Оглавление

[ВВЕДЕНИЕ 3](#_Toc41227028)

[1.АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc41227029)

[1.1 Тестирующая программа по C# 4](#_Toc41227030)

[1.2 Выбор языка и среды разработки 5](#_Toc41227031)

[1.3 Постановка задачи 6](#_Toc41227032)

[2.РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 7](#_Toc41227033)

[2.1 Разработка основного алгоритма программы 7](#_Toc41227034)

[2.2 Реализация основных блоков кода 8](#_Toc41227035)

[2.2.1 Класс Answers 8](#_Toc41227036)

[2.2.2 Класс Tasks 9](#_Toc41227037)

[2.2.3 Класс ConnectionToDatabase 9](#_Toc41227038)

[2.2.4 Класс MainWindow 10](#_Toc41227039)

[2.3 Контроль программы во время её выполнения 11](#_Toc41227040)

[2.4 Визуальная составляющая приложения 17](#_Toc41227041)

[ЗАКЛЮЧЕНИЕ 20](#_Toc41227042)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc41227043) 21

# ВВЕДЕНИЕ

Для начала давайте обозначим объект нашего исследования. Что же такое тест? Сегодня этот термин встречается довольно часто и в областях далеких от образования.

Тестирование (от слова *test* -- испытание, проверка). Тестирование применяется для определения соответствия предмета испытания заданным спецификациям. В задачи тестирования не входит определение причин несоответствия заданным требованиям. Тестирование - один из разделов диагностики. Тестирование применяется в технике, медицине, психиатрии, образовании для определения пригодности объекта тестирования для выполнения тех или иных функций. Качество тестирования и достоверность его результатов в значительной степени зависит от тестера.

Итак, можно определить тест, как стандартизированные, краткие, ограниченные во времени испытания, предназначенные для установления количественных и качественных индивидуальных различий.

В науке проводят существенные различия между простым переводом слова и смыслом понятия. Чаще всего мы встречаемся с упрощенным восприятием понятия "тест" как простой выбор одного ответа из нескольких предложенных к вопросу. Многочисленные примеры таких, казалось бы, "тестов" легко найти в газетно-журнальной периодике, в различных конкурсах и в многочисленных книжных публикациях под названием "Тесты". Но и это часто оказываются не тесты, а нечто внешне похожее на них. Обычно это сборники вопросов и задач, рассчитанных на выбор одного правильного ответа из числа предложенных. Они только по внешней видимости похожи на настоящий тест. Различия в понимании сущности тестов порождают различия в отношении к тестам. В наши дни существует много видов тестов, поэтому дать универсальное определение для всех этих видов вряд ли можно.

Традиционный тест представляет собой стандартизованный метод диагностики уровня и структуры подготовленности. В таком тесте все испытуемые отвечают на одни и те же задания, в одинаковое время, в одинаковых условиях и с одинаковыми правилами оценивания ответов. Главная цель применения традиционных тестов - установить уровень знаний. И на этой основе определить место (или рейтинг) каждого на заданном множестве тестируемых испытуемых. Для достижения этой цели можно создать бесчисленное количество тестов, и все они могут соответствовать достижению поставленной задаче.

И тогда возникает один из главных вопросов теории тестов - вопрос выбора наилучшего теста из практически неограниченного множества всех возможных тестов. Каждый тест может отличаться от других по числу заданий и другим характеристикам. С прагматической точки зрения выгодней делать тест, имеющий сравнительно меньшее число заданий, но обладающий большинством достоинств, присущих более длинным, как говорят в зарубежной теории, тестам. Понятие "длина теста" введено в начале XX века Ч. Спирманом и обозначает, на русском языке, количество заданий в тесте. Чем длиннее тест, тем больше в нем заданий. От числа заданий некоторым образом зависит точность педагогического измерения.

В тест стараются отобрать минимально достаточное количество заданий, которое позволяет сравнительно точно определить уровень и структуру подготовленности. Интерпретация результатов тестирования ведется преимущественно с опорой на среднюю арифметическую и на так называемые процентные нормы, показывающие, - сколько процентов испытуемых имеют тестовый результат худший, чем у любого другого испытуемого. Такая интерпретация тестовых результатов называется нормативно-ориентированной.

# 1.АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Тестирующая программа по C#

Необходимо создать тестирующую программу, которая будет проверять знания тестируемого человека. При этом должно быть не меньше восьми вопросов. В ходе тестирования будет видно количество заданных вопросов и количество не верных ответов. Вопросы будут выводиться в случайном порядке.

Специфическая форма тестовых заданий отличается тем, что задания теста представляют собой не вопросы и не задачи, а задания, сформулированные в форме высказываний, истинных или ложных, в зависимости от ответов. Традиционные вопросы, напротив, истинными или ложными не бывают, а ответы на них нередко настолько неопределенны и многословны, что для выявления их правильности требуются заметные, в суммарном исчислении, затраты интеллектуальной энергии преподавателей. В этом смысле традиционные вопросы и ответы нетехнологичны, и потому их лучше не включать в тест.

Ответ на задание теста представляет собой краткое суждение, связанное по содержанию и по форме с содержанием задания.

Каждому заданию ставятся в соответствие ответы правильные и неправильные.

Критерии правильности заранее определяются авторами теста. Вероятность правильного ответа на любое задание зависит от соотношения уровня знаний испытуемого и уровня трудности задания.

Существует множество разнообразных программ, которые производят проверку знаний учащихся, по различны предметам обучения.

Все обучающие программы могу быть написаны на разны языках программирования: PHP, Python, Delphi,C, C++, С #, Turbo Pascal и др.

Основная проблема всех существующих текстов - большой размер исполняемого файла, больше количество совершенно не нужных функций, не совместимость со старыми моделями компьютеров.

Например: Тестирующую программу по программированию на C# будет реализована на языке объектно-ориентированного языка программирования C#с использованием платформы пользовательского интерфейса для создания клиентских приложений для настольных систем WPF.

## 1.2 Выбор языка и среды разработки

Для написания этой курсовой работы выбран язык С#.

C# — компилируемый в CIL байт код, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как объектно-ориентированное программирование, процедурное программирование , обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C# сочетает свойства как высокоуровневых, так и низкоуровневых языков.

Интегрированная Среда Разработки (ИСР) − это среда, в которой есть все необходимое для проектирования, запуска и тестирования приложений, где все нацелено на облегчение процесса создания программ. ИСР интегрирует в себе редактор кодов, отладчик, инструментальные панели, редактор изображений, инструментарий баз данных, т.е. все, с чем приходится работать. Результатом является быстрая разработка сложных прикладных программ. Таким образом, IDE дает возможность получить ЕХЕ файл, не используя другие программы.

Для разработки программ на языке С# будет использована среда программирования Microsoft Visual Studio 2019 (ее английская аббревиатура − IDE: Integrated Development Environment), которая содержит в себе средства создания программы, ее компиляции, отладки и запуска на выполнение. В этой связи рассмотрим кратко структуру этой среды, а точнее ее интерфейс. Интерфейс − это аппарат, который позволяет удобно взаимодействовать пользователю со средой.

## 1.3 Постановка задачи

В задачу курсового проекта входит разработать программное средство на языке С#, которое:

* Будет позволять пользователю проверить свои знания по языку C#

# 2.РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 2.1 Разработка основного алгоритма программы

Основной алгоритм программы написан на языке программирования С#, который является объектно-ориентированным языком программирования.

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

* абстрагирование для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса;
* инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;
* наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах;
* полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

То есть фактически речь идёт о прогрессирующей организации информации согласно первичным семантическим критериям: «важное/неважное», «ключевое/подробности», «родительское/дочернее», «единое/множественное». Прогрессирование, в частности, на последнем этапе даёт возможность перехода на следующий уровень детализации, что замыкает общий процесс.

Обычный человеческий язык в целом отражает идеологию ООП, начиная с инкапсуляции представления о предмете в виде его имени и заканчивая полиморфизмом использования слова в переносном смысле, что в итоге развивает[3] выражение представления через имя предмета до полноценного понятия-класса.

Поэтому программа разделена на классы:

* Answers
* Tasks
* ConnectionToDatabase
* MainWindow

## 2.2 Реализация основных блоков кода

Так как код программы достаточно большой, то весь он разделён на файлы. Каждый класс находится в отдельном файле .cs, объекты которых затем создаются в основном классе программы MainWindow.xaml.cs, где находится класс формы приложения.

Будет приведен только частичный код классов : создание объектов и свойств, конструкторов классов, а из класса ConnectionToDatabase будет приведена одна из функций подключения к базе данных.

### 2.2.1 Класс Answers

Этот класс представляет собой ответ на задание теста, выводимый в компонент формы ListBox.

public class Answers

{

public int Id { get; set; }

public string Answer { get; set; }

public int PreviousId { get; set; }

public Answers(int id, string answer)

{

Id = id;

Answer = answer;

}

}

### 2.2.2 Класс Tasks

Этот класс представляет собой задание, считываемое из базы данных и выводимое в компонент формы ListBox.

public class Tasks

{

public int Id { get; set; }

public string Task { get; set; }

public string TaskId { get; set; }

public int Current { get; set; }

public int CorrectAnswer { get; set; }

public int UserAnswer { get; set; }

public Tasks(int id, string value, int current, int correct)

{

Id = id;

Task = value;

TaskId = current.ToString() + "." + " " + Task;

UserAnswer = -1;

CorrectAnswer = correct;

}

}

### 2.2.3 Класс ConnectionToDatabase

Этот класс и его методы считывают информацию с базы данных MySql с помощью пространства имён MySql.Data.MySqlClient.

class ConnectionToDatabase

{

public int CurrentTestId { get; set; }

public ObservableCollection<string> splitReadView = new ObservableCollection<string>();

public ArrayList ConnectionToTestNumbers()

{

string connStr = "server=localhost;user=root;database=kursach;password=0000;";

MySqlConnection connect = new MySqlConnection(connStr);

connect.Open();

string sql = "SELECT id,name FROM testnumbers";

MySqlCommand comm = new MySqlCommand(sql, connect);

MySqlDataReader read = comm.ExecuteReader();

ArrayList readInt = new ArrayList();

while (read.Read())

{

readInt.Add(int.Parse(read[0].ToString()));

}

connect.Close();

return readInt;

}

### 2.2.4 Класс MainWindow

Это класс формы приложения, который является главным классом всей программы.

public partial class MainWindow : Window

{

//Коллекция для ComboBox

public ObservableCollection<string> list = new ObservableCollection<string>();

//Коллекция для вывода заданий теста в TaskList

public ObservableCollection<Tasks> listOfTasks { get; set; }

//Коллекция для вывода ответов теста в AnswerList

public ObservableCollection<Answers> listOfAnswers { get; set; }

ConnectionToDatabase model = new ConnectionToDatabase();

DispatcherTimer timer;

DispatcherTimer timerOne;

TimeSpan timeForTest = new TimeSpan(0, 10, 0);

private int TestIdFromComboBox;

public MainWindow()

{

InitializeComponent();

TestIdFromComboBox = 1;

ArrayList source = model.ConnectionToTestNumbers();

foreach (int i in source)

{

list.Add(i.ToString());

}

ComboMain.ItemsSource = list;

ComboMain.SelectedIndex = 0;

TaskList.FontSize = 12;

AnswerList.FontSize = 12;

Button1.IsEnabled = true;

Button2.IsEnabled = false;

MessageBox.Show("Вас приветствует программа для решения тестов по языку программирования C#!" + "\n" +

"Для того, чтобы начать тест нажмите кнопку <Начать тест>." + "\n" +

"Вам будут предоставлены 10 вопросов с 4 вариантами ответов и 10 минут для их решения." + "\n" +

"После того, как вы выполнили задания, нажмите кнопку <Завершить тест> и вам будут предоставлены ваши результаты." + "\n" +

"Желаем удачи!");

}

## 2.3 Контроль программы во время её выполнения

Все алгоритмы контроля запущенной программы находятся в файле формы и описаны в событиях.

Форма является одним из важнейших элементов архитектуры C# WPF.

Технология WPF использует многоуровневую архитектуру. На вершине ваше приложение взаимодействует с высокоуровневым набором служб, которые полностью написаны на управляемом коде C#. Действительная работа по трансляции объектов .NET в текстуры и треугольники Direct3D происходит "за кулисами", с использованием низкоуровневого неуправляемого компонента по имени **milcore.dll**. Библиотека milcore.dll реализована в неуправляемом коде потому, что ей требуется тесная интеграция с Direct3D, и вдобавок для нее чрезвычайно важна производительность.

На рисунке показаны уровни, на которых построена работа приложения WPF:

Изображение выглядит как снимок экрана

Автоматически созданное описание

Описание нескольких ключевых компонентов архитектуры WPF:

Milcore.dll

Ядро системы визуализации WPF и фундамент уровня медиаинтеграции (англ. Media Integration Layer — MIL). Его составной механизм транслирует визуальные элементы в треугольники и текстуры, которых ожидает Direct3D. Хотя milcore.dll считается частью WPF, это также важнейший компонент операционных систем Windows Vista и Windows 7. В действительности DWM (англ. Desktop Window Manager — диспетчер окон рабочего стола) использует milcore.dll для отображения рабочего стола.

Сбоpку milcore.dll иногда называют механизмом "управляемой графики". Подобно тому, как общеязыковая исполняющая среда (*англ. Common Language Runtime — CLR*) управляет жизненным циклом приложения .NET, milcore.dll управляет состоянием дисплея. И так же, как CLR избавляет от забот об освобождении объектов и восстановлению памяти, milcore.dll избавляет от необходимости думать о недействительности и перерисовке окна. Вы просто создаете объекты с содержимым, которое хотите отобразить, a milcore.dll рисует соответствующие части окна, когда оно перемещается, скрывается и раскрывается, сворачивается и восстанавливается, и т.д.

Direct3D

Низкоуровневый API-интерфейс, через который визуализируется вся графика в WPF.

User32

Используется для определения того, какое место на экране к какой программе относится. В результате он по-прежнему вовлечен в WPF, но не участвует в визуализации распространенных элементов управления.

Наиболее важный факт, который потребуется осознать, состоит в том, что Direct3D визуализирует все рисование в WPF. При этом не важно, установлена на компьютере видеокарта со скромными возможностями или же более мощная, используются базовые элементы управления или рисуется более сложное содержимое, запускается приложение в Windows ХР, Windows Vista или Windows 7. Даже двумерные фигуры и обычный текст трансформируются в треугольники и проходят по трехмерному конвейеру. Какие-либо обращения к GDI+ или User32 отсутствуют.

Процесс редактирования формы происходит при добавлении к форме компонентов, изменении их свойств, создании обработчиков событий. Когда создаётся новый проект приложение WPF в Microsoft Visual Studio 2019, сразу в проекте создаётся 2 файла: 1) .xaml 2) .xaml.cs

При добавлении компонента к форме модифицируется файл .xaml. Все изменения формы можно произвести и отследить в элементе управления “Окно закладок”.

События. Каждый экземпляр компонента имеет свой собственный набор функций - обработчиков событий. Создавая обработчик события, вы поручаете программе выполнить указанную функцию, если это событие произойдет. Чтобы добавить обработчик события, нужно выбрать компонент, затем открыть страницу событий Инспектора объектов и дважды щелкнуть левой клавишей мыши рядом с событием. Это заставит Microsoft Visual Studio 2019 сгенерировать текст пустой функции с курсором в том месте, где следует вводить код. Далее нужно ввести код, который должен выполняться при наступлении данного события.

private void Button1\_Click(object sender, RoutedEventArgs e)

{

model.CurrentTestId = 0;

Button1.IsEnabled = false;

Button2.IsEnabled = true;

ComboMain.IsEnabled = false;

var selectedItem = ComboMain.SelectedItem;

if (selectedItem != null)

{

try

{

string selectedItemStr = selectedItem.ToString();

TestIdFromComboBox = int.Parse(selectedItemStr);

}

catch (InvalidCastException ex)

{

MessageBox.Show(ex.Message);

}

}

ArrayList task = model.ConnectionToTestTasks(TestIdFromComboBox);

listOfTasks = new ObservableCollection<Tasks>();

foreach (Object obj in task)

{

listOfTasks.Add((Tasks)obj);

}

TaskList.ItemsSource = listOfTasks;

timeForTest = TimeSpan.FromMinutes(10);

timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromSeconds(1);

timer.Tick += timer\_Tick;

timer.Start();

}

void timer\_Tick(object sender, EventArgs e)

{

timeForTest = timeForTest.Subtract(TimeSpan.FromSeconds(1));

lblTime.Content = timeForTest.ToString();

}

private void TaskList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

var selectedItem = ComboMain.SelectedItem;

if (selectedItem != null)

{

try

{

string selectedItemStr = selectedItem.ToString();

TestIdFromComboBox = int.Parse(selectedItemStr);

}

catch (InvalidCastException ex)

{

MessageBox.Show(ex.Message);

}

}

//Создание объеста класса Tasks

Tasks task = (Tasks)TaskList.SelectedItem;

if (task != null)

{

ArrayList answer = model.ConnectionToTestAnswers(TestIdFromComboBox, task.Id);

listOfAnswers = new ObservableCollection<Answers>();

foreach (Object obj in answer)

{

listOfAnswers.Add((Answers)obj);

}

AnswerList.ItemsSource = listOfAnswers;

if (task.UserAnswer != -1)

{

timerOne = new DispatcherTimer();

timerOne.Interval = TimeSpan.FromMilliseconds(10);

timerOne.Tick += timerOne\_Tick;

timerOne.Start();

}

}

}

void timerOne\_Tick(object sender, EventArgs e)

{

Tasks task = (Tasks)TaskList.SelectedItem;

if (task != null)

{

ListBoxItem lbi2 = (ListBoxItem)AnswerList.ItemContainerGenerator.ContainerFromIndex(task.UserAnswer - 1);

lbi2.Foreground = Brushes.Blue;

}

timerOne.Stop();

}

private void AnswerList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

//Создание объекта класса Answers

Answers answer = (Answers)AnswerList.SelectedItem;

Tasks task = (Tasks)TaskList.SelectedItem;

if (answer != null)

{

int answerId = answer.Id + 1;

if (task.UserAnswer != -1)

{

ListBoxItem lbp = (ListBoxItem)AnswerList.ItemContainerGenerator.ContainerFromIndex(task.UserAnswer - 1);

lbp.Foreground = Brushes.Black;

}

task.UserAnswer = answerId;

ListBoxItem lbi = (ListBoxItem)TaskList.ItemContainerGenerator.ContainerFromIndex(TaskList.SelectedIndex);

lbi.Foreground = Brushes.Blue;

ListBoxItem lbi2 = (ListBoxItem)AnswerList.ItemContainerGenerator.ContainerFromIndex(AnswerList.SelectedIndex);

lbi2.Foreground = Brushes.Blue;

}

}

private void Button2\_Click(object sender, RoutedEventArgs e)

{

timer.Stop();

Button2.IsEnabled = false;

Button1.IsEnabled = true;

ComboMain.IsEnabled = true;

int i = 0;

int tempAnswer = 0;

int nonAnswer = 0;

foreach (Object obj in listOfTasks)

{

Tasks task = (Tasks)obj;

ListBoxItem lbi = (ListBoxItem)TaskList.ItemContainerGenerator.ContainerFromIndex(i);

if (task.CorrectAnswer == task.UserAnswer)

{

lbi.Foreground = Brushes.Green;

tempAnswer++;

}

else

{

lbi.Foreground = Brushes.Red;

nonAnswer++;

}

i++;

}

if (tempAnswer >= 6 && tempAnswer <= 10)

{

MessageBox.Show("Ваш результат составил: "

+ tempAnswer.ToString() + "\n"

+ "Вы не ответили на "

+ nonAnswer.ToString() + " вопроса(ов)" + "\n"

+ "Поздравляем! Вы сдали тест!");

}

else

{

MessageBox.Show("Ваш результат составил: "

+ tempAnswer.ToString() + "\n"

+ "Вы не ответили на "

+ nonAnswer.ToString() + " вопроса(ов)" + "\n"

+ "К сожалению, в этот раз тест сдать не получилось.");

}

}

## 2.4 Визуальная составляющая приложения

Интерфейс приложения создаётся с помощью компонентов

Типовые элементы управления WPF. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы объектов, называемых компонентами. Microsoft Visual Studio 2019 позволяет разработчику создавать собственные компоненты с помощью языка разметки, используемого для инициализации объектов в технологиях на платформе .NET.

Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются как во время выполнения, так и во время проектирования. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они не видны во время выполнения, но обладают функциональностью. Для добавления компонента в форму можно выбрать мышью нужный компонент в Панеле элементов и щелкнуть левой клавишей мыши в нужном месте проектируемой формы. Компонент появится на форме, и далее его можно перемещать и изменять. Каждый компонент C++ Builder имеет три характеристики: свойства, события и методы.

Обозреватель решений автоматически показывает свойства и события, которые могут быть использованы с компонентом. Свойства являются атрибутами компонента, определяющими его внешний вид и поведение.

На форму добавлен набор компонентов:

* Тег(Label)
* Кнопки(Button)
* Комбинированный список(ComboBox)
* Окно списка(ListBox)

Главная форма

Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

Автоматически созданное описание

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта была разработана тестирующая программа по C#, проверяющая знания пользователя по данному языку программирования. В ходе его разработки были получены теоретические и практические знания по языку программирования С#, а также по принципам объектно-ориентированного программирование. Была создана база данных MySQL с помощью программы MySQL Workbench 8.0, содержащая все тесты используемые в программе, при добавлении новых тестов в базу данных они будут автоматически добавляться в тестирующую программу.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.WIKIPEDIA [Электронный ресурс] – WIKIPEDIA: https://ru.wikipedia.org

2.METANIT[Электронный ресурс] – METANIT:

https://metanit.com

3.Язык программирования C# 2010 и платформа .NET 4 5-е издание Эндрю Троелсен Вильямс, 2011. 995-1213 с.