МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
УНИВЕРСИТЕТСКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Допустить к защите
Заместитель директора
по УМР
_____ Е.Г. Конакина
(Подпись) (ФИО)
«___»_____ 2017 г.

КУРСОВОЙ ПРОЕКТ

Тема «ПРИЛОЖЕНИЕ GUITAR KEYBOARD»

специальность 09.02.03 группа 32928/2

Студент (ка) _____ Тимушев Ф.А.
(подпись) (ФИО)
Преподаватель _____ Девятко Н.С.
(подпись) (ФИО)

Санкт-Петербург
2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В наше время технологии шагнули далеко, упрощая жизнь обычных людей. Те, в свою очередь, даже не подозревая, как работает устройство, используют его повседневно и повсеместно.

Однако они имеют представление о том, что устройство использует программы. Пользователь достаточно продвинут для того, чтобы найти, установить и, наконец, понять, как работать с той или иной программой.

Программы же по своей цели бывают узконаправленные и могут решать лишь одну или небольшую часть проблем или целей, а бывают и те, что могут работать масштабно, глобально – решая задачи, порой, не одного, а сразу огромного количества пользователей разных направленностей.

Разработанная в данном курсовом проекте программа может быть использована как развлекательных, так и в практических целях и является узконаправленным приложением для круга лиц, интересующихся музыкой или музыкальным оборудованием.

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ

1.1 Описание предметной области

Приложение — это прикладной компьютерный сервис, который обладает набором определенных функций и является одним из компонентов программного обеспечения. Проще говоря, это программа, которая выполняет некоторые действия, чтобы облегчить жизнь пользователю или решить ту или иную проблему.

Приложения могут быть использованы как в рабочих или учебных целях, так и для хобби, развлечения ли общения. Люди творческих профессий (художники, дизайнеры и т.д.) с помощью приложений создают рисунки и эскизы. Учителя разрабатывают планы уроков. Архитекторы рисуют чертежи. Для водителей есть приложения-карты, которые помогают отследить свое место нахождения и проложить путь с одной точки в другую. Поклонники здорового питания или люди, следящие за весом, могут воспользоваться программой для подсчета калорий в том или ином блюде. Незаменимым помощником для спортсмена часто становится программа, которая считает затраты энергии и контролирует показатели жизнедеятельности (пульс, давление). Этот список можно продолжать до бесконечности, ведь существует несколько тысяч приложений с разными функциями.

Приложения бывают разными. Компьютерное приложение — это сервис, который устанавливается на ПК или ноутбук. Веб-приложение — это программа, которая работает с браузером. Программы, которые можно установить на смартфон, планшет или тому подобный «умный» гаджет, называются мобильными. Каждое приложение разрабатывается под конкретную операционную систему: программы для ПК, мобильные приложения пишутся под Android, BlackBerry OS, Apple IOS, Symbian, Windows.

Платное или бесплатное приложение — что лучше? Приобрести платное приложение или установить бесплатную программу — решать только пользователю. Обычно у платных приложений функционал шире, поэтому для работы, учебы или творчества лучше купить нужную программу с максимальным набором возможностей. Для развлечений подойдут и программы попроще — сегодня на рынке программного обеспечения есть масса бесплатных игр и тому подобных приложений.

Многие разработчики предлагают перед покупкой воспользоваться пробной версией программы, которая действует от нескольких недель до нескольких месяцев. После окончания «испытательного срока» программы можно купить лицензию, которая даст право использовать приложение без ограничений во времени. Перед тем как купить программу, имеет смысл проверить, удовлетворяют ли ее функции все требования пользователя. Можно также поинтересоваться отзывами людей, у которых есть опыт использования того или иного приложения. Использование прикладных программ помогает пользователю выполнять те или иные действия с максимальной эффективностью и минимальными затратами времени и сил.

Многие приложения могут стать хорошими помощниками в учебе и работе, некоторые служат развлечением или средством общения, а есть и такие программы, которые помогают создавать картины, музыку (как в нашем случае) или другие произведения современного искусства.

1.2 Анализ методов решения

Мое приложение является компьютерным и может быть применено несколькими способами:

1) использование записи нот в «письменный» или «буквенный» код, подобно табулатурам – это может быть альтернативный способ записи нотной грамоты для более простого запоминания;
2) средство альтернативной записи и передачи данных, звучит необычно и отдаленно, но представим, что у вас неисправна клавиатура – помимо экранной можно использовать гитару, что достаточно весело и, как я уже говорил, необычно);
3) настройка музыкального оборудования (зачастую – электрогитар), что является немаловажным, можно сказать – основным, аспектом при игре на музыкальных инструментах.

1.3 Обзор средств программирования

Средством программирования данного курсового проекта является среда разработки C#.

Требования к курсовому проекту включали удобный пользовательский интерфейс, поэтому при написании данного курсового проекта моя задача состояла в выборе между Java, C#, Delphi.

Java и Delphi я не стал выбирать, так как попросту их не знаю.

При моих текущих знаниях, C# оказался достаточно удобным в плане написания программы и интерфейса к ним, благодаря не сколько простоте языка, но огромных возможностей в подключении дополнительных или сторонних модулей для стилизации и усовершенствования программного продукта.

1.4 Описание языка C#

Средством программирования данного курсового проекта является среда разработки C#.

На сегодняшний момент язык программирования C# одни из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

По сравнению с другими языками C# достаточно молодой, но в то же время он уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 6.0, которая вышла в 20 июля 2015 года вместе с Visual Studio 2015.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче.

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

Алфавит языка C#

Все тексты на языке пишутся с помощью его алфавита. В C# используется кодировка символов Unicode. Кодировкой, или кодовой таблицей (character set),

называется соответствие между символами и кодирующими их числами. Кодировка Unicode позволяет представить символы всех существующих алфавитов одновременно. Каждому символу соответствует свой уникальный код.

Алфавит C# включает:

- буквы (латинские и национальных алфавитов) и символ подчеркивания (_), который употребляется наряду с буквами;

- цифры;

- специальные символы, например +, *, { и &;

- пробельные символы (пробел и символы табуляции);

- символы перевода строки.

Из символов составляются более крупные строительные блоки: лексемы, директивы препроцессора и комментарии.

Лексема (token) - это минимальная единица языка, имеющая самостоятельный смысл. Существуют следующие виды лексем:

- имена (идентификаторы);

- ключевые слова;

- знаки операций;

- разделители;

- литералы (константы).

Лексемы языка программирования аналогичны словам естественного языка. Например, лексемами являются число 128 (но не его часть 12), имя Vasia, ключевое слово goto и знак операции сложения +. Далее мы рассмотрим лексемы подробнее.

Директивы препроцессора пришли в C# из его предшественника - языка C++. Препроцессором называется предварительная стадия компиляции, на которой формируется окончательный вид исходного текста программы. Например, с помощью директив (инструкций, команд) препроцессора можно включить или выключить из процесса компиляции фрагменты кода. Директивы препроцессора не играют в C# такой важной роли, как в C++.

Комментарии предназначены для записи пояснений к программе и формирования документации. Правила записи комментариев мы рассмотрим чуть позже.

Из лексем составляются выражения и операторы. Выражение задает правило вычисления некоторого значения. Например, выражение a + b задает правило вычисления суммы двух величин.

Операторы языка C#

Любое выражение, завершающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении выражения. Частным случаем выражения является пустой оператор; (он используется, когда по синтаксису оператор требуется, а по смыслу - нет). Примеры:

i++; // выполняется операция инкремента*= b + c; // выполняется умножение с присваиванием(i, k); // выполняется вызов функции(true); // цикл из пустого оператора (бесконечный)

Блок, или составной оператор, - это последовательность описаний и операторов, заключенная в фигурные скобки. Блок воспринимается компилятором как один оператор и может использоваться всюду, где синтаксис требует одного оператора, а алгоритм - нескольких. Блок может содержать один оператор или быть пустым.

Условный оператор if используется для разветвления процесса вычислений на два направления.

Оператор switch (переключатель) предназначен для разветвления процесса вычислений на несколько направлений.

Операторы цикла используются для вычислений, повторяющихся многократно. Блок, ради выполнения которого и организуется цикл, называется телом цикла. Остальные операторы служат для управления процессом повторения вычислений: это начальные установки, проверка условия продолжения цикла и модификация параметра цикла. Один проход цикла называется итерацией.

Начальные установки служат для того, чтобы до входа в цикл задать значения переменных, которые в нем используются.

Проверка условия продолжения цикла выполняется на каждой итерации либо до тела цикла (тогда говорят о цикле с предусловием), либо после тела цикла (цикл с постусловием).

Параметром цикла называется переменная, которая используется при проверке условия продолжения цикла и принудительно изменяется на каждой итерации, причем, как правило, на одну и ту же величину. Если параметр цикла целочисленный, он называется счетчиком цикла.

Цикл завершается, если условие его продолжения не выполняется. Возможно принудительное завершение как текущей итерации, так и цикла в целом. Для этого служат операторы break, continue, return и goto. Передавать управление извне внутрь цикла запрещается.

Ключевые слова C#:

abstract	extern	null	struct
as	false	object	switch
base	finally	operator	this
bool	fixed	out	throw
break	float	override	true
byte	for	params	try
case	foreach	private	typeof
catch	goto	protected	uint
const	if	public	ulong
continue	implicit	readonly	unchecked
decimal	in	ref	unsafe
default	int	return	ushort
delegate	interface	sbyte	using
do	internal	sealed	virtual
double	is	short	void
else	lock	sizeof	while
enum	long	stackalloc	
event	namespace	static	
explicit	new	string	

Контекстные ключевые слова C#

add	equals	join	set
ascending	from	let	value
async	get	on	var
await	global	orderby	where
by	group	partial	yield
descending	in	remove	
dynamic	into	select	

Основные алгоритмические конструкции C#

Оператор присваивания	`string hello = "hello " + "world"; //результат равен "hello world"`
	`int x1 = 2 + 4; // результат равен 6`
	`int x2 = 10 - 6; //результат равен 4`
	`int x3 = 10 * 6; //результат равен 60`
	`double x4 = 10.0 / 4.0; //результат равен 2.5`
	`double x5 = 10.0 % 4.0; //результат равен 2`
	`int y1 = 5;`
	`int z1 = ++y1; // z1=6; y1=6`
	`int y2 = 5;`
	`int z2 = y2++; // z2=5; y2=6`
	`int y3 = 5;`
	`int z3 = --y3; // z3=4; y3=4`
	`int y4 = 5;`
	`int z4 = y4--; // z4=5; y4=4`

Условный оператор	**if** (*условие)* **{**(*действие)* } **else** {(*альтернатива)}* ;
	`int num1 = 8;`
	`int num2 = 6;`
	`if(num1 > num2)`
	`{`
	`  Console.WriteLine("Число {0} больше числа {1}", num1, num2);`
	`}`
	`else`
	`{`
	`  Console.WriteLine("Число {0} меньше числа {1}", num1, num2);`
	`}`

	Конструкция switch/case аналогична конструкции if/else, так как позволяет обработать сразу несколько условий:
	`Console.WriteLine("Нажмите Y или N");`
	`string selection = Console.ReadLine();`
	`switch (selection)`
	`{`
	`  case "Y":`
	`  Console.WriteLine("Вы нажали букву Y");`
	`  break;`
	`  case "N":`
	`  Console.WriteLine("Вы нажали букву N");`
	`  break;`
	`  default:`
	`  Console.WriteLine("Вы нажали неизвестную букву");`
	`  break;`
	}
	После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора case. И если совпадение будет найдено, то

	будет выполняться определенный блок case. *В конце блока case ставится оператор break, чтобы избежать выполнения других блоков.* *Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default, как в примере выше.*

Арифметический цикл (применяется, когда известно количество повторений цикла)	*for ([инициализация счетчика]; [условие]; [изменение счетчика])*
	`for (int i = 0; i < 9; i++)`
	`{`
	`  Console.WriteLine("Квадрат числа {0} равен {1}", i, i * i);`
	`}`

Цикл с предусловием (применяется, когда неизвестно количество повторений цикла)	Этот цикл будет выполняться до тех пор, пока истинно *условие* (логическое выражение, возвращающее значение типа **Boolean**). При этом если это выражение сразу равно **false**, *тело цикла* не будет выполнено ни разу. Нужно очень внимательно следить за написанием *условия* и контролем завершения цикла, так как в результате ошибки цикл **while** будет повторяться бесконечное количество раз, что приведёт к "зацикливанию" и "зависанию" программы.
Цикл с постусловием (применяется, когда неизвестно количество повторений цикла)	*do { тело цикла } while условие;* Повторения сначала выполняет *тело цикла*, а затем уже проверяет выполнение *условия*: Таким образом, этот вариант цикла гарантирует, что *тело цикла* будет выполнен по крайней мере один раз. И будет выполняться до тех пор, пока *условие* не станет истинным (т.е. **true**).

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Постановка задачи

2.1.1 Основания для разработки

Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.02 «Прикладное программирование» и утверждена Университетским политехническим колледжем.

2.1.2 Назначение программы

Приложение «Guitar Keyboard»: пользователь активирует приложение, с помощью которого он может, посредством электрогитары, подключенной к компьютеру, или акустической гитары, записывающейся через микрофон, считывать входной сигнал, определять его уровень, сопоставлять значение этого уровня с определенным символом и выводить эти символы в любом доступном текстовом поле.

2.1.3 Требования к программе

2.1.3.1 Требования к функциональным характеристикам

- поочередная игра нот на гитаре;
- считывание нот (уровня входного сигнала) программой;
- сопоставление определенного уровня с определённым символом;
- вывод символа в текстовое поле;
- сопоставление определенного уровня с эталоном;

- вывод уровня на экран для помощи в настройке гитары;

1 версия:

- программа должна включаться/выключаться посредством специальной кнопки;
- программа должна определять частоту и уровень входного сигнала;
- программа должна выводить уровень входного сигнала;
- программа должна печатать символы, соответствующие определенной частоте в специальном окне или в полях ввода;
- программа должна выводить подсказку для удобства настройки гитары;

2 версия:

- программа должна графически отображать уровень входного сигнала;
- программа должна менять язык интерфейса по желанию пользователя;
- программа должна переходить в режим тюнера и обратно в режим ввода посредством специальной кнопки;
- программа должна иметь выбор тюна для настройки гитары (3 вида);
- программа должна иметь выбор струны в тюне для помощи в настройке гитары.

2.1.3.2 Требования к надежности

- использование лицензированного программного обеспечения;
- проверка программы на наличие вирусов;
- организация бесперебойного питания.

2.1.3.3 Требования к условиям эксплуатации

Программа должна эксплуатироваться в условиях вычислительного центра при температуре от +15 до +35 С и относительной влажности воздуха от 25 до 85%.

Программа не требует специального обслуживания, а ее сопровождение выполняется самим автором.

Работа с программой не должна требовать специальных навыков, кроме умения работать с клавиатурой компьютера и гитары.

2.1.3.4 Требования к техническим средствам

Для нормального функционирования данной информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

- процессор Intel или другой совместимый;
- объем свободной оперативной памяти ~500 Кб;
- объем необходимой памяти на жестком диске ~20Мб;
- стандартный VGA-монитор или совместимый;
- стандартная клавиатура;
- манипулятор «мышь»;
- наличие электро/акустической гитары;
- наличие микрофона (необязательное);

2.1.3.5 Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы выше Microsoft Windows 95 или совместимой. Язык интерфейса – не важен, наличие текстовых редакторов типа Блокнот, WordPad.

2.1.3.6 Требования к маркировке и упаковке

Программа должна поставляться на диске в виде исполняемого (exe) файла, документации и проекта. На диске должна быть наклейка с надписью "Guitar Keyboard". Диск должна быть упакован в пластиковую коробку.

2.1.3.7 Требования к транспортировке и хранению

Диск с программой должен храниться вдали от электромагнитных полей и не подвергаться механической деформации. Место и условия хранения должны соответствовать санитарным требованиям отрасли. Сроки хранения устанавливаются в соответствии с гарантийными сроками поставщика магнитных носителей.

Основные требования к транспортировке – создание условий, исключающих механические повреждения магнитного носителя.

2.1.4 Требования к программной документации

- «Техническое задание»;
- разрабатываемые программные модули должны быть самодокументированы, т.е. тексты программ должны содержать все необходимые комментарии;
- разрабатываемое программное обеспечение должно включать справочную систему.

2.1.5 Виды испытаний

Приемо-сдаточные испытания проводятся преподавателем.

2.2 Описание схем

2.2.1 Описание схемы основного модуля

Программа содержит 3 модуля:

Form1-модуль формы приложения, класс ComplexNumber.cs содержит методы работы при помощи комплексных чисел, для работы класса FFT.cs (преобразование Фурье). Эти классы использует модуль Form1 во время авторизации и использования.

Схемы отдельных методов приведены в приложении В.

2.2.2 Описание схем методов класса ComplexNumber.cs

Класс ComplexNumber.cs предназначен для организации работы класса FFT.cs, который содержит обычные математические функции, необходимые для вычисления комплексных чисел. Связность модуля информационная (последовательная) (CC=9), так как выходные части используются как входные части для другой.

2.2.3 Описание схем методов класса FFT.cs

Класс FFT.cs предназначен для исполнения преобразования Фурье. Связность модуля информационная (последовательная) (CC=9), так как выходные части используются как входные части для другой.

Это преобразование даёт частотный спектр аналогового сигнала. То есть любой сигнал можно представить как сумму синусоидальных кривых разной частоты и разной амплитуды. Таким образом, если сигнал сам по себе является синусоидой, то в разложении по частотам у него будет только один член, то есть в сигнале будет присутствовать одна частота. Если сигнал сложный, то и спектр сложный.

```
public static int Log2(int n)
    Получает ряд значительных байтов. “n”-число
    Возвращает количество минимальных битов для хранения числа.

private static int ReverseBits(int n, int bitsCount)
    Перводит биты в число. "bitsCount" - значимые биты в числе.
    Возвращает обратное двоичное число.

private static bool IsPowerOfTwo(int n)
    Проверяет, является ли число степенью числа 2. “n” – число.
    Возращает true если n=2^k и k положительное целое число.

private static float[] Calculate(float[] x)
    Вычисляет БПФ с помощью алгоритма Кули-Тьюки быстрого преобразования Фурье. “x” =
    исходные данные.
    Возвращает спектрограмму данных
```

Если количество элементов данных не кратно 2, то алгоритм автоматически переходит с 0s к самому минимальному значению кратному 2.

```
public static float[] FillSampleAgr(byte[] buffer, int BytesRecorded)
```
Вычисление уровня громкости, запись значения в буфер.

Возвращает заполненную переменную в размере до 8192(макс.)

```
public static float Level(float[] sampleAggregator)
```
Присваивает найденное ранее значение переменной level.

Возвращает level(уровень громкости входного сигнала).

```
public static int Generative(float[] sampleAggregator)
```
Работает при помощи Calculate.

Присваивает найденное значение переменной hz.

Возвращает hz(значение частоты входного сигнала).

2.2.3 Описание схем методов класса Form1.cs

Класс Form1.cs предназначен для организации работы приложения, данная форма является главной. Связность модуля информационная (последовательная) (CC=9), так как действия внутри модуля связаны с данными, а также важна последовательность действий.

Модуль содержит 7 методов:

```
public void WaveOnDataAvailable(object sender, WaveInEventArgs e)
```
Вычисляет значения уровня громкости и частоты входного сигнала.

```
private void timer1_Tick(object sender, EventArgs e)
```
Запуск первого таймера, получения уровня громкости из выбранного девайса.

```
private void timer2_Tick(object sender, EventArgs e)
```
Запуск второго таймера, настройки набора тюнов для гитары.

```
private void русскийToolStripMenuItem_Click(object sender, EventArgs e)
```
Установка русскоязычного интерфейса.

```
private void englishToolStripMenuItem_Click(object sender, EventArgs e)
```
Установка англоязычного интерфейса.

```
private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
```
Вывод справки.

```
private void button3_Click(object sender, EventArgs e)
```
Вывод таблицы сопоставлений уровня частоты с буквами для удобства пользования.

```
public void main_func()
```
Запуск считывания входного сигнала устройства и его преобразования.

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

Включение считывателя. Сопоставление входных данных частоты с нужными символами и вывод их в поля ввода.

```
private void button2_Click(object sender, EventArgs e)
```

Включение тюнера. Сопоставление входных данных частоты с нужными значениями тюна и вывод соответствующего значения для ориентира пользователя в настройке инструмента.

```
void waveIn_RecordingStopped(object sender, StoppedEventArgs e)
```

Остановка считывания входного сигнала устройства.

```
private void button4_Click(object sender, EventArgs e)
```

Выключение считывателя.

```
private void button5_Click(object sender, EventArgs e)
```

Выключение тюнера.

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

Выход из программы.

2.3 Текст программы

Текст программы в соответствии с ГОСТ 19.101-77 (СТ СЭВ 1626-79) и ГОСТ 19.401-79 (СТ СЭВ 3746-82) представляет собой запись программы на исходном языке программирования с необходимыми комментариями. Текст программы представляет собой документ, выполненный машинным способом, и приведен в приложении В.

2.4 Описание программы

2.4.1 Общие сведения

Приложение «Guitar Keyboard»: пользователь активирует приложение, с помощью которого он может, посредством электрогитары, подключенной к компьютеру, или акустической гитары, записывающейся через микрофон, считывать входной сигнал, определять его уровень, сопоставлять значение этого уровня с определенным символом и выводить эти символы в любом доступном текстовом поле.

2.4.2 Функциональное назначение

Основное назначение программного продукта заключается в организации работы приложения «Guitar Keyboard», с помощью которого пользователь может, посредством гитары, подключенной к компьютеру, печатать символы в любом доступном текстовом поле, а также настроить гитару пользователя.

Музыкальные инструменты играют какую-то определённую ноту и одновременно множество призвуков (обертонов), что значит, что спектр имеет сложную структуру, однако играемая нота (частота) в спектре имеет наибольшую амплитуду. Как правило, имеется, также, некоторое число гармоник в сигнале, то есть частот, кратных основной (удвоенная, утроенная, т.д.). Таким образом, для определения звучащей на гитаре ноты необходимо снять со звуковой карты звуковой поток, взять от него Фурье-образ, найти частоту с наибольшей амплитудой, провести соответствие частоты и ноты, в нашем случае еще и символа или определенной ноты, которую выдает нам открытая струна гитары.

2.4.3 Описание логической структуры

Необходимо выводить в выбранные текстовые поля и поля вводы символы, совпадающие с уровнем входного сигнала, а также выводить данные об уровне для сопоставления его эталоном, с целью настройки инструмента.

Пользователь поочередно играет ноты на гитаре, затем программа определяет частоту и уровень входного сигнала. После чего значение уровня сопоставляется с определенным значением, за которым следует символ. В итоге – символ выводится в выбранное поле.

Пользователь поочередно играет открытые струны на гитаре, затем программа определяет частоту и уровень входного сигнала. После чего значение уровня сопоставляется с эталоном настройки. Так будет продолжаться до тех пор, пока пользователь его не достигнет. В итоге пользователь настраивает инструмент для игры.

Программа использует функции следующих библиотек среды C#:

System; System.Collections.Generic; System.Drawing; System.Linq; System.Windows.Forms; NAudio.Wave; SoundAnalysis; NAudio.CoreAudioApi;

Исполняемый файл программы создан средствами среды C#, имеет имя keyboard_guitar.exe и размер 44500 байт.

2.4.4 Используемые технические и программные средства

Для нормального функционирования данной информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

- процессор Intel или другой совместимый;
- объем свободной оперативной памяти ~500 Кб;
- объем необходимой памяти на жестком диске ~3Мб;
- стандартный VGA-монитор или совместимый;
- стандартная клавиатура;
- манипулятор «мышь».
- наличие электро/акустической гитары;
- наличие микрофона;

2.4.5 Вызов и загрузка

Программа может быть загружена как с диска, так и с жесткого диска. В последнем случае требуется предварительно переписать программу с диска на жесткий диск.

Исполняемым файлом программы является файл keyboard_guitar.exe. Для его запуска необходимо дважды щелкнуть по исполняемому файлу левой кнопкой мышки.

2.5 Руководство оператора

2.5.1 Назначение программы

Основное назначение программного продукта заключается в организации работы приложения «Guitar Keyboard», с помощью которого пользователь может, посредством гитары, подключенной к компьютеру, печатать символы в любом доступном текстовом поле, а также настроить гитару пользователя.

2.5.2 Выполнение программы и сообщения оператору

Для запуска программы дважды щелкните левой кнопкой мыши по исполняемому файлу keyboard_guitar.exe.

Процесс, этапы работы приложения показаны в виде иллюстраций в приложении А.

При запуске работы приложения загружается главная форма, на которой появляется интерфейс самой программы. Он содержит в себе: основную часть – приложение для использования гитары в качестве устройства ввода, а также второстепенную часть – тюнер для настройки инструмента.

Основная часть. Для начала работы, пользователю необходимо выбрать устройство ввода, в зависимости от его подключенных внешних/внутренних устройств обработки входного сигнала, в выпадающей вкладке. Затем ему необходимо выбрать язык, в котором будет выводиться текст в поля ввода. После завершения предварительной настройки пользователь нажимает кнопку «Включить/Power On», которая активирует считывание входного сигнала.

Далее дело за самой программой – она выведет уровень громкости и частоты в необходимые для этого места в интерфейсе. За пользователем лишь останется выбрать нужное поле ввода и начать поочередно играть ноты.

Второстепенная часть. Данной частью является тюнер. Для его использования пользователю необходимо выбрать устройство ввода, в зависимости от его подключенных внешних/внутренних устройств обработки входного сигнала, в выпадающей вкладке, затем

выбрать необходимый вид тюна в следующей выпадающей вкладке и, в конце, выбрать нужную струну для настройки.

Активация тюнера происходит посредством кнопки – «Включить/Power On». Далее пользователя необходимо применить свои навыки по настройке инструмента, а также наблюдать за поступающими изменениями в интерфейсе – большая белая буква будет обозначать струну, которая настраивается в данный момент, как только пользователь достигает нужного натяжения, соответственно и нужной частоты звучания, буква загорится зеленым цветом, если струна будет недотянута – желтым, перетянута – красным.

Выключение обоих частей программы происходит посредством кнопки «Выключить/Power Off», которая имеет 2 экземпляра для каждого блока соответсвенно.

2.6 Программа и методика испытаний

2.6.1 Объект испытаний

Объектом испытаний является игровая программа keyboard_guitar.exe. Пользователь поочередно играет ноты на гитаре, затем программа определяет частоту и уровень входного сигнала. После чего значение уровня сопоставляется с определенным значением, за которым следует символ. В итоге – символ выводится в выбранное поле.

2.6.2 Цель испытаний

Целью испытаний является проверка соответствия программы требованиям Технического Задания.

2.6.3 Требования к программе

В процессе испытаний подлежат проверке следующие требования к программе:

2.6.3.1 Требования к функциональным характеристикам

- поочередная игра нот на гитаре;
- считывание нот (уровня входного сигнала) программой;
- сопоставление определенного уровня с определённым символом;
- вывод символа в текстовое поле;
- сопоставление определенного уровня с эталоном;
- вывод уровня на экран для помощи в настройке гитары;

2.6.3.2 Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы выше Microsoft Windows 7 или совместимой. Язык интерфейса – не важен, наличие текстовых редакторов типа Блокнот, WordPad.

2.6.3.3 Требования к маркировке и упаковке

Программа должна поставляться на диске в виде исполняемого (exe) файла, документации и проекта. На диске должна быть наклейка с надписью "Приложение «Guitar Keyboard»". Диск должна быть упакован в пластиковую коробку.

2.6.4 Требования к программной документации

На испытания должны быть представлены следующие программные документы:

- техническое задание
- текст программы
- описание программы
- руководство оператора
- описание языка

2.6.5 Средства и порядок испытаний

Для проведения испытаний необходимы следующие технические средства:

- процессор Intel или другой совместимый;
- объем свободной оперативной памяти ~500 Кб;
- объем необходимой памяти на жестком диске ~3Мб;
- стандартный VGA-монитор или совместимый;
- стандартная клавиатура;
- манипулятор «мышь».
- - наличие электро/акустической гитары;
- - наличие микрофона.

Для проведения испытаний необходимы следующие программные средства:

Операционная система Windows 7

Текстовый редактор Блокнот или WordPad.

Испытания проводятся в следующем порядке:

1) проверяется наличие и комплектность программной документации (п.2.6.4)

2) проверяется соответствие требованиям к маркировке и упаковке (п.2.6.3.3)

3) проверяется соответствие требованиям к функциональным характеристикам (п.2.6.3.1)

4) проверяется соответствие требованиям к информационной и программной совместимости (п.2.6.3.2)

2.6.6 Методы испытаний

2.6.6.1 Для проверки способности программы пошаговой активации, необходимо:

- запустить программу;
- выбрать устройство ввода в нужной графе на вкладке «Guitar Keyboard»;
- выбрать язык ввода;

- убедиться, что предыдущие действия выполнены, иначе кнопка активации останется недоступной;
- нажать кнопку «Включить/Power On»;

2.6.6.2 Для проверки способности программы считывать входной сигнал, необходимо:

- выкрутить регуляторы инструмента и устройств считывания на значение, отличное от нуля;
- сыграть любую ноту;
- убедится в способности считывания при помощи изменений в нижней части интерфейса;

2.6.6.3 Для проверки способности программы сопоставлять частоту водного сигнала с буквами и вывода их в поля ввода, необходимо:

- выбрать нужное поле ввода;
- играть ноты на гитаре попеременно;
- убедится в способности считывания при помощи изменений в нижней части интерфейса, обозначенной как «Частота»

2.6.6.4 Для проверки способности программы пошаговой активации в тюнере, необходимо:

- запустить программу;
- выбрать устройство ввода в нужной графе на вкладке – «Тюнер»;
- выбрать нужный тюн;
- выбрать струну, которую необходимо настроить;
- убедиться, что предыдущие действия выполнены, иначе кнопка активации останется недоступной;
- нажать кнопку «Включить/Power On»;

2.6.6.5 Для проверки способности программы помогать в настройке инструмента, необходимо:

- ориентируясь по элементам интерфейса – настроить инструмент (при зеленом цвете большой буквы – струна настроена, при желтом – недотянута, при красном – перетянута).

2.7 Протокол испытаний

Результаты испытаний программы представлены в таблице 1, рисунки приведены в приложении А.

Результаты испытаний программы

Таблица 1

Проверяемые требования	**Сообщения программы и вводимые значения**	**Результаты**
для проверки способности программы пошаговой активации, необходимо	- запустить программу; - выбрать устройство ввода в нужной графе на вкладке «Guitar Keyboard»; - выбрать язык ввода; - убедиться, что предыдущие действия выполнены, иначе кнопка активации останется недоступной; - нажать кнопку с надписью «Включить/Power On»;	смотри Рис. 1

Проверяемые требования	Сообщения программы и вводимые значения	Результаты
способность программы считывать входной сигнал, необходимо	- выкрутить регуляторы инструмента и устройств считывания на значение, отличное от нуля; - сыграть любую ноту; - убедится в способности считывания при помощи изменений в нижней части интерфейса;	смотри Рис. 2
способность программы сопоставлять частоту водного сигнала с буквами и вывода их в поля ввода	- выбрать нужное поле ввода; - играть ноты на гитаре попеременно; - убедится в способности считывания при помощи изменений в нижней части интерфейса, обозначенной как «Частота»	Смотри Рис. 3
способности программы пошаговой активации в тюнере, необходимо	- запустить программу; - выбрать устройство ввода в нужной графе на вкладке – «Тюнер»; - выбрать нужный тюн; - выбрать струну, которую необходимо настроить; - убедиться, что предыдущие действия выполнены, иначе кнопка активации останется недоступной; - нажать кнопку «Включить/Power On»;	смотри Рис. 4

Проверяемые требования	Сообщения программы и вводимые значения	Результаты
способность программы помогать в настройке инструмента, необходимо:	- ориентируясь по элементам интерфейса – настроить инструмент (при зеленом цвете большой буквы – струна настроена, при желтом – недотянута, при красном – перетянута).	смотри Рис. 5

ЗАКЛЮЧЕНИЕ

Разработанная в ходе выполнения курсового проекта программа удовлетворяет всем требованиям технического задания, что подтверждается протоколом испытаний.

Разработанная программа может быть использована в развлекательных целях, а также в практических или развивающих.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Стандарты Единой Системы Программной Документации:

   ГОСТ 19.105-78 Общие требования к программным документам

   ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

   ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению

   ГОСТ 19.301-78 Программа и методика испытаний. Требования к содержанию и оформлению

   ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению

   ГОСТ 19.402-78 Описание программы. Требования к содержанию и оформлению

   ГОСТ 19.505-79 Руководство оператора. Требования к содержанию и оформлению

   ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

   2 – Интернет ресурс – GitHub https://github.com/

   3 - Подбельский В.В. Язык C#. Базовый курс. 2-е изд.

   4 – Интернет ресурс – CyberForum http://www.cyberforum.ru/

   5 - Бен Ватсон C# 4.0 на примерах

   6 – Интернет ресурс – HabrHabr https://habrahabr.ru

   7 - Гриффитс И. Программирование на C# 5.0

   8 – Интернет ресурс – MSDN.microsoft https://msdn.microsoft.com/

   9 - Албахари Д., Албахари Б. C# 6.0. Справочник. Полное описание языка. 6-е изд.

   10 – Интеренет ресурс – YouTube www.youtube.com

ПРИЛОЖЕНИЕ А

Рисунок 1 Запуск программы

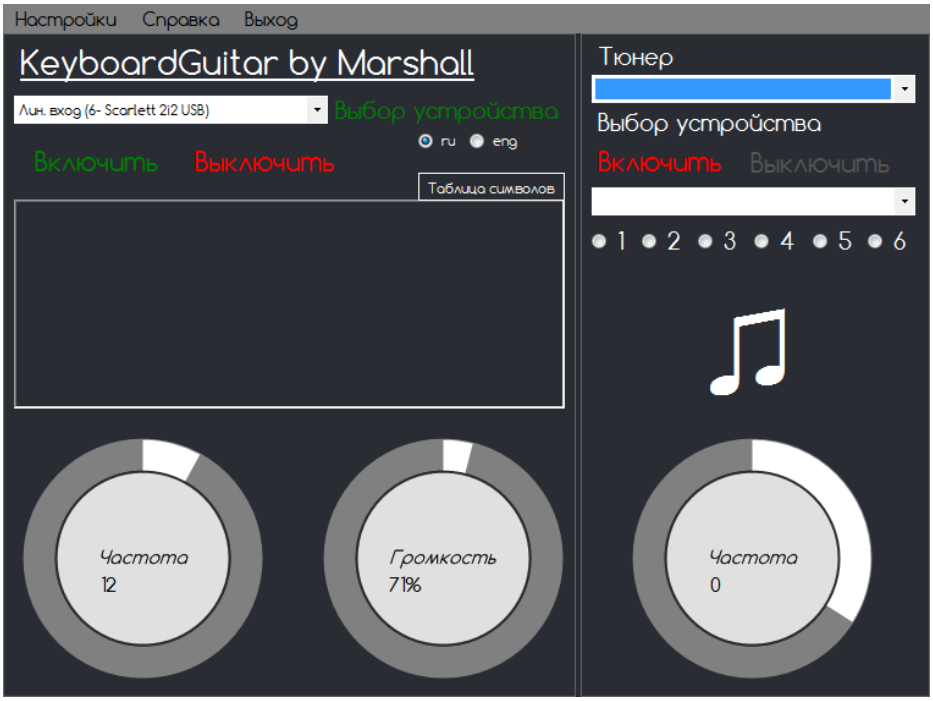

Рисунок 2 Выбор устройства и языка ввода

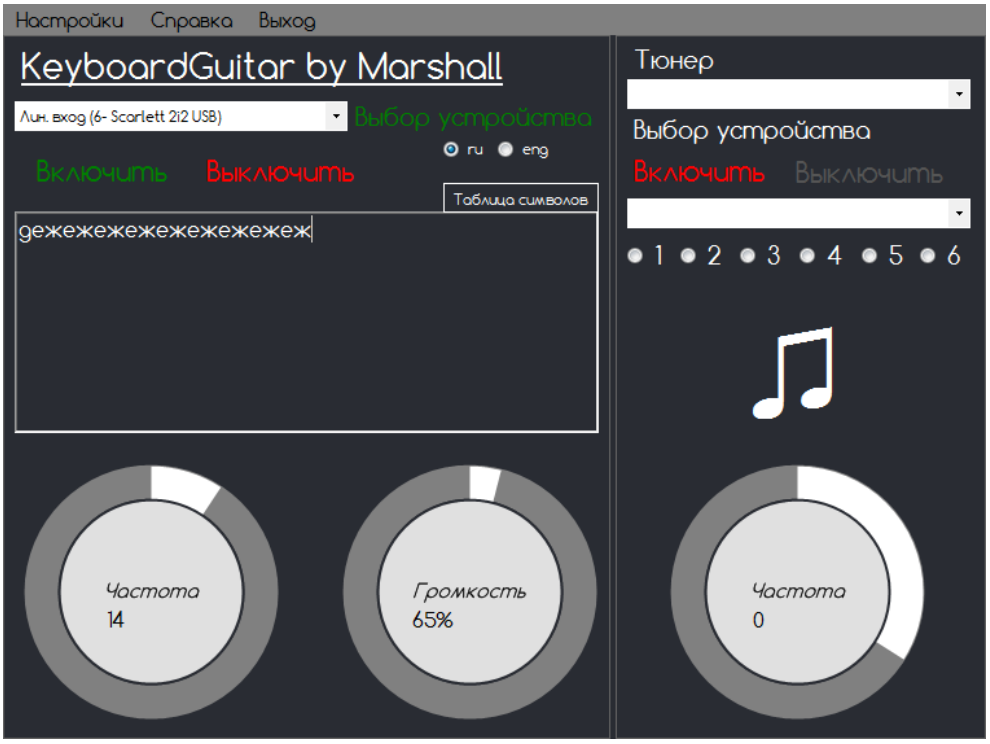

Рисунок 3 Игра на гитаре – вывод текста в спец. поле

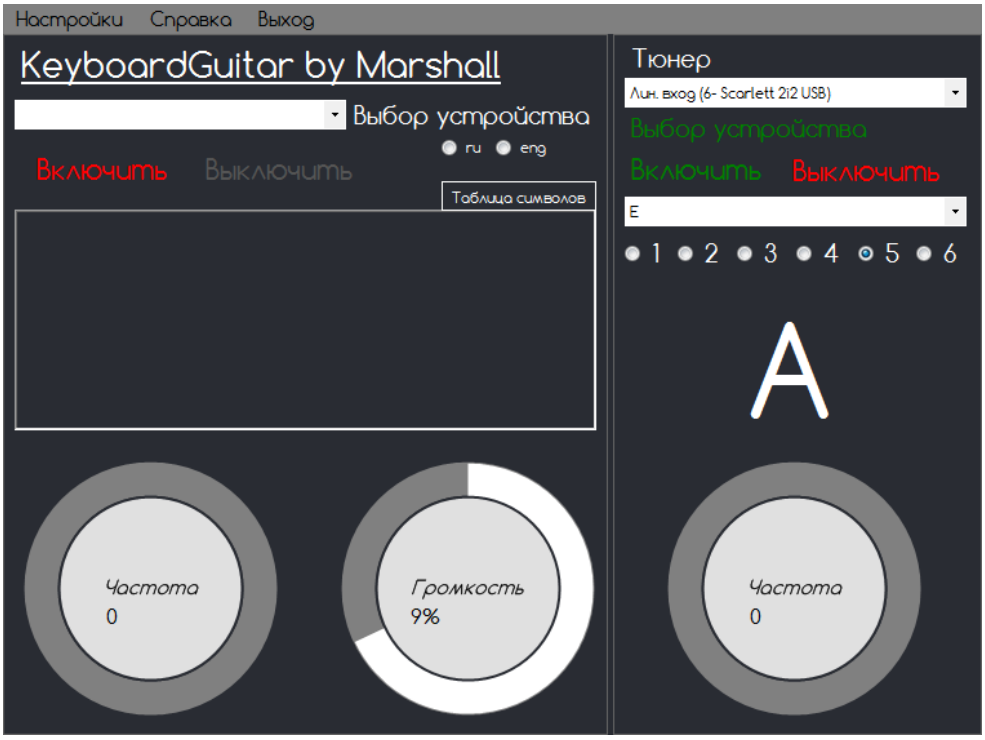

Рисунок 4 Выключение программы

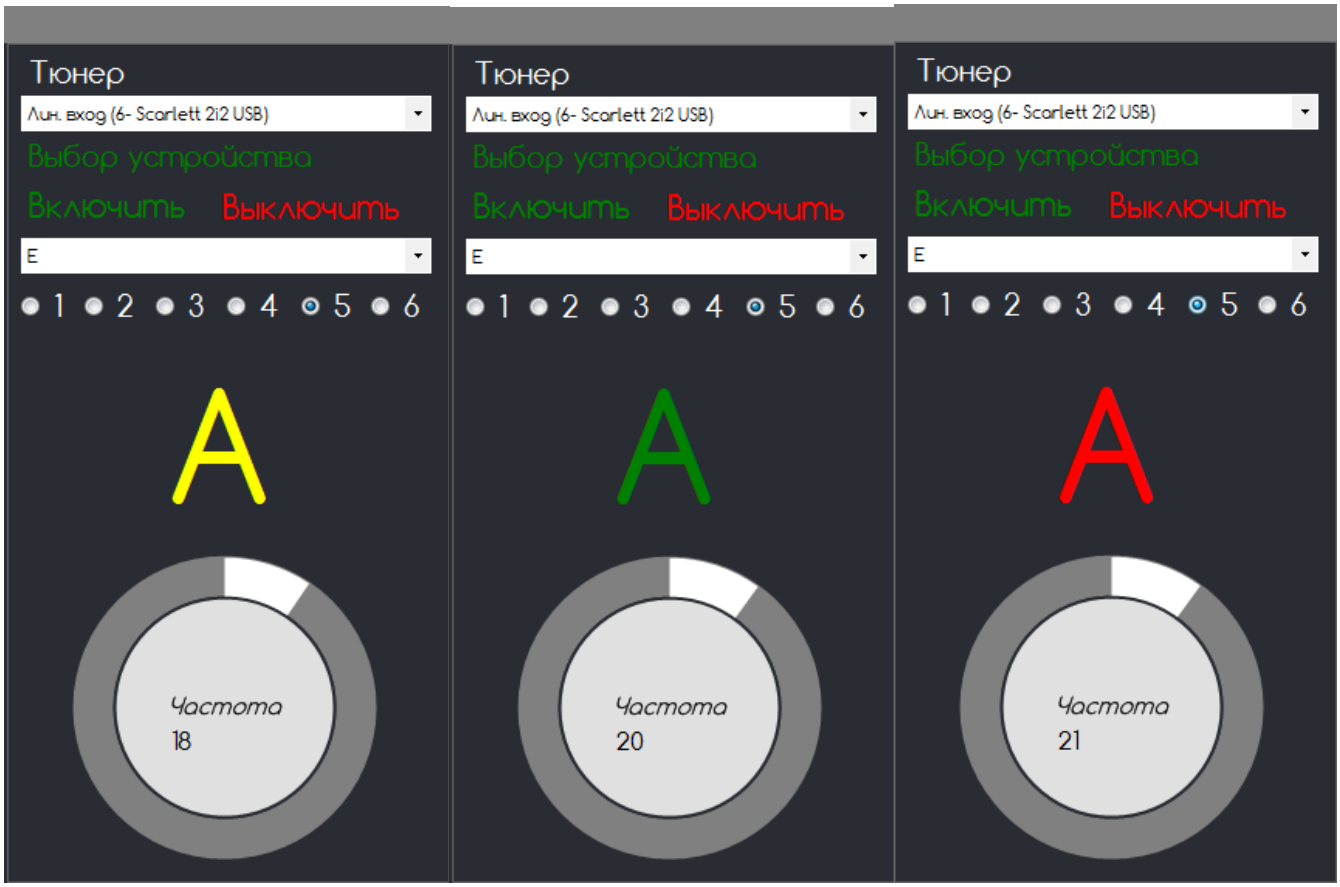

Рисунок 5 Процесс настройки (тюнинга)

ПРИЛОЖЕНИЕ Б

Для FFT.cs

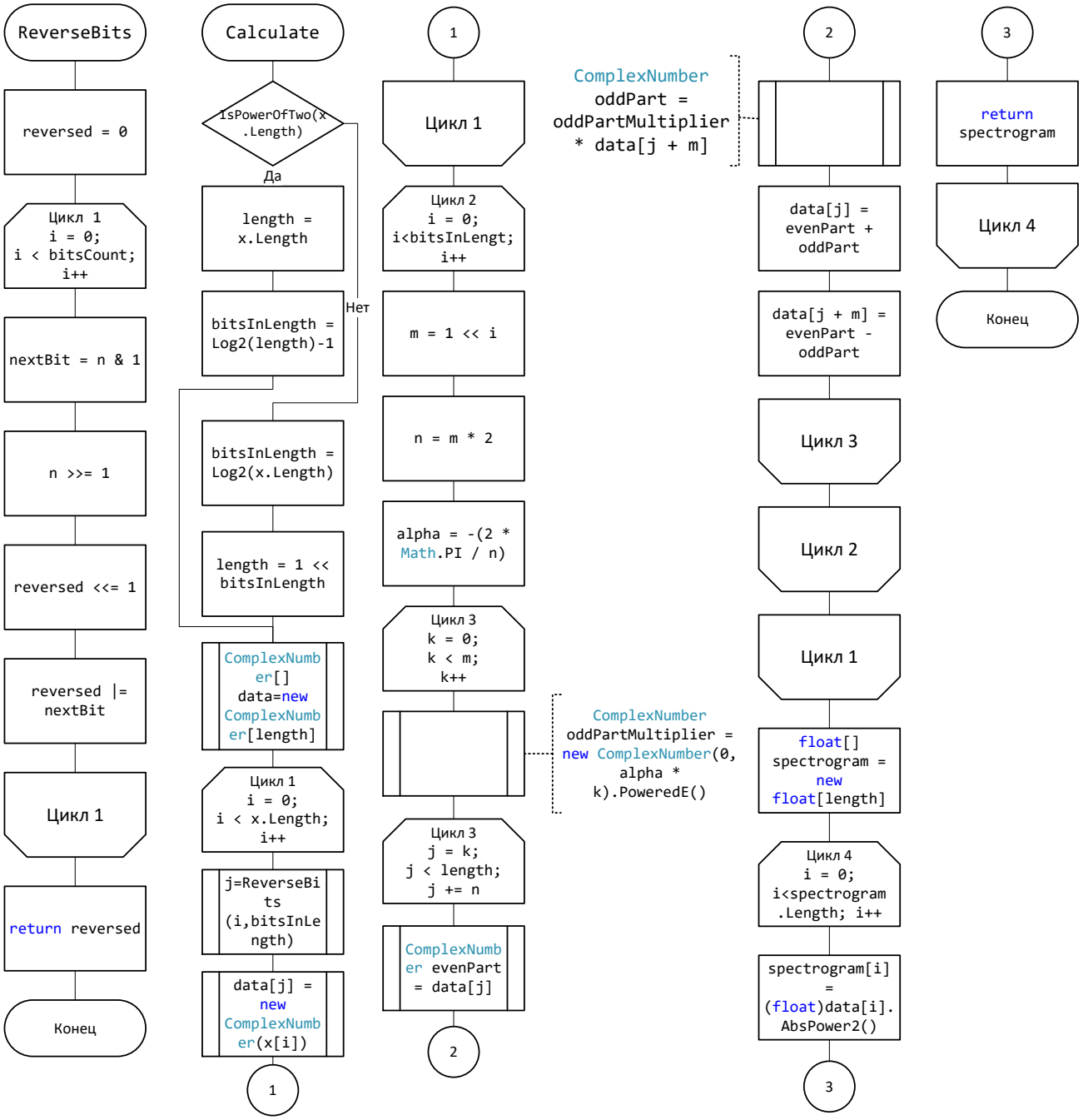

Рисунок 1 Схема метода `private static int ReverseBits(int n, int bitsCount)`

Рисунок 2 Схема метода `private static float[] Calculate(float[] x)`

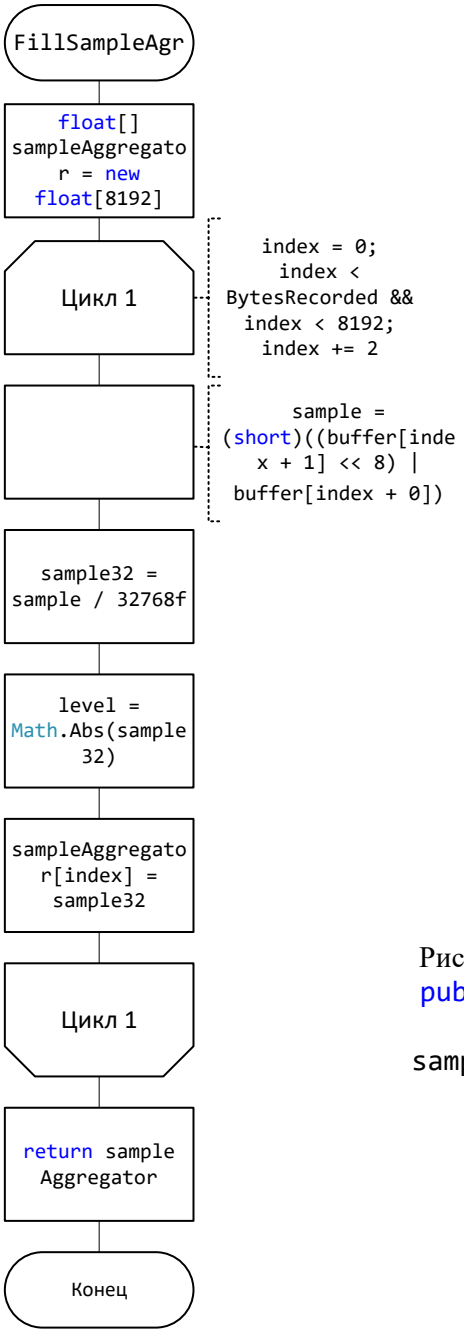

Рисунок 3 Схема метода `public static float[] FillSampleAgr(byte[] buffer, int`

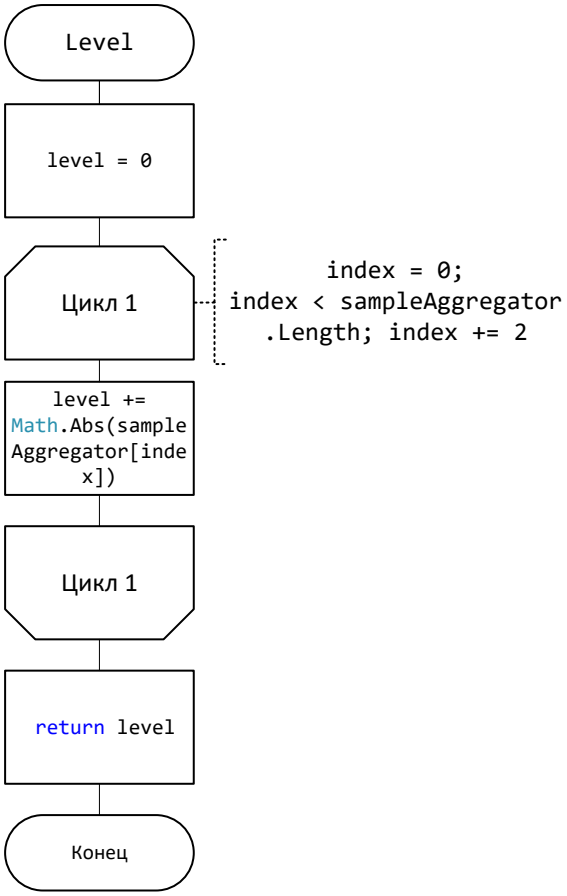

Рисунок 4 Схема метода `public static float Level(float[] sampleAggregator)BytesRecorded)`

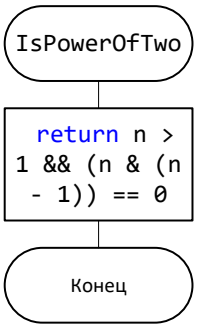

Рисунок 5 Схема метода `private static bool IsPowerOfTwo(int n)`

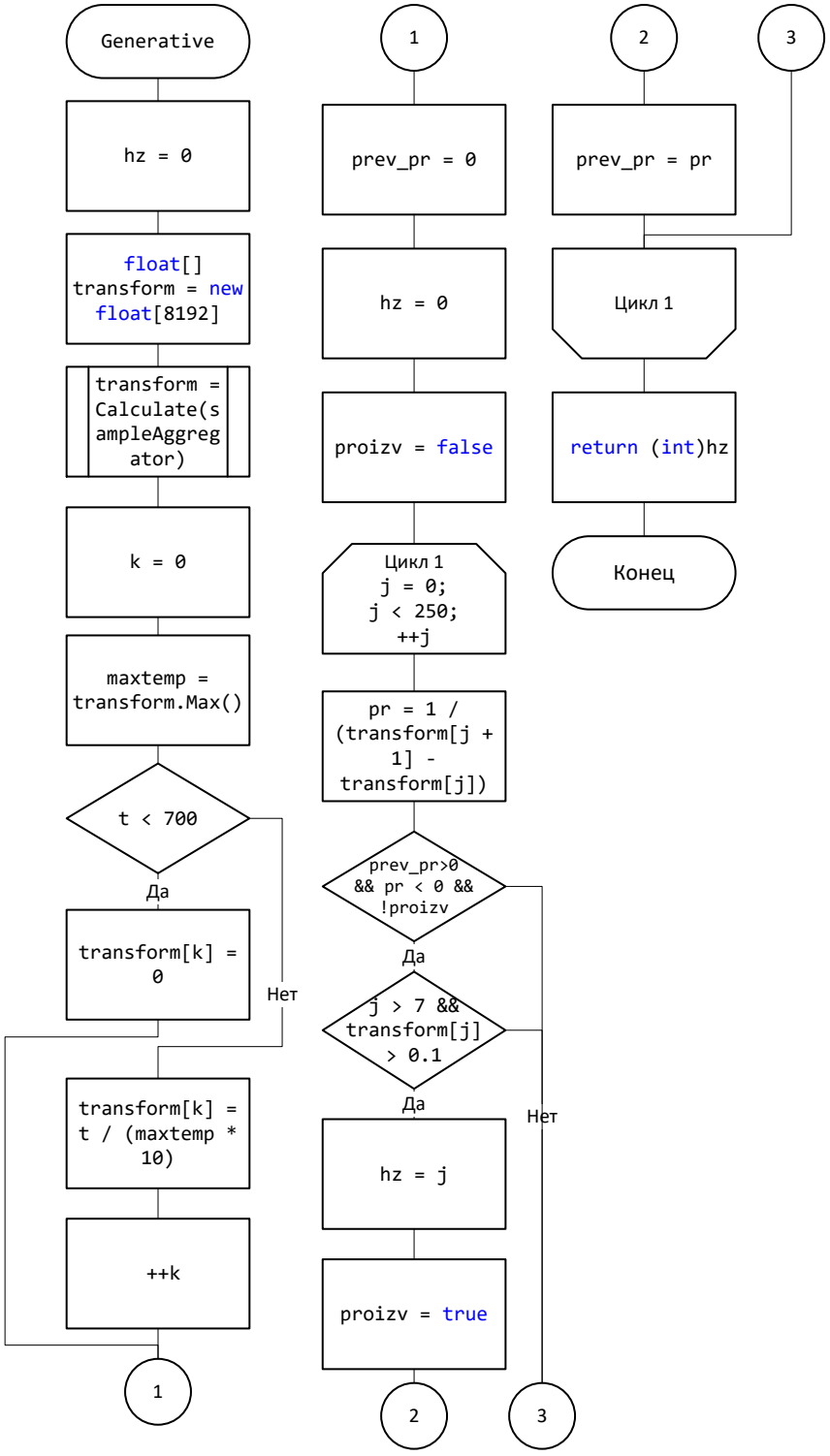

Рисунок 6 Схема метода `public static int Generative(float[] sampleAggregator)`

Для Form1.cs

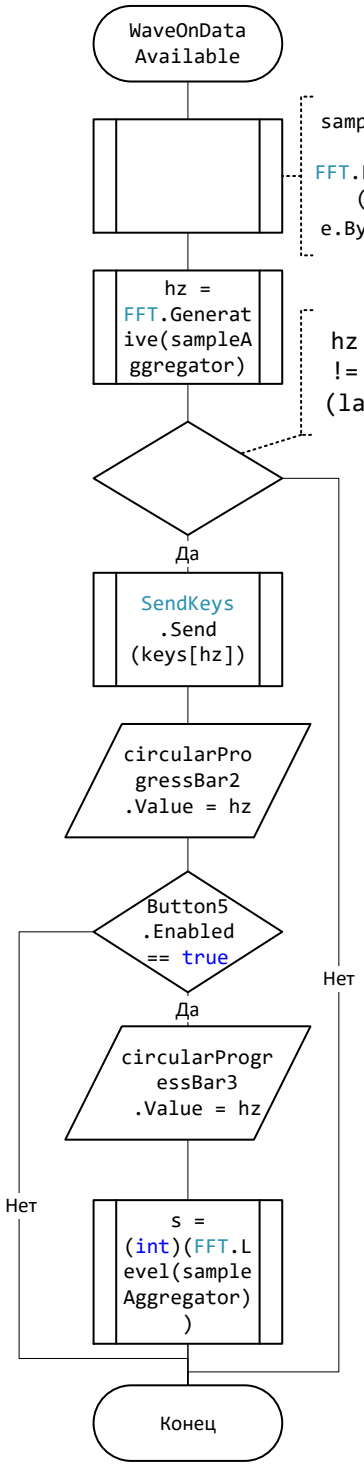

Рисунок 7 Схема метода `public void WaveOnDataAvailable( object sender, WaveInEventArgs e)`

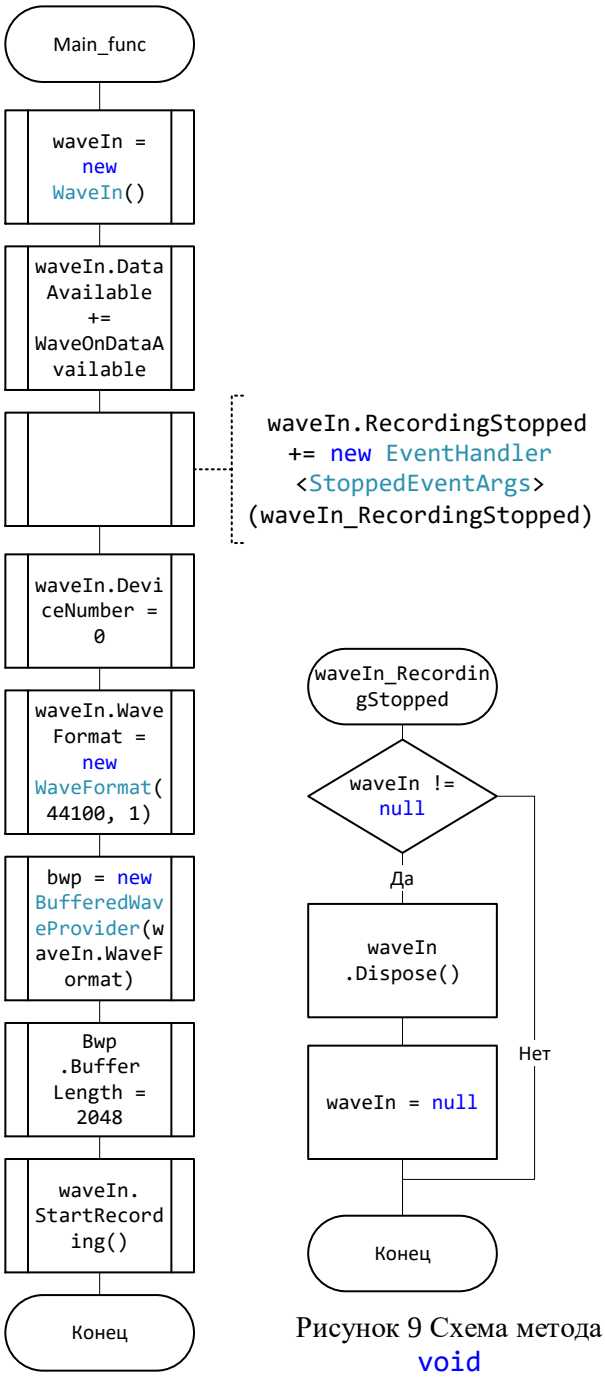

Рисунок 8 Схема метода `public void main_func()`

Рисунок 9 Схема метода `void waveIn_RecordingStop ped(object sender, StoppedEventArgs e)`

ПРИЛОЖЕНИЕ В

Текст программы

ComplexNumber.cs

```
using System;

namespace SoundAnalysis
{
    /// <summary>
    /// Complex number.
    /// </summary>
    struct ComplexNumber
    {
        public double Re;
        public double Im;

        public ComplexNumber(double re)
        {
            this.Re = re;
            this.Im = 0;
        }

        public ComplexNumber(double re, double im)
        {
            this.Re = re;
            this.Im = im;
        }

        public static ComplexNumber operator *(ComplexNumber n1, ComplexNumber n2)
        {
            return new ComplexNumber(n1.Re * n2.Re - n1.Im * n2.Im,
                n1.Im * n2.Re + n1.Re * n2.Im);
        }

        public static ComplexNumber operator +(ComplexNumber n1, ComplexNumber n2)
        {
            return new ComplexNumber(n1.Re + n2.Re, n1.Im + n2.Im);
        }

        public static ComplexNumber operator -(ComplexNumber n1, ComplexNumber n2)
        {
            return new ComplexNumber(n1.Re - n2.Re, n1.Im - n2.Im);
        }

        public static ComplexNumber operator -(ComplexNumber n)
        {
            return new ComplexNumber(-n.Re, -n.Im);
        }

        public static implicit operator ComplexNumber(double n)
        {
            return new ComplexNumber(n, 0);
        }
```

```
        public ComplexNumber PoweredE()
        {
            double e = Math.Exp(Re);
            return new ComplexNumber(e * Math.Cos(Im), e * Math.Sin(Im));
        }

        public double Power2()
        {
            return Re * Re - Im * Im;
        }

        public double AbsPower2()
        {
            return Re * Re + Im * Im;
        }

        public override string ToString()
        {
            return String.Format("{0}+i*{1}", Re, Im);
        }
    }
}
```

FFT.cs

```
using System;

namespace SoundAnalysis
{
    /// <summary>
    /// Complex number.
    /// </summary>
    struct ComplexNumber
    {
        public double Re;
        public double Im;

        public ComplexNumber(double re)
        {
            this.Re = re;
            this.Im = 0;
        }

        public ComplexNumber(double re, double im)
        {
            this.Re = re;
            this.Im = im;
        }

        public static ComplexNumber operator *(ComplexNumber n1, ComplexNumber n2)
        {
            return new ComplexNumber(n1.Re * n2.Re - n1.Im * n2.Im,
                n1.Im * n2.Re + n1.Re * n2.Im);
        }

        public static ComplexNumber operator +(ComplexNumber n1, ComplexNumber n2)
```

```
        {
            return new ComplexNumber(n1.Re + n2.Re, n1.Im + n2.Im);
        }

        public static ComplexNumber operator -(ComplexNumber n1, ComplexNumber n2)
        {
            return new ComplexNumber(n1.Re - n2.Re, n1.Im - n2.Im);
        }

        public static ComplexNumber operator -(ComplexNumber n)
        {
            return new ComplexNumber(-n.Re, -n.Im);
        }

        public static implicit operator ComplexNumber(double n)
        {
            return new ComplexNumber(n, 0);
        }

        public ComplexNumber PoweredE()
        {
            double e = Math.Exp(Re);
            return new ComplexNumber(e * Math.Cos(Im), e * Math.Sin(Im));
        }

        public double Power2()
        {
            return Re * Re - Im * Im;
        }

        public double AbsPower2()
        {
            return Re * Re + Im * Im;
        }

        public override string ToString()
        {
            return String.Format("{0}+i*{1}", Re, Im);
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using NAudio.Wave;
using SoundAnalysis;
using NAudio.CoreAudioApi;

namespace keyboard_guitar
{

    public partial class Form1 : Form
    {
        public List<int> note = new List<int>();
        public bool f = false;
        public string[] keys = new string[250];
        public int temp_note = 0;
        public BufferedWaveProvider bwp;
        public WaveIn waveIn = null;
        int hz;


        public Form1()
        {
            InitializeComponent();
            MMDeviceEnumerator enumirator = new MMDeviceEnumerator();
            var devices = enumirator.EnumerateAudioEndPoints(DataFlow.All,
DeviceState.Active);
            comboBox1.Items.AddRange(devices.ToArray());
            comboBox3.Items.AddRange(devices.ToArray());
            label3.Text = "\u266B";
            button1.ForeColor = Color.Red;
            button2.ForeColor = Color.Red;
            button4.Enabled = false;
            button5.Enabled = false;
        }

        public delegate void lblDelegate(string message);

        public void main_func()
        {

            waveIn = new WaveIn();
            waveIn.DataAvailable += WaveOnDataAvailable;
            waveIn.RecordingStopped += new
EventHandler<StoppedEventArgs>(waveIn_RecordingStopped); //объявление события об окончании
записи
            waveIn.DeviceNumber = 0;
            waveIn.WaveFormat = new WaveFormat(44100, 1); //определение формата входной
волны - по стандарту 44100 hrz
            bwp = new BufferedWaveProvider(waveIn.WaveFormat);
            bwp.BufferLength = 2048; //длина буфера для записи
            waveIn.StartRecording(); //запуск считывания

        }
```

```
        void waveIn_RecordingStopped(object sender, StoppedEventArgs e) //событие об
окончании записи
        {
            if (waveIn != null)
            {
                waveIn.Dispose();
                waveIn = null;
            }
        }

        public void WaveOnDataAvailable(object sender, WaveInEventArgs e)
        {

            float[] sampleAggregator = FFT.FillSampleAgr(e.Buffer, e.BytesRecorded);
            hz = FFT.Generative(sampleAggregator);
            //здесь мы нажимаем кнопку
            if (hz > 0 && hz != int.Parse(label2.Text)) SendKeys.Send(keys[hz]);
            //выводим число, на которое ориентируемся
            label2.Text = hz + "";
            circularProgressBar2.Value = hz; //заполнение прогрессбара числом частоты
            if (button5.Enabled == true)
            {
                label6.Text = hz + "";
                circularProgressBar3.Value = hz;
            }
            int s = (int)(FFT.Level(sampleAggregator)); //определение уровня громкости
            label1.Text = s + "%";

        }
        private void button1_Click(object sender, EventArgs e)
        {
            {

                if (comboBox1.SelectedIndex == -1)
                {
                    label4.ForeColor = Color.Red;
                    return;
                }
                if (radioButton7.Checked == false && radioButton8.Checked == false)
                {
                    radioButton7.ForeColor = Color.Red;
                    radioButton8.ForeColor = Color.Red;
                    return;
                }
                radioButton7.ForeColor = Color.White;
                radioButton8.ForeColor = Color.White;
                button4.Enabled = true;
                button4.ForeColor = Color.Red;
                button1.ForeColor = Color.Green;
                main_func();

                if (radioButton8.Checked == true)
                {
                    keys[15] = "{BACKSPACE}";
                    keys[8] = "a";
                    keys[9] = "b";
                    keys[10] = "c";
                    keys[11] = "d";
```

```
        keys[12] = "e";
        keys[13] = "f";
        keys[14] = "g";
        keys[15] = "h";
        keys[17] = "i";
        keys[19] = "j";
        keys[21] = "k";
        keys[23] = "l";
        keys[25] = "m";
        keys[27] = "n";
        keys[29] = "o";
        keys[31] = "p";
        keys[33] = "q";
        keys[35] = "r";
        keys[37] = "s";
        keys[39] = "t";
        keys[41] = "u";
        keys[43] = "v";
        keys[45] = "w";
        keys[47] = "x";
        keys[49] = "y";
        keys[51] = "z";
        keys[90] = "{ENTER}";
    }
    else
        if (radioButton7.Checked == true)
    {
        keys[15] = "{BACKSPACE}";
        keys[8] = "а";
        keys[9] = "б";
        keys[10] = "в";
        keys[11] = "г";
        keys[12] = "д";
        keys[13] = "е";
        keys[14] = "ж";
        keys[15] = "ж";
        keys[17] = "и";
        keys[19] = "к";
        keys[21] = "л";
        keys[23] = "м";
        keys[25] = "н";
        keys[27] = "о";
        keys[29] = "п";
        keys[31] = "р";
        keys[33] = "с";
        keys[35] = "т";
        keys[37] = "у";
        keys[39] = "ф";
        keys[41] = "х";
        keys[43] = "ц";
        keys[45] = "ч";
        keys[47] = "ш";
        keys[49] = "щ";
        keys[51] = "ъ";
        keys[53] = "ы";
        keys[55] = "ь";
        keys[57] = "э";
        keys[59] = "ю";
```

```
                keys[61] = "я";
                keys[90] = "{ENTER}";

            }
            //задаём клавиши

        }
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        if (comboBox1.SelectedItem != null)
        {
            var device = (MMDevice)comboBox1.SelectedItem;
            circularProgressBar1.Value =
(int)(Math.Round(device.AudioMeterInformation.MasterPeakValue * 100));
        }
    }

    private void button2_Click(object sender, EventArgs e)
    {
        if (comboBox3.SelectedIndex == -1)
        {
            label8.ForeColor = Color.Red;
            return;
        }
        if (comboBox2.SelectedIndex == -1)
        {
            comboBox2.BackColor = Color.Red;
            return;
        }
        comboBox2.ForeColor = Color.Black;
        button5.Enabled = true;
        button5.ForeColor = Color.Red;
        button2.ForeColor = Color.Green;
        timer2.Enabled = !timer2.Enabled;
        main_func();
    }

    private void timer2_Tick(object sender, EventArgs e)
    {

        string s = comboBox2.SelectedItem.ToString();
        switch (s)
        {
            case "E":
                if (radioButton6.Checked == true)
                {
                    label3.Text = "E";
                    switch (hz)
                    {

                        case 0: label3.ForeColor = Color.White;
                            break;
                        case 22: label3.ForeColor = Color.Yellow;
                            break;
                        case 23: label3.ForeColor = Color.Green;
                            break;
```

```
            case 24: label3.ForeColor = Color.Red;
                break;
        }

    }
    if (radioButton5.Checked == true)
    {
        label3.Text = "A";
        switch (hz)
        {

            case 0: label3.ForeColor = Color.White;
                break;
            case 19: label3.ForeColor = Color.Yellow;
                break;
            case 20: label3.ForeColor = Color.Green;
                break;
            case 21: label3.ForeColor = Color.Red;
                break;
        }

    }
    if (radioButton4.Checked == true)
    {
        label3.Text = "D";
        switch (hz)
        {

            case 0: label3.ForeColor = Color.White;
                break;
            case 26: label3.ForeColor = Color.Yellow;
                break;
            case 27: label3.ForeColor = Color.Green;
                break;
            case 28: label3.ForeColor = Color.Red;
                break;
        }

    }
    if (radioButton3.Checked == true)
    {
        label3.Text = "G";
        switch (hz)
        {

            case 0: label3.ForeColor = Color.White;
                break;
            case 53: label3.ForeColor = Color.Yellow;
                break;
            case 54: label3.ForeColor = Color.Green;
                break;
            case 55: label3.ForeColor = Color.Red;
                break;
        }

    }
    if (radioButton2.Checked == true)
    {
```

```
                label3.Text = "B";
                switch (hz)
                {

                    case 0: label3.ForeColor = Color.White;
                        break;
                    case 68: label3.ForeColor = Color.Yellow;
                        break;
                    case 69: label3.ForeColor = Color.Green;
                        break;
                    case 70: label3.ForeColor = Color.Red;
                        break;
                }

            }
            if (radioButton1.Checked == true)
            {
                label3.Text = "e";
                switch (hz)
                {

                    case 0: label3.ForeColor = Color.White;
                        break;
                    case 91: label3.ForeColor = Color.Yellow;
                        break;
                    case 92: label3.ForeColor = Color.Green;
                        break;
                    case 93: label3.ForeColor = Color.Red;
                        break;
                }

            }
            break;

    }

}

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

private void русскийToolStripMenuItem_Click(object sender, EventArgs e)
{
    button1.Text = "Включить";
    button2.Text = "Включить";
    button3.Text = "Таблица символов";
    button4.Text = "Выключить";
    button5.Text = "Выключить";
    label4.Text = "Выбор устройства";
    label8.Text = "Выбор устройства";
    circularProgressBar2.Text = "Частота";
    circularProgressBar3.Text = "Частота";
    circularProgressBar1.Text = "Громкость";
    label7.Text = "Тюнер";
    настройкиToolStripMenuItem.Text = "Настройки";
    языкlanguageToolStripMenuItem.Text = "Язык (language)";
```

```
            справкаToolStripMenuItem.Text = "Справка";
            выходToolStripMenuItem.Text = "Выход";
            comboBox2.Text = "Дроп";
        }

        private void englishToolStripMenuItem_Click(object sender, EventArgs e)
        {
            button1.Text = "Power on";
            button2.Text = "Power on";
            button4.Text = "Power off";
            button5.Text = "Power off";
            button3.Text = "Table of symbols";
            label4.Text = "Choose the device";
            label8.Text = "Choose the device";
            circularProgressBar2.Text = "Frequency";
            circularProgressBar3.Text = "Frequency";
            circularProgressBar1.Text = "Volume";
            label7.Text = "Tuner";
            настройкиToolStripMenuItem.Text = "Settings";
            языкlanguageToolStripMenuItem.Text = "Language (язык)";
            справкаToolStripMenuItem.Text = "Info";
            выходToolStripMenuItem.Text = "Exit";
            comboBox2.Text = "Drop";
        }
        private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("lol");
        }
        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (comboBox1.SelectedIndex >= 0)
            {
                label4.ForeColor = Color.Green;
            }
        }
        private void comboBox3_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (comboBox3.SelectedIndex >= 0)
            {
                label8.ForeColor = Color.Green;
            }
        }
        private void button3_Click(object sender, EventArgs e)
        {
            MessageBox.Show("sas");
        }
        private void button4_Click(object sender, EventArgs e)
        {
            waveIn.StopRecording();
            button4.Enabled = false;
        }
        private void button5_Click(object sender, EventArgs e) {
        waveIn.StopRecording();
        button5.Enabled = false;
        }
    }
}
```