Содержание

Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата	Вариант №4		
Разраб.		Белым А.А.			Пояснительная записка к курсовой работе по курсу «Программирование на ЯВУ» по теме «Быстрое преобразование Фурье»	Лит.	Лист 2 / Листов 74
Пров.		Сулимова В.Е.					
Н. контр.						ТулГУ гр. 220601	
Утв.							

ВВЕДЕНИЕ

В данной работе представлен процесс разработки программы, базирующийся на концепциях объектно-ориентированного программирование (ООП), результатом которого является полнофункциональное приложение, написанное на языке Object Pascal, на его варианте, представленном в среде Delphi 7. Поэтому во вступительном слове расскажем немного о методологии ООП и языке Object Pascal.

Объектно-ориентированная парадигма программирования не нова. Её истоки восходят к Симуле-67, хотя первая полная реализация была в Smalltalk-80. ООП (Объектно-ориентированное программирование) стало популярным во второй половине 80-х в таких языках, как C++, Objective C (другое расширение C), Object Pascal и Turbo Pascal, CLOS (OO-расширение Lisp'a), Eiffel, Ada (в её последних воплощениях) и недавно — в Java. Эта работа сосредоточена на C++,Object Pascal и Java, иногда упоминая и другие языки.

Ключевые черты ООП хорошо известны:

Первая — инкапсуляция — это определение классов — пользовательских типов данных, объединяющих своё содержимое в единый тип и реализующих некоторые операции или методы над ним. Классы обычно являются основой модульности, инкапсуляции и абстракции данных в языках ООП.

Вторая ключевая черта, — наследование — есть способ определения нового типа, наследуя элементы (свойства и методы) существующего и модифицируя или расширяя их. Это способствует выражению специализации и генерализации.

Третья черта, известная как полиморфизм, позволяет единообразно ссылаться на объекты различных классов (обычно внутри некоторой иерархии). Это делает классы ещё более удобными и делает программы, основанные на них, легче для расширения и поддержки.

Инкапсуляция, наследование и полиморфизм — фундаментальные свойства, требуемые от языка, претендующего называться объектно-ориентированным (языки, не

имеющие наследования и полиморфизма, но имеющие только классы, обычно называются основанными на классах). Различные ОО языки используют совершенно разные подходы. Мы можем различать ОО языки, сравнивая механизм контроля типов, способность поддерживать различные программные модели и то, какие объектные модели они поддерживают.

Алан Кей в свое время вывел пять основных черт языка Smalltalk — первого удачного ОО языка: Все является объектом. Объект как хранит информацию, так и способен ее преобразовывать. В принципе любой элемент решаемой задачи (дом, собака, услуга, химическая реакция, город, космический корабль и т. д.) может представлять собой объект. Объект можно представить себе как швейцарский нож: он является набором различных ножей и «открывашек» (хранение), но в то же самое время им мы можем резать или открывать что-либо (преобразование).

Программа — совокупность объектов, указывающих друг другу что делать. Для обращения к одному объекту другой объект «посылает ему сообщение». Как вариант возможно и «ответное сообщение». Программу можно представить себе как совокупность к примеру 3 объектов: писателя, ручки и листа бумаги. Писатель «посылает сообщение» ручке, которая в свою очередь «посылает сообщение» листу бумаги — в результате мы видим текст (посыл сообщения от листа к писателю). Каждый объект имеет свою собственную «память» состоящую из других объектов. Таким образом программист может скрыть сложность программы за довольно простыми объектами. К примеру дом (достаточно сложный объект) состоит из дверей, комнат, окон, проводки и отопления. Дверь в свою очередь может состоять из собственно двери, ручки, замка и петель. Проводка так-же состоит из проводов, розеток и к примеру щитка.

У каждого объекта есть тип. Иногда тип называют еще и классом. Класс (тип) определяет какие сообщения объекты могут посылать друг другу. Например, аккумуляторная батарея может передавать электролампе ток, а вот момент или физическое усилие - нет.

Все объекты одного типа могут получать одинаковые сообщения. К примеру у

нас есть 2 объекта: синяя и красная кружки. Обе разные по форме и материалу. Но из обеих мы можем пить (или не пить, если они пустые). В данном случае кружка — это тип объекта.

Самое лаконичное описание объекта предложил Буч: «Объект обладает состоянием, поведением и индивидуальностью».

Языки программирования можно оценить по тому, насколько они строги к типам. Контроль типов включает проверку существования вызываемых методов, типов их параметров, проверку границ массивов и подобное.

Различаются чистые и гибридные объектно-ориентированные языки. Чистые — это те, которые позволяют использовать только одну модель программирования — объектно-ориентированную. Вы можете объявлять классы и методы, но не можете завести глобальные переменные и обычные функции и процедуры старого типа.

Третий элемент, по которому различаются языки ООП - их объектная модель. Некоторые традиционные языки ООП позволяют программистам создавать объекты в стеке, в куче (в хипе - heap) или в статической памяти. В этих языках переменная типа класс соответствует объекту в памяти. Так работает C++. В последнее время появилась тенденция использовать другую модель, часто называемую ссылочно-объектной моделью. В этой модели каждый объект динамически размещается в куче, а переменная типа класс фактически является ссылкой или хэндлом объекта в памяти (технически это нечто вроде указателя). Java и Object Pascal оба используют эту ссылочную модель.

Опишем некоторые особенности обьектно-ориентированных языков;

Если вы создали и использовали объект, вам нужно уничтожить его, чтобы не занимать неиспользуемую память.

При создании объекта какого либо класса вызывается специальный метод этого класса, называемый конструктором, который выполняет все действия по подготовке объекта: выделение памяти, инициализация параметров и т.д.

Деструктор играет роль противоположную конструктору и обычно вызывается при уничтожении объекта. Если конструктор нужен большинству классов, то только

некоторые из них нуждаются в деструкторе. Деструктор в основном используется для освобождения ресурсов, зарезервированных конструктором (или другими методами во время жизни объекта). Эти ресурсы включают память, файлы, базы данных, ресурсы ОС и т. д.

Общим элементом всех ООП языков является присутствие трех спецификаторов доступа, указывающих на различные уровни инкапсуляции класса: public, protected, и private. Public означает: видимый любым другим классом, protected означает: видимый производными классами, private означает: отсутствие видимости извне. В деталях, однако, есть различия.

ОО языки обычно разрешают заводить методы и данные, относящиеся к классу целиком, а не к отдельным объектам. Метод класса обычно может быть вызван как для объекта класса, так и применён к классу в целом. Данные класса не повторяются для каждого объекта, а разделяются между всеми объектами данного типа.

Наследование у классов — одно из оснований ООП. Оно может быть использовано для выражения генерализации или специализации. Основная идея в том, что вы определяете новый тип, расширяя или модифицируя существующий, другими словами, производный класс обладает всеми данными и методами базового класса, новыми данными и методами и, возможно, модифицирует некоторые из существующих методов. Различные ОО языки используют различные жаргоны для описания этого механизма (derivation, inheritance, sub-classing), для класса, от которого вы наследуете (базовый класс, родительский класс, суперкласс) и для нового класса (производный класс, дочерний класс, подкласс).

В некоторых ОО языках каждый класс происходит по крайней мере от некоторого базового класса по умолчанию. Этот класс, часто называемый Object, или подобно этому, обладает некоторыми основными способностями, доступными всем классам. Фактически, все другие классы в обязательном порядке его наследуют. Этот подход является общим ещё и потому, что так первоначально делалось в Smalltalk.

Когда вы пишете метод класса или перекрываете метод базового класса, вам нередко надо сослаться на методы базового класса. Если этот метод переопределен в

производном классе, то, используя его имя, вы получите новую версию. В ОО языках есть некоторые приёмы или ключевые слова, позволяющие решить эту проблему.

Когда различные классы в иерархии переопределяют некоторый метод, очень полезна возможность ссылаться на общий объект этих классов (благодаря совместимости подклассов) и вызывать этот метод, результатом чего будет вызов метода надлежащего класса. Для этого компилятор должен поддерживать позднее связывание, то есть не генерировать вызов специфической функции, а ждать, пока во время выполнения не определятся фактический тип объекта и функция, которую нужно вызвать.

При построении сложной иерархии, для обеспечения полиморфизма программисты часто вынуждены вводить методы в классы верхнего уровня, даже если эти методы ещё не определены для этой специфической абстракции. Здесь можно было бы оставить пустые методы, но многие ОО языки предлагают такой специфический механизм, как определение абстрактных методов, то есть методов без реализации. Классы, имеющие хотя бы один абстрактный метод, часто называются абстрактными классами.

Некоторые ОО языки допускают наследование более чем одному базовому классу. Другие языки позволяют вам наследовать только от одного класса, но дополнительно позволять вам наследовать также от многочисленных интерфейсов или чисто абстрактных классов, то есть классов, состоящих только из виртуальных функций.

В строго типизованных ОО языках компилятор осуществляет весь контроль типов, так что нет особой необходимости хранить информацию о классах и типах в работающей программе. Тем не менее, есть случаи (как, например, динамическое преобразование типов), которые требуют информацию о типе. По этой причине все три ОО языка, рассматриваемые здесь, более или менее поддерживают Идентификацию/Информацию о Типе Времени Выполнения (RTTI).

Основная идея обработки исключений — упростить код обработки ошибок в программе, предоставив стандартный встроенный механизм, с целью сделать программы более устойчивыми. Обработка исключений — это тема, требующая отдель-

ного рассмотрения, поэтому я только очерчу некоторые ключевые элементы и различия.

Теперь скажем несколько слов о языке Delphi.

Delphi — императивный, структурированный, объектно-ориентированный язык программирования, диалект Object Pascal] Начиная со среды разработки Delphi 7.0, в официальных документах Borland стала использовать название Delphi для обозначения языка Object Pascal. Начиная с 2007 года уже язык Delphi (производный от Object Pascal) начал жить своей самостоятельной жизнью и претерпевал различные изменения связанные с современными тенденциями (например, с развитием платформы .NET) развития языков программирования: появились class helpers, перегрузки операторов и другое.

Изначально среда разработки была предназначена исключительно для разработки приложений Microsoft Windows, затем был реализован также для платформ Linux (как Kylix), однако после выпуска в 2002 году Kylix 3 его разработка была прекращена, и, вскоре после этого, было объявлено о поддержке Microsoft .NET. Реализация среды разработки проектом Lazarus (Free Pascal, компиляция в режиме совместимости с Delphi) позволяет использовать его для создания приложений на Delphi для таких платформ, как Linux, Mac OS X и Windows CE. Также предпринимались попытки использования языка в проектах GNU (например, Notepad GNU) и написания компилятора для GCC.

Object Pascal — результат развития языка Турбо Паскаль, который, в свою очередь, развился из языка Паскаль. Паскаль был полностью процедурным языком, Турбо Паскаль, начиная с версии 5.5, добавил в Паскаль объектно-ориентированные свойства, а в Object Pascal — динамическую идентификацию типа данных с возможностью доступа к метаданным классов (то есть к описанию классов и их членов) в компилируемом коде, также называемом интроспекцией — данная технология получила обозначение RTTI. Так как все классы наследуют функции базового класса TObject, то любой указатель на объект можно преобразовать к нему, после чего воспользоваться методом ClassType и функцией TypeInfo, которые и обеспечат интро-

спекцию.

Итак, на языке программирования Delphi с использованием концепций объектно-ориентированного программирования было разработано приложение. Данное приложение распознаёт ноты в файлах формата WAV, и записывает их в файл формата MIDI. Для этого используется быстрое преобразование Фурье - алгоритм быстрого вычисления дискретного преобразования Фурье, которое позволяет разложить звуковую волну, записанную в исходном файле, на элементарные составляющие — гармонические колебания с разными частотами. Далее после анализа этих гармоник мы и получаем ноты, которые записываются в выходной файл.

Итого, в работе содержится 74 страниц, 9 рисунков и 7 таблиц.

1. ЗАДАЧА РАСПОЗНАВАНИЯ МУЗЫКИ

1.1. Содержательное описание задачи

Задачей данной работы является построение системы, осуществляющей идентификацию музыкальных структур в данных, представленных в виде отсчетов акустических сигналов (WAV-файл) и запись их в некоторой музыкальной нотации (MIDI-файл), т.е. задача, известная как "распознавание музыки"(music recognition).

"Распознавание музыки"может быть определено как процесс прослушивания частей музыкального произведения и запись в музыкальной нотации нот, которые присутствуют в этих частях. Это реализуется с помощью с помощью извлечения специфической информации из музыкальных звуковых сигналов, и в результате получается символическое представление совокупности нот, их высоты, пауз, динамики и, возможно, информацию о использованных инструментах.

Люди без музыкального образования часто испытывают больше трудностей при распознавании полифонической (в которой одновременно звучит больше одной ноты) музыки, чем музыкально-подготовленные люди. Опыт музыкального стиля, звуков инструментов и познания в музыкальной теории может дать слушателям возможность понимания более сложной и богатой полифонии, сочетающей большое количество различных инструментов и музыкальных стилей.

Автоматическое распознавание музыки привлекает внимание музыкантов и компьютерных специалистов в течение более двадцати пяти лет.

Исследования в области автоматического распознавания музыки проводятся в рамках различных областей науки.

Психоакустика пытается установить зависимость между акустическими сигналами, психологии человеческой системы слуха и восприятия звука. Эта наука основывается еще на знаниях Древней Греции, когда Пифагор заметил, что колеблющиеся одновременно струны звучат хорошо,когда их длины относятся друг к другу как малые целые числа.

Уже в XX-ом веке ряд ученых положили начало эволюции в области психоакустики, которая стала нацелена на глубокое понимание психофизики слуха. Благодаря этим людям, были созданы точные и хорошо протестированные модели, фокусирующиеся на базовых способностях восприятия (восприятие высоты и громкости) как на простых стимулах и на том, что один простой звук маскирует другой во временном пространстве.

Кроме того, частью процесса автоматического распознавания музыки должна быть идентификация музыкальных инструментов, участвующих в музыкальном фрагменте, и автоматический расчет темпа и ритма музыки. Эти параметры очень важны для системы распознавания, так как именно они помогают человеческому мозгу создать точную и целостную картину музыкального образа.

Существует множество способов применения систем восприятия музыки, но все они ограничены из за низкой надежности результатов, предоставляемых текущими решениями.

Однако, достаточно просто указать некоторые области применения для уже существующих и создаваемых в будущем систем:

1. Собственно системы распознавания музыки. Эти системы заинтересуют композиторов и музыкантов, которые хотят эффективно анализировать композиции, имя у себя лишь звуковые записи. Получаемое в результате такой системы символическое представление музыки является гибким для музыкального анализа, редактирования и смешивания, которое вручеую было бы слишком сложно или попросту не возможно.

2. Алгоритмическая композиция. Имеющиеся на текущий системы компьютерной композиции могут улучшить результаты в оценке человеком. Если машины смогут понимать созданную ими музыку, они смогут проверять созданную ими музыку и будут "самокритичными". Это также позволит увеличить степень автоматизации процесса композиции.

3. Визуализация музыки. Можно создавать различные мультимедийные приложения, создающие зрительные и иные образы, каким-либо способом синхронизиро-

ванные с музыкой.

4. Создание музыкальных баз данных. С использованием систем распознавания музыки можно создавать базу данных, в которой можно будет найти полную информацию о фрагменте музыкального произвеления с помощью небольшого его фрагмента. Кроме того, основываясь на данных этой базы можно будет подбирать незнакомую раннее музыку, просто указав свои предпочтения.

5. Структурное кодирование музыки. Использование системы распознавания музыки может использоваться для поиска эквивалентно звучащих звуков, и позволит увеличить качество и уменьшить размер закодированной музыки, как это делается в архиваторах.

6. Автоматизированные системы обучения. Использование систем распознавания музыки позволит создать новое поколение систем обучение музыки, котрое позволит изучать музыку на расстоянии (например, через Интернет), и позволит получать более глубокие познания в музыке.

Распознавание музыки - сложная задача, поскольку в данном процессе необходимо учитывать большое количество параметров и наличие сложных взаимосвязей между ними. Поэтому ниже вводятся некоторые, самые важные, понятия и определения, которые могут получить представление о структуре и этапах решения данной задачи.

К сожалению, трудно найти объективное и формализованное определения музыки. Давалось множество определений как теоретиками музыки, так и музыкантами. Приведём некоторые из них: Музыка - искусство стройного и согласного сочетания звуков, как последовательных (мелодия, напев, голос), так и совместных (гармония, согласие, созвучие); равно искусство это в действии.

Музыка - есть искусство пения и пляски, позже совокупность всех изящных искусств, необходимых для гармонического развития духа, в противоположность гимнастике, искусству воспитания красивого тела. 2) Искусство воспроизведения в звуках чувств и настроений с целью вызвать в слушателе соответствующие чувства и настроения. Главные элементы музыки: ритм, мелодия и гармония. Различают

по содержанию музыку: церковную и светскую, по средствам исполнения: инструментальную и вокальную. Инструм. музыка делится на оркестровую и камерную, по роду инструментов на духовую и смычковую.

Музыка (от греч. musike, буквально - искусство муз), вид искусства, который отражает действительность и воздействует на человека посредством осмысленных и особым образом организованных звуковых последований, состоящих в основном из тонов (звуков определённой высоты). Музыка - специфическая разновидность звуковой деятельности людей. С др. разновидностями (речь, инструментально-звуковая сигнализация и т. д.) её объединяет способность выражать мысли, эмоции и волевые процессы человека в слышимой форме и служить средством общения людей и управления их поведением. В наибольшей степени Музыка сближается с речью, точнее, с речевой интонацией, выявляющей внутреннее состояние человека и его эмоциональное отношение к миру путём изменений высоты и др. характеристик звучания голоса. Это родство позволяет говорить об интонационной природе Музыки. Вместе с тем Музыка существенно отличается от всех остальных разновидностей звуковой деятельности людей. Сохраняя некоторое подобие звуков реальной жизни, музыкального звучания принципиально отличаются от них строгой высотной и временной (ритмической) организованностью. Эти звучания входят в исторически сложившиеся системы, основу которых составляют тоны, отобранные музыкальной практикой данного общества.

Наверное, после рассмотрения вышеуказанных определений, можно лишь сказать что музыка - понятие достаточно философское и многогранное, поэтому мы будем говорить не столько о музыке, сколько об её акустической стороне - совокупности гармонических колебаний.

В традиционной музыкальной нотации Запада (на Востоке существует множество различных музыкальных нотаций, и причем все они достаточно специфические и экзотичные с точки зрения западного человека) основным элементом и понятием является нота - фундаментальный символ, отображающий звук, получаемый с помощью музыкального инструмента. Такое определение ноты включает в себя такие

свойства, как высота, громкость и ритм, и, во многих случаях также и тембр.

Высота ноты кореллирует с частотой одного гармонического тона, поэтому можно упорядочить музыкальные звуки по высоте - от низших к высшим.

Когда мы берем единственную ноту на музыкальном инструменте с частотой, допустим f, это означает на самом деле, что звук также имеет ещё и отзвуки с частотами, кратными данной частоте. При понимании этого эффекта может помочь теория Фурье, которая утверждает, что любой гармонический звук может быть разложен на сумму различных синусоид(чистых тонов) с различными фазами и частотами, кратными основной частоте, т.е. $2f, 3f...$ и т.д. Тогда частота f называется основной частотой гармонического звука. Противоположностью гармонического звука являются не-периодические, хаотические колебания, которые очень сложно подогнать под эту модель.

Синусоида с частотой, кратной основной частоте называется обертоном. Можно считать, что гармоника с базовой частотой является нулевым обертоном.

Громкость является ощущаемым свойством звука, которое имеет связь с физической величиной интенсивности звука. По этому свойств звук может иметь характеристики от "тихого"до "громкого". Связь между громкостью и интенсивностью похода на связь между высотой и частотой.

В музыкальном контексте, это свойство является базой динамики музыкальной фразы (долговременного последовательного изменения громкости последовательности музыкальных нот).

Определение громкости может быть основано на подсчете мощности записанного сигналов

Темп звука является свойством ощущения того, как часто звук повторяется с некоторым интервалом - обычно от 250 миллисекунд до 2 секунд. Темп является достаточно относительным ощущением, однако при указании минимальной длительности ноты его можно восстановить из ритмического рисунка - отношения длительностей различных нот.

Тембр может быть определен как набор качеств, по которым слушатели мо-

гут определить играющий музыкальный инструмент. Он зависит от относительных магнитуд гармоник.

1.2. Математическая формулировка задачи

Дана последовательность отсчетов некоторого сигнала $f(t)$. Разложить данный сигнал на сумму сигналов с определенными частотами: $\omega_i = f_0 \cdot 2^i$, где f_0 - частота камертона - равна 440 Гц.

1.3. Обсуждение задачи

Итак, определим - какой же главный параметр должен быть определен на этапе распознавания? Как видно из описания задачи, а особенно её математической формулировки - это высота звука.

Высота звука, как было указано, напрямую связана с частотой гармоник, простых тонов, поэтому для успешного решения задачи нам необходим алгоритм, который мог бы дать нам данные об амплитудах этих гармоник на выходе, т.е. зависимость амплитуды от частоты, имея на входе зависимость амплитуды от времени. В терминах теории обработки сигналов такие методы называются методами преобразования из временного пространства в частотное пространство , и такие методы рассматриваются в следующем разделе.

2. ВЫБОР И ОБОСНОВАНИЕ ЧИСЛЕННОГО МЕТОДА РЕШЕНИЯ ЗАДАЧИ

2.1. Методы промежуточного представления акустических сигналов

Как уже говорилось, музыка может быть рассмотрена на разных уровнях: от музыки теоретическом уровне (как правило, связано с символическим представлением музыки - ноты) до уровня восприятия (как человеческое сознание воспринимает акустический сигнал) и как физическое явление (связанное с акустикой понятие музыки).

Далее в тексте, форма волны акустического музыкального сигнала будет рассматриваться как представление низкого уровня, а нотная запись будет рассматриваться как представление высокого уровня. Так как музыкальные символы не могут быть непосредственно выделены из акустического сигнала, сигнал должен быть сначала приведен к представлению среднего уровня. При определении вида такого среднего уровня, нужно принимать во внимание ограничения и сильные стороны, которые он накладывает на "верхний уровень"музыкального сигнала во время процесса сокращения числа объектов "нижнего уровня"в нем. Выбранное представление должно легко отвечать на вопросы более высокого уровня переработки и с использованием наиболее эффективных вычислительный методов.

В следующих пунктах представлены некоторые из наиболее важных представлений среднего уровня для музыкальных сигналов.

Ом и Гельмгольц впервые заметили, что ухо, как анализатор Фурье,делит звук на спектральные составляющие, положили начало обсуждениям обработки сигнала в ухе. Их выводы широко используются в спектральном представлении звука в аудиоприложениях. Такой вид представления среднего уровня был предметом многих исследований, кроме того, развитие быстрого преобразования Фурье (БПФ), как вычислительно-эффективного способа вычисления ДПФ добавило популярности Фурье-анализу и синтезу во многих научных и технических приложениях.

Однако есть некоторые различия между спектральным анализом, выполняемым человеческой слуховой системой, и стандартным анализом по методам Фурье. Наиболее важным отличием является то, что слуховая система раскладывает спектр по логарифмической шкале, тогда как традиционный анализ Фурье вычисляет спектр с линейной шкалой. Другое отличие состоит в том, что реализация на основе спектра Фурье не рассматривает другие особенности слуховой системы, такие как маскировка явлений во временном и частотном пространстве, ни различий в восприятии громкости в связи с частотой. Эти различия должны быть приняты во внимание, с связи с тем, что они могут быть исправлены или, по крайней мере приняты во внимание при толковании результаты анализа.

В любом случае, Фурье-представление является очень эффективным решением для анализа звуковых сигналов, в частности в случае транскрипции акустических музыкальных сигналов. Вышеуказанные недочеты должны быть приняты во внимание лишь тогда, когда они чрезмерно искажают результат, что случается не так уж и часто.

Для того чтобы преодолеть недостатки ДПФ в области восприятия, некоторые авторы предложили Constant Q Transform(CQT) в качестве базы для анализа сигналов в системах восприятия музыки. В кратце, благодаря экспоненциальной частоте дискретизации некоторого банка фильтров имитируется человеское восприятия интервалов - октава звучит одинаково в широком диапазон частот.

По сравнению с БПФ, CQT является лучшим приближением к стандартной модели улитки человеческого уха. Тем не менее, расчет частот с помощью БПФ имеет гораздо большую эффективность. Кроме того, каждая подпоследовательность при БПФ имеет одинаковую ширину, что также приводит к более простой архитектуре.

Кроме того, в некоторых работах промежуточное представление сигналов в системе автоматической транскрипции музыки было основано на использовании кореллограмм. Применение данного метода дает весьма неплохие результаты, однако весьма трудоёмко.

Исходя из вышесказанных соображений, в данной работе для получения про-

межуточного представления сигнала используется быстрое преобразование Фурье - как мощный, качественный и проверенный метод для получения частотного спектра.

2.2. Быстрое преобразование Фурье

Для разложения входного сигнала на базовые гармоники можно использовать алгоритм дискретного преобразования Фурье (ДПФ). Однако расчет ДПФ, содержащего N коэффициентов, потребует N^2 пар операций «умножение-сложение». Число операций возрастает пропорционально квадрату размерности ДПФ. Однако, если N не является простым числом и может быть разложено на множители, процесс вычислений можно ускорить, разделив анализируемый набор отсчетов на части, вычислив их ДПФ и объединив результаты. Такие способы вычисления ДПФ называются быстрым преобразованием Фурье (БПФ; английский термин — Fast Fourier Transform, FFT) и повсеместно используются на практике. При реализации БПФ возможно несколько вариантов организации вычислений в зависимости от способа деления последовательности отсчетов на части (прореживание по времени либо по частоте) и от того, на сколько фрагментов производится разбиение последовательности на каждом шаге (основание БПФ). БПФ с прореживанием по времени Рассмотрим идею БПФ с прореживанием по времени (decimation in time, DIT) на примере деления набора отсчетов пополам. Итак, пусть N — четное число. Выделим два слагаемых, соответствующих элементам исходной последовательности с четными и нечетными номерами:

$$X(n) = \sum_{m=0}^{N/2-1} x(2m)e^{-j\frac{2\pi 2mn}{N}} + \sum_{m=0}^{N/2-1} x(2m+1)e^{-j\frac{2\pi(2m+1)n}{N}}$$

Введем обозначения $y(m) = x(2m)$ и $z(m) = x(2m+1)$, а также вынесем из второй суммы общий множитель $e^{\frac{-2j\pi n}{N}}$:

$$X(n) = \sum_{m=0}^{N/2-1} y(m)e^{-j\frac{2\pi 2mn}{N}} + e^{\frac{-2j\pi n}{N}} \sum_{m=0}^{N/2-1} z(m)e^{-j\frac{2\pi 2mn}{N}}$$

Две суммы представляют собой ДПФ последовательностей $\{y(m)\}$ (отсчеты с четными номерами) и $\{z(m)\}$ (отсчеты с нечетными номерами). Каждое из этих ДПФ имеет размерность $N/2$. Таким образом,

$$X(n) = Y(n) + e^{\frac{-2j\pi n}{N}} Z(n),$$

где $Y(n)$ и $Z(n)$ – ДПФ соответственно последовательностей отсчетов с четными и нечетными номерами:

$$Y(n) = \sum_{m=0}^{N/2-1} y(m) e^{-j\frac{2\pi 2mn}{N}}$$
$$Z(n) = \sum_{m=0}^{N/2-1} z(m) e^{-j\frac{2\pi 2mn}{N}}$$

Так как ДПФ размерности $N/2$ дает лишь $N/2$ спектральных коэффициентов, непосредственно использовать вышеприведенные две формулы можно только при $0 < n < N/2$. Для остальных $n(N/2 < n < N)$ следует воспользоваться периодичностью спектра дискретного сигнала (и, соответственно, периодичностью результатов ДПФ):

$$Y(n + \frac{N}{2}) = Y(n), Z(n + \frac{N}{2}) = Z(n)$$

С учетом этого при $n \geq N/2$ получаем:

$$X(n) = Y(n - \frac{N}{2}) + e^{\frac{-2j\pi}{N}(n-\frac{N}{2})} Z(n - \frac{N}{2})$$

Оценим количество операций, необходимое для вычисления ДПФ указанным способом. Каждое из двух ДПФ половинной размерности требует $N^2/4$ операций. Кроме того, при вычислении окончательных результатов каждый спектральный коэффициент $Z(n)$ умножается на экспоненциальный комплексный множитель. Это добавляет еще $N/2$ операций. Итого получается $2N^2/4 + N/2 = N(N+1)/2$, что почти вдвое меньше, чем при вычислении ДПФ прямым способом. Если $N/2$ тоже является четным числом (то есть если N делится на 4), можно продолжить описанную процедуру, выразив результат через четыре ДПФ размерности $N/4$. Это позволяет еще больше сократить число требуемых вычислительных операций.

Делить исходную последовательность можно на любое количество частей. Таким образом, приведенный алгоритм позволяет уменьшить число операций в случае любого N, не являющегося простым числом. Степень ускорения вычислений зависит от числа фрагментов последовательности и является максимальной при делении на две части, как в рассмотренном примере. Наибольшая степень ускорения вычислений может быть достигнута при $N = 2^k$, в этом случае деление последовательностей на две части можно продолжать до тех пор, пока не получатся двухэлементные последовательности, ДПФ которых рассчитывается вообще без использования операций умножения (достаточно вычислить сумму и разность двух отсчетов). Число требуемых при этом пар операций «умножение — сложение» можно оценить как $N \log_2(N)$. Таким образом, вычислительные затраты по сравнению с непосредственным использованием ДПФ уменьшаются в $N \log_2(N)$ раз. При больших N это отношение становится весьма велико (например, $1024/\log_2(1024) = 102,4$, то есть при $N = 1024$ достигается более чем 100-кратное ускорение).

Формулы прямого и обратного ДПФ отличаются только знаком в показателе экспоненты и множителем перед суммой. Поэтому можно получить еще один вариант алгоритма БПФ. Этот способ вычислений называется прореживанием по частоте (decimation in frequency, DIF). Разделим исходную последовательность x(k) на две следующие друг за другом половины (как и в предыдущем случае, N должно быть четным числом):

$$X(n) = \sum_{m=0}^{N/2-1} x(2m) e^{-j\frac{2\pi 2mn}{N}} + \sum_{m=0}^{N/2-1} x(m+N/2) e^{-j\frac{2\pi(m+N/2)n}{N}}$$

Из второй суммы можно выделить множитель

$$e^{-j\frac{2\pi(N/2)n}{N}} = e^{(-j\pi n)} = (-1)^n.$$

Этот множитель равен 1 или -1 в зависимости от четности номера вычисляемого спектрального отсчета n, поэтому дальше рассматриваем четные и нечетные n по отдельности. После выделения множителя 1 комплексные экспоненты в обеих сум-

мах становятся одинаковыми, поэтому выносим их за скобки, объединяя две суммы:

$$X(2k) = \sum_{m=0}^{N/2-1} \left(x(2m) + x(m+N/2)\right) e^{-j\frac{2\pi mk}{N/2}}$$

$$X(2k+1) = \sum_{m=0}^{N/2-1} \left(x(2m) + x(m+N/2)\right) e^{-j\frac{2\pi mk}{N/2}} e^{-j\frac{2\pi m}{N}}$$

Фигурирующие здесь суммы представляют собой ДПФ суммы и разности половин исходной последовательности, при этом разность перед вычислением ДПФ умножается на комплексные экспоненты $exp(-j2nm/N)$. Каждое из двух используемых здесь ДПФ имеет размерность $N/2$.

В названиях алгоритмов БПФ можно встретить слово «RADIX» («основание» — в математическом смысле). Следующее после него число обозначает число фрагментов, на которое разбивается сигнал на каждом этапе прореживания (а также минимальный размер «кусочков» входного вектора, который достигается в результате его последовательных разбиений). В алгоритмах «RADIX-2» размер анализируемой последовательности должен быть равен степени двойки, а ее половинное деление производится вплоть до получения двухэлементных последовательностей. Вычисление их ДПФ не требует операций умножения — два спектральных отсчета представляют собой сумму и разность отсчетов временных:

$$X(0) = x(0) + x(1),$$
$$X(1) = x(0) - x(1).$$

В алгоритмах «RADIX-4» количество отсчетов сигнала должно быть равно степени четверки, при каждом прореживании сигнал делится на четыре фрагмента, а последней стадией деления являются четырехэлементные последовательности. При вычислении их ДПФ умножение производится только на j, а такое умножение сводится к взаимной перестановке вещественной и мнимой частей комплексного числа с изме-

нением знака у одной из них:

$$X(0) = x(0) + x(1) + x(2) + x(3),$$
$$X(1) = x(0) - jx(1) - x(2) + jx(3),$$
$$X(2) = x(0) - x(1) + x(2) - x(3),$$
$$X(3) = x(0) + jx(1) - x(2) - jx(3).$$

3. РАЗРАБОТКА АЛГОРИТМА

3.1. Структуры данных

Итак, после выбора главного алгоритма разрабатываемой программы, можно приступить к её разработке и реализации.

Введём сначала обозначения и поясним предназначение переменных и структур данных, которые могут встретиться в программе.

Как уже говорилось, входные данные для программы будут представленные в формате WAV, который в свою очередь является подмножеством формата RIFF (Rich Interchange File Format), разработанного Microsoft. Данный формат является достаточно общим (достаточно сказать, что формат MIDI, который имеет совершенно другую логику, также описывается практически тем же форматом), поэтому он насыщен различного рода метаданными, однако с нашей точки зрения кажется важным лишь то, что сами полезные данные WAV могут быть представлены как последовательность отсчетов, модули которых имеют абсолютное значение - импульсно-кодовая модуляция (ИКМ; Pulse-Code Modulation, PCM), или являются разницей от предыдущего отсчета - в данном случае говорят об адаптивной дельта ИКМ (Adaptive-Delta PCM,ADPCM). Для нас более удобен первый вариант, и поэтому будем испольлзовать именно его, поэтому структуре данных, представляющей входные данные дадим имя WavPCM.

Также нам понадобятся данные о количестве точек, на которых будет применено БПФ. Обозначим данную переменную как FCount. Также потребуется массив длинной FCount для временного хранения считанных из WavPCM отсчетов, обозначим его как Temp.

При использовании итеративного преобразования Фурье, которое имеет преимущества в быстродействии перед рекурсивным, нам потребуется переставить исходные данные определенным образом. Обозначим массив, значения которого указывают на место элемента с данным индексом в преобразованном массиве, как

IndexTable. Кроме того, комплексные экспоненты, участвующие в преобразовании, можно рассчитать заранее, что может повысить скорость обработки. Поэтому сохраним данные экспоненты (также называемые поворачивающими множителями) в массив под названием ExpTable.

Так как классическое преобразование Фурье предполагает, что входной сигнал является стационарным (то есть является постоянным, единым процессом в течение своей продолжительности), то музыкальное произведение, в котором сигналы-ноты могут сменятся до нескольких раз в секунду, не подходит под эти требования. Однако, если считать сами ноты достаточно стационарными сигналами, то появляется возможность провести оконное преобразование Фурье, и рассчитывать БПФ на небольшом временном отрезке. Поэтому при анализе исходного сигнала приходится проводить не одно, а несколько преобразований Фурье. Соотвественно, будем сохранять результаты этих преобразований в массив. Получается, что тогда результаты анализа представляют собой двумерный массив, верхний индекс которого указывает номер временного участка, на котором проводилось преобразование, и нижний индекс которого является номером гармоники, полученным в результате данного преобразования. Обозначим данную структуру данных как Fourier.

После этого мы готовы будем приступить к получению MIDI-файла. Данный формат файла имеет с форматом WAV схожую форму, но совершенно другое смысловое наполнение. Если WAV-файл содержит просто замер амплитуды сигнала в каждый момент времени, то в MIDI-файле сохранены команды инструментам, когда исполнять ноту той или иной высоты в каждый момент времени. Так как имеется прямая зависимость между высотами нот и некоторыми соответствующими гармониками, то для получения информации о нотах мы должны провести анализ гармоник соответствующих частот.

Итак, после завершения спектрального анализа Фурье, мы должны провести гармонический анализ, который, используя информацию о гармониках, даст нам информацию о нотах, звучащих во время каждого временного участка оконного преобразования Фурье, то есть получим данные о высотах нот и времени из звучания.

Этих данных достаточно, чтобы получить MIDI-файл. Итак, обозначим эти данные как MIDIData.

Теперь иы ввели все основные структуры данных, что позволяет перейти к этапу разработки алгоритма.

3.2. Структура алгоритма

Итак, обрисуем словами алгоритм по получению MIDI-файла из файла WAV.

Сначала мы получаем входные данные WavPCM и параметр FCount.

Далее мы считываем данные из WavPCM порциями по FCount отсчетов, и применяем к каждой такой порции БПФ. В результате этой операции мы получаем двумерный массив Fourier, в котором верхний индекс указывает номер участка, на котором проводилось преобразование, и нижний индекс которого является номером гармоники, полученным в результате данного преобразования.

Далее к полученным данным применяется процедура гармонического анализа, и в результате мы получаем выходные данные MidiData. Эти данные и являются результатом работы алгоритма.

3.3. Схема алгоритма

На рисунке 1 представлена схема общего алгоритма преобразования WAV-файла в MIDI-файл.

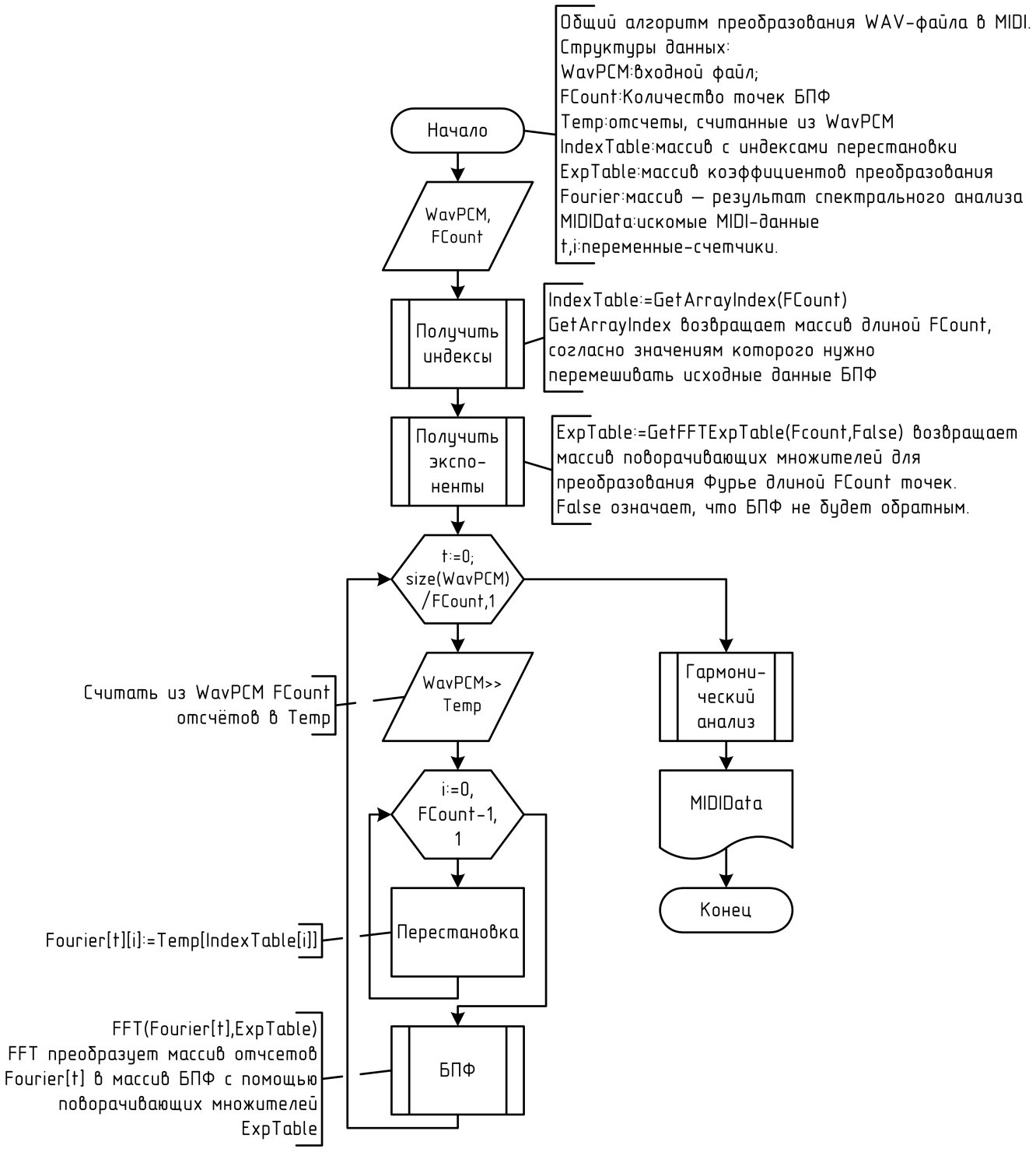

Рисунок 1 – Схема общего алгоритма преобразования WAV-файла в MIDI-файл

На рисунке 2 представлена схема алгоритма получения для перемешивания исходных данных при подготовке их к БПФ.

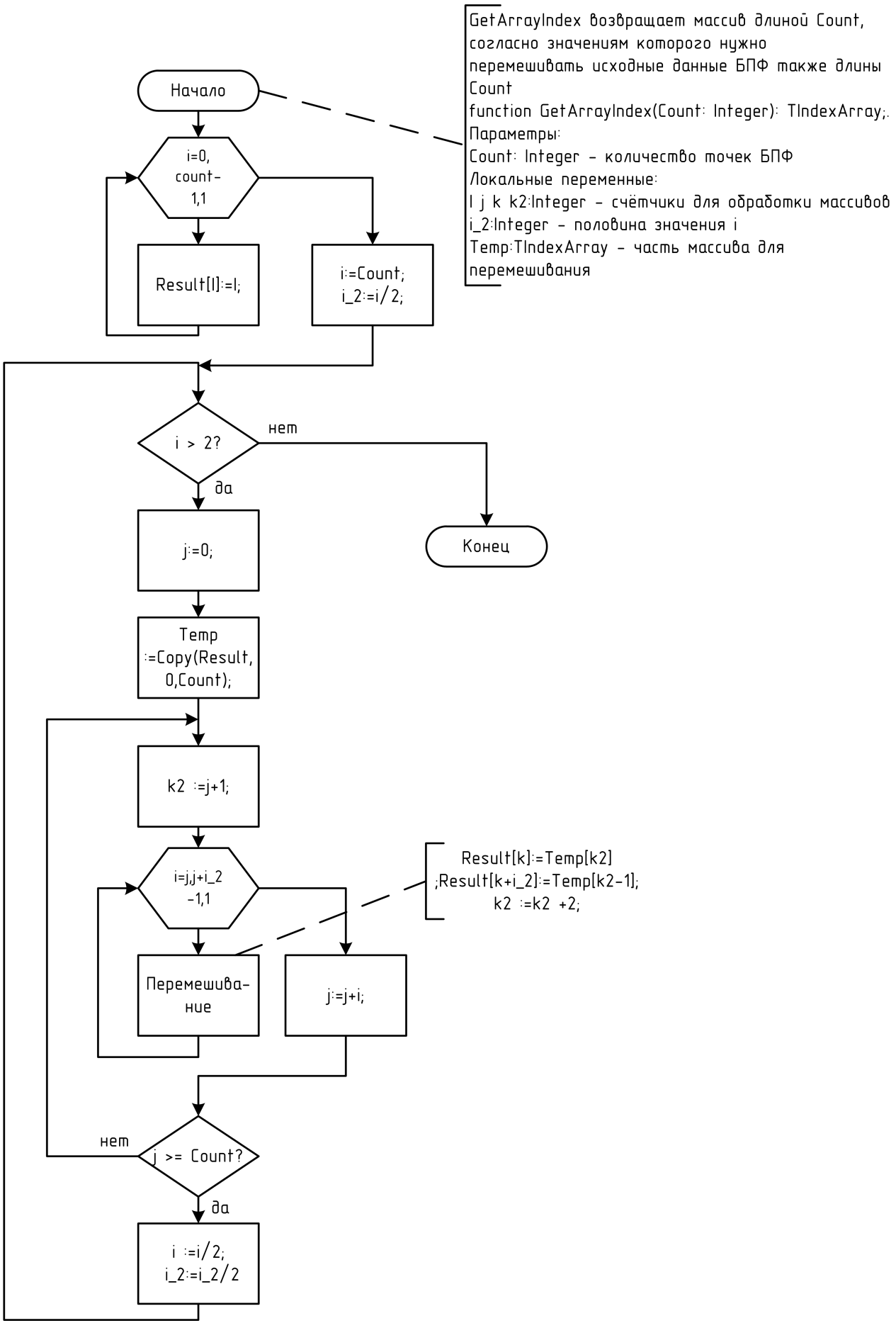

Рисунок 2 – Схема алгоритма перемешивания данных

На рисунке 3 представлена схема алгоритма получения поворачивающих множителей - комплексных экспонент, коэффициентов БПФ.

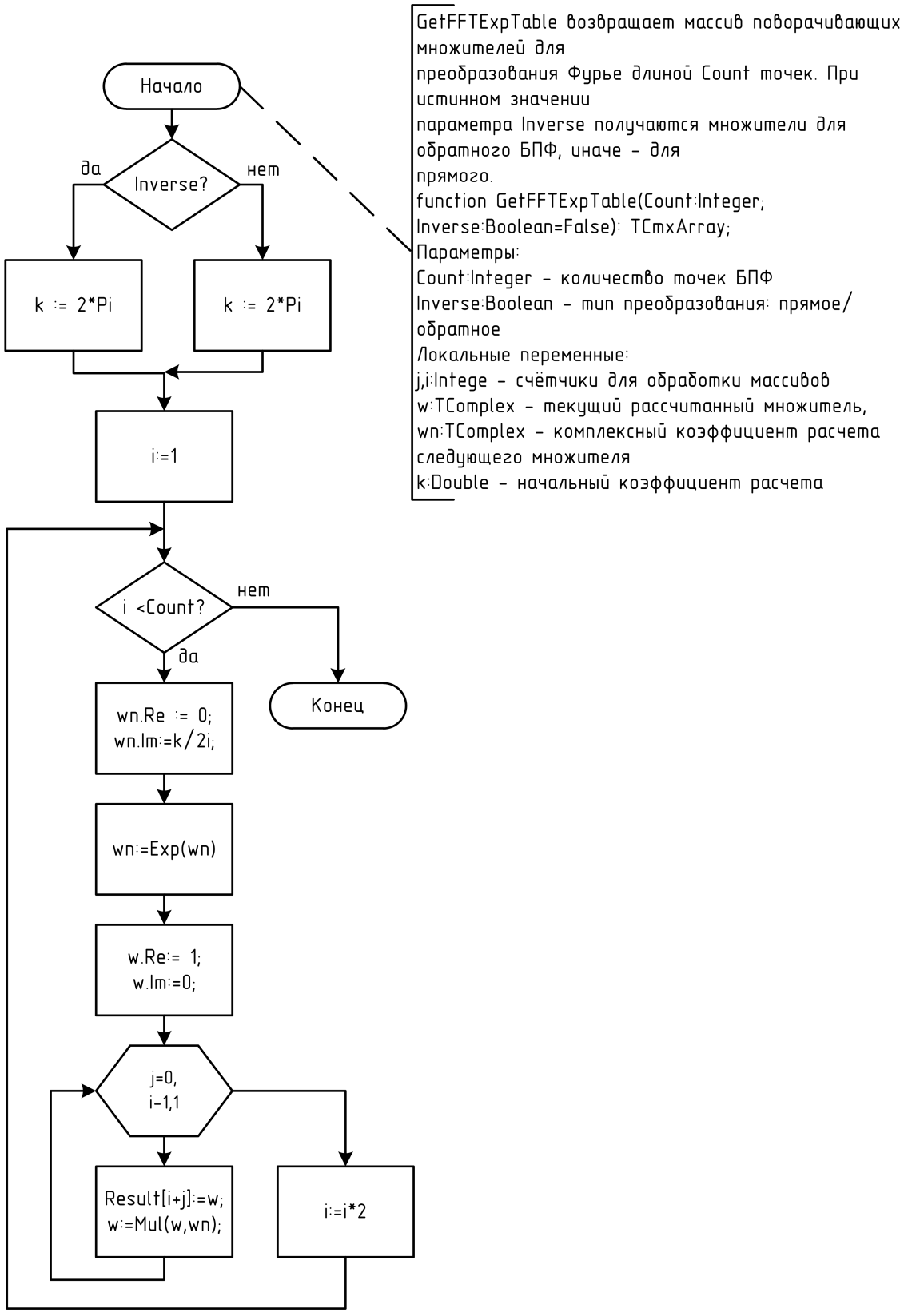

Рисунок 3 – Схема алгоритма получения поворачивающих множителей

На рисунке 4 представлена схема алгоритма БПФ - быстрого преобразования Фурье.

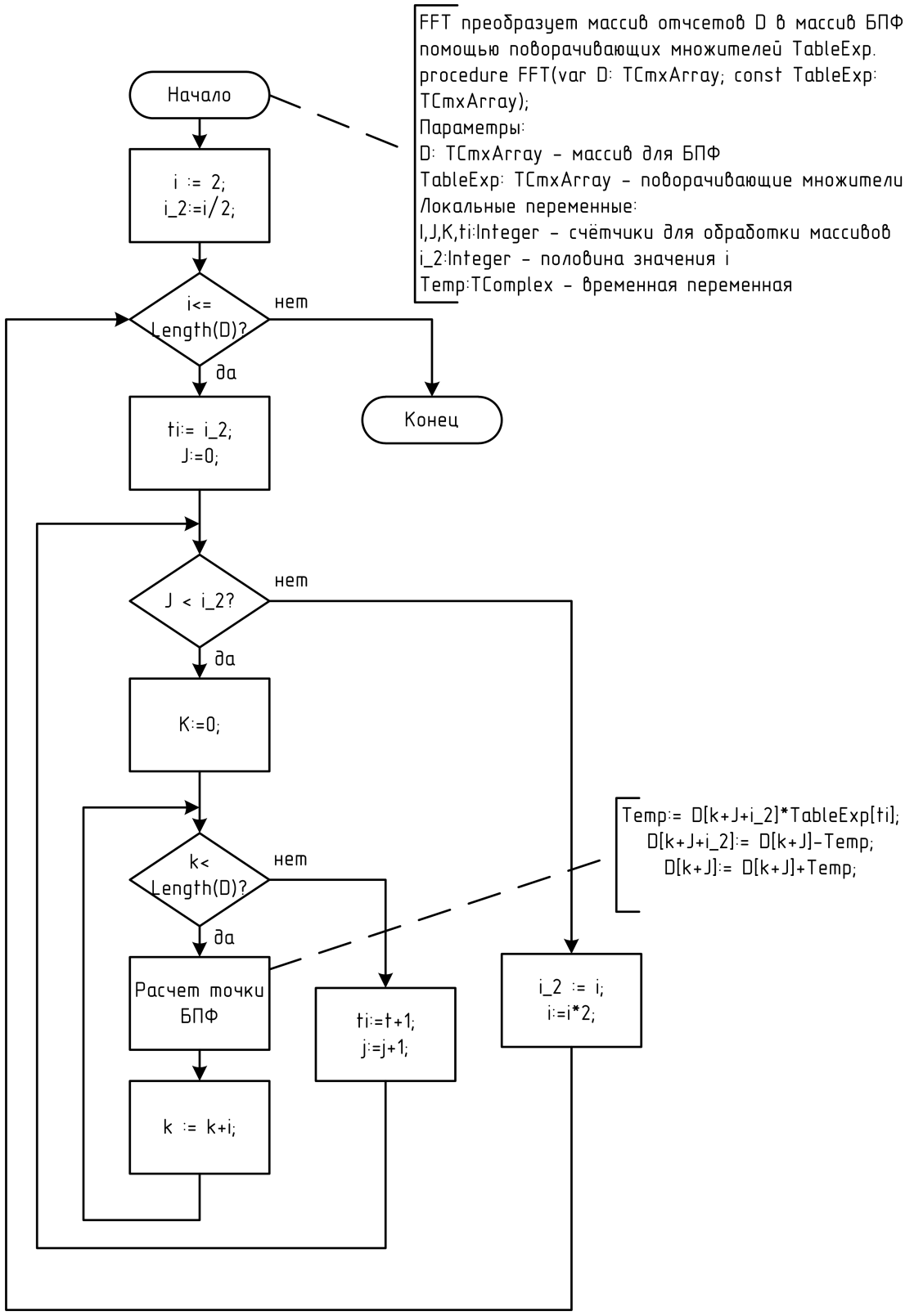

Рисунок 4 – Схема алгоритма БПФ

4. ТЕКСТ ПРОГРАММЫ

4.1. Описание переменных и структур данных

Переменные программы распознавания нот в WAV-файле представлены в таблице 1 .

Таблица 1 – Переменные программы распознавания нот

имя	тип	предназначение
MidiPort,MidiStatus	Integer	Порт MIDI и статус его открытия
instr_indexes	array of byte	индексы используемых инструментов MIDI
pcount	integer	количество точек при построении графика звуковой волны
UsedWindow	byte	индекс используемой оконной функции
FCount	Integer	количество точек для преобразования
FCount_1	Integer	количество точек для преобразования минус 1
FCountDiv2	Integer	половина точек для преобразования
FCountDiv2_1	Integer	половина точек для преобразования минус 1
norm	Double	нормирующий множитель амплитуды
MaxAmplitude	Double	максимальная амплитуда среди гармоник
Eps	Double	минимальная учитываемая амплитуда гармоники
UpdateTrack	Boolean	флаг обновления ползунка проигрывателя
AnalyzeComplete	Boolean	флаг проведения спетроанализа
WaveLoaded	Boolean	флаг получения WAV-файла
MidiCreated	Boolean	флаг создания MIDI-данных
MidiSaved	Boolean	флаг сохранения MIDI-данных
Form1	TForm1	форма программы
WavPCM	TPCMWaveFile	WAV-файл
Fourier	array of TCmxArray	массив гармоник

4.2. Описание функций

1. Функция GetFFTExpTable возвращает массив поворачивающих множителей для преобразования Фурье длиной Count точек. При истинном значении параметра Inverse получаются множители для обратного БПФ, иначе - для прямого.

function GetFFTExpTable(Count:Integer; Inverse:Boolean=False): TCmxArray;

Параметры функции представлены в таблице 2 :

Таблица 2 – Параметры функции расчета поворачивающих множителей

имя	тип	предназначение
Count	Integer	количество точек БПФ
Inverse	Boolean	тип преобразования: прямое/обратное

Локальные переменные функции представлены в таблице ?? :

Таблица 3 – Локальные переменные функции расчета поворачивающих множителей

имя	тип	предназначение
j,i	Integer	счётчики для обработки массивов
w	TComplex	текущий рассчитанный множитель
wn	TComplex	комплексный коэффициент расчета следующего множителя
k	Double	начальный коэффициент расчета

2. Функция GetArrayIndex возвращает массив длиной Count, согласно значениям которого нужно перемешивать исходные данные БПФ также длины Count.

function GetArrayIndex(Count: Integer): TIndexArray;

Параметры функции представлены в таблице 4 :

Таблица 4 – Параметры функции расчета индексов перемешивания

имя	тип	предназначение
Count	Integer	количество точек БПФ

Локальные переменные функции представлены в таблице 5 :

Таблица 5 – Локальные переменные функции расчета индексов перемешивания

имя	тип	предназначение
I,j,k,k2	Integer	счётчики для обработки массивов
i_2	Integer	половина значения i
Temp	TIndexArray	часть массива для перемешивания

3. Процедура FFT преобразует массив отсчетов D в массив БПФ с помощью поворачивающих множителей TableExp.

procedure FFT(var D: TCmxArray; const TableExp: TCmxArray);

Параметры процедуры представлены в таблице 6 :

Таблица 6 – Параметры процедуры расчета БПФ

имя	тип	предназначение
D	TCmxArray	массив для БПФ
TableExp	TCmxArray	поворачивающие множители

Локальные переменные процедуры представлены в таблице 7 :

Таблица 7 – Локальные переменные процедуры расчета БПФ

имя	тип	предназначение
I,J,K,ti	Integer	счётчики для обработки массивов
i_2	Integer	половина значения i
Temp	TComplex	временная переменная

4.3. Текст программы

Далее приводится текст модуля, написанного на языке Delphi 7, содержащего преобразование Фурье и связанные с ним процедуры:

```
Unit ModuleFFT;
Interface
Uses ModuleComplex,math;
type TIndexArray  = array of Integer;
function HannWindow(t,N:integer):Extended;
function HammingWindow(t,N:integer):Extended;
```

```
function BlackmanWindow(t,N:integer):Extended;
procedure FFT(var D: TCmxArray; const TableExp: TCmxArray);
function GetFFTExpTable(Count:Integer; Inverse:Boolean=False): TCmxArray;
function GetArrayIndex(Count: Integer): TIndexArray;
implementation
function HammingWindow(t,N:integer):Extended;
begin
  result:=0.53836-0.46164*cos(2*pi*t/(N-1));
end;

function HannWindow(t,N:integer):Extended;
begin
  result:=0.5*(1-cos(2*pi*t/(N-1)));
end;

function BlackmanWindow(t,N:integer):Extended;
const a=0.16;
      a0=(1-a)/2;
      a1=1/2;
      a2=a/2;
begin
  result:=a0-a1*cos(2*pi*t/(N-1))+a2*cos(4*pi*t/(N-1));
end;
(*
FFT преобразует массив отчсетов D в массив БПФ с помощью поворачивающих
множителей TableExp.
Параметры:
D: TCmxArray - массив для БПФ
TableExp: TCmxArray - поворачивающие множители
Локальные переменные:
I,J,K,ti:Integer - счётчики для обработки массивов
i_2:Integer - половина значения i
Temp:TComplex -временная перменная
*)
procedure FFT(var D: TCmxArray; const TableExp: TCmxArray);
var
    I,J,K,ti,i_2:Integer;
    Temp:TComplex;
begin
  i := 2;i_2:=i shr 1;
  while i <= Length(D) do
    begin
      ti:= i_2;
      J:=0;
      while J < i_2 do
        begin
          K:=0;
          while k<Length(D) do
            begin
              Temp       := CmpMul(D[k+J+i_2],TableExp[ti]);
              D[k+J+i_2]  := CmpSub(D[k+J],Temp);
              D[k+J]:= CmpAdd(D[k+J],Temp);
              k            := k+i;
            end;
          Inc(ti);
          Inc(J);
        end;
      i_2 := i;
      i  := i shl 1;
```

```
    end;
end;
(*
GetFFTExpTable возвращает массив поворачивающих множителей для
преобразования Фурье длиной Count точек. При истинном значении
параметра Inverse получаются множители для обратного БПФ, иначе — для
прямого.
Параметры:
Count:Integer — количество точек БПФ
Inverse:Boolean — тип преобразования: прямое/обратное
Локальные переменные:
j,i:Intege — счётчики для обработки массивов
w:TComplex — текущий рассчитанный множитель,
wn:TComplex — комплексный коэффициент расчета следующего множителя
k:Double — начальный коэффициент расчета
*)
function GetFFTExpTable(Count:Integer; Inverse:Boolean=False): TCmxArray;
var j,i:Integer;
    w,wn:TComplex;
    k:Double;
begin

  k := −2*Pi;

  if Inverse then
     k:= −k;


  SetLength(Result,Count +1);

  i:=1;

  while i <Count do
    begin
      wn.Re := 0;
      wn.Im := k/(i shl 1);
      wn    := CmpExp(wn);
      w.Re     := 1;
      w.Im     :=0;
      For j:=0 to i−1 do
        begin

          Result[i+j]:=w;
          w:=CmpMul(w,wn);
        end;
    i  := i shl 1;
    end;
end;

(*
GetArrayIndex возвращает массив длиной Count, согласно значениям которого
нужно
перемешивать исходные данные БПФ также длины Count.
Параметры:
Count: Integer — количество точек БПФ
Локальные переменные:
I j k k2:Integer — счётчики для обработки массивов
i_2:Integer — половина значения i
Temp:TIndexArray — часть массива для перемешивания
```

```
*)
function GetArrayIndex(Count: Integer): TIndexArray;
Var I,i_2,j,k,k2:Integer;
    Temp:TIndexArray;
begin
  SetLength(Result,Count);
  For I:=0 to count-1 do
      Result[I]:=I;
  i   :=Count;
  i_2  :=i shr 1;
  while i > 2 do
    begin
      j:=0;
      Temp :=Copy(Result,0,Count);
      repeat
        k2 :=j;
        for k:=j to j+i_2-1 do
          begin
            Result[k]          :=Temp[k2];
            Result[k+i_2]:=Temp[k2+1];
            k2 :=k2 +2;
          end;
        j:=j+i;
      Until j >= Count;
      i   :=i shr 1;
      i_2  :=i_2 shr 1;
    end;
  Temp:=Nil;
end;

end.
```

Далее представлен текст основного модуля программы поиска нот в WAV-файле, написанной на языке Delphi 7.

```
unit UnitMain;
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs,ModuleFFT,ModuleComplex,ModuleWAV, StdCtrls, ExtCtrls, TeEngine,
  Series, TeeProcs, Chart,Math, ComCtrls, cmpBarControl, cmpPianoRoll,
  cmpKeyboard, cmpMidiData,mmsystem, Menus, Grids, MPlayer,
  Buttons, ToolWin,IniFiles,UnitAbout;
const
      tempos:array[1..9] of byte=
        (44,50,55,60,90,105,120,175,200);
      notes_name:array[1..12]of string=
      ('C','C#','D','D#','E','F','F#','G','G#','A','B','H');
      default_freq:array[1..12]of double=
      (261.63,277.18,293.66,311.13,329.63,349.23,
        369.99,392.00,415.30,440.00,466.16,493.88);
      octaves:array [1..10] of string=
    ('субконтр', 'контр' ,'большой ','малой ',
      '1-й ','2-й ','3-й ','4-й ','5-й ','6-й ');
      rus_names:array[1..12] of string=
    ('До', 'До диез', 'Ре', 'Ре диез', 'Ми', 'Фа',
    'Фа диез', 'Соль', 'Соль диез', 'Ля', 'Си бемоль', 'Си');
```

```
      Instruments:array [0..127] of String=(
'AcousticGrandPiano','BrightAcousticPiano','ElectricGrandPiano',
'HonkyTonkPiano','ElectricPiano1','ElectricPiano2','Harpsichord','Clavinet',
'Celesta','Glockenspiel','MusicBox','Vibraphone','Marimba','Xylophone',
'TubularBells','Dulcimer',
'DrawbarOrgan','PercussiveOrgan','RockOrgan','ChurchOrgan',
'ReedOrgan','Accordion','Harmonica','TangoAccordion',
'AcousticNylonGuitar','AcousticSteelGuitar','JazzElectricGuitar',
'CleanElectricGuitar','MutedElectricGuitar','OverdrivenGuitar',
'DistortionGuitar','GuitarHarmonics','AcousticBass',
'FingeredElectricBass','PickedElectricBass','FretlessBass',
'SlapBass1','SlapBass2','SynthBass1','SynthBass2',
'Violin','Viola','Cello','Contrabass',
'TremoloStrings','PizzicatoStrings','OrchestralHarp','Timpani',
'StringEnsemble1','StringEnsemble2','SynthStrings1',
'SynthStrings2','ChoirAahs','VoiceOohs','SynthVoice','OrchestraHit',
'Trumpet','Trombone','Tuba','MutedTrumpet','FrenchHorn',
'BrassSection','SynthBrass1','SynthBrass2',
'SopranoSax','AltoSax','TenorSax','BaritoneSax',
'Oboe','EnglishHorn','Bassoon','Clarinet',
'Piccolo','Flute','Recorder','PanFlute','BlownBottle',
'Shakuhachi','Whistle','Ocarina',
'SquareLead','SawtoothLead','CalliopeLead','ChiffLead',
'CharangLead','VoiceLead','FifthsLead','BassandLead',
'NewAgePad','WarmPad','PolySynthPad','ChoirPad',
'BowedPad','MetallicPad','HaloPad','SweepPad',
'SynthFXRain','SynthFXSoundtrack','SynthFXCrystal','SynthFXAtmosphere',
'SynthFXBrightness','SynthFXGoblins','SynthFXEchoes','SynthFXSciFi',
'Sitar','Banjo','Shamisen','Koto','Kalimba',
'Bagpipe','Fiddle','Shanai',
'TinkleBell','Agogo','SteelDrums','Woodblock',
'TaikoDrum','MelodicTom','SynthDrum','ReverseCymbal',
'GuitarFretNoise','BreathNoise','Seashore','BirdTweet',
'TelephoneRing','Helicopter','Applause','Gunshot');

type
  TFreqArray=array [1..12] of Double;
  TInstrSet=set of 0..127;
  TMyForm =class (Tform)
    clbSpectum: TColorBox;//цвет спектра
    chkWaveform: TCheckBox; //рисовать график волны
    chkListSpectr: TCheckBox; //прокручивать спектрограмму
    chkAutosave: TCheckBox; //автосохранение
    clbWaveform: TColorBox; //цвет звуковой волны
    edtWavPoints: TEdit;  //количество точек графика
    rgWindowFuncs: TRadioGroup; //оконные функции
    cbxFFTCount: TComboBox;   //количество БПФ
    cbxInstruments: TComboBox; //инструмент
    cbxTempo: TComboBox;    //темп
    procedure update_instruments;
    public
    notes_freq:TFreqArray;
    instr:TInstrSet;
  end;
  TForm1 = class(TMyForm)
    MidiData: TMidiData;//данные MIDI
    MainMenu: TMainMenu;//меню
    N1: TMenuItem; //пункт меню
    N2: TMenuItem; //пункт меню
```

```
odgWaveOpen: TOpenDialog;//диалог открытия файла
WAV1: TMenuItem; //пункт меню
Timer1: TTimer;  //таймер
grpWAV: TGroupBox; //группа парметров
btnWavSel: TBitBtn; //кнопка выбора файлов
stWavName: TStaticText; //имя файла
Label2: TLabel; //надпись
Label3: TLabel; //надпись
Label4: TLabel; //надпись
Label11: TLabel; //надпись
chtWaveform: TChart; // график звуковой волны
Label1: TLabel;  // надпись
grpAnalyze: TGroupBox; //группа параметров анализа
PianoRoll: TPianoRoll; //табулатура
MIDIKeys: TMIDIKeys;  //пианинко
Bevel1: TBevel;      // рамочка
stMidiName: TStaticText; //имя файла
btnMidiSel: TBitBtn;   //кнопка выбора файла
grpMIDI: TGroupBox;   //группа параметров
chtSpectrum: TChart;  //спектрограма
Series1: TBarSeries;  //данные спектрограммы
btnAnalyze: TBitBtn;  //кнопка анализа
Label13: TLabel;     //надпись
stFocusedNote: TLabel; //текущая нота
Bevel2: TBevel;       //рамка
Label14: TLabel;    //надпись
trkSpectrum: TTrackBar; //ползунок спектрограммы
trkMinAmp: TTrackBar;  //ползунок амплитуды
Label12: TLabel;   //надпись
btnFindNotes: TBitBtn;  //кнопка поиска нот
sdgMidiSave: TSaveDialog; //сохранение файла
Label15: TLabel;   //надпись
btnSaveMidi: TBitBtn;      //сохранение файла
Label18: TLabel;  //надпись
MediaPlayer: TMediaPlayer;   //проигрыватель
trkMPlayer: TTrackBar;     // ползунок проигрывателя
Label9: TLabel;  //надпись
Label5: TLabel;  //надпись
Label6: TLabel;  //надпись
N3: TMenuItem; //пункт меню
N4: TMenuItem;  //пункт меню
MIDI1: TMenuItem; //пункт меню
MIDI2: TMenuItem; //пункт меню
N5: TMenuItem;  //пункт меню
N6: TMenuItem;  //пункт меню
N7: TMenuItem;  //пункт меню
N8: TMenuItem;  //пункт меню
N9: TMenuItem;  //пункт меню
N10: TMenuItem; //пункт меню
N11: TMenuItem;     //пункт меню
N12: TMenuItem;     //пункт меню
N13: TMenuItem;     //пункт меню
Series2: TFastLineSeries; //данные графика звуковой волны
Label7: TLabel;           //надпись
sdgIniSave: TSaveDialog;   //сохранение настроек
odgIniOpen: TOpenDialog;   //открытие настроек
N15: TMenuItem;            //пункт меню
MIDI3: TMenuItem;          //пункт меню
Label8: TLabel;             //надпись
```

```
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure trkSpectrumChange(Sender: TObject);
    procedure trkMinAmpChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure WAV1Click(Sender: TObject);
    procedure PianoRollScroll(Sender: TObject; ScrollCode: TScrollCode;
      var ScrollPos: Integer);
    procedure PianoRollFocus(Sender: TObject);
    procedure PianoRollMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure PianoRollMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure MediaPlayerNotify(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure MediaPlayerPostClick(Sender: TObject; Button: TMPBtnType);
    procedure OpenWAVE(const filename:string);
    procedure DrawWaveform(pcount:cardinal);
    procedure btnWavSelClick(Sender: TObject);
    procedure chkWaveformClick(Sender: TObject);
    procedure clbWaveformChange(Sender: TObject);
    procedure trkMPlayerChange(Sender: TObject);
    procedure btnAnalyzeClick(Sender: TObject);
    procedure FreeFourier;
    procedure FreeWAV;
    procedure FreeMidi;
    procedure btnFindNotesClick(Sender: TObject);
    procedure btnMidiSelClick(Sender: TObject);
    procedure btnSaveMidiClick(Sender: TObject);
    procedure cbxInstrumentsSelect(Sender: TObject);
    procedure chkAutosaveClick(Sender: TObject);
    procedure cbxFFTCountChange(Sender: TObject);
    procedure clbSpectumSelect(Sender: TObject);
    procedure rgWindowFuncsClick(Sender: TObject);
    procedure edtWavPointsKeyPress(Sender: TObject; var Key: Char);
    procedure MIDI1Click(Sender: TObject);
    procedure MIDI2Click(Sender: TObject);
    procedure N6Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure N7Click(Sender: TObject);
    procedure N8Click(Sender: TObject);
    procedure N10Click(Sender: TObject);
    procedure N11Click(Sender: TObject);
    procedure N12Click(Sender: TObject);
    procedure N13Click(Sender: TObject);
    procedure MIDI3Click(Sender: TObject);
    procedure edtWavPointsExit(Sender: TObject);
    procedure UpdateWaveForm;
    procedure UpdateOptions;
    procedure cbxInstrumentsChange(Sender: TObject);
    procedure cbxTempoChange(Sender: TObject);
    procedure MIDIKeysNoteOn(Sender: TObject; var note,
      velocity: Integer);
    procedure MIDIKeysNoteOff(Sender: TObject; var note,
      velocity: Integer);
    procedure N15Click(Sender: TObject);

  private
    { Private declarations }
```

```
    PlayingNote:integer;
  public
    { Public declarations }
   // procedure Notify (tp : TActiveFormNotify); override;
  end;
var
  MidiPort,MidiStatus:Integer;  //Порт MIDI и статус его открытия
  instr_indexes:array of byte; //индексы используемых инструментов MIDI
  //количество точек при построении графика звуковой волны
  pcount:integer =250;
  UsedWindow:byte=2; //индекс используемой оконной функции
  FCount:Integer       = 2*4096; //количество точек для преобразования
  //количество точек для преобразования минус 1
  FCount_1:Integer     = 2*4096-1;
  FCountDiv2:Integer   = (2*4096 div 2); //половина точек для преобразования
  //половина точек для преобразования минус 1
  FCountDiv2_1:Integer = (2*4096 div 2)-1;
  norm:Double      =  1/(2*4096); // нормирующий множитель амплитуды
  MaxAmplitude:Double;  //максимальная амплитуда среди гармоник
  Eps:Double;    //минимальная учитываемая амплитуда гармоники
  UpdateTrack:Boolean=true;  //флаг обновления ползунка проигрывателя
  AnalyzeComplete:Boolean=false; //флаг проведения спетроанализа
  WaveLoaded:Boolean=false;   //флаг получения WAV-файла
  MidiCreated:Boolean=false;  //флаг создания MIDI-данных
  MidiSaved:Boolean=false;   //флаг сохранения MIDI-данных
  Form1: TForm1;       //форма программы
  WavPCM      : TPCMWaveFile;  //WAV-файл
  Fourier     : array of TCmxArray;  //массив гармоник


implementation
{$R *.dfm}
uses cmpMidiIterator,unitMidiGlobals,UnitSettings;
procedure TForm1.UpdateOptions;
var midimsg:integer;
var pkey:char;
begin
  if Form2.NeedUpdate then begin
    btnAnalyzeclick(btnAnalyze);
    btnFindNotesclick(btnFindNotes);
    clbSpectumSelect(clbSpectum);
    clbWaveformChange(clbWaveform);
    update_instruments;
    cbxInstrumentsSelect(cbxInstruments);
    cbxFFTCountChange(cbxFFTCount);
    chkAutosaveClick(chkAutosave);
    pkey:=#13;
    edtWavPointsKeyPress(edtWavPoints,pkey);
  end;
  if MidiStatus=MMSYSERR_NOERROR then begin
  Midimsg:=$C0 + (instr_indexes [ cbxInstruments.ItemIndex] *$100);
  midiOutShortMsg (midiport, midimsg);
end;
end;
procedure TMyForm.update_instruments;
var i:byte;  temp:integer;
begin
  temp:=cbxInstruments.ItemIndex;
  cbxInstruments.Items.Clear;
```

```
  setlength(instr_indexes,0);
  for i:=0 to 127 do
    if i in instr then begin
       setlength(instr_indexes,length(instr_indexes)+1);
       instr_indexes[high( instr_indexes)]:=i;
       cbxInstruments.Items.Add(instruments[i]);
    end;
  if temp<cbxInstruments.Items.Count then
    cbxInstruments.ItemIndex:=temp
  else
    cbxInstruments.ItemIndex:=0;
end;

procedure TForm1.UpdateWaveForm;
var key:char;
begin
  key:=#13;
  edtWavPointsKeyPress(edtWavPoints,key);
end;
procedure TForm1.N12Click(Sender: TObject);
var settings:TMemIniFile;   pkey:char;
begin
if odgIniOpen.Execute then begin
  settings:=TMemIniFile.Create(odgIniOpen.FileName);
  LoadSettings(form1,settings);
  FreeAndNil(settings);
  btnAnalyzeclick(btnAnalyze);
    btnFindNotesclick(btnFindNotes);
    clbSpectumSelect(clbSpectum);
    clbWaveformChange(clbWaveform);
    cbxInstrumentsSelect(cbxInstruments);
    chkAutosaveClick(chkAutosave);
    pkey:=#13;
    edtWavPointsKeyPress(edtWavPoints,pkey);
end;
end;
procedure TForm1.FreeMidi;
begin
  FreeAndNil(MidiData);
  PianoRoll.MidiData:=nil;
  PianoRoll.Repaint;
  MidiCreated:=false;
  MidiSaved:=false;
end;

procedure TForm1.FreeWAV;
begin
if WavPCM<>NIL then
    FreeAndNil(WavPCM);
WaveLoaded:=False;
    UpdateTrack:=false;
    trkMPlayer.Max:=0;
    label11.Caption:='Статус: N/A';
    label4.Caption:='Разрядность: N/A';
    label3.Caption:='Каналы: N/A';
    label2.Caption:='Данные: N/A';
    label11.Font.Color:=clBlack;
    label2.Font.Color:=clBlack;
    label3.Font.Color:=clBlack;
```

```
    label4.Font.Color:=clBlack;
    btnAnalyze.Kind:=bkCancel; btnAnalyze.Enabled:=False;
    btnAnalyze.Caption:='Спектроанализ';
    btnAnalyze.Cancel:=false;
end;

procedure TForm1.FreeFourier;
var i:cardinal;
begin
  MaxAmplitude:=0;
  label5.Caption:='0';
  trkMinAmp.Min:=0;
  Label18.Caption:='0';
  Series1.Clear;
AnalyzeComplete:=false;
    btnFindNotes.Enabled:=false;
    btnFindNotes.Kind:=bkCancel;
    btnFindNotes.Caption:='Поиск нот';
    btnFindNotes.Cancel:=false;
if length(fourier)<>0 then
    for i:=0 to high(fourier) do
      fourier[i]:=nil;
  fourier:=nil;
  trkSpectrum.Max:=0;;
end;
procedure TForm1.OpenWAVE(const filename:string);
begin
 stWavName.Caption:=filename;
  if MediaPlayer.filename<>'' then begin
    MediaPlayer.Close;
    MediaPlayer.FileName:='';
  end;
  FreeFourier;
  FreeWAV;
  WavPCM:= TPCMWaveFile.Create;
  WavPCM.LoadFromFile(filename);
   if WavPCM.IdError<>NoError then
       begin
         Label2.caption:='Данные: Не RIFF WAV';
         Label2.Font.Color:=clRed;
       end
   else
     begin
         Label2.caption:='Данные: RIFF WAV';
         Label2.Font.Color:=clGreen;
   label3.Caption:='Каналы: '+inttostr(WavPCM.Channels);
   if WavPCM.Channels<>1 then
          Label3.Font.Color:=clRed
   else
          Label3.Font.Color:=clGreen;

   label4.Caption:='Разрядность: '+ inttostr(WavPCM.BitsPerSample);
   if WavPCM.BitsPerSample<>16 then
       Label4.Font.Color:=clRed
   else
       Label4.Font.Color:=clGreen;
   end;
   if (label2.Font.Color=clGreen)and (label3.Font.Color=clGreen)
      and (label4.Font.Color=clGreen)
```

```
   then begin
      WaveLoaded:=True;
      label11.Caption:='Статус: подходит для анализа';
      label11.Font.Color:=clGreen;
      Label7.Caption:='/'+inttostr(wavpcm.Samples.Size div 2);
      btnAnalyze.Kind:=bkOK; btnAnalyze.Enabled:=True;
      btnAnalyze.Caption:='Спектроанализ';
      btnAnalyze.Default:=false;
      MediaPlayer.FileName:=filename;
      MediaPlayer.Open;
      if chkWaveform.Checked then
        UpdateWaveform;
   end else begin
    WaveLoaded:=False;
    label11.Caption:='Статус: не подходит для анализа';
    label11.Font.Color:=clRed;
   end;
end;

procedure TForm1.DrawWaveform(pcount:cardinal);
var i,h:cardinal; sample:smallint; pos:cardinal;
begin
  if WaveLoaded and (pcount>1) then begin
  Series2.SeriesColor:=clbWaveform.Selected;
  Series2.Clear;
  pos:=WavPCM.Samples.Position;
  WavPcm.Samples.Seek(0,soFromBeginning);
  i:=1;
  h:=(WavPCM.Samples.Size)div (2*(pcount-1));
  if h<=1 then h:=1 else h:=h-1;
  while i+h<=(WavPCM.Samples.Size)div 2 do begin
    Wavpcm.Samples.Read(sample,2);
    WavPcm.Samples.Seek((h-1)*2,soCurrent);
    Series2.AddXY((i-1)/3600/24/WavPCM.SamplesPerSec, sample );
    i:=i+h;
  end;
  Wavpcm.Samples.Seek(-1,soEnd);
  Wavpcm.Samples.Read(sample,2);
    Series2.AddXY((i-1)/3600/24/WavPCM.SamplesPerSec, sample );
  WavPCM.Samples.Seek(pos,soFromBeginning);
  end;
  chtWaveform.Refresh;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var i:integer;
begin
  FreeandNil(WavPCM);
  for i:=0 to high(Fourier) do
    fourier[i]:=nil;
  fourier:=nil;

end;

procedure TForm1.trkSpectrumChange(Sender: TObject);
var i:cardinal;
begin
  if AnalyzeComplete then begin
    Series1.Clear;
```

```
    for i:=0 to FCountdiv2-1 do
      Series1.AddXY(i*WavPCM.SamplesPerSec/FCount_1,
      fourier[trkSpectrum.Position][i].Re);
   end;
end;

procedure TForm1.trkMinAmpChange(Sender: TObject);
begin
  if trkMinAmp.Min<>0 then
   Label18.Caption:=inttostr(round(trkMinAmp.Position/
                                   trkMinAmp.Min*MaxAmplitude));
end;

procedure TForm1.FormCreate(Sender: TObject);
var Settings:TMemIniFile;
begin
MIDISTATUS:=midiOutOpen(@MIDIPort,MIDI_MAPPER,0,0,0);
MIDIKeys.MIDIPort:=Midiport;
MIDIKeys.MIDIPortOk:=Midistatus=MMSYSERR_NOERROR;
MediaPlayer.TimeFormat:=tfMilliseconds;
MediaPlayer.Notify:=true;
PlayingNote:=-1;
MIDIKeys.BaseOctave:=11-PianoRoll.VertScrollBar.Position;

 PianoRoll.MidiData:=MidiData;
 Settings:=TMemIniFile.Create('config.ini');
 LoadSettings(form1,settings);
 update_instruments;
end;
procedure TForm1.PianoRollFocus(Sender: TObject);
var
 noteOnEvent : PMidiEventData;
begin
  inherited;
  with PianoRoll do
    GetFocusedNote (noteOnEvent);

  if Assigned (noteOnEvent) and Assigned (noteOnEvent.OnOffEvent) then
  begin
    stFocusedNote.Caption := GetNoteName (noteonevent.data.b2)

  end
  else
  begin
    stFocusedNote.Caption := '---';
  end
end;
procedure TForm1.WAV1Click(Sender: TObject);
begin
 if odgWaveOpen.Execute then
 begin
    openWaVe(odgWaveOpen.FileName);
 end;
end;

procedure TForm1.PianoRollScroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
inherited;
```

```
  MIDIKeys.BaseOctave:=11-ScrollPos;
end;


procedure TForm1.PianoRollMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if PlayingNote<>-1 then begin
    MIDIKeys.ReleaseNote (PlayingNote, 0, True);
    PlayingNote:=-1;
  end;
end;

procedure TForm1.PianoRollMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var   noteOnEvent : PMidiEventData;
begin
  with PianoRoll do if Assigned(MidiData) then  begin
    GetFocusedNote(noteonEvent);
    if noteonevent<>NIL then begin
      MIDIKeys.PressNote (Noteonevent^.data.b2,Noteonevent^.data.b3, True);
      playingNote:= Noteonevent^.data.b2;
    end;
  end;
end;

procedure TForm1.MediaPlayerNotify(Sender: TObject);
begin
  MediaPlayer.Notify:=true;
  MediaPlayer.AutoEnable:=True;
  if MediaPlayer.Mode=mpPlaying then begin
    trkMPlayer.Max:=MediaPlayer.Length;
    UpdateTrack:=false;
    trkMPlayer.Position:=MediaPlayer.Position;
    if chkListSpectr.Checked And AnalyzeComplete then
      trkSpectrum.Position:=round(WavPCM.SamplesPerSec/
                                  1000*MediaPlayer.Position)
                                    div FCount;
    if MediaPlayer.Position=MediaPlayer.Length then begin
      MediaPlayer.Notify:=true;
      MediaPlayer.Stop;
    end else
      Timer1.Enabled:=true;
  end
  else begin
   if (MediaPlayer.Mode=mpPaused) and
        (MediaPlayer.NotifyValue=nvSuccessful) then begin
       MediaPlayer.AutoEnable:=False;
       MediaPlayer.EnabledButtons:=MediaPlayer.EnabledButtons+[btPlay];
   end;
   if  (MediaPlayer.Mode=mpStopped) then
    if (MediaPlayer.NotifyValue=nvSuccessful) then begin
      trkMPlayer.Position:=0;
    end else MediaPlayer.Notify:=false;
    Timer1.Enabled:=False;
   end;

end;
```

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  UpdateTrack:=false;
  if MediaPlayer.Mode=mpPlaying then begin

      trkMPlayer.Position:=MediaPlayer.Position;
    if chkListSpectr.Checked And AnalyzeComplete then
      trkSpectrum.Position:=round(WavPCM.SamplesPerSec/
                                  1000*MediaPlayer.Position)
                                 div FCount;
    Timer1.Enabled:=true;
  end else
  if MediaPlayer.Mode=mpStopped then begin
      UpdateTrack:=False;
      trkMPlayer.Position:=0;
      if chkListSpectr.Checked And AnalyzeComplete then
        trkSpectrum.Position:=0;
   end;

end;

procedure TForm1.MediaPlayerPostClick(Sender: TObject;
  Button: TMPBtnType);
begin
  MediaPlayer.AutoEnable:=true;
  if Button=btPlay then begin

    trkMPlayer.Max:=MediaPlayer.Length;
    UpdateTrack:=false;
    MediaPlayer.Position:=trkMPlayer.Position;
    if chkListSpectr.Checked And AnalyzeComplete then
      trkSpectrum.Position:=round(WavPCM.SamplesPerSec/
                                  1000*MediaPlayer.Position)
                                 div FCount;
    MediaPlayer.Notify:=true;
    Timer1.Enabled:=true;
    MediaPlayer.Play;
  end else begin
    if button=btStop then begin
      MediaPlayer.Stop;
      MediaPlayer.Position:=0;
      UpdateTrack:=false;
      trkMPlayer.Position:=0;
      if chkListSpectr.Checked And AnalyzeComplete then
        trkSpectrum.Position:=0;
    end;
    If button=btStep then
      trkMPlayer.Position:=trkMPlayer.Position+trkMPlayer.Max div 100;
    if button=btBack then
      trkMPlayer.Position:=trkMPlayer.Position-trkMPlayer.Max div 100;
    Timer1.Enabled:=False;
  end;
end;

procedure TForm1.btnWavSelClick(Sender: TObject);
begin
 if odgWaveOpen.Execute then
 begin
    openWaVe(odgWaveOpen.FileName);
```

```
 end;
end;

procedure TForm1.chkWaveformClick(Sender: TObject);
begin
  If chkWaveform.Checked then
    DrawWaveform(pcount)
  else
    Series2.Clear;
end;

procedure TForm1.clbWaveformChange(Sender: TObject);
begin
  Series2.SeriesColor:=clbWaveform.Selected;
end;

procedure TForm1.trkMPlayerChange(Sender: TObject);
begin
  if waveloaded and updateTrack then begin
    MediaPlayer.PauseOnly;
    //MediaPlayer.Position:=trkMPlayer.Position;
    //MediaPlayer.Play;
  end else updatetrack:=true;
end;

procedure TForm1.btnAnalyzeClick(Sender: TObject);
var i,t:cardinal;
  TempSamples  : T16BitsPerSample;
  ExpTable    : TCmxArray;
  IndexTable : TIndexArray;
  LeftAxis:TChartAxis;
  pos:int64;
begin
IndexTable:=nil;ExpTable:=nil;TempSamples:=nil;
if WaveLoaded then begin
try
  t:=0;
  FreeFourier;
  SetLength(TempSamples,FCount);
  ExpTable:=GetFFTExpTable(FCount);
  IndexTable:=GetArrayIndex(FCount);
  pos:=WavPCM.Samples.Position;
  WavPcm.Samples.Seek(0,soBeginning);
  While  (WavPCM.Samples.Size- WavPCM.Samples.Position)>FCount*2  do
     begin
         WavPCM.Samples.ReadBuffer(TempSamples[0],FCount*2);
          setlength(fourier,t+1);
          setlength(fourier[t],FCount);
          for I:=0 to FCount_1 do begin
              UsedWindow:= 0;
              case rgWindowFuncs.ItemIndex of
              1:TempSamples[i]:=round(HannWindow(I,FCount)*TempSamples[i]);
              2:TempSamples[i]:=round(HammingWindow(I,FCount)*TempSamples[i]);
              3:TempSamples[i]:=round(BlackmanWindow(I,FCount)*TempSamples[i]);
              else TempSamples[i]:=round(TempSamples[i]);
              end;

              fourier[t][I].Re:=TempSamples[IndexTable[I]];
              fourier[t][I].Im:=0;
```

```
          end;
          FFT(fourier[t],ExpTable);
          fourier[t][0].Re:=0;
          for i:=1 to FCountdiv2_1 do begin
              fourier[t][i].Re:=sqrt(fourier[t][i].Re*fourier[t][i].Re+
                                     fourier[t][i].Im*fourier[t][i].Im)*norm*2;
              if maxAmplitude<fourier[t][i].Re then
                maxAmplitude:=fourier[t][i].Re;
          end;
          for i:=FCountdiv2 to FCount_1 do
              fourier[t][i].Re:=0;
          inc(t);
        end;
      WavPCM.Samples.Seek(pos,soBeginning);
      ExpTable    :=Nil;
      IndexTable :=Nil;
      TempSamples  :=Nil;
      AnalyzeComplete:=True;
      btnFindNotes.Enabled:=true;
      btnFindNotes.Kind:=bkOK;
      btnFindNotes.Caption:='Поиск нот';
      btnFindNotes.Default:=false;
      leftaxis:=Series1.GetVertAxis;
      leftaxis.Maximum:=MaxAmplitude;
      label5.Caption:=inttostr(round(MaxAmplitude));
      if round(maxAmplitude)>maxint then
        trkMinAmp.Min:=-maxint
      else
        trkMinAmp.Min:=-round(MaxAmplitude);
      trkMinAmp.Position:=round(0.8*trkMinAmp.Min);
      trkSpectrum.Max:=High(fourier);
      trkSpectrumChange(self);
    except else
      AnalyzeComplete:=false;
      FreeFourier;
    end;
  end;
end;

procedure TForm1.btnFindNotesClick(Sender: TObject);
var i,j,k:integer;t:cardinal;
  channels:array of byte;
  data:TEventData;  note:byte;
  lg,ppqn:integer;
  tempo,ft:cardinal; head:array of byte; added:boolean;
begin
if analyzecomplete then try
  if MidiCreated and not MidiSaved then
    if MessageDlg('Изменения не сохранены! Продолжить?',
                      mtWarning,mbOKCancel,0)<>1 then
      exit;
  t:=0;
  channels:=nil;

    FreeMidi;
    MidiData:=TMidiData.Create(Form1);
    MidiData.New;

  eps:=trkMinAmp.Position/trkMinAmp.Min*MaxAmplitude;
```

```
if MidiData.AddNewTrack(0)then  begin
  MidiData.Tracks[0].BeginUpdate;
  setlength(head,3);
  tempo:=round(60000000/tempos[cbxTempo.ItemIndex+1]);
  head[0]:=(tempo div $10000) mod $100;head[1]:=(tempo div $100) mod $100;
  head[2]:=tempo mod $100;
  tempo:=tempo div 1000;

  MidiData.Tracks[0].InsertMetaEvent(0,$51,pchar(head),3);
  lg:=trunc(log10(tempo/FCount*WavPCM.SamplesPerSec))-2;
  if abs(lg)<5 then begin
      PPQN:=trunc(tempo/(FCount_1/WavPCM.SamplesPerSec)*power(10,1-lg));
      MidiData.PPQN:=ppqn;
      ft:=round(power(10,4-lg));
  end else ft:=180;
  if length(fourier)>0 then
  for j:=-4 to 6 do   begin
       for k:=1 to 12 do
         if abs(fourier[t][round(notes_freq[k]*power(2,j-1)*FCount_1/
                   WavPCM.SamplesPerSec)].Re)>=eps then begin
              i:=length(channels);
              setlength(channels,i+1);
              data.status:=$90; data.b2:=(j+4)*12+(k-1); data.b3:=127;
              channels[i]:=data.b2;
              MidiData.Tracks[0].InsertEvent(t*ft,data,0);
            end;
  end;
  for t:=1 to high(fourier) do begin
     for j:=0 to high(channels) do
       if channels[j]<>$FF then begin
        note:=channels[j];
        if abs(fourier[t][round(notes_freq[(note mod 12)+1]*
                power(2,(note div 12) -5)*FCount_1/
                    WavPCM.SamplesPerSec)].Re)<eps then
            begin
              data.status:=$80; data.b2:=note; data.b3:=127;
              channels[j]:=$FF;
              MidiData.Tracks[0].InsertEvent(t*ft,data,0);
            end;
       end;
     for j:=-4 to 6 do
       for k:=1 to 12 do
         if abs(fourier[t][round(notes_freq[k]*power(2,j-1)*FCount_1/
                   WavPCM.SamplesPerSec)].Re)>=eps then begin
            added:=false;
            for i:=0 to high(channels) do
             if (channels[i]=$FF) or (channels[i]=(j+4)*12+(k-1)) then
              begin
              added:=true;
               if (channels[i]=(j+4)*12+(k-1)) then break;
              data.status:=$90; data.b2:=(j+4)*12+(k-1); data.b3:=127;
              channels[i]:=data.b2;
              MidiData.Tracks[0].InsertEvent(t*ft,data,0);
               break
             end;
            if not added then begin
              i:=length(channels);
              setlength(channels,i+1);
              data.status:=$90; data.b2:=(j+4)*12+(k-1); data.b3:=127;
```

```
              channels[i]:=data.b2;
              MidiData.Tracks[0].InsertEvent(t*ft,data,0);
            end;
      end;
  end;
  for j:=0 to high(channels) do
    if channels[j]<>$ff then begin
        note:=channels[j];
        data.status:=$80; data.b2:=note; data.b3:=127;
        channels[j]:=$FF;
        MidiData.Tracks[0].InsertEvent(high(fourier)*ft,data,0);
  end;
  channels:=nil;
  MidiData.Tracks[0].SetPatch(cbxInstruments.ItemIndex);
  MidiData.Tracks[0].EndUpdate;
  MidiCreated:=true;
  PianoRoll.MidiData:=MidiData;
  PianoRoll.Refresh;
  if chkAutosave.Checked and btnSaveMidi.Enabled then
   try
    MidiData.FileName:=stMidiName.Caption;
    MidiData.Save;
    MidiSaved:=true;
   except else
      MidiSaved:=false;
   end;
end;
except else
  channels:=nil;
  MidiSaved:=false;
  MidiCreated:=false;
  FreeMidi;
end;
end;

procedure TForm1.btnMidiSelClick(Sender: TObject);
begin
if sdgMidiSave.Execute then
 begin
    stMidiName.Caption:=sdgMidiSave.FileName;
    btnSaveMidi.Enabled:=true;
    if midicreated and not midisaved then
    try
      MidiData.FileName:=stMidiName.Caption;
      MidiData.Save;
      MidiSaved:=true;
    except else MidiSaved:=false;
    end;
 end;
end;

procedure TForm1.btnSaveMidiClick(Sender: TObject);
begin
  if midicreated then
  try
  MidiData.FileName:=stMidiName.Caption;
  MidiData.Save;
  MidiSaved:=true;
  except else MidiSaved:=false;
```

```
  end
  else MessageDlg('Сначала проведите анализ и поиск нот!',
              mtInformation,[mbok],0);
end;

procedure TForm1.cbxInstrumentsSelect(Sender: TObject);
var midimsg:integer;
begin
if MidiStatus=MMSYSERR_NOERROR then begin
  Midimsg:=$C0 + (instr_indexes [ cbxInstruments.ItemIndex] *$100);
  midiOutShortMsg (midiport, midimsg);
  end;
if AnalyzeComplete and MidiCreated then begin
  MidiData.Tracks[0].BeginUpdate;
   MidiData.Tracks[0].SetPatch(instr_indexes[cbxInstruments.ItemIndex]);
  MidiData.Tracks[0].EndUpdate;
  PianoRoll.Update;
  PianoRoll.Repaint;
end;
end;

procedure TForm1.chkAutosaveClick(Sender: TObject);
begin
  if chkAutosave.Checked then
    btnSaveMidi.Visible:=false
  else
    btnSaveMidi.Visible:=true;
end;

procedure TForm1.cbxFFTCountChange(Sender: TObject);
var x:integer;
begin
  x:= 1 shl (cbxFFTCount.ItemIndex+10) ;
  if fcount<>x then begin

    FCount:= x;
    FCount_1:= FCount-1;
    FCountDiv2:= (FCount div 2);
    FCountDiv2_1:= (FCount div 2)-1;
    norm:=1/Fcount;

    if AnalyzeComplete then
      btnAnalyzeClick(btnAnalyze);
  end;
end;

procedure TForm1.clbSpectumSelect(Sender: TObject);
begin
  Series1.SeriesColor:=clbSpectum.Selected;
end;

procedure TForm1.rgWindowFuncsClick(Sender: TObject);
begin
  if (UsedWindow<>rgWindowFuncs.ItemIndex) and AnalyzeComplete then
    btnAnalyzeClick(btnAnalyze);
end;

procedure TForm1.edtWavPointsKeyPress(Sender: TObject; var Key: Char);
var x:integer;
```

```
begin
  x:=pcount;
  if (key<>#8) then
  try
    pcount:=strtoint(edtWavPoints.Text);
    if WaveLoaded then begin
      if pcount>wavpcm.Samples.size div 2 then begin
        pcount:=wavpcm.Samples.size div 2;
        edtWavPoints.text:=inttostr(pcount);
      end;
    if {(pcount<>x) and }chkWaveform.Checked then
      DrawWaveform(pcount);
    end;
  except else
    pcount:=x;
    edtWavPoints.text:=inttostr(x);
    edtWavPoints.SelStart:=length(edtWavPoints.Text);
  end;

end;

procedure TForm1.MIDI1Click(Sender: TObject);
begin
if midicreated then begin
if sdgMidiSave.Execute and midiCreated then
 if (stMidiName.Caption<>sdgMidiSave.FileName) or not MidiSaved then begin
    if stMidiName.Caption<>sdgMidiSave.FileName then
      stMidiName.Caption:=sdgMidiSave.FileName;
    btnSaveMidi.Enabled:=true;
  try
  MidiData.FileName:=stMidiName.Caption;
  MidiData.Save;
  MidiSaved:=true;
  except else MidiSaved:=false;
  end;
 end
end else MessageDlg('Сначала проведите анализ!',mtInformation,[mbok],0);
end;

procedure TForm1.MIDI2Click(Sender: TObject);
begin
if midicreated then begin
if sdgMidiSave.Execute and midiCreated then
  try
  MidiData.FileName:=sdgMidiSave.FileName;
  MidiData.Save;
  MidiSaved:=true;
  except else MidiSaved:=false;
  end;
end else MessageDlg('Сначала проведите анализ!',mtInformation,[mbok],0);
end;

procedure TForm1.N6Click(Sender: TObject);
begin
  FreeAndNil(Form1);
  Application.Terminate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
```

```
begin
  FreeWav;
  FreeFourier;
  FreeMidi;
  instr_indexes:=nil;
  midioutClose(MidiPort);
end;

procedure TForm1.N7Click(Sender: TObject);
begin
  Form2.PageControl1.TabIndex:=0;
  Form2.ShowModal;
  UpdateOptions;
end;

procedure TForm1.N8Click(Sender: TObject);

begin
  Form2.PageControl1.TabIndex:=1;
  Form2.ShowModal;
  UpdateOptions;
end;

procedure TForm1.N10Click(Sender: TObject);
var settings:TMemIniFile;
begin
if sdgIniSave.Execute then begin
  settings:=TMemIniFile.Create(sdgIniSave.FileName);
  SaveSettings(form1,settings);
  settings.UpdateFile;
  FreeAndNil(settings);
end;
end;

procedure TForm1.N11Click(Sender: TObject);
var settings:TMemIniFile;
begin
if sdgIniSave.Execute then begin
  settings:=TMemIniFile.Create(sdgIniSave.FileName);
  SaveSettings(form1,settings);
  settings.UpdateFile;
  FreeAndNil(settings);
end;
end;

procedure TForm1.N13Click(Sender: TObject);
begin
Application.HelpCommand(HELP_FINDER, 0);
end;

procedure TForm1.MIDI3Click(Sender: TObject);
begin
Form2.PageControl1.TabIndex:=2;
  Form2.ShowModal;
  UpdateOptions;
end;

procedure TForm1.edtWavPointsExit(Sender: TObject);
var key:char;
```

```
begin
  key:=#13;
  edtWavPointsKeyPress(edtWavPoints,key);
end;

procedure TForm1.cbxInstrumentsChange(Sender: TObject);
begin
 cbxInstrumentsSelect(Sender);
end;

procedure TForm1.cbxTempoChange(Sender: TObject);
begin
  if MidiCreated then
    btnFindNotesClick(btnFindNotes);
end;




procedure TForm1.MIDIKeysNoteOn(Sender: TObject; var note,
  velocity: Integer);
begin
  stFocusedNote.Caption:=GetNoteName(note);
end;

procedure TForm1.MIDIKeysNoteOff(Sender: TObject; var note,
  velocity: Integer);
begin
  stFocusedNote.Caption:='';
end;

procedure TForm1.N15Click(Sender: TObject);
begin
  AboutDlg.ShowModal;
end;

end.
```

Ниже представлен текст модуля управления настройками, написанного на языке Delphi 7:

```
unit UnitSettings;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms,
  Dialogs, Grids, StdCtrls, ExtCtrls, ComCtrls,UnitMain,math,IniFiles,
  mmsystem;

type
  TForm2 = class(TMyForm)
    PageControl1: TPageControl;//страницы настроек
    TabSheet1: TTabSheet;   //страница 1
    TabSheet2: TTabSheet;  //страница 2
```

```
    Button1: TButton;     //
    Button2: TButton;     //
    Label2: TLabel;       //
    StringGrid1: TStringGrid; //
    GroupBox1: TGroupBox;     //
    Label13: TLabel;        //
    Label14: TLabel;       //
    Bevel2: TBevel;         //
    GroupBox2: TGroupBox;    //
    Label15: TLabel;       //
    GroupBox3: TGroupBox;   //
    Label9: TLabel;        //
    Bevel1: TBevel;        //
    Label1: TLabel;        //
    Button3: TButton;       //
    Button4: TButton;       //
    TabSheet3: TTabSheet;   //
    ListBox1: TListBox;     //
    ListBox2: TListBox;     //
    Button5: TButton;       //
    Button6: TButton;       //
    Label3: TLabel;        //
    Label4: TLabel;         //
    ComboBox3: TComboBox;    //
    ComboBox4: TComboBox;    //
    ComboBox5: TComboBox;    //
    Button7: TButton;      //
    Label5: TLabel;       //

    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure TabSheet2Show(Sender: TObject);
    procedure StringGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
      var CanSelect: Boolean);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure StringGrid1SetEditText(Sender: TObject; ACol, ARow: Integer;
      const Value: String);
    procedure TabSheet1Show(Sender: TObject);
    procedure update_lists;
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure TabSheet3Show(Sender: TObject);
    procedure Button7MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Button7MouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormDestroy(Sender: TObject);
    procedure ComboBox3Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    NeedUpdate:Boolean;
  end;

var
```

```
  Form2: TForm2;
  PlayingNote:integer=-1;
procedure LoadSettings(form:tmyform;settings:TMemIniFile);
procedure SaveSettings(form:tmyform;settings:TMemIniFile);

implementation
var list1,list2:array of byte;
{$R *.dfm}
procedure TForm2.update_lists;
var i:byte;
begin
  ListBox1.Items.Clear;
  ListBox2.Items.Clear;
  setlength(list1,0);
  setlength(list2,0);
  for i:=0 to 127 do
    if i in instr then begin
       setlength(list2,length(list2)+1);
       list2[high(list2)]:=i;
       ListBox2.Items.Add(instruments[i]);
    end else begin
       setlength(list1,length(list1)+1);
       list1[high(list1)]:=i;
       ListBox1.Items.Add(instruments[i]);
    end;
end;
procedure SaveDefSettings(form:tmyform;settings:TMemIniFile);
begin

  settings.WriteInteger('Default','cbxFFTCount',
      Form.cbxFFTCount.ItemIndex);
  settings.WriteInteger('Default','cbxTempo',
      Form.cbxTempo.ItemIndex);
  settings.WriteInteger('Default','rgWindowFuncs',
      Form.rgWindowFuncs.ItemIndex);
  settings.WriteInteger('Default','clbSpectum',
      Form.clbSpectum.Selected);
  settings.WriteInteger('Default','chkAutosave',
      integer(Form.chkAutosave.Checked));
  settings.WriteInteger('Default','cbxInstruments',
      Form.cbxInstruments.ItemIndex);
  settings.WriteInteger('Default','chkWaveform',
      integer(Form.chkWaveform.Checked));
  settings.WriteInteger('Default','clbWaveform',
      form.clbWaveform.Selected);
  try
  settings.WriteInteger('Default','edtWavPoints',strtoint(Form.edtWavPoints.Text));;
  except else
    MessageDlg('Неправильное количество точек!'+
                #13#10+'Значение проигнорировано.',mtWarning,[mbok],0);
  end;
  settings.WriteInteger('Default','chkListSpectr',
      integer(Form.chkListSpectr.Checked));
end;
procedure LoadDefSettings(form:tmyform;settings:TMemIniFile);
begin

  Form.cbxFFTCount.ItemIndex:=
    settings.ReadInteger('Default','cbxFFTCount',3);
```

```
  Form.cbxTempo.ItemIndex:=
    settings.ReadInteger('Default','cbxTempo',3);
  Form.rgWindowFuncs.ItemIndex:=
    settings.ReadInteger('Default','rgWindowFuncs',2);
  Form.clbSpectum.Selected:=
    settings.ReadInteger('Default','clbSpectum',clRed);
  Form.chkAutosave.Checked:=
    boolean(settings.ReadInteger('Default','chkAutosave',1));
  Form.cbxInstruments.ItemIndex:=
    settings.ReadInteger('Default','cbxInstruments',0);
  Form.chkWaveform.Checked:=
    boolean(settings.ReadInteger('Default','chkWaveform',1));
  form.clbWaveform.Selected:=
    settings.ReadInteger('Default','clbWaveform',clblue);
  Form.edtWavPoints.Text:=
    inttostr(settings.ReadInteger('Default','edtWavPoints',250));
  Form.chkListSpectr.Checked:=
    boolean(settings.ReadInteger('Default','chkListSpectr',1));

end;

procedure LoadFreqSettings(form:tmyform;settings:TMemIniFile);
var i:byte;
begin
  for i:=1 to 12 do
    form.notes_freq[i]:=settings.ReadFloat('Frequencies',notes_name[i],
                        default_freq[i]);
end;

procedure SaveFreqSettings(form:tmyform;settings:TMemIniFile);
var i:byte;
begin
  for i:=1 to 12 do
    settings.WriteFloat('Frequencies',notes_name[i],form.notes_freq[i]);
end;

procedure SaveInstrSettings(form:tmyform;settings:TMemIniFile);
type pinteger=^integer;
var i:byte; p:pinteger;
begin
p:=addr(form.instr);
for i:=1 to 4 do begin
   settings.WriteInteger('Instruments','Part'+inttostr(i),p^);
   inc(p);
end;
end;

procedure LoadInstrSettings(form:tmyform;settings:TMemIniFile);
type pinteger=^integer;
var i:byte; p:^integer;
begin
p:=addr(form.instr);
for i:=1 to 4 do begin
   p^:=settings.ReadInteger('Instruments','Part'+inttostr(i),-1);
   inc(p);
end;
end;

procedure LoadSettings(form:tmyform;settings:TMemIniFile);
```

```
begin
   LoadDefSettings(form,Settings);
   LoadFreqSettings(form,Settings);
   LoadInstrSettings(form,Settings);
end;
procedure SaveSettings(form:tmyform;settings:TMemIniFile);
begin
  SaveDefSettings(form,settings);
  SaveFreqSettings(form,settings);
  SaveInstrSettings(form,settings);
end;
procedure TForm2.Button3Click(Sender: TObject);
var settings:TMemIniFile;
begin
  settings:=TMemIniFile.Create('config.ini');
  case PageControl1.TabIndex of
  0:begin
    SaveDefSettings(form1,settings);
    LoadDefSettings(form2,settings);
    TabSheet1Show(TabSheet1);
  end;
  1: begin
    SaveFreqSettings(form1,settings);
    LoadFreqSettings(form2,settings);
    TabSheet2Show(TabSheet2);
  end;
  2:
  begin
    SaveInstrSettings(form1,settings);
    LoadInstrSettings(form2,settings);
    TabSheet3Show(TabSheet3);
  end;
  end;
  FreeAndNil(settings);

end;

procedure TForm2.Button4Click(Sender: TObject);
var settings:TMemIniFile;
begin
  settings:=TMemIniFile.Create('config.ini');
  case PageControl1.TabIndex of
  0:begin
    SaveDefSettings(form2,settings);
    LoadDefSettings(form1,settings);
    end;
  1: begin
    SaveFreqSettings(form2,settings);
    LoadFreqSettings(form1,settings);
  end;
  2: begin
    SaveInstrSettings(form2,settings);
    LoadInstrSettings(form1,settings);
  end;
  end;
  FreeAndNil(settings);
  self.NeedUpdate:=true;
end;
```

```
procedure TForm2.FormShow(Sender: TObject);
var settings:TMemIniFile;
begin
  instr:=[0];
  Self.NeedUpdate:=false;
  settings:=TMemIniFile.Create('config.ini');
  LoadSettings(form2,settings);
  TabSheet2Show(TabSheet2);
  TabSheet1Show(TabSheet1);
  TabSheet3Show(TabSheet3);
  FreeAndNil(settings);
  //midiStatus:=midiOutOpen(@MidiPort,MIDI_MAPPER,0,0,0);
end;

procedure TForm2.TabSheet2Show(Sender: TObject);
var i:byte; var canselect:boolean;
begin
 for i:=1 to 12 do begin
    StringGrid1.Cells[0,i-1]:=notes_name[i];
    StringGrid1.Cells[1,i-1]:=floattostr(notes_freq[i]);
 end;
  canselect:=true;
  StringGrid1SelectCell(StringGrid1,1,0,canselect);

end;

procedure TForm2.StringGrid1SelectCell(Sender: TObject; ACol,
  ARow: Integer; var CanSelect: Boolean);
var i:shortint;
begin
  Label2.Caption:='';
  for i:=-4 to 5 do begin
    label2.Caption:=label2.Caption+
        rus_names[Arow+1]+' '+octaves[i+5]+'октавы: '+
          floattostr(strtofloat(StringGrid1.Cells[1,Arow])*power(2,i))+#13#10;
  end;
end;

procedure TForm2.Button1Click(Sender: TObject);
var settings:TMemIniFile;
begin
  //self.NeedUpdate:=true;
  settings:=TMemIniFile.Create('config.ini');
  SaveSettings(form2,settings);
  settings.UpdateFile;
  //LoadSettings(form1,settings);
  FreeAndNil(settings);

  Self.Close;
end;

procedure TForm2.Button2Click(Sender: TObject);
var settings:TMemIniFile;
begin
  settings:=TMemIniFile.Create('config.ini');
  LoadSettings(form2,settings);
  FreeAndNil(settings);
  Self.Close;
end;
```

```
procedure TForm2.FormCreate(Sender: TObject);
var settings:TMemIniFile; i:byte;
begin
  Self.NeedUpdate:=false;
  settings:=TMemIniFile.Create('config.ini');
  LoadSettings(form2,settings);
  FreeAndNil(settings);

  for i:=0 to 127 do
    ComboBox3.Items.Add(instruments[i]);
  ComboBox3.ItemIndex:=0;
  for i:=1 to 12 do
    ComboBox4.Items.Add(rus_names[i]);
  for i:=1 to 10 do
    ComboBox5.Items.Add(octaves[i]+'октавы');
  ComboBox4.ItemIndex:=0; ComboBox5.ItemIndex:=0;
end;


procedure TForm2.StringGrid1SetEditText(Sender: TObject; ACol,
  ARow: Integer; const Value: String);
var temp:Double;  canselect:boolean;
begin
  try
    if value<>'' then begin
    temp:=strtofloat(Value);
    notes_freq[ARow+1]:=temp;
    canselect:=true;
    StringGrid1SelectCell(StringGrid1,acol,arow,canselect);
    end;
  except else
    ShowMessage('Ошибка!')
  end;
end;

procedure TForm2.TabSheet1Show(Sender: TObject);
begin
  update_instruments;
end;

procedure TForm2.Button5Click(Sender: TObject);
var i:shortint;
begin
  for i:=0 to ListBox1.Items.Count-1 do
    if ListBox1.Selected[i] then
      instr:=instr+[list1[i]];
  update_lists;
end;

procedure TForm2.Button6Click(Sender: TObject);
var i:shortint;
begin
  for i:=0 to ListBox2.Items.Count-1 do
    if ListBox2.Selected[i] then
      instr:=instr-[list2[i]];
  update_lists;
end;
```

```
procedure TForm2.TabSheet3Show(Sender: TObject);
begin
  update_lists;
end;

procedure TForm2.Button7MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var midimsg:record
        case Boolean of
         true:(bytes:array [1..4] of byte);
         false:(l:integer;)
        end;
begin
  if MidiStatus=MMSYSERR_NOERROR then begin
  PlayingNote:=ComboBox4.ItemIndex+ComboBox5.ItemIndex*12;
  Midimsg.bytes[1]:=$90;
  midimsg.bytes[2]:=PlayingNote;
  midimsg.bytes[3]:=127;
  midimsg.bytes[4]:=0;
  midiOutShortMsg (Form1.MIDIKeys.MIDIPort, midimsg.l);
end;
end;

procedure TForm2.Button7MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var midimsg:integer;
begin
  if PlayingNote<>-1 then begin
    midimsg:=$80+PlayingNote*$100;

    midiOutShortMsg(midiport,midimsg);
    PlayingNote:=-1;

  end;
end;

procedure TForm2.FormDestroy(Sender: TObject);
begin
  midiOutClose(MidiPort);
  list1:=nil;
  list2:=nil;
  instr_indexes:=nil;
end;

procedure TForm2.ComboBox3Change(Sender: TObject);
var midimsg:integer;
begin
  if MidiStatus=MMSYSERR_NOERROR then begin
  Midimsg:=$C0 + (ComboBox3.ItemIndex *$100);
  midiOutShortMsg (midiport, midimsg);
  end;
end;

end.
```

Далее представлен текст модуля, написанного на языке Delphi 7, содержащего объекты для работы с WAV-файлами:

```
unit ModuleWav;

{$H+}

interface

uses
  Classes, SysUtils;

Const
  NoError            = 0;
  ReadError          = 1;
  HeaderError        = 2;
  DataError          = 3;
  FileCorrupt        = 4;
  IncorectFileFormat = 5;
  FileDontFound      = 6;

  MaxBlock = 200;

 type
  T3Bytes = array [0..2] of Byte;

  T8BitsPerSample  = array of Byte;
  T16BitsPerSample = array of SmallInt;
  T24BitsPerSample = array of T3Bytes;
  T32BitsPerSample = array of Integer;

  //TAudioData = array of integer;

  TWaveHeaderChank = record
      wFormatTag     : Smallint;
      wChannels      : WORD;
      wSamplesPerSec : Cardinal;
      wAvgBytesPerSec: Cardinal;
      wBlockAlign    : WORD;
      wBitsPerSample : WORD;
      wcbSize        : WORD;
    end;


  { TPCMWaveFile }

  TPCMWaveFile = class
    private
      WaveFile       : TFileStream;
      FSamples        : TMemoryStream;

      F_8BitsSamples  :T8BitsPerSample;
      F_16BitsSamples :T16BitsPerSample;
      F_24BitsSamples :T24BitsPerSample;
      F_32BitsSamples :T32BitsPerSample;

      FAvgBytesPerSec: Cardinal;
      FCountFFTPoint: Word;

      FFileSize      : Cardinal;
      FSamplesPerSec : Cardinal;
      FBlockAlign    : Word;
```

```
      FChannels      : Word;
      FBitsPerSample : Word;
      FFormatTag     : Word;
      FIdError       : Word;
      procedure SetCountFFTPoint(const AValue: Word);

    public
      property _8BitsSamples  :T8BitsPerSample  read F_8BitsSamples;
      property _16BitsSamples :T16BitsPerSample  read F_16BitsSamples;
      {Массив сэмплов (16 бит)}
      property _24BitsSamples :T24BitsPerSample  read F_24BitsSamples;
      //Массив сэмплов (24 бит)
      property _32BitsSamples :T32BitsPerSample  read F_32BitsSamples;
      //Массив сэмплов (32 бит)

      property IdError       :Word     read FIdError;
      //ID ошибки при разборе файла
      property FileSize      :Cardinal read FFileSize;
      //Размер файла в байтах
      property FormatTag     :Word     read FFormatTag;
      // Категория формата
      property Channels      :Word     read FChannels;
      // Число каналов
      property BitsPerSample :Word     read FBitsPerSample;
      // Бит на сэмпл
      property SamplesPerSec :Cardinal read FSamplesPerSec;
      // Частота дискретизации
      property AvgBytesPerSec:Cardinal read FAvgBytesPerSec;
      // Байт в секунду
      property BlockAlign    :Word     read FBlockAlign;
      // Выравнивание данных в data-чанке

      property CountFFTPoint:Word read FCountFFTPoint write SetCountFFTPoint;
              //Число точек расчета в БПФ

      property Samples:TMemoryStream read  FSamples;

      function NextSamples:Boolean;
      //получения следующей порции точек из файла
      procedure LoadFromFile(Const FileName:String);
      constructor Create;
      destructor Destroy; override;
    end;


implementation

uses math;
{ TPCMWaveFile }

procedure TPCMWaveFile.SetCountFFTPoint(const AValue: Word);
begin
  if FCountFFTPoint=AValue then exit;
  FCountFFTPoint:=AValue;
end;

function TPCMWaveFile.NextSamples: Boolean;
var
   CountSamples:Int64; //Число семплов в массиве _8BitsSamples.._32BitsSamples
```

```
begin
  Result:=False;

    If  (WaveFile=Nil) or (FIdError<>0) then exit;

    //CountSamples – Вернет сколько осталось не обработанных байт
    CountSamples:=WaveFile.Size–WaveFile.Seek(0, soFromCurrent);

    //Переведем кол–во байт в число семплов
    CountSamples:=CountSamples div (BitsPerSample div 8);

    if CountSamples < CountFFTPoint*Channels then exit;
    //Конец файла, не стоит замарачивается

    CountSamples:=Min(CountSamples,MaxBlock*CountFFTPoint*Channels);


    Case BitsPerSample of
    1..8: begin
            SetLength(F_8BitsSamples,CountSamples);
            //Выделяем память под данные
            WaveFile.ReadBuffer(F_8BitsSamples[0],CountSamples);
            //Копируем данные в память
            Result:=True;
            Exit;
          end;
     9..16: begin
            SetLength(F_16BitsSamples,CountSamples);
            //Выделяем память под данные
            WaveFile.ReadBuffer(F_16BitsSamples[0],CountSamples*2);
            //Копируем данные в память
            Result:=True;
            Exit;
           end;
   17..24: begin
            SetLength(F_24BitsSamples,CountSamples);
            //Выделяем память под данные
            WaveFile.ReadBuffer(F_24BitsSamples[0],CountSamples*3);
            //Копируем данные в память
            Result:=True;
            Exit;
           end;
   25..32: begin
            SetLength(F_32BitsSamples,CountSamples);
            //Выделяем память под данные
            WaveFile.ReadBuffer(F_32BitsSamples[0],CountSamples*4);
            //Копируем данные в память
            Result:=True;
            Exit;
           end;
    else
          begin
            F_8BitsSamples  :=nil;
            F_16BitsSamples :=nil;
            F_24BitsSamples :=nil;
            F_32BitsSamples :=nil;
          end;
    end;//Case BitsPerSample of
end;
```

```
procedure TPCMWaveFile.LoadFromFile(const FileName: String);
var
  wFileSize     : Cardinal;
  wChankSize    : Cardinal;
  ID            : array[0..3] of Char;
  Header        : TWaveHeaderChank;
  RealFileSize  : Cardinal;
  h:Integer;
begin

  FSamples.Clear;

  FIdError:=FileDontFound;
  if not FileExists(FileName) then exit; //Файла нет выходим

   Try
     WaveFile := TFileStream.Create(FileName, fmOpenRead or fmShareDenyNone);
     WaveFile.Seek(0, soFromBeginning);  //В начало файла

     WaveFile.ReadBuffer(ID[0], 4);      //читаем тип файла
    if String(ID) <> 'RIFF' then  //Определяем тип файла
      begin
      FIdError:=IncorectFileFormat; //Файл не корректен
      FreeAndNil(WaveFile);
      Exit;
      end;

      WaveFile.ReadBuffer(wFileSize, 4);   //читаем размер файла
      FFileSize:=wFileSize;

    if WaveFile.size <> (wFileSize + 8) then
    //Определяем соответствие указанного размера
     begin
     //и размера файла(на случай если был поврежден)
      FIdError:=IncorectFileFormat;   //Файл не корректен
      FreeAndNil(WaveFile);
      Exit;
      end;

      WaveFile.ReadBuffer(ID[0], 4);
      //Читаем очередной Id ожидаем что это 'WAVE'
    if String(ID) <> 'WAVE' then //Определяем формат файла
      begin
        FIdError:=IncorectFileFormat;  //Файл не корректен
        FreeAndNil(WaveFile);
        Exit;
      end;

    wChankSize := 0;
    repeat                                  //Ищем чанк формата
      WaveFile.Seek(wChankSize, soFromCurrent);
      //Пропускаем все дополнительные чанки
      WaveFile.ReadBuffer(ID[0], 4);           //Читаем идентификатор чанка
      WaveFile.ReadBuffer(wChankSize, 4);      //Читаем размер чанка

      if wChankSize > High(integer) then
      //Проверяем размер загловка на разумность
        begin
```

```
          FIdError:=DataError;
          FreeAndNil(WaveFile);
          exit;
        end;

    until (String(ID)='fmt ') or (String(ID)='data');

    if String(ID)='data' then    //Проверяем найден ли заголовок формата
      begin
        FIdError:=HeaderError;
        //Ошибка т.к должны были найти  String(ID)='fmt '
        FreeAndNil(WaveFile);
        Exit;
      end;

      //Читаем заголовок меньше нашей структуры
      WaveFile.ReadBuffer(Header, Min(wChankSize, SizeOf(TWaveHeaderChank)));

      FFormatTag     :=Header.wFormatTag;
      FChannels      :=Header.wChannels;
      FSamplesPerSec :=Header.wSamplesPerSec;
      FBlockAlign    :=Header.wBlockAlign;
      FBitsPerSample :=Header.wBitsPerSample;


    //Смещаем указатель чтения в конец блока
    //нужно только для больших заголовков
    if wChankSize > SizeOf(TWaveHeaderChank) then
      WaveFile.Seek(wChankSize - SizeOf(TWaveHeaderChank), soFromCurrent);

     wChankSize := 0;
     repeat                                  //Ищем чанк данных
       WaveFile.Seek(wChankSize, soFromCurrent);
       //Пропускаем все дополнительные чанки
       WaveFile.ReadBuffer(ID[0], 4);
       //Читаем идентификатор чанка
       WaveFile.ReadBuffer(wChankSize, 4);
       //Читаем  размер чанка
     until  String(ID)='data';

       if String(ID)='data' then
         begin
          FIdError:=NoError;
          FSamples.CopyFrom(WaveFile,wChankSize);
          FSamples.Seek(0,soFromBeginning);
         end;



   except

    end;
 end;

constructor TPCMWaveFile.Create;
begin
  F_8BitsSamples  :=nil;
  F_16BitsSamples :=nil;
  F_24BitsSamples :=nil;
```

```
  F_32BitsSamples :=nil;

  FIdError := NoError;
  FFileSize:= 0;
  FSamples  := TMemoryStream.Create;
  //WaveFile := TFileStream.Create;
end;

destructor TPCMWaveFile.Destroy;
begin
  F_8BitsSamples  :=nil;
  F_16BitsSamples :=nil;
  F_24BitsSamples :=nil;
  F_32BitsSamples :=nil;

  FAvgBytesPerSec:=0;
  FFileSize      :=0;
  FSamplesPerSec :=0;
  FBlockAlign    :=0;
  FChannels      :=0;
  FBitsPerSample :=0;
  FFormatTag     :=0;
  FIdError       :=0;

  {FreeAndNil(WaveFile)};
  FreeAndNil(WaveFile);
  FreeAndNil(FSamples);
  inherited Destroy;
end;

end.
```

Ниже приводится текст модуля, написанного на языке Delphi 7, реализующего операции с комплексными числами:

```
unit ModuleComplex;
{$H+}

interface
uses
   SysUtils,Math;
  {$ifdef ComplexIsSingle}
    const
      MinComplex  =  1.5e-45;
      MaxComplex  =  3.4e+38;

    Type
      PComplex =^TComplex;
      TComplex = record
         Re,
         Im:Single;
    end;// TComplex = record
  {$else}
    const
      MinComplex  =  5.0e-324;
      MaxComplex  =  1.7e+308;

    Type
```

```
      PComplex =^TComplex;
      TComplex = record
         Re,
         Im:Double;
    end;// TComplex = record
  {$endif}
    type
      TCmxArray    = array of TComplex;

    function CmpAdd(const a,b:TComplex):TComplex;
    function CmpSub(const a,b:TComplex):TComplex;
    function CmpMul(const a,b:TComplex):TComplex;
    function CmpExp(const X:TComplex): TComplex;
    //function CmpSub(const a,b:TComplex):TComplex;
implementation
    function CmpAdd;
    var c:TComplex;
    begin
        c.Re:=a.Re+b.Re;
        c.Im:=a.Im+b.Im;
        result:=c;
    end;

    function CmpSub;
    var c:TComplex;
    begin
        c.Re:=a.Re—b.Re;
        c.Im:=a.Im—b.Im;
        result:=c;
    end;

    function CmpMul;
    var c:TComplex;
    begin
        c.Re:=a.Re*b.Re—a.Im*b.Im;
        c.Im:=a.Re*b.Im+a.Im*b.Re;
        result:=c;
    end;
    function CmpExp(const X:TComplex): TComplex;
var TempExp:Real;
    ImCos,ImSin:Extended;
begin
  TempExp:=Exp(X.Re);
  SinCos(X.Im,ImSin,ImCos);
  Result.Re:=TempExp*ImCos;
  Result.Im:=TempExp*ImSin;
end;
end.
```

5. ТЕСТОВАЯ ЗАДАЧА

5.1. Аналитическое решение и умозрительные результаты

Итак, приступим к тестированию полученной программы.

Пусть входными данными будем WAV-файл, в котором содержится запись проигранных на фортепиано аккордов, представленных на рисунке 5.

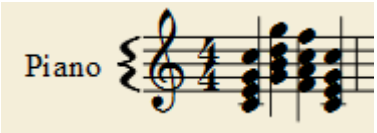

Рисунок 5 – Тестовый пример для распознавания

Какими же критериями следует руководствоваться при оценке полученных результатов?

Основным показателем правильности работы алгоритма БПФ является присутствие всех без исключения нот в выходных данных. Мы допускаем присутствие лишних нот вследствие возможных шумовых и иных искажений спектра входных данных, а также регистрации обертонов.

Отношение к временным показателям (т.е. как распознанные ноты располагаются по времени) также не должно быть слишком строгим, так как в случае фортепиано после нажатия ноты она ещё некоторое время продолжает звучать (если не была нажата педаль). Поэтому следует ожидать небольшое "смазывание"результирующих аккордов вследствие регистрации отзвуков.

5.2. Решение, полученное с использованием разработанного ПО

Ниже на рисунке 6 представлен пример работы программы при обработке вышеуказанных входных данных.

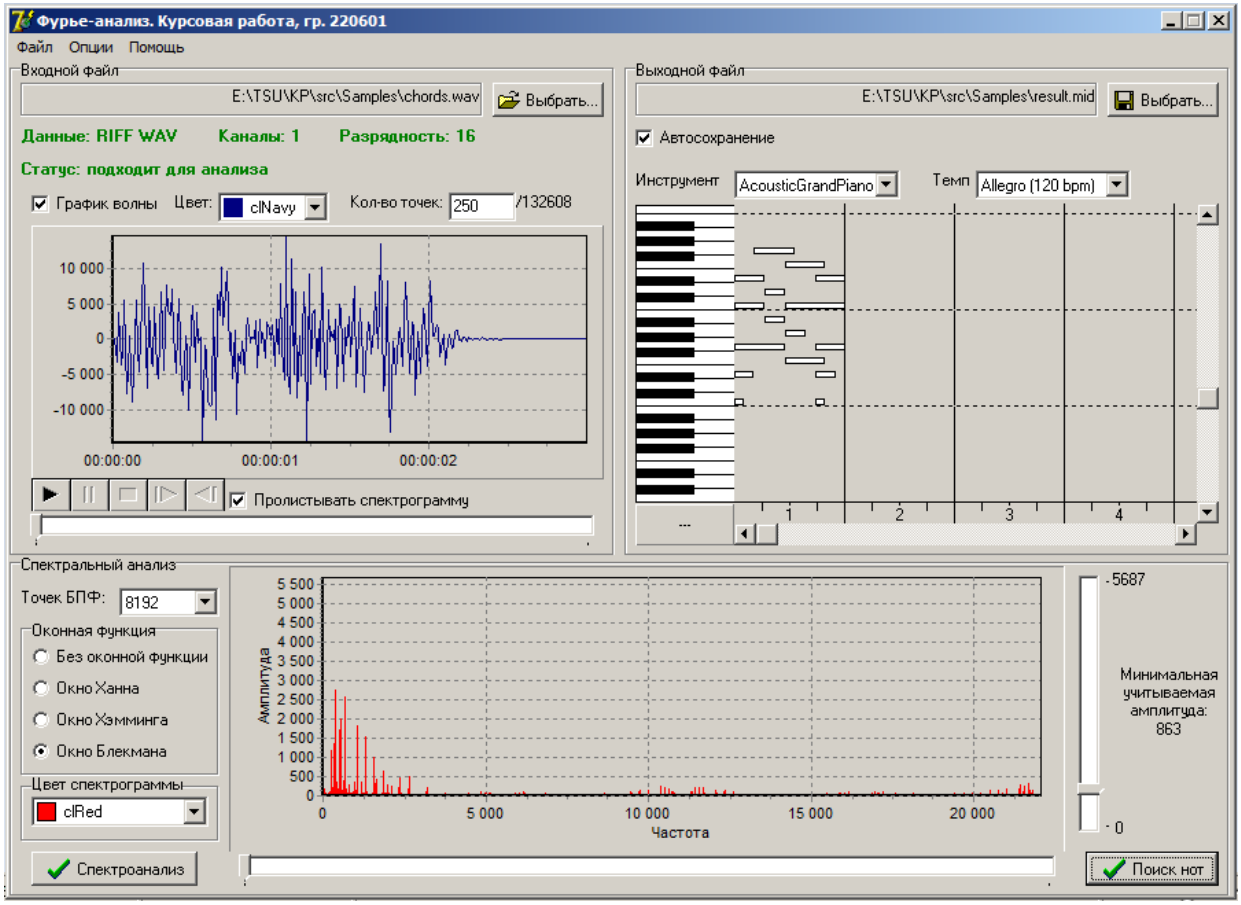

Рисунок 6 – Пример работы программы при обработке данных

На рисунке 7 представлена клавишная табулатура, полученная программой в результате работы.

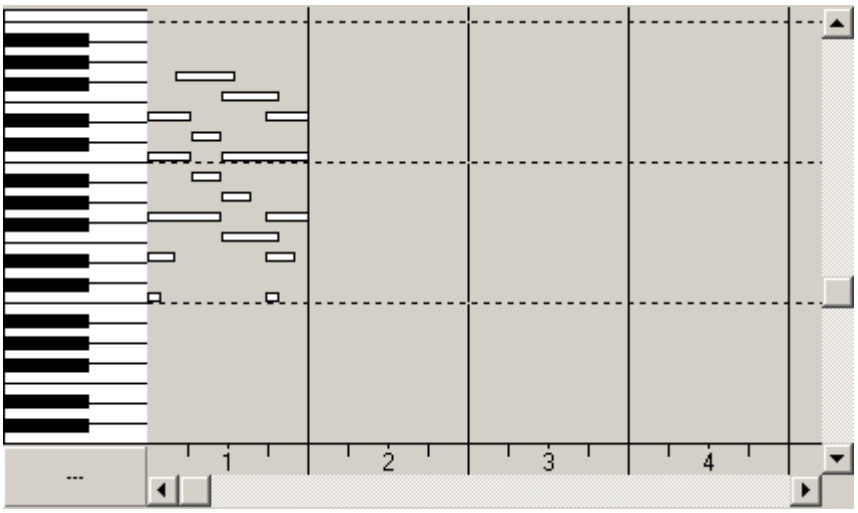

Рисунок 7 – Клавишная табулатура, полученная программой

На рисунке 8 представлен полученный MIDI-файл, открытый в нотном редакторе. Импортируемые длины нот округлялись до $\frac{1}{4}$.

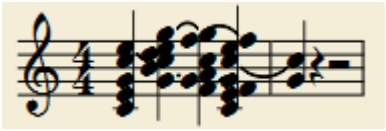

Рисунок 8 – Результат с округлением до $\frac{1}{4}$

На рисунке 9 открыт тот же файл, но когда импортируемые длины нот округлялись до $\frac{1}{8}$.

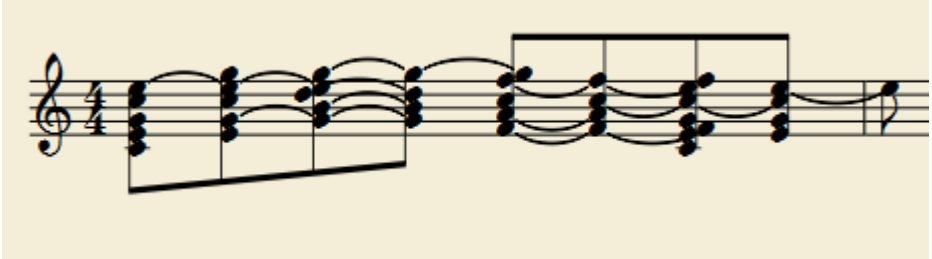

Рисунок 9 – Результат с округлением до $\frac{1}{8}$

5.3. Выводы

Итак, рассматривая рисунки 8 и 9, убеждаемся, что программа работает верно:

1. Все присутствующие ноты были распознаны программой.

2. К каждому аккорду сверху добавился первый обертон третьей ступени, т.е. третья ступень на октаву выше.

3. Вследствие отзвуков переход к текущему аккорду происходит не резко, а постепенно, что хорошо видно на рисунке 9. Это обстоятельство и дало в аккордах лишние ноты на рисунке 8.

Полученные результаты полностью соответствуют описанным раннее ожиданиям, поэтому можно считать, что программа работает верно и получает удовлетворительные результаты.

6. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ

Данная программа предназначена для нахождения нот в WAV-файле и сохранения их в MIDI-файл.

Сначала выберите исходный файл с помощью кнопки выбора файла на форме программы, либо с помощью меню. После этого можно перейти к спектральному анализу, а можно построить график звуковой волны, сохраненной в файле.

Для проведения спектроанализа выберите необходимые параметры, после чего нажмите соотвествующую кнопку.

После этого выберите имя выходного файла с помощью соответствующей кнопки на форме программы. Впрочем, та операция может быть проведена позже с помощью меню.

После этого выберите минимальную учитываемую амплитуду. Чем меньше величина, тем больше может быть найдено нот, однако к результату могут примешаться и шумы. После этого нажмите кнопку "Поиск нот". Программа проведет анализ, и сохранит ноты в указанном файле, а также нарисует клавишную табулатуру. Повторяйте три предыдущих шага, пока не добьётесь наилучшего по вашему мнению результата.

ЗАКЛЮЧЕНИЕ

Итак, в данной работе было разработано программное обеспечение, успешно реализующее численный метод и применяющее его для решения инженерной задачи.

Программа написана на языке Delphi 7 с использованием методологии объектно-ориентированного программирования, которое чрезвычайно мощным и удобным инструментом при решении многих задач.

Алгоритм быстрого преобразования Фурье, используемый программой для нахождения нот в WAV-файле и сохранения их в MIDI-файл, показал себя как мощный инструмент спектрального анализа, при этом являясь одним из наиболее эффективным в вычислительном плане среди других алгоритмов своего класса.

Однако не стоит останавливаться на достигнутом, ведь данные, получаемые методом быстрого преобразования Фурье, можно значительно улучшить с помощью последних достижений в алгоритмике: кореллограмм, нейронных сетей, генетических алгоритмов и т.д.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А.Б. Сергиенко. Цифровая обработка сигналов. - СПб.:Питер,2003.

2. Распознавание аудиообразов с применением обертонового ряда. Волков А.В., ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, № 3 ,2010.

3. PCM TO MIDI TRANSPOSITION. Luis Gustavo Pereira Marques Martins, Faculdade de Engenharia da Universidade do Porto, 2001

4 . Music: A Mathematical Offering, Dave Benson, Department of Mathematics, University of Aberdeen, 2008.

5. Волошинов А.В. Математика и искусство - 2-е изд. дораб и доп. - М.:Просвещение,2000.