Add”. Но машинный код этой команды может быть таким “000010”. Действительно, **ассемблер** намного упростил жизнь программистам того времени. Он является языком низкого уровня. Но в данном случае нельзя отождествлять “низкий” и “плохой”. Имеется в виду, что его операторы приближены к машинным командам и “заточены” под конкретные команды для процессора.

Но на **языках низкого уровня** достаточно сложно создавать большие проекты и программные комплексы. Этот факт и привёл к созданию языков третьего поколения – языков высокого уровня. Но это не означает, что все прекратили пользоваться языками низкого уровня. До сих пор они пользуются популярностью, для решения определенных задач.

**Структуры**

Структура – тип данных, который состоит других типов данных. Так же структуры могут быть управляющими. Например, программные блоки. Благодаря этому подходу программу можно разбивать на составляющие элементы, тем самым улучшая читабельность кода и удобство для программиста. Но и этого было недостаточно для комфортного написания больших проектов. Нужен был новый подход к программированию. И этим подходом стали принципы **ООП.**

**.NET**

.NET является патентом компании Microsoft, и, естественно рассчитана на работу под операционными системами семейства Microsoft Windows. Но существуют отдельные независимые проекты (Mono, Portable .NET), которые позволяют запускать .NET приложения на других операционных системах. Так же хочется отметить, что .NET Framework, в отличии от Java, в данный момент поддерживается и получает обновления в виде .NET Core.

**C#**

**С#** - один из представителей семейства языков с “СИ” – подобным синтаксисом. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, **LINQ**, исключения, комментарии в формате **XML**.

Приложение 2

|  |
| --- |
| C:\Users\maksi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\37wTMVHGfhw.jpg |
| Рис.1 Консольный решатель уравнений ver. 0.7.1. |

|  |
| --- |
| Безымянный |
| Рис.2 Оконный решатель квадратных уравнений ver. 1.0.4. |

|  |
| --- |
| Безымянный |
| Рис.3 Интерфейс PhysicsLabs |
|  |
| Рис.4 Концепт меню. |
|  |
| Рис.5 Концепт окна лабораторной. |
| Схема структуры |
| Рис.6 Схема структуры «PhysicsLabs». |
| Структура |
| Рис.7 Структура «PhysicsLabs». |
| C:\Users\maksi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Безымянный.png |
| Рис.8 GitHub – изменения от 15 ноября 2017 |
| Безымянный |
| Рис.9 Часть исходного кода в программе Visual Studio 2017. |
| C:\Users\maksi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PhysicLabs.png |
| Рис.10 Старый вариант логотипа. |
| C:\Users\maksi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Logo.png |
| Рис.11 Новый вариант логотипа. |