Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Исследование методов обработки речи для передачи по каналу связи

направление: 09.03.01 - Информатика и вычислительная техника

Выполнил:
Алексеев Даниил Михайлович
Подпись_____

Руководитель: доцент, к.т.н.,
Богач Наталья Владимировна
Подпись_____

Санкт-Петербург

2016

Реферат

СПОСОБЫ ОБРАБОТКИ РЕЧИ, РЕЧЕВЫЕ КОДЕКИ, ВОКОДЕР, ЛИПРИДЕР, ПОМЕХОЗАЩИЩЁННОЕ КОДИРОВАНИЕ, КОД ХЭММИНГА, ИНТЕРЛИВИНГ, ЯЗЫК ПРОГРАММИРОВАНИЯ СИ, ИНТЕРФЕЙСЫ.

В выпускной работе рассматриваются методы обработки речевого сигнала. Кратко описаны виды речевых кодеков, теория канального кодирования блочными и свёрточными кодами, рассказана причина применения интерливинга. В работе описано проектирование и разработка модулей библиотек для обработки речевого сигнала. Представлена методология выбора метода обработки речевого сигнала по имеющимся параметрам канала передачи данных.

Содержание

Список рисунков

Введение

Не существует настолько надёжных каналов связи, которые могут обеспечить полное отсутствие помех, воздействующих на передаваемую речь. Поэтому были изобретены различные методы защиты от помех. Из них можно выделить три наиболее популярных: аппаратное улучшение аппаратуры и канала связи, повышение отношения сигнал-шум, обнаружение и исправление ошибок в принятых сообщениях. Первый способ защиты позволяет улучшить качество передаваемой речи только до определённых пределов, второй подразумевает работу на физическом уровне (что также имеет свои пределы). Поэтому мы рассмотрим методы обнаружения и исправления ошибок в речевом сигнале.

Кодирование речи можно разбить на два основных этапа: непосредственно речевое кодирование и канальное кодирование (которое, как правило, делится на блочное и свёрточное кодирования, а также деинтерливинг (перемежение)). На Рис. 1 приведена поясняющая схема последовательности кодирования.

Рис. 1.1. Последовательность кодирования речи

Речевое кодирование не является помехоустойчивым, поэтому данные с речевого кодера поступают на канальный. Его задача – сделать передаваемую информацию помехоустойчивой, т.е. дать возможность приёмнику обнаружить (и, в некоторых случаях, исправить) ошибки, которые возникают при передаче информации. Помимо этого, канальное кодирование может выполнять такие функции, как добавление управляющей информации и шифрование.

При блочном кодировании информация делится на блоки определенной длины, и каждый блок кодируется отдельно. Простейшим примером блочного кодирования является дополнение до четности. В этом случае каждый блок делится на группы, которые дополняются одним битом со значением единица или ноль, в зависимости от четного или нечетного количество единиц в исходной группе.

При свёрточном кодировании работа ведётся сразу со всей поступившей информацией (без разделения её на части). Самый простой способ свёрточного кодирования – это добавление между информационных бит проверочных, которые зависят от рядом стоящих информационных бит.

Оба описанных выше метода кодирования позволяют не только определить наличие ошибки в передаваемом коде, но и устранить её. Последнее не всегда является обязательным требованием: бывают ситуации, когда намного проще повторно отправить данные, или же оставить всё, как есть (в таком случае мы получим небольшие помехи во время разговора). К минусам данных методов можно отнести то, что они не могут восстановить правильную последовательность, если ошибкам подверглось слишком много подряд идущих бит. Решением данной проблемы является интерливинг.

Интерливинг нужен для того, чтобы переставить местами закодированную последовательность: если при передаче информация подверглась пачечным ошибкам, то на приёмной стороне после сборки последовательности для декодирования повреждённые биты окажутся на значительном расстоянии друг от друга. Для надёжности можно выполнить процедуру интерливинга несколько раз подряд.

Постановка задачи

Исходя из вышесказанного, мы можем сформулировать задачи, которые требуется решить при выполнении данной работы.

Необходимо исследовать существующие методы обработки речи и предложить свою реализацию библиотек, предназначенных для обработки речи, а также методологию выбора более подходящего варианта обработки с учётом имеющегося канала связи.

К канальным кодекам выдвигаются следующие требования:

- простота реализации;
- допустимая избыточность;
- возможность коррекции ошибок.

К речевым кодекам выдвигаются следующие требования:

- минимальное снижение качества сигнала;
- небольшие задержки.

Стоит отметить, что не всегда удаётся соответствовать всем требованиям. Чаще всего они даже противоречат друг другу.

1. Исследование методов обработки речевого сигнала

1.1. Канальный уровень: блочные кодеки

Блочному кодированию подвергается не весь речевой сигнал, а только его часть. Это связано с тем, что блочными кодами, как правило, не могут кодировать длинные последовательности бит, или при повышении числа информационных бит избыточность становится неприемлемой.

По ходу выполнения работы было выделено три наиболее популярных вида блочных кодов, а также предложены конкретные варианты их применения (по числу информационных и контрольных бит).

1.1.1. Проверка на чётность

Данный вид кодирования выделяется следующими свойствами:

- простота реализации;
- возможность находить и восстанавливать ошибки;
- не защищён от ошибок в контрольных битах.

Нами предлагается проверка на чётность через матрицу; при этом можно выделить два основных вида матриц: двумерные и трёхмерные.

В ходе исследований было установлено, что для первого вида наиболее оптимальным является размер матрицы 3x3 информационных бит: в таком случае мы передаём 9 информационных и 6 контрольных бита, и можем исправить 1 ошибку. На Рис. 2 изложена основная идея алгоритма с применением матрицы 2x2.

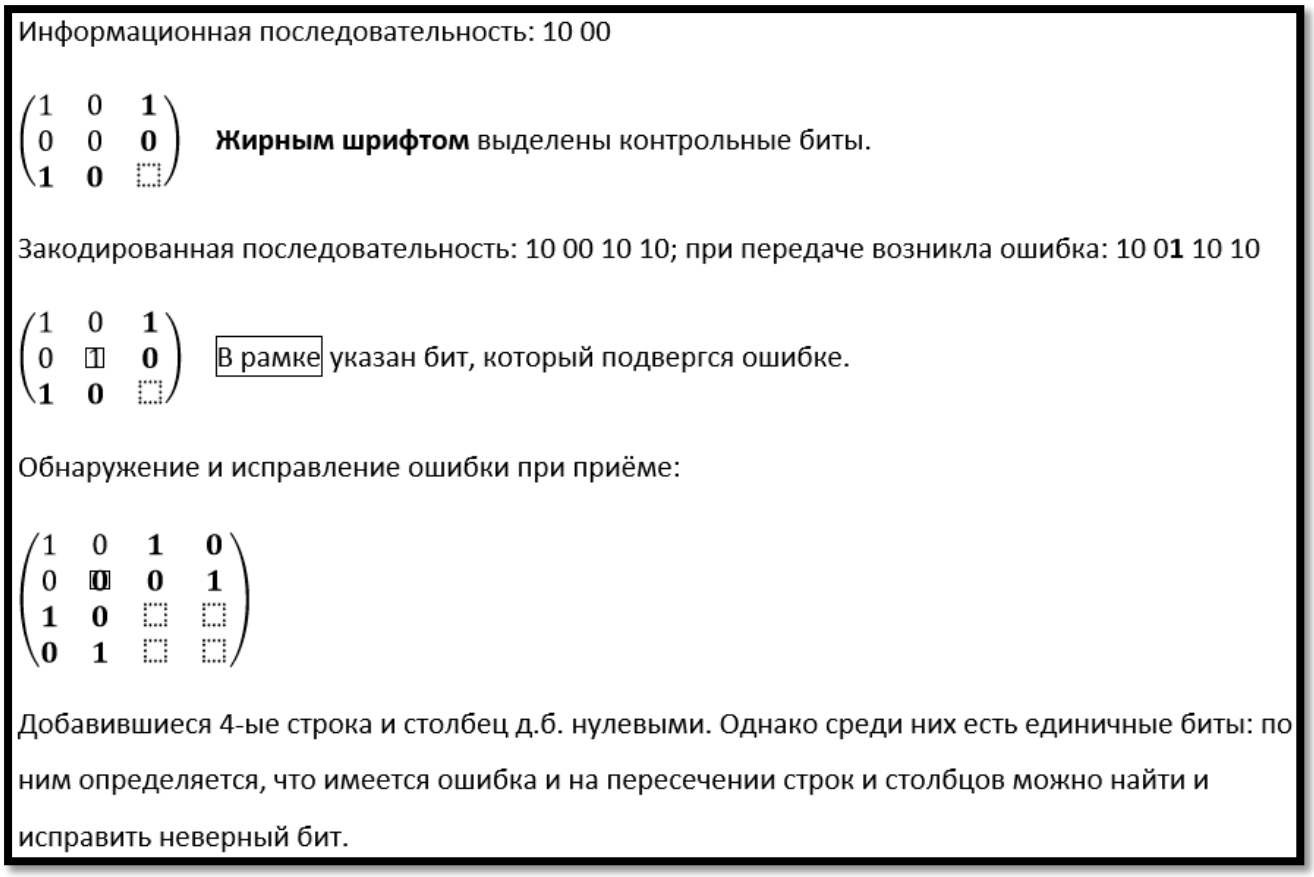

Рис. 1.2. Алгоритм дополнения до чётности, двумерный вариант

Для трёхмерного представления более удобно использовать кодирование бит в виде трёх матриц 3x3. Помимо чётности по столбам и строкам, добавляется матрица проверки «по глубине» - см. Рис. 3. Пример работы с такой матрицей рассмотрен в [1]. На каждые 27 информационных бит мы получаем 27 контрольных, и можем исправить 3 ошибки.

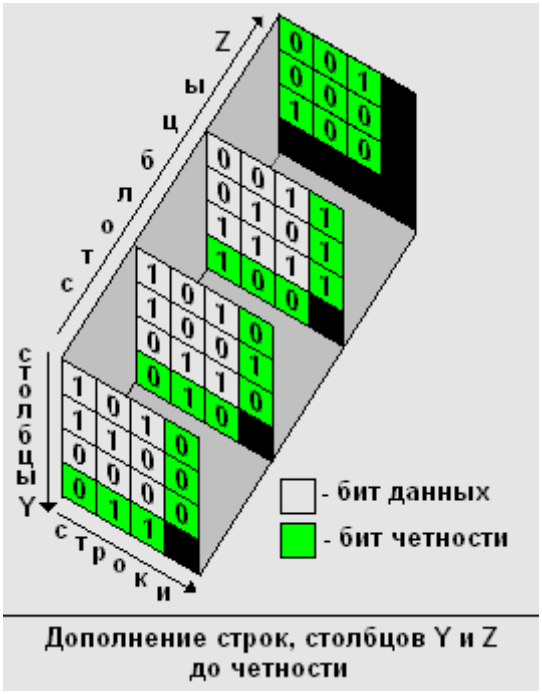

Рис. 1.3. Представление закодированных бит в трёхмерном варианте контроля на чётность

Следует напомнить, что данный вид кодирования не защищает от ошибок в контрольных битах. Однако по сравнению с одномерным вариантом при чётном числе ошибок мы можем узнать, что принятая последовательность – битая. К минусам данного алгоритма можно отнести высокую избыточность.

1.1.2. Код Хэмминга

Данный алгоритм весьма популярен. Обычно код Хэмминга характеризуется двумя числами: общим числом бит, и числом информационных бит.

Общее число бит определяется как 2^n-1, где n – число контрольных бит. Количество информационных бит определяется как 2^n-1-n.

Код Хэмминга при вводе большой избыточности позволяет обнаружить и исправить более одной ошибки в блоке. В результате исследований было определено, что наиболее оптимальным вариантом будет код Хэмминга (26, 31): 26 информационных бита, 5 контрольных (31 всего).

Причины такого выбора кроются в следующем: если взять наиболее длинный из применяемых блоков в коде Хэмминга (127 бит), то получим следующее: кодом (120, 127) мы можем исправить одну ошибку, кодом (99, 127) – четыре, но при этом информационных бит будет ~78%. Если отправить 4 блока, закодированных кодом (26, 31), то мы получим всего 124 бита, 104 информационных и их содержание в пакетах будет около 84%.

Следует заметить: если использовать код (11, 15), то при отправке 8 блоков мы получим всего 120 бит, 88 информационных и их содержание в пакетах будет ~73%. Однако данная избыточность нецелесообразна в большинстве случаев.

К плюсам данного алгоритма можно отнести то, что кодирующий полином может быть задан разработчиком: как следствие мы получим дополнительную защиту от прослушки. Кроме того, данный алгоритм не ломается при наличии ошибки в контрольном бите. При вводе наибольшей возможной избыточности для 4 информационных бит мы выигрываем у проверки на чётность: будет всего 7 бит (а не 8).

Пример работы с кодом Хэмминга приведён в [2].

1.1.3. Циклические коды

Данный вид кодов является обобщённым вариантом кода Хэмминга. Они были введены для упрощения алгоритмов, цель которых – найти и исправить более одной ошибки. Примеры работы с данным видом кодов можно найти в [3].

Всё, сказанное про избыточность и число информационных бит в коде Хэмминга, справедливо и для циклических кодов.

Помимо указанных ранее достоинств кодов Хэмминга, говоря про циклические коды, можно добавить следующее: декодер циклических кодов строится на операциях циклического сдвига, что позволяет реализовать его даже аппаратно.

1.2. Канальный уровень: свёрточные кодеки

Канальное кодирование применяется сразу ко всему передаваемому пакету, без разделения его на блоки. По результатам исследований было выделено два наиболее популярных свёрточных кодека, а также предложено улучшение к одному из них.

1.2.1. Ввод контрольных бит

Данный метод является наиболее простым. Пример работы с данным методом описан в [1], однако сам предложенный там алгоритм требует доработки.

При кодировании между информационными битами вставляются проверочные (они представляют собой сумму по модуль два соседних бит). Декодирование выглядит следующим образом: суммируются соседние биты исходных данных и сравниваются с их проверочным битом.

- Если для двух соседних проверочных битов была зафиксирована ошибка, то общий информационный бит для этих двух проверочных битов - неверен. Для исправления ошибки необходимо заменить его на противоположный.
- Если для одного проверочного символа была зафиксирована ошибка, а два соседних проверочных символа ошибку не показали, это означает, что сбой произошел в проверочном символе, а информационные биты корректны.

Очевидная проблема данного алгоритма – ошибка в крайних битах. Для её устранения предложен следующий подход: добавлять в начало и конец последовательности по два дополнительных бита, значение которых нам всегда известно. Независимо от принятых данных, мы будем декодировать код исходя из того, что первые и последние три бита – это наши константы и проверочный бит, образованный ими. Таким образом, даже при наличии ошибки в

первых/последних двух переданных битах мы сможем без проблем декодировать последовательность.

К плюсам данного алгоритма можно отнести следующее: простота реализации, ошибка в контрольных битах не приводит к краху алгоритма.

Серьёзным минусом данного алгоритма можно считать следующее: если расстояние между ошибочными битами меньше 2 (они стоят рядом или через один), то алгоритм породит дополнительные ошибки.

1.2.2. Кодирование полиномом с задержкой

Данный метод подразумевает использование полинома с задержкой. Кодеры такого типа имеют скорость ½ или ¼ (т.е. на каждый входной бит приходится 2 или 4 закодированных бита). Алгоритм кодирования описан в [4] и [5], алгоритм декодирования – в [6].

К плюсам данного метода можно отнести высокую надёжность: по результатам опытов декодер справлялся примерно с 14-20% ошибок (в зависимости от установленного режима). Кроме того, несколько подряд идущих ошибок (не более 3) не ломали алгоритм декодирования.

К минусам данного метода можно отнести большую избыточность (минимально возможный режим работы – ½). Кроме того, из всех представленных здесь алгоритмов помехоустойчивого кодирования данный алгоритм самый сложный.

Однако несмотря на указанные минусы, именно этот метод является самым распространённым и используемым в сетях связи.

1.3. Канальный уровень: перемежение (interleaving)

Ранее отмечалось, что не все кодеки могут справиться с несколькими ошибками в переданном пакете/блоке информации. Такие ошибки называются пачечными: они возникают из-за того, что длительность воздействующего

вредного сигнала достаточно высока для того, чтобы повредить несколько рядом стоящих бит.

Для борьбы с пачечными ошибками было введено перемежение (интерливинг – interleaving). Перед отправкой в канал связи биты перестраиваются таким образом, чтобы соседние биты оказались как можно дальше друг от друга. При приёме происходит деинтерливинг: мы получаем исходную последовательность; в результате повреждённые биты оказываются на довольно большом расстоянии друг от друга. Примеры интерливинга можно найти в [4].

1.4. Речевые кодеки

Можно выделить два основных вида речевых кодеров, которые в свою очередь порождают гибридный вариант: это вокодеры и липридеры. Нередко бывает так, что одно устройство поддерживает сразу оба варианта.

1.4.1. Вокодер

Принцип действия вокодеров основан на том, что они сопоставляют принятый голосовой сигнал с имеющимся фиксированным словарём, достают из него номер фонемы, который соответствует сигналу, и передают его. Такой подход значительно снижает количество передаваемой информации, но сам голосовой сигнал теряет характерное для человека звучание. Более подробно с принципом действия вокодеров можно ознакомиться в [7].

Сейчас почти не используется в чистом виде, однако применяется в сочетании с липридером.

1.4.2. Липридер

Данный вид речевых кодеров является наиболее распространённым в наши дни. Липридеры снимают характеристики принятого голосового сигнала и передают их. Данный подход обеспечивает более человеческое звучание, но

количество передаваемой информации заметно возрастает. Более подробно с принципом действия липридеров можно ознакомиться в [7] и [8].

Следует заметить: из-за использования линейного предсказания в липридерах речь собеседника не всегда удаётся передать более точно (возможно искажение звуков).

1.4.3. Гибрид из вокодера и липридера

При сочетании вокодера и липридера удаётся избежать минусов устройств, в которых данные подходы реализованы отдельно (однако в таком случае повышается количество передаваемой информации: помимо номера фонемы передаются параметры речевого сигнала).

Основой аналитической части таких устройств служат фиксированный и адаптивный кодовые словари, которые позволяют более точно подобрать параметры фильтра для воспроизводящего устройства. Более подробно можно ознакомиться с гибридным вариантом в [8] и [9].

2. Проектирование библиотек

С учётом того, что обработка речевого сигнала м.б. разделена на два основных этапа, мы можем выделить библиотеки для работы с канальным уровнем и работы с самим речевым сигналом.

Библиотека для канального уровня будет содержать четыре модуля: кодирования, декодирования, интерливинга и деинтерливинга.

Для обработки речевого сигнала требуется четыре модуля: два кодирования и два декодирования (по одному на каждый вид кодека).

Было решено написать библиотеки на языке Си. Принятию данного решения способствовали следующие факторы:

- язык Си наравне с ассемблерами используется для программирования микроконтроллеров – таким образом область применения написанных библиотек значительно расширяется;
- написанное на классическом Си приложение будет кроссплатформенным (на уровне исходных кодов): данные библиотеки получат возможность встраиваться в приложения, написанные на разных ОС;
- согласно разным источникам [10][11], язык программирования Си довольно долгое время занимает верхние позиции по популярности (таким образом, найти специалиста для поддержки и сопровождения библиотек будет просто).

Данные библиотеки разрабатывались по такой технологии проектирования ПО, как прототипирование. В пользу данной технологии были следующие факторы:

- нам известны не все требования (формат, в котором мы будем передавать данные в канал связи; скорость передачи данных);
- данная технология позволила нам быстро увидеть некоторые свойства продукта (применимость, удобство);

Подробнее с данной моделью можно ознакомиться в [12].

Также было решено определить такой тип данных, как Voice_type: он описывает речевой сигнал. В данном типе предлагается хранить принятый речевой сигнал, а также содержимое словарей (фиксированного и адаптивного).

2.1. Модули для обработки речевого сигнала

Данные модули можно использовать отдельно от предлагаемых модулей для работы на канальном уровне. В таком случае, если взаимодействие предлагаемых кодеков для обработки речи с канальными кодеками (отличными от предложенных в данной работе) будет невозможно из-за различающихся форматов представления данных, предлагается использовать такой шаблон проектирования, как адаптер (adapter/wrapper).

Все модули для обработки речевого сигнала представлены в виде интерфейсов. Это было сделано по следующим причинам:

- существует множество способов реализации речевых кодеков, каждый из которых требует довольно большой теоретической базы (математическое описание, особенности речи). Ввод хотя бы в один способ реализации займёт крайне много времени.
- используя интерфейс, мы можем без проблем рассказать, какие действия должен уметь выполнять данный алгоритм и, следовательно, что в данной функции должно быть реализовано.

2.1.1. Модуль для кодирования речевого сигнала

Предполагается, что речевой сигнал уже получен, и мы имеем его в цифровом виде. Пользователю требуется указать, откуда считать полученный сигнал и вид кодера.

Создав объект типа Voice_type, в него следует считать полученный речевой сигнал. Далее идёт вызов функций, которые переводят речевой сигнал в формат кодека. Это отдельные функции для вокодера и липридера (подробнее

см. 3.3.1); для использования гибрида рекомендуется последовательно применить данные функции последовательно, или модифицировать функцию липридера – в таком случае можно будет обойтись вызовом только функций липридера.

После, используя функцию вызова канального кодирования, пользователь подвергает полученный речевой сигнал помехоустойчивому кодированию.

2.1.2. Модуль для декодирования речевого сигнала

При декодировании пользователь вызывает функцию синтезатора речи, в которую передаёт следующие параметры: тип кодека и массив, в который нужно поместить полученный с выхода канального декодера результат (номер фонемы, число нулей интенсивности и прочие параметры речевого сигнала), а также ID свёрточного и блочного кодера (см. 2.2.1).

Далее к указанному массиву обращается модуль, который воспроизводит полученную информацию (в данной работе этот модуль не рассматривается).

2.2. Модули для работы на канальном уровне

Часть модулей для работы на канальном уровне представлены в виде интерфейсов. В данном случае это сделано по следующим причинам:

- алгоритм хорошо известен и/или его написание не составляет особого труда;
- пользователю предлагается самому подобрать нужный кодирующий полином/образующую матрицу.

2.2.1. Модуль для канальных кодеков

Данный модуль служит для помехоустойчивого кодирования и декодирования речевого сигнала. По-умолчанию он предоставляет пользователю 3 описанных в данной работе кодека для блочного кодирования и 2 свёрточных кодека (см. 1.1 и 1.2).

За вызов функций данного модуля отвечает функция, которая принимает на вход тип речевого кодека, ID блочного кодека и ID свёрточного кодека.

Ниже приведены таблицы 2.1 и 2.2, в которых указано соответствие между ID кодеков и реализуемых ими алгоритмами.

Таблица 2.1.

Соответствие между ID блочного кодека и его алгоритмом

ID кодека	**Реализованный алгоритм**
1	**2**
1	Проверка на чётность (двумерная)
2	Код Хэмминга (4, 7)
3	Циклический код (4, 7)

Таблица 2.2.

Соответствие между ID свёрточного кодека и его алгоритмом

ID кодека	**Реализованный алгоритм**
1	**2**
1	Ввод контрольных бит
2	Кодирование полиномом с задержкой

При желании пользователь может добавить в библиотеку свои кодеки.

2.2.2. Модуль интерливинга и деинтерливинга

Данный модуль предназначен для перемежения речевого сигнала, который уже подвергся помехоустойчивому кодированию.

Функции данного модуля вызываются из модулей канальных кодеков: применение интерливинга без помехоустойчивого кодирования не имеет смысла! Данный модуль содержит две функции: интерливинга (на вход подаются биты, подвергнутые помехоустойчивому кодированию), и деинтерливинга (на вход подаются принятые из канала передачи данных биты).

После интерливинга данные отправляются в канал передачи. Из канала передачи принятые биты поступают в деинтерливер.

3. Разработка библиотек

Перед описанием процесса разработки стоит отметить, что подключение библиотек к проекту выполняется путём добавления .h и .c файлов в папку проекта.

3.1. Структура библиотек и описание

Можно выделить четыре основных заголовочных файла разработанных библиотек: это Channel_coding.h, Analytic.h, Synthetic.h и Speech_decoder.h

Первый файл содержит прототипы функций для канальных кодеков, а последующие три содержат прототипы функций для речевого кодека.

В библиотеке имеется одна сущность: Voice_type. Она описывает свойства речевого сигнала. Выглядит она следующим образом:

Листинг 1. Содержимое Voice_type.h (описание сущности Voice_type)

```
	#ifndef VOICE_TYPE
	#define VOICE_TYPE

	typedef struct
	{
	    int amplitude;
	    int frequency;
	    int phoneme;
	    /*
	    Иные характерные для речевого сигнала параметры:
	    -кратковременная энергия речевого сигнала;
	    -число нулей интенсивности (мгновенная частота);
	    -форманты речевого сигнала (концентрация энергии в
ограниченной частотной области);
	    -коэффициенты линейного предсказания (ковариационный и
автокорреляционный методы);
	    -распределение энергии сигнала по частотным группам;
	    -длительность пауз.
	    */
	} Voice_type;

	#endif
```

3.2. Разработка канальной библиотеки

Разработка библиотеки проходит по следующему плану:

1) разработка одного из описанных в 1.1 и 1.2 методов;
2) сравнение полученной закодированной последовательности с результатом эталонной модели. В случае успеха переход к следующему пункту, в случае неудачи – приступаем к отладке;
3) разработка декодера для созданного метода кодирования;
4) тестирование декодера; в случае успеха переходим к реализации следующего метода, в случае неудачи – приступаем к отладке.

Пример листинга алгоритма на языке С, реализующий один из методов канального кодека (Хэмминг (4, 7), код ошибки совпадает с номером бита), приведён в приложении 1.

Следует заметить, что данная библиотека имеет 2 заголовочных файла: Channel_coding.h и Interleaving.h. Их содержимое приведено в приложении 2.

3.2.1. Описание функций канальной библиотеки

Ниже приведена таблица 3.1, в которой описаны функции предлагаемой канальной библиотеки. Следует заметить: ID – идентификатор метода канального кодирования; подробнее см. 2.2.1.

Таблица 3.1.

Соответствие между ID блочного кодека и его алгоритмом

Функция	**Аргументы**	**Описание**
1	**2**	**3**
void bm_ID	int to_encode[], int out[]	Функция bm_ID реализовывает блочное кодирование речевого сигнала согласно заданному алгоритму.

Продолжение таблицы 3.1.

1	2	3
	int to_encode[], int out[]	Данная функция не возвращает никаких результатов. to_encode – указатель на массив данных, которые нужно закодировать. out – указатель на массив, куда нужно сложить результат. Входит в заголовочный файл "Channel_coding.h"
void cm_ID	int to_encode[], int out[]	Функция cm_ID реализовывает свёрточное кодирование принятого речевого сигнала согласно заданному алгоритму. Данная функция не возвращает никаких результатов. to_encode – указатель на массив данных, которые нужно закодировать. out – указатель на массив, куда нужно сложить результат. Входит в заголовочный файл "Channel_coding.h"
void De_bm_ID	int to_decode[], int out[]	Функция De_bm_ID реализовывает декодирование принятого блочного кода согласно заданному алгоритму. Данная функция не возвращает никаких результатов. to_decode – указатель на массив данных, которые нужно декодировать. out – указатель на массив, куда нужно сложить результат. Входит в заголовочный файл "Channel_coding.h"

Продолжение таблицы 3.1.

1	2	3
void De_cm_ID	int to_decode[], int out[]	Функция De_cm_ID реализовывает декодирование принятого свёрточного кода согласно заданному алгоритму. Данная функция не возвращает никаких результатов. to_decode – указатель на массив данных, которые нужно декодировать. out – указатель на массив, куда нужно сложить результат. Входит в заголовочный файл "Channel_coding.h"
Interleaving	int to_interleaving[]	Функция Interleaving реализовывает перемежение бит перед отправкой в канал связи. Данная функция не возвращает никаких результатов. to_interleaving – указатель на массив данных, которые нужно переставить. Входит в заголовочный файл "Interleaving.h"
DE_interleaving	int to_DEinterleaving[]	Функция DE_interleaving реализовывает перемежение бит перед отправкой в канал связи. Данная функция не возвращает никаких результатов. to_ DEinterleaving – указатель на массив данных, которые нужно переставить. Входит в заголовочный файл "Interleaving.h"

3.3. Разработка библиотеки для речевого сигнала

Данная библиотека представлена в виде интерфейсов и состоит из 6 файлов: 3 заголовочных и 3 – исходных кодов (Analytic, Synthetic, Speech_decoder).

Рассмотрим процедуру кодирования речи. Она состоит из следующих шагов:

1) Создаём объект VT_obj типа Voice_type.
2) Получаем речевой сигнал (сторонний модуль, в данной работе не рассмотрен) и записываем его в VT_obj.
3) Определяем тип кодека, который нам нужен.
4) В зависимости от типа кодека вызываем методы Phoneme_number_generator(2) или LPC(2), на вход которых подаём тип кодера и VT_obj.
5) Вызывается метод get_signal_parameter(3). На его вход подаётся тип кодера, а также VT_obj (третий параметр – значение VT_obj на предыдущем шаге (если выбран липридер) или текущее значение VT_obj(если выбран вокодер)).
6) В зависимости от типа кодека вызываются методы Compare_with_fixed_dictionary(1)/Compare_with_adaptive_dictionary(2), на вход которых подаётся VT_obj (и его значение на предыдущем шаге – если выбран липридер).

Листинг заголовочных файлов, содержащих прототипы функций, реализующих речевой кодек, приведён в приложении 3.

3.3.1. Описание функций библиотеки для речевой обработки сигнала

Далее приведена таблица 3.2, в которой описаны функции предлагаемой библиотеки для речевой обработки сигнала.

Таблица 3.2.

Соответствие между ID блочного кодека и его алгоритмом

Функция	Аргументы	Описание
1	**2**	**3**
Search_pause	Voice_type VT	Функция Search_pause отвечает за поиск пауз в речи человека. Данная функция возвращает true, если пауза найдена, и false в противном случае. VT – полученный речевой сигнал. Входит в заголовочный файл "Analytic.h"
LPC	char coder_type, Voice_type VT	Функция LPC отвечает за реализацию алгоритма линейного предсказания. Данная функция ничего не возвращает. coder_type – тип речевого кодека. VT – полученный речевой сигнал. Входит в заголовочный файл "Analytic.h"
Set_current_sound	Voice_type VT	Функция Set_current_sound заполняет линию задержки. Данная функция ничего не возвращает. VT – полученный речевой сигнал. Входит в заголовочный файл "Analytic.h"

Продолжение таблицы 3.2.

1	2	3
Get_previous_sound	-	Функция Get_previous_sound извлекает из линии задержки предыдущий речевой сигнал. Данная функция возвращает предыдущий речевой сигнал (тип – Voice_type). Входит в заголовочный файл "Analytic.h"
Compare_with_adaptive_dictionary	Voice_type VT, Voice_type VT_old	Функция Compare_with_adaptive_dictionary сравнивает полученный речевой сигнал с адаптивным словарём. Данная функция ничего не возвращает. Результат сравнения заносится в глобальную переменную NEW_VOICE (тип – Voice_type). VT – полученный речевой сигнал. VT_old – предыдущий речевой сигнал. Входит в заголовочный файл "Analytic.h"

Продолжение таблицы 3.2.

1	2	3
get_signal_parameter	char coder_type, Voice_type VT1, Voice_type VT2	Функция get_signal_parameter вытаскивает из полученного сигнала его параметры. Данная функция ничего не возвращает. coder_type – вид речевого кодера VT1, VT2 –речевые сигналы (определяются видом кодека). Входит в заголовочный файл "Analytic.h"
Phoneme_number_generator	char coder_type, Voice_type VT	Функция Phoneme_number_generator определяет номер фонемы по полученному сигналу. Данная функция возвращает номер фонемы. coder_type – вид речевого кодера VT – полученный речевой сигнал. Входит в заголовочный файл "Analytic.h"
Save_best_phoneme_id	int id	Функция Save_best_phoneme_id сохраняет идентификатор фонемы лучшей подобранной фонемы на данный момен. Данная функция ничего не возвращает. id – идентификатор фонемы, который надо сохранить. Входит в заголовочный файл "Analytic.h"

Продолжение таблицы 3.2.

1	2	3
Compare_with_fixed_dictionary	Voice_type VT	Функция Compare_with_fixed_dictionary сравнивает полученный речевой сигнал с фиксированным словарём. Данная функция ничего не возвращает. Результат сравнения заносится в глобальную переменную BEST_PHONEM_ID (тип – int). VT – полученный речевой сигнал. Входит в заголовочный файл "Analytic.h"
Set_filter_parametres	Voice_type VT	Функция Set_filter_parametres устанавливает текущие значения фильтра (используется для линейного предсказания). Данная функция ничего не возвращает. VT – полученный речевой сигнал. Входит в заголовочный файл "Synthetic.h"
Set_phoneme_num	int phoneme_num	Функция Set_phoneme_num устанавливает текущее значение идентификатора фонемы (используется для линейного предсказания). Данная функция ничего не возвращает. VT – полученный речевой сигнал. Входит в заголовочный файл "Synthetic.h"

Продолжение таблицы 3.2.

1	2	3
Code_generator	char coder_type, int block_code_id, int conv_code_id	Функция Code_generator вызывает функции библиотеки канального кодирования. Данная функция ничего не возвращает. Результат обработки данных кладётся в глобальную переменную CONV_CODE[500] block_code_id – идентификатор блочного кодера. conv_code_id – идентификатор свёрточного кодера. Входит в заголовочный файл "Synthetic.h"
get_filter_parameters	int input_data[], int block_code_id, int conv_code_id	Функция get_filter_parameters определяет параметры речевого фильтра. Данная функция возвращает полученные параметры речевого фильтра (тип – Voice_type). input_data[] – массив принятых бит. block_code_id – идентификатор блочного кодера. conv_code_id – идентификатор свёрточного кодера. Входит в заголовочный файл "Speech_decoder.h"

Продолжение таблицы 3.2.

1	2	3
get_phoneme_number	int input_data[], int block_code_id, int conv_code_id	Функция get_phoneme_number определяет идентификатор фонемы. Данная функция возвращает идентификатор фонемы (тип – int). input_data[] – массив принятых бит. block_code_id – идентификатор блочного кодера. conv_code_id – идентификатор свёрточного кодера. Входит в заголовочный файл "Speech_decoder.h"
get_ALL	int input_data[], int block_code_id, int conv_code_id	Функция get_ALL определяет идентификатор фонемы и параметры речевого фильтра. Данная функция возвращает речевой сигнал (тип – Voice_type). input_data[] – массив принятых бит. block_code_id – идентификатор блочного кодера. conv_code_id – идентификатор свёрточного кодера. Входит в заголовочный файл "Speech_decoder.h"

Продолжение таблицы 3.2.

1	2	3
made_voice	char coder_type, int input_data[], int block_code_id, int conv_code_id	Функция made_voice восстанавливает речевой сигнал по полученным данным. Данная функция возвращает речевой сигнал (тип – Voice_type). input_data[] – массив принятых бит. block_code_id – идентификатор блочного кодера. conv_code_id – идентификатор свёрточного кодера. Входит в заголовочный файл "Speech_decoder.h"
channel_decoding	int input_data[], int block_code_id, int conv_code_id	Функция channel_decoding декодирует принятые из канала передачи биты. Данная функция возвращает речевой сигнал (тип – Voice_type). input_data[] – массив принятых бит. block_code_id – идентификатор блочного кодера. conv_code_id – идентификатор свёрточного кодера. Входит в заголовочный файл "Speech_decoder.h"

4. Методология выбора более подходящего варианта обработки речевого сигнала с учётом имеющегося канала связи

Следует заметить, что при наличии канала связи с высокой скоростью передачи данных проблема выбора метода обработки речи не столь актуальна – можно выбрать любой речевой кодек и использовать любой из имеющихся подходов канальной обработки данных.

В данной работе мы предполагаем, что пропускная способность нашего канала низкая (или мы можем выделить лишь небольшую часть пропускной способности на каждого абонента).

При разработке или улучшении алгоритма для обработки речевого сигнала главный вопрос – это «Насколько хорошо надо организовать/повысить помехозащищённость канала связи?».

Для ответа на данный вопрос в первую очередь следует рассчитать вероятность возникновения ошибки при передаче информации. Подобные исследования уже проводились ранее для различных каналов связи (см. [13]-[16]), поэтому здесь мы приведём основные заключения из них:

- при отсутствии помехоустойчивого кодирования шанс приёма ошибочного символа в пакете составляет чуть меньше 1%. Исключение – канал радиотелеграфа (шанс ошибки ~1.37%, но данный канал уже не используется);
- на прохождение сигнала влияют такие факторы, как метеоусловия, промышленные помехи, взаимные помехи, а также человеческий фактор, что приводит к пакетным ошибкам. В связи с этим нахождение вероятности ошибок затруднительно и можно получить только их приближённое значение.

После определения вероятности возникновения ошибки надо оценить пропускную способность канала связи и размер данных, которые мы можем передать. В случае, если пропускная способность крайне мала, рекомендуется

использовать вокодер. Это связано с тем, что максимальное число возможных фонем в языке (по разным источникам) колеблется в пределах от 81 (языки Кавказа) до 141 (языки Южной Африки) (см. [17][18]), в то время как среднее число фонем в большинстве языков – 40-50 единиц. Для кодирования числа 141_{10} нужно 8 бит ($1000\ 1011_2$). Если мы будем использовать липридер, то нам нужно будет как минимум закодировать 6 чисел (см. Листинг 1 в 3.1), на кодирование каждого из которых требуется больше одного байта.

После определения числа информационных бит в посылке, надо сопоставить их с числом бит, которые могут быть повреждены (с учётом интерливинга), и с доступной для абонента пропускной способностью канала.

Разность между пропускной способностью и числом информационных бит – это размер избыточной информации, который мы можем ввести.

Заключение

В результате выполнения работы были исследованы методы обработки речи для передачи по каналу связи и предложены наиболее удачные варианта из предложенных методов.

На основании исследования были спроектированы и разработаны библиотеки, отвечающие поставленной задаче и выполняющей следующие действия:

- канальное кодирование и декодирование данных;
- перемежение;
- обработка речевого сигнала.

Стоит отметить, что свободно распространяемых аналогов для данной библиотеки.

Список использованных источников

1. Избыточное кодирование информации: [Электронный документ]. (http://all-ht.ru/inf/systems/p_0_11.html). Проверено: 13.05.2016.
2. Код Хэмминга: [Электронный документ]. (http://informkod.narod.ru/5_3item.htm). Проверено: 13.05.2016.
3. Исправление ошибок с помощью циклических кодов: [Электронный документ]. (http://ido.tsu.ru/iop_res1/kodi/index.php-mod=article&id=196.htm). Проверено: 13.05.2016.
4. GEO-Mobile Radio Interface Specifications; Part 5: Radio interface physical layer specifications; Sub-part 3: Channel Coding: [Электронный документ]. (http://www.etsi.org/deliver/etsi_ts/101300_101399/1013770503/01.01.01_60/ts_1013770503v010101p.pdf). Проверено: 13.05.2016.
5. Способы задания сверточного кода: [Электронный документ]. (http://sernam.ru/book_tec.php?id=95). Проверено: 13.05.2016.
6. Алгоритм декодирования Витерби: [Электронный документ]. (http://sernam.ru/book_tec.php?id=96). Проверено: 13.05.2016.
7. Что такое вокодер и липридер?: [Электронный документ]. (http://www.bnti.ru/showart.asp?aid=720&lvl=01.02.09.). Проверено: 13.05.2016.
8. Кодирование и декодирование речевого сигнала: [Электронный документ]. (http://www.sbi-telecom.ru/kodirovanie-i-dekodirovanie-rechevogo-signala.html). Проверено 13.05.2016.
9. Речевой кодер с линейным предсказанием и использованием анализа через синтез: [Электронный документ]. (http://www.freepatent.ru/patents/2163399). Проверено 13.05.2016.
10. TIOBE Index: [Электронный документ]. (http://www.tiobe.com/tiobe_index). Проверено 16.05.2016.
11. Какой язык программирования будет наилучшим для изучения в 2015 году?: [Электронный документ].

(https://habrahabr.ru/post/260797/). Проверено 18.05.2016.

12. Основы программной инженерии. Жизненный цикл ПО. 2015: [Электронный документ]. (http://kspt.icc.spbstu.ru/media/files/2015/course/se/SE2015_01_LifeCycle.pdf). Проверено 16.05.2016.
13. Расчёт вероятности ошибки в цифровых каналах связи: [Электронный документ]. (http://www.telesputnik.ru/archive/pdf/181/70.pdf). Проверено: 19.05.2016.
14. Вероятность ошибки при передаче информации по каналам связи с пакетным распределением ошибок: [Электронный документ]. (http://info.sernam.ru/book_codb.php?id=54). Проверено: 19.05.2016.
15. Расчет эквивалентной вероятности ошибочного приема двоичного элемента: [Электронный документ]. (http://edu.dvgups.ru/METDOC/GDTRAN/YAT/TELECOMM/TEOR_PERED_SIGN/METOD/HARAK_SV/Stroev_7.htm). Проверено: 19.05.2016.
16. Методика оценки вероятности ошибочного приёма кодового слова с учётом разбиения на блоки и локализации участков:[Электронный документ]. (http://www.science-education.ru/ru/article/view?id=9789). Проверено: 19.05.2016.
17. Языки – статистика и факты: [Электронный документ]. (http://langopedia.ru/2012/07/02/языки-–-статистика-и-факты/). Проверено: 19.05.2016.
18. Основы лингвистической типологии: Учебно-методическое пособие : [Электронный документ]. (http://window.edu.ru/catalog/pdf2txt/538/59538/29606?p_page=4). Проверено: 19.05.2016.

Приложение 1

Листинг 1. Метод кодирования Хэмминга (4, 7), код ошибки совпадает с номером ошибочного бита

```
void bm_2(int to_encode[], int out[]){
     int to_encode_size, i,
          bit_1, bit_2, bit_3, bit_4, bit_5, bit_6, bit_7;
…
/*
     Алгоритм кодирования Хэмминга (4,7):
     биты 3, 5, 6, 7 - информационные; биты 1, 2, 4 -
контрольные.
     Контрольные биты образуются через сумму по модуль 2 между:
     1: 3, 5, 7 | 2: 3, 6, 7 | 4: 5, 6, 7)
     */
     for (i = 0; i < to_encode_size; i++){
          printf("Data to ecnode: %d\n", to_encode[i]);
          //Информационные биты:
          bit_7 = (to_encode[i] & 1);
          bit_6 = (to_encode[i] & 2) >> 1;
          bit_5 = (to_encode[i] & 4) >> 2;
          bit_3 = (to_encode[i] & 8) >> 3;

          //Контрольные биты:
          bit_1 = bit_3 ^ bit_5 ^ bit_7;
          bit_2 = bit_3 ^ bit_6 ^ bit_7;
          bit_4 = bit_5 ^ bit_6 ^ bit_7;

          //Заполняем результат:
          out[i] = bit_1 | (bit_2 << 1) | (bit_3 << 2) | (bit_4
<< 3)
               | (bit_5 << 4) | (bit_6 << 5) | (bit_7 << 6);

     }
}
```

Листинг 2. Метод декодирования Хэмминга (4, 7), код ошибки совпадает с номером ошибочного бита

```
void De_bm_2(int to_decode[], int out[])
{
int to_decode_size, i,
          bit_1, bit_2, bit_3, bit_4, bit_5, bit_6, bit_7,
          control_bit_1, control_bit_2, control_bit_3, control_code
= 0;
…
     for (i = 0; i < to_ decode _size; i++){

          //Разбиваем на биты:
          bit_1 = (to_decode[i] & 1);
```

```
          bit_2 = (to to_decode[i] & 2) >> 1;
          bit_3 = (to to_decode[i] & 4) >> 2;
          bit_4 = (to to_decode[i] & 8) >> 3;
          bit_5 = (to to_decode[i] & 16) >> 4;
          bit_6 = (to to_decode[i] & 32) >> 5;
          bit_7 = (to to_decode[i] & 64) >> 6;

          //Составляем контрольные биты
          control_bit_1 = bit_1 ^ bit_3 ^ bit_5 ^ bit_7;
          control_bit_2 = bit_2 ^ bit_3 ^ bit_6 ^ bit_7;
          control_bit_3 = bit_4 ^ bit_5 ^ bit_6 ^ bit_7;
          control_code = control_bit_1 | (control_bit_2 << 1) |
(control_bit_3 << 2);

          if (control_code == 0)
               out[i] = bit_3 | (bit_5 << 2) | (bit_6 << 1) | (bit_7
<< 3);
          else{
               //Повреждён только контрольный бит
               if (control_code == 1 || control_code == 2 ||
control_code == 4)
                    out[i] = bit_3 | (bit_5 << 2) | (bit_6 << 1) |
(bit_7 << 3);
               else{//Повреждён информационный бит.
                    switch (control_code){
                    case 3:
                         bit_3 = (~bit_3 & 1);
                         break;
                    case 5:
                         bit_5 = (~bit_5 & 1);
                         break;
                    case 6:
                         bit_6 = (~bit_6 & 1);
                         break;
                    case 7:
                         bit_7 = (~bit_7 & 1);
                         break;
                    default:
                         break;
                    }
                    out[i] = bit_3 | (bit_5 << 2) | (bit_6 << 1) |
(bit_7 << 3);
               }

          }
     }
}
```

Приложение 2

Листинг 3. Файл Channel_coding.h

```
/*
Данный файл содержит прототипы ф-ций для канальных кодеков.
*/

#ifndef CHANNEL_CODING
#define CHANNEL_CODING

void bm_1(int to_encode[], int out[]);
void bm_2(int to_encode[], int out[]);
void bm_2(int to_encode[], int out[]);
void cm_1(int to_encode[], int out[]);
void cm_2(int to_encode[], int out[]);
void De_bm_1(int to_decode[], int out[]);
void De_bm_2(int to_decode[], int out[]);
void De_bm_3(int to_decode[], int out[]);
void De_cm_1(int to_decode[], int out[]);
void De_cm_2(int to_decode[], int out[]);

#endif
```

Листинг 4. Файл Interleaving.h

```
/*
Данный файл содержит прототипы ф-ций для интерливинга.
*/

#ifndef INTERLEAVING_LIB
#define INTERLEAVING_LIB

void Interleaving(int to_interleaving[]);
void DE_interleaving(int to_DEinterleaving[]);

#endif
```

Приложение 3

Листинг 5. Файл Analytic.h

```
/*
Данный файл содержит прототипы ф-ций аналитической части
речевого кодера.

Тип кодера задаётся символом (char):
V - обычный вокодер (определяет и передаёт № фонемы).
L - липредер (вокодер с линейным предсказанием - снимает и
передаёт
     параметры фильтра).
H - гибридный вариант (определяет как № фонемы, так и параметры
фильтра).
*/

#ifndef ANALYTIC
#define ANALYTIC

#include "Voice_type.h"

bool Search_pause(Voice_type VT);
Voice_type LPC(char coder_type, Voice_type VT); //Linear
Prediction
//Next 2 functions - for delay line:
void Set_current_sound(Voice_type VT);
Voice_type Get_previous_sound();
void Compare_with_adaptive_dictionary(Voice_type VT, Voice_type
VT_old);
void get_signal_parameter(char coder_type, Voice_type VT1,
Voice_type VT2);
int Phoneme_number_generator(char coder_type, Voice_type VT);
void Save_best_phoneme_id(int id);
void Compare_with_fixed_dictionary(Voice_type VT);

#endif
```

Листинг 6. Файл Synthetic.h

```
/*
Данный файл содержит прототипы ф-ций синтетической части
речевого кодера.

Тип кодера задаётся символом (char):
V - обычный вокодер (определяет и передаёт № фонемы).
L - липредер (вокодер с линейным предсказанием - снимает и
передаёт
параметры фильтра).
H - гибридный вариант (определяет как № фонемы, так и параметры
фильтра).
*/
```

```
#ifndef SYNTHETIC
#define SYNTHETIC

#include "Voice_type.h"
#include "Channel_coding.h"

void Set_filter_parametres(Voice_type VT);
void Set_phoneme_num(int phoneme_num);
//Используем библиотеку для канального кодирования:
void Code_generator(char coder_type, int block_code_id, int
conv_code_id);

extern int CONV_CODE[500];
extern int BLOCK_CODE[100];
#endif
```

Листинг 7. Файл Speech_decoder.h

```
/*
Данный файл содержит прототипы ф-ций для речевого декодера.
*/

#include "Voice_type.h"
#include "Channel_coding.h"
#include "Interleaving.h"

#ifndef SPEECH_DECODER
#define SPEECH_DECODER

Voice_type get_filter_parameters(int input_data[], int
block_code_id, int conv_code_id);
int get_phoneme_number(int input_data[], int block_code_id, int
conv_code_id);
Voice_type made_voice(char coder_type, int input_data[], int
block_code_id, int conv_code_id);
Voice_type get_ALL(int input_data[], int block_code_id, int
conv_code_id);
Voice_type channel_decoding(int input_data[], int block_code_id,
int conv_code_id);

#endif
```

Заключительный лист работы

Выпускная работа выполнена мною самостоятельно. Использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

Список использованных источников содержит _____ наименований.

Работа выполнена на_____ страницах.

Приложения к работе на _____ страницах.

Один экземпляр сдан в директорат.

Подпись_____/_____/

(фамилия, инициалы)

Дата «_____» _____ 20_____г.