Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

ОСНОВНЫЕ НАПРАВЛЕНИЯ ЛИНГВИСТИЧЕСКОГО ОБЕСПЕЧЕНИЯ НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ (ЯЗЫКОВЫЕ ТЕХНОЛОГИИ)

сборник учебно-методических материалов
для направления подготовки 45.04.03 – Фундаментальная
и прикладная лингвистика

Направленность (профиль) образовательной программы «Иностранные языки и речевые технологии»

Благовещенск, 2017

Печатается по решению
редакционно-издательского совета
филологического факультета
Амурского государственного
университета

Составители: Андросова С. В., Андросов Е. Ю.

Основные направления лингвистического обеспечения новых информационных технологий (языковые технологии): сборник учебно-методических материалов для направления подготовки 45.03.03. – Благовещенск: Амурский гос. ун-т, 2017.

ТЕМА 1. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ПРИКЛАДНАЯ ЛИНГВИСТИКА

Прикладная лингвистика (далее – ПЛ) требует строгого структурного подхода к языку и отводит важную роль математике. ПЛ рассматривает *язык как сложную алгоритмическую систему* со своими алгоритмами функционирования и законами развития. Это напрямую связывает ее с информационными технологиями (далее – ИТ).

Информационные технологии (ИТ)

Определение

ИТ — это совокупность законов, методов и средств получения, хранения, передачи, распространения, преобразования информации с помощью компьютеров.

Предпосылки появления и развития ИТ (Зубов А.В., Зубова И.И., с. 8)

1. Создание ПК с большой памятью и большой скоростью выполнения операций.
2. Разработка звуковых плат, дающих возможность воспроизводить и записывать речь, звуки и музыку в большом диапазоне частот.
3. Изобретение видеоплат, позволяющих выводить на экран компьютеров изображение с телеэкранов и видеомагнитофонов.
4. Разработка мультимедийных компьютеров, позволяющих воспроизводить на экране дисплеев цвет, звук, музыку, движение.
5. Создание процессоров и устройств, способных передавать информацию в сети от одного компьютера к другому.
6. Разработка модемов, позволяющих передавать информацию на далекие расстояния.
7. Создание оргтехники, позволяющей осуществлять высокоскоростную печать документов и их копирование.

ИТ в лингвистике — это совокупность законов, методов и средств получения, хранения, передачи, распространения, преобразования информации о языке и законах его функционирования с помощью компьютеров.

ИТ в основном соотносятся с задачами ПЛ.

Существуют разные перечни задач. Приведем два.

Перечень 1

1. Создание автоматических систем анализа и синтеза речи
2. Автоматические методы переработки текстовой информации
3. Создание автоматизированных систем информационного поиска
4. Составление автоматических словарей и систем машинного перевода
5. Разработка методов автоматического аннотирования, реферирования и перевода
6. Разработка экспертных систем
7. Лингвистическое обеспечение АСУ(автоматизированные системы управления)
8. Стандартизация научно-технической терминологии

Перечень 2 (Зубов, Зубова, 2004: 8-9)

Комплексные задачи в соотнесении с задачами ПЛ

Создание систем

1) искусственного интеллекта;
2) автоматического перевода;
3)автоматического аннотирования и реферирования;
4) порождения текстов;
5) обучения языку;
6) понимания устной речи;
7) генерации устной речи;
8) автоматизированных информационно-поисковых систем;
9) атрибуции и дешифровки анонимных и псевдоанонимных текстов;

Разработка

10) баз данных (электронные словари, каталоги, реестры);
11) программных оболочек электронных словарей;
12) систем передачи информации в сети Интернет.

Частные задачи в соотнесении с задачами ПЛ

Автоматизация процессов

1) построения словарей текстов;
2) морфологического анализа слова;
3) определения значения многозначного слова;
4) синтаксического анализа предложения;
5) поиска слова в словаре;
6) порождения предложения и т.д.

Структурные составляющие прикладной лингвистики

1. Структурная лингвистика – это совокупность взглядов на язык и методов его исследования, в основе которых лежит понимание языка как знаковой системы с четко выделенными структурными элементами (единицами языка, их классами и пр.) и стремление к строгому (как в точных науках) формальному описанию языку. Свое название СЛ получила благодаря особому вниманию к структуре языка, которая представляет собой сеть отношений

(противопоставлений) между элементами языковой системы, упорядоченных и находящихся в иерархической зависимости в пределах определенных уровней.
Структурное описание языка предполагает такой анализ реального текста, который позволяет выделить обобщенные инвариантные единицы (схемы предложений, морфемы, фонемы) и соотнести их с конкретными речевыми сегментами на основе строгих правил реализации Эти правила определяют границы допустимого варьирования языковых единиц в речи. В зависимости от уровня анализа правила реализации формулируются как правила позиционного распределения конкретных, например, принцип дополнительной дистрибуции в фонологии и морфологии (дистрибутивный анализ), или как трансформационные правила в синтаксисе (при трансформационном анализе) регулирующие переход от инвариантной глубинной структуры предложения к множеству ее реализации.
На базе СЛ развилась порождающая грамматика (генеративная лингвистика); идеи структурного анализа во многом определили постановку и решение задач, связанных с машинным переводом; СЛ открыла дорогу для широкого проникновения в лингвистику математических методов (математическая лингвистика).
На СЛ оказали влияние: Сепир, Блумфилд. Ф.де Соссюр, Н.С. Трубецкой, Р. Якобсон, Реформатский.
Успешно применяли методы СЛ: Апресян, Арутюнова, Гак, Зализняк, Звегинцев, Мельчук, Успенский и др.
2. Контрастивная лингвистика (сопоставительная лингвистика) – сопоставительное изучение двух, реже нескольких языков для выявления их сходств и различий на всех уровнях языковой структуры с целью типологической классификации языков. Как правило, контрастивная лингвистика оперирует материалами на синхронном срезе языка.
КЛ появилась и интенсивно развивалась в 50 гг. 20 в., однако ее появление подготовили работы Е.Д. Поливанова, Бодуена де Куртенэ, Л.В. Щербы с изложением теоретических основ сравнения родного и ин. Языков.
В 70 гг. контрастивные исследования в отдельных странах (гл. образом в США) использовали порождающую модель Хомского, с возведением явлений двух сопоставляемых языков к общей глубинной структуре; в наст. время наблюдается отход от этой методики в пользу сруктурно-функционального подхода.
3. Математическая лингвистика (МЛ) — это математическая дисциплина, предметом которой является разработка формального аппарата для описания строения естственных и некоторых искусственных языков.
Возникла в 50 годы 20 в. Одним из главных стимулов появления математической лингвистики послужила назревшая потребность в уточнения основных лингвистических понятий.
Методы МЛ имеют много общего с с методами мат. логики — мат. дисциплины, занимающейся изучением строения мат. рассуждений, - и в особенности таких ее разделов, как теория алгоритмов и теория автоматов.
4. Компьютерная лингвистика (КЛ) задает общую ориентацию на использование компьютеров для решения разнообразных научных и практических задач, никак не ограничивая способы решения этих задач.
5. Вычислительная лингвистика (ВЛ) может пониматься более узко, чем КЛ, так как даже при широкой трактовке понятия «вычисление» за его пределами остаются такие стороны решения лингвистических задач, как, например, представление знаний, организация банков языковых данных, психолингвистические аспекты взаимодействия человека и компьютера и другие.

Модели

Метод моделирования - центральный исследовательский метод в науке. Моделирование в науке - это выяснение свойств какого-либо предмета при помощи построения его модели.

Моделью можно назвать образ какого-либо объекта, используемый в определенных условиях в качестве его заместителя (фотография в паспорте - модель человека).

Свойства моделей:

· условность

· образ может быть не только материальным, но и мысленным и передаваться посредством знаковой системы

· моделью может быть не только образ, но и прообраз оригинала

·модель чаще всего является гомоморфной оригиналу (то есть многим элементам оригинала соответствует меньшее количество элементов модели в отличие от изоморфизма).

Типы моделей: модели объяснительного типа или объясняющие модели и модели воспроизводящего типа. От первых не ожидается порождения языкового продукта; эта модель должна непротиворечиво объяснять действие напр, языка в целом, если он моделируется, или каких-либо его частей моделируемого явления, если моделируются эти части (Марчук, 200...: 21). Вторые обретают ценность, когда воспроизводимый ими результат подобен тому, который был бы получен в результате деятельности человека (там же).

Понятие лингвистической модели возникло в структурной лингвистике, но вошло в научный обиход в 60-70 гг. 20 в. с возникновением мат. лингвистики и проникновением в лингвистику мат. Методов. Содержание термина "модель" в современной лингвистике в значительной степени охватывалось ранее термином "теория" (особенно Ельмслевым). Считается, что наименования модель заслуживает лишь такая теория. которая достаточно эксплицитно изложена и в достаточной степени формализована (в идеале каждая модель должна допускать реализацию на ЭВМ).

Главная цель моделирования в лингвистике - это моделирование целостной языковой способности человека.

Устройство и функционирование языка недоступны непосредственному наблюдению, поэтому изучение языка и продуктов его деятельности — текстов разного рода — осуществляется главным образом с помощью моделирования (Марчук, 200...: 20-21. Марчук Ю.Н. Компьютерная лингвистика.).

Собственно лингвистические модели:

- модели речевой деятельности, процессуальный модели (самые сложные)
- модели языковой системы, языковой структуры (тоже очень сложные)
- модель памяти и др.

Лингвистическое моделирование необходимо предполагает использование абстракции и идеализации. Отображая релевантные существенные (с точки зрения исследования) свойства оригинала и отвлекаясь от несущественных, модель выступает как некоторый абстрактный идеализированный объект.

Всякая модель строится на основе гипотезы о возможном устройстве оригинала и представляет собой функциональный аналог оригинала. что позволяет переносить знания с модели на оригинал.

Критерием адекватности модели является эксперимент.

В идеале модель должна быть формальной (т.е. в ней должны быть в явном виде и однозначно заданы исходные объекты, связывающие их отношения и правила обращения с ними) и обладать объяснительной силой (т.е. не только объяснять факты или данные

экспериментов, необъяснимые с точки зрения уже существующей теории, но и предсказывать неизвестное раньше, хотя и принципиально возможное поведение оригинала, которое позднее должно подтверждаться данными наблюдения или экспериментов).
Конструирование модели - не только одно из средств отображения языковых явлений, но и объективный практический критерий проверки истинности знаний о языке. В единстве с другими методами изучения языка моделирование выступает как средство углубления познания скрытых механизмов речевой деятельности, его движения от относительно примитивных к более содержательным моделям, полнее раскрывающим сущность языка.
Внутри языка как системы существует принцип моделирования: одни его подсистемы моделируют другие, например, система письменной речи является моделью устной речи; внутри письменной речи мы имеем дело с несколькими моделями (печатной, рукописной); план выражения является моделью плана содержания.
Метод моделирования обычно опирается на знаковые систем, но язык - сам знаковая система, т.е. слова мы моделируем при помощи слов.

ОСНОВНЫЕ СОСТАВЛЯЮЩИЕ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

1. Структура информационных технологий
2. Теоретические основы информационных технологий.
3. Будущее информационных технологий.
4. Алгоритмы и их свойства.
5. Элементы языка программирования Turbo Pascal.

Структура информационных технологий

В составе современных информационных технологий можно выделить следующие составляющие:

1) теоретические основы информационных технологий;
2)методы решения задач информационными технологиями;
3)средства решения задач, используемые в информационных технологиях:
   а) аппаратные средства;
   б) программные средства.

Рассмотрим подробнее эти составляющие.

2. Теоретические основы информационных технологий.
Теоретическую основу информационных технологий составляют важнейшие понятия и законы информатики. В свою очередь понятие «информатика» тесно связано с понятием «информация».
Слово информация (от лат. informatio — разъяснение, изложение) в обычном житейском понимании обозначает некоторые сведения о внешнем и внутреннем мире, которые мы используем Для регулирования своего поведения. Более строго это понятие раскрывается разными способами. Выберем один из них, который более всего подходит к рассматриваемым в пособии лингвистическим задачам, и определим информацию как определенным образом связанные сведения, данные, понятия, отраженные в нашем сознании и изменяющие наши представления о реальном мире.

Информация обладает разными свойствами. Наиболее важными из них являются: ценность, достоверность, полнота, актуальность, логичность, компактность.
Ценность информации определяется тем, насколько она важна для позволяющих получателю информации достичь своей цели.

Полнота — насколько много имеется сведений, позволяющих получателю достичь своей цели.
Актуальность информации определяется необходимостью ее немедленного использования для достижения какой-либо цели.
Компактность информации — способность представить ее в наиболее сжатом виде. Достоверность и логичность (не требуют особых пояснений).

Выделяют различные виды информации. При этом для ее классификации по видам разработано много подходов, использующих разнообразные признаки и особенности информации. Так, в зависимости от того, какими органами чувств воспринимается информация, ее делят на визуальную, аудиальную (звуковую, фонетическую), аудиовизуальную, тактильную. По направленности информации всем членам общества или каким-то его группам различают информацию массовую, предназначенную для всех членов общества, и специальную — для специалистов в различных областях науки, техники, культуры, производства. Специальную информацию подразделяют на научную, техническую, производственную, эстетическую и т.п.
В каждом виде специальной информации выделяют подвиды. Например, в зависимости от области науки и научной информации выделяют информацию физическую, математическую, биологическую, лингвистическую и т.д. Так, лингвистической информацией называют множество определенным образом связанных сведений, данных, понятий о языке и правилах его функционирования, отраженных в нашем сознании и влияющих на наше речевое поведение.

Слово «информатика» также не имеет единого определения. С современной точки зрения информатика — это наука о законах и методах получения, хранения, передачи, распространения, преобразования и использования информации в естественных и искусственных системах с применением компьютера.
В зависимости от вида информации выделяют различные типы информатики. Так, различают информатику социальную, экономическую, научную научно-техническую, статистическую, биологическую, медицинскую и т. п.
Наука, изучающая законы и методы организации и переработки с помощью компьютера лингвистической информации, называется лингвистической информатикой. Вспомнив, что понимается под лингвистической информацией, можно сказать, что объектом исследования лингвистической информатики будет структура слов, словосочетаний, предложений, текстов. Ее интересуют правила, объединяющие нижестоящие языковые единицы в вышестоящие, правила перевода предложений и текстов, способы построения рефератов и аннотаций, пути обучения языкам и целый ряд других вопросов, связанных с языком и речью.

ТЕМА 2. АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИТ

- Средства аппаратного обеспечения: компьютер и периферийные устройства
- Программное обеспечение (ПО):

1. Системное ПО (операционные системы — ОС, дополнительные системные программы — утилиты и драйверы);
2. Прикладное ПО;
3. Прикладные инструментальные средства.

ОС — главная программа, загружаемая в оперативную память компьютера после его включения.

Функции ОС

1. Управление работой ПК (внутренние функции ПК, контроль за выполнением операций, распределение памяти и т.д.)
2. Запуск и выполнение прикладных программ.
3. Обеспечение пользователю удобного способа общения с ПК.

Утилита — программа, расширяющая возможности ОС.

Виды утилитов

1. Программы-архиваторы;
2. Программы для создания резервных копий;
3. Антивирусные программы;
4. Программы для диагностики копьютера.

Драйверы — программы для управления устройствами компьютера (чаще ввода-вывода): драйверы клавиатуры, мыши, принтера, сканера и т.д.

Прикладные программы

1. Текстовые процессоры (MS Words, Open Office writer, Notepad etc)
2. Программы автоматического преобразования графической информации в текстовую (скан в текстовый файл) (FineReader etc);
3. Системы машинного перевода;
4. Системы автоматического аннотирования и реферирования;
5. Настольно-издательские системы (PageMaker, Scribus);
6. Обучающие программы;
7. Экспертные системы, напр. Для диагностики неисправности приборов, при определении болезни и метода ее лечения и т.п.;
8. Табличные процессоры (MS Excel, OpenOffice Calc) для обработки экономических и статистических данных;
9. СУБД (Access для Windows, MySQL для Linux) позволяют осуществлять создание и модификацию больших совокупностей структурированных определенным образом данных (баз данных), а так же поиск в них информации.

Прикладные инструментальные средства. Языки программирования

Изначально никаких языков программирования не существовало. Программирование осуществлялось в машинных кодах процессора (любой набор цифр), под который писалась программа. Это было неудобно, поскольку слабо читаемо. И писать такие программы, и находить в них ошибки тоже было сложно. Для упрощения этого процесса был придуман язык низкого уровня Ассемблер. В нем машинные команды записывались командами

естественного языка. Напр., команда сложения обозначалась тремя буквами «ADD», команда вычитания «SUB» (subtract), команда сравнения «CMP» (compare), команда перехода к другому адресу «JMP» (jump) и т.п.

Проблема состояла в том, что каждый процессор имел свой собственный набор машинных команд и большинство семейств процессоров имели сильно отличающийся друг от друга язык Ассемблера. Поэтому стали разрабатывать так называемые языки программирования высокого уровня, которые в больше походили на естественный язык общения, программы на этом языке могли писаться быстрее, ошибки на этих языках было легче обнаруживать и исправлять. Но перед тем как запускать такие программы, они должны были быть переведены в язык, понятный для компьютера, т.е. язык двоичных машинных кодов. Процесс такого перевода называется трансляцией. Трансляторы бывают двух типов:

1) программы-интерпретаторы;
2) программы-компиляторы.

Интерпретаторы берут программу на языке высокого уровня и исполняют ее строчка за строчкой, останавливаясь в случае ошибки.

Достоинства и недостатки: интерпретаторы позволяют быстро разрабатывать небольшие программы, но вместе с тем требуют больших накладных расходов, поскольку помимо собственно программы пользователя в памяти компьютера должна находиться еще и программа-интерпретатор. Написание сложных программ поэтому затруднено.

Компиляторы сначала переводят всю программу в машинные коды, которые исполняются уже без участия компилятора. Перед исполнением программы необходимо пройти стадию компиляции, т.е. перевода программы на языке высокого уровня в машинные коды.

Достоинства и недостатки: исполняется быстро, поскольку не требует дополнительных накладных расходов, но исправление ошибок в этих программах требует исправления первоначальной версии высокого уровня, а затем проведения повторной компиляции.

Одним из самых распространенных *интерпретаторов* до недавнего времени являлся язык BASIC. В настоящее время в связи с распространением интернета широкое распространение получили языки JAVA, PERL (Practical Extraction and Report Language — практический язык для извлечения данных и составления отчётов), PYTHON,

В 60-70 гг. самыми распространенными *компиляторами* были COBOL (Common Business Oriented Language), FORTRAN (Formula Translator), PL/1 (Program Language 1). В настоящее время для программирования в основном используется универсальный язык C/C++ (C с добавлением объектно-ориентированного программирования), а для обучения программированию — язык PASCAL.

Компилятор обрабатывает программу, написанную на языке Free Pascal в два этапа.

1. Компилятор анализирует, какие внешние библиотеки3 нужно подключить, разбирает текст программы на составляющие элементы, проверяет синтаксические ошибки и в случае их отсутствия формирует объектный код (в Windows — файл с расширением .obj, в Linux — файл с расширением .o). Получаемый на этом этапе двоичный файл (объектный код) не включает в себя объектные коды подключаемых библиотек.

2. Компоновщик подключает к объектному коду программы объектные коды библиотек и генерирует исполняемый код программы. Этот этап называется компоновкой или сборкой программы. Полученный на данном этапе исполняемый код программы можно запускать на выполнение.

Особняком стоит PHP (Hypertext Preprocessor), который служит для оформления и динамического создания страниц HTML (hyper text markup language – язык гипертекстовой разметки).

Состав алгоритмического языка

В письменном тексте на естественном языке выделяют четыре основных элемента: символы, слова, словосочетания и предложения. Аналогично и в алгоритмическом языке с тем отличием, что слова там принято называть лексемами, словосочетания — выражениями, а предложения — операторами.

1. Символы языка = алфавит — основные неделимые знаки (прописные и строчные латинские буквы, знак подчеркивания; знаки препинания; различные виды скобок; слэши, проценты, больше, меньше, плюс, тильда и т.п.).

2. Лексемы. Состоят из символов алфавита. К лексемам относят идентификаторы, ключевые / зарезервированные слова, знаки операций, константы, разделители: скобки, точка, запятая, пробельные символы.

2.1. Идентификаторы.

Идентификатор — это имя программного объекта. Общий для разных языков момент — в имени могут быть использованы только буквы, цифры и знак нижнего подчеркивания. Детали могут различаться. Например, на языке C / C++ чередование или выбор строчных и прописных букв образует разные имена, а на языке Free Pascal — нет.

2.2. Зарезервированные / ключевые слова

Это идентификаторы, которые имеют специальное значение для компилятора. Это значение строго определено, использование в другом значении невозможно. Ключевые слова варьируют от языка к языку, хотя ряд слов совпадают. Например, do, while, const, for, if и ряд других совпадают у C++ и Free Pascal.

ПРИНЦИПЫ РЕШЕНИЯ ЛИНГВИСТИЧЕСКИХ ЗАДАЧ МЕТОДОМ МОДЕЛИРОВАНИЯ

Экспериментальные методы и модели

При использовании информационных технологий в лингвистике на первый план выходит метод компьютерного моделирования.

Выделяют следующие типы моделей.

1. Структурные модели, служащие для изучения и описания внутреннего строения некоторого объекта. Например, такая модель строится, если необходимо изучить систему согласных какого-либо языка или устройство речевого аппарата человека.
2. Функциональные модели, позволяющие изучать поведение некоторого объекта, течение некоторого процесса или же этапы реализации некоторого явления. Например, функциональная модель строится, если необходимо смоделировать процесс создания некоторого текста человеком. Такая же модель создается для объяснения процесса перевода текста с одного языка на другой.
3. Динамические модели, которые создаются при необходимости найти объяснение некоторых процессов или явлений в их временном развитии. Так, если требуется узнать, как со временем менялось произношение некоторого слова, строят динамическую модель такого процесса.

Модели различаются по направленности на определенный объект и по используемым средствам моделирования — алгоритму или исчислению.

Алгоритм - строгая последовательность предписывающих правил .

Исчисление - множество разрешающих правил (порядок выполнения не важен) .

Понятие лингвистической модели возникло в структурной лингвистике, но вошло в научный обиход в 60-70 гг. 20 в. с возникновением математической лингвистики и проникновением в лингвистику математических методов.

Особая роль в лингвистике отводится функциональным моделям, позволяющим раскрыть суть функционирования языка, механизма производства и восприятия речи и текста. Нельзя заглянуть в мозг человека и посмотреть, как в нем осуществляются операции с буквами, звуками, словами, предложениями при всевозможном использовании языка. Поэтому для решения таких задач в рамках функциональных моделей выделяют воспроизводящие инженерно-лингвистические модели (ВИЛМ). Они представляют собой компьютерные системы, поведение которых, с одной стороны, имитирует поведение реальных лингвистических объектов, а с другой — позволяет хотя бы частично воспроизвести эти реальные объекты.

Типы лингвистических моделей:
1. По охвату структуры языка:
а) общие (глобальные) стремятся охватить весь язык: <VG> (vocabulary, grammar) ;
б) частные: фонетическая модель русского языка, модель системы гласных .
2. По типологическому статусу:
а) универсальные стремятся охватить все языки мира: <VG> ;
б) специфические характерны для определенного языка или группы языков: мягкость - твердость согласных рус. языка (не действует в англ., нем., франц.) .
3. По гносеологическому статусу:
а) модели языка ;
б) модели лингвистических знаний различные фонетические школы ;
в) модели деятельности лингвиста .
4. По отраженному аспекту языка и речевой деятельности:
а) анализирующие модели моделируют процесс понимания, используют логическое средство — алгоритм ;
б) синтезирующие модели моделируют процесс вербализации, смысла речевого отрезка
в) порождающие модели: порождающая грамматика Н. Хомского объект моделирования — множество правильных речевых отрезков составляются правила различения приемлемого и неприемлемого; логический средство - исчисление; не служат выражением смысла; на выходе - цепочки элементов (грамматически правильных предложений);
г) собственно структурные модели — основа всех остальных; объект моделирования - структура языка как таковая; логический аппарат — логика отношений и классов. Пример: грамматический словарь Зализняка.
5. По конечной цели исследования:
а) теоретические;
б) описательные;
в) прикладные.
6. По используемым методам
а) математические модели;
б) психологические модели;
в) социологические модели.
7. По функциональному статусу:
а) абстрактно обобщающие модели;
б) действующие.
8. По используемым материальным средствам:

а) графические;
б) символьные;
в) компьютерные.

Частная модель обычно входит в набор частных моделей, описывающий определенный уровень языка: фонологический, морфологический, синтаксический, лексико-семантический.

Лингвистическое моделирование необходимо предполагает использование абстракции и идеализации: отображает релевантные свойства оригинала и отвлекается от несущественных свойств. Всякая модель строится на основе гипотезы о возможном устройстве оригинала и представляет собой функциональный аналог оригинала. что позволяет переносить знания с модели на оригинал.
Критерием адекватности модели является эксперимент.
В идеале модель должна быть формальной (т.е. в ней должны быть в явном виде и однозначно заданы исходные объекты, связывающие их отношения и правила обращения с ними) и обладать объяснительной силой (т.е. не только объяснять факты или данные экспериментов, необъяснимые с точки зрения уже существующей теории, но и предсказывать неизвестное раньше, хотя и принципиально возможное поведение оригинала, которое позднее должно подтверждаться данными наблюдения или экспериментов).

Основные теоретические требования к модели:
1. Полнота модели - способность отражать все факты, на которые она рассчитана, на охват которых она претендует.
2. Простота - удобство, использования как можно меньшего числа средств (символов, правил) для достижения поставленной научной цели.
3. Объяснительная сила - способность модели вскрывать причины наблюдаемых фактов и предсказывать новые факты (например. модели исторического изменения слова; !! системы машинного перевода в очень малой степени объяснительные).
4. Адекватность - свойство максимальной похожести на моделируемый объект, на оригинал, можно свести к объяснительной силе или теоретико- множественному соответствию.
5. Экономность - экономичное использование энергетических и временных ресурсов при применении модели.
6. Точность - возможность выполнения операций представляемым моделью формальным аппаратом.
7. Эстетические свойства - красота модели.

Главный прикладной критерий - удобство модели.

Для моделирования языка очень важны логические средства реализации модели (компьютерное воплощение модели).

Конструирование модели - не только одно из средств отображения языковых явлений, но и объективный практический критерий проверки истинности знаний о языке. В единстве с другими методами изучения языка моделирование выступает как средство углубления познания скрытых механизмов речевой деятельности, его движения от относительно примитивных к более содержательным моделям, полнее раскрывающим сущность языка. Внутри языка как системы существует принцип моделирования: одни его подсистемы моделируют другие, например, система письменной речи является моделью устной речи; внутри письменной речи мы имеем дело с несколькими моделями (печатной, рукописной); план выражения является моделью плана содержания.

Метод моделирования обычно опирается на знаковые систем, но язык - сам знаковая система, т.е. слова мы моделируем при помощи слов.

ТЕМА 3. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ОБРАБОТКЕ ТЕКСТОВ

План

1. Основные задачи обработки текстов.
2. Анализ отдельных слов.
3. Анализ отдельных предложений.
4. Семантический анализ.

1. Основные задачи обработки текстов

Задачи обработки текстов возникли практически сразу вслед за появлением вычислительной техники. Но, несмотря на полувековую историю исследований в области искусственного интеллекта, огромный скачок в развитии ИТ и смежных дисциплин, удовлетворительного решения большинства практических задач обработки текста пока нет. Решением задач обработки текстов с применением ИТ занимается компьютерная лингвистика.

Компьютерная лингвистика – это раздел науки, изучающий применение математических моделей для описания лингвистических закономерностей. Эту область можно разделить на две большие части. Первая изучает способы применения вычислительной техники в лингвистических исследованиях – применение известных математических подходов (например, статистической обработки) для выявления закономерностей. Данные закономерности используются второй частью компьютерной лингвистики, изучающей вопросы осмысления текстов, написанных на естественном языке – создание математических моделей для решения лингвистических задач и разработка программного обеспечения, функционирующего на основе этих моделей.

Вторая часть компьютерной лингвистики тесно соприкасается с разделом искусственного интеллекта, занимающимся разработкой систем обработки текстов, написанных на естественном языке (NLP, Natural Language Processing).

Существует наиболее общая схема обработки текстов (см. рис. 1), инвариантная по отношению к выбору естественного языка (т.е. независимо от того, на каком языке написан исходный текст, его анализ будет проходить все указанные в схеме стадии): 1. Разбиение текста на отдельные предложения; 2. Разбиение текста на отдельные слова; 3. Определение характеристик отдельных слов; 4. Синтаксический анализ

Первые две стадии практически одинаковы для большинства естественных языков. Единственное, где могут проявляться черты, специфичные для выбранного языка - это обработка сокращений слов и обработка знаков препинания (точнее, определение того, какие из знаков препинания являются концом предложения, а какие нет).

Последующие две стадии (определение характеристик отдельных слов и синтаксический анализ), наоборот, очень сильно зависят от выбранного естественного языка. Последняя стадия (семантический анализ), как и первые стадии, мало зависит от выбранного языка, но, это проявляется только в общих подходах к проведению анализа.

Семантический анализ полностью основывается на результатах работы предыдущих фаз обработки текста, которые всегда специфичны для конкретного языка. Следовательно, способы представления их результатов тоже могут сильно варьироваться, оказывая большое влияние не реализацию методов семантического анализа. Результаты анализа, произведенного на ранних стадиях, могут быть многозначны - для выходных параметров указывается не одно, а сразу несколько возможных значений (например, может существовать несколько способов трактовки одного и того же слова). В таком случае последующие стадии должны определять наиболее вероятные значения результатов ранних стадий анализа и уже на их основе проводить дальнейший анализ текста.

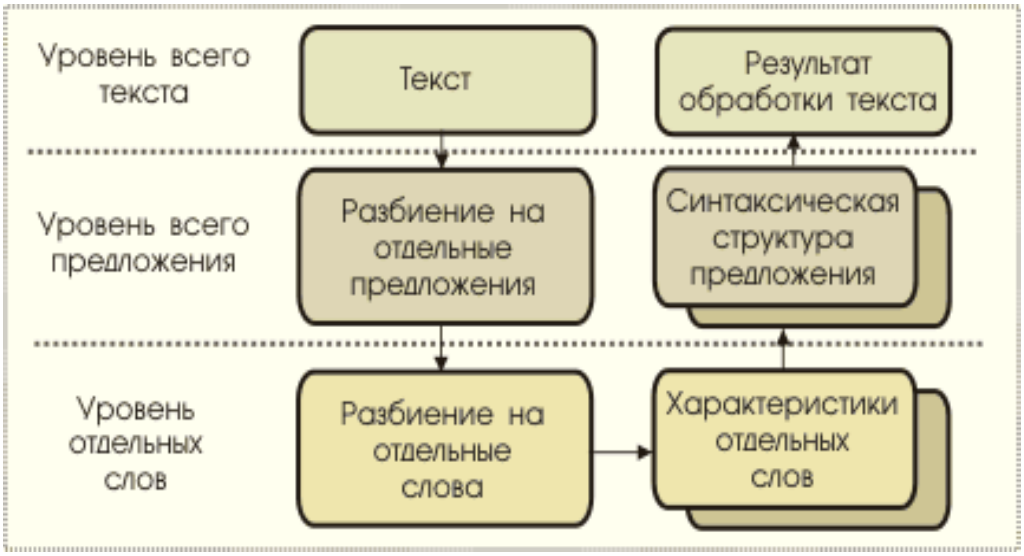

Рисунок 1. Общая схема обработки текстов

Специфические моменты обработки текстов на естественном языке

- обработка сокращений слов;
- обработка знаков препинания: определение того, какие из знаков препинания являются
- концом предложения, а какие нет;
- определение характеристик отдельных слов;
- синтаксический анализ;
- семантический анализ.

Более подробно каждая из стадия анализа текста будет рассмотрена позже в рамках дисциплины «Автоматическая обработка звучащей речи и письменного текста». Компьютерная обработка текстов состоит в поиске и классификации тех или иных языковых объектов, например, грамматических. Посмотрим, как это делается на примере простой задачи по поиску грамматических форм одного из английских глаголов – *to be*.

Автоматический поиск грамматических форм

ЗАДАЧА

Найти в тексте все формы глагола *to be* и вывести на экран с указанием количества употреблений в тексте в абсолютных единицах и в процентах по сравнению с общим количеством слов в тексте.

1. Необходимо объяснить компьютеру, что такое слово. **Слово** — это последовательность буквенных символов или знака «дефис», при этом слово начинается и заканчивается только на буквенные символы; перед и после слова идут один или более разделителей, в качестве которых выступают все небуквенные символы, а также начало или конец файла.

2. Поскольку язык Free Pascal изначально не содержит в себе средства работы с текстовыми форматами текстовых процессоров (Writer), для обработки текстов в нем необходимо перевести текст, в котором будет производиться поиск, в файл PLAIN TEXT (плоский = простой текстовый файл), чтобы не мешало форматирование

(достигается выбором в меню «сохранить как» для writer — текст .txt, для MS Word — текст MS-DOS).

3. Формы глагола to be: is, am, are, was, were, be, been

В области **констант** через массив задаем формы глагола to be с указанием количества форм. Формы записываем большими буквами, поскольку в тексте они могут употребляться в разных регистрах (как большие, так и маленькие), поэтому мы все буквы приводим к верхнему регистру (строки 4 и 28).

В области **переменных** в глобальной области видимости мы описываем следующие переменные: текстовый файл, в котором будет произведен подсчет (строка 6), массив счетчиков для форм глагола to be (строка 7), символ, который читается из файла (читается по одному символу (строка 8)); переменная для промежуточного хранения слова (строка 9); количество всех слов в тексте (строка 10).

В программе используемые следующие **процедуры**.

1. Процедура init – начальная инициализация переменных, т.е. Обнуление счетчиков (строки 11-18) и открытие файлов (строки 19-21).

2. Процедура пропуска разделителей слов (строки 22-30). Строка 24: присваиваем символу (C) небуквенное значение, чтобы нормально работал цикл с предусловием (while – строка 25), который читает символы пока не прочитает буквенный символ или не наступит конец файла.

3. Процедура чтения слова: читает символы из файла, добавляет их в переменную S пока символы являются буквами (строки 31–41). Если слово заканчивается на дефис, то дефис выбрасываем (строки 42–43). Далее прибавляем единицу к счетчику слов (44–45).

4. Процедура сравнения прочитанного слова с формами искомого глагола (46–54). Если строка соответствует i-той форме глагола to be, то к этому счетчику добавляется единица (51).

5. Финальная процедура: закрытие текстового файла и выдача результатов на экран.

Программа

```
1.  Program Count_to_be;
2.  const
3.  Counttobe = 7;
4.  Arraytobe: array [1..Counttobe] of string = ('IS', 'AM', 'ARE', 'WAS', 'WERE', 'BE',
    'BEEN');
5.  var
6.  F : text;
7.  Tcounttobe : array [1..Counttobe] of integer;
8.  C : char;
9.  S : string;
10. CountWord : integer;
11. procedure init;
12. var i : integer;
13. begin
14.    For i := 1 to Counttobe do begin Tcounttobe [i] := 0;
15. end;
16.    CountWord := 0;
17.    C := ' ';
18.    S := '';
```

```
19.    assign (F, 'script1.txt');
20.    reset (F);
21. end;
22. procedure SkipToWord;
23. begin
24.    C:=' ';
25.     while not eof(F) and not (C in ['A'..'Z']) do
26.     begin
27.       read(F, C);
28.       C := UpCase (C);
29.    end;
30. end;
31. procedure ReadWord;
32. begin
33.   if not eof(F) and (C in ['A'..'Z']) then S := C;
34.   while not eof(F) and (C in ['A'..'Z', '-']) do
35.     begin
36.       read(F, C);
37.       C := UpCase (C);
38.       If (C in ['A'..'Z', '-']) then begin
39.         S := S+C;
40.       end;
41.    end;
42.    if S [length (S)] = '-' then begin delete (S, length(S), 1);
43.    end;
44.    CountWord := CountWord + 1;
45. end;
46. procedure CheckToBe;
47. var i : integer;
48. begin
49.   For i := 1 to Counttobe do begin
50.     if S = Arraytobe [i] then begin
51.       Tcounttobe [i] := Tcounttobe [i]+1;
52.     end;
53.    end;
54. end;
55. procedure final;
56. var i : integer;
57. begin
58.   close(F);
59.   Writeln ('words in text ', CountWord);
60.   For i := 1 to Counttobe do begin
61.          Writeln ('Count of ', Arraytobe [i], '=', Tcounttobe [i],', or ',100.0*Tcounttobe
    [i]/CountWord:7:3,'%');
62.   end;
63. end;
64. begin
65.   init;
66.   while not eof (F) do begin
67.     SkipToWord;
```

```
68.     ReadWord;
69.     CheckToBe;
70.   end;
71.   final;
72.         end.
```

Компьютерное моделирование на Free Pascal для автоматического транскриптора

ЗАДАЧА

На языке Паскаль представьте алгоритм выбора варианта прочтения диграфа «ea».

Условия задачи

Вариант 1: [ei]: только для great, break, steak

Вариант 2: [e]: heavy, heaven, breast, meadow, weapon, wealthy, ready, read (Past Simple), dead, death, breath, measure, pleasure, health, healthy.

Вариант 3: [iei]: только create

Вариант 4: [i]: все остальные.

Простейший вариант обработки строк на языке Pascal представлен десятью стороками (стр.)

```
1. program ReadDigraf;
2. var
  3. sInp : String;
4. begin
 5. ReadLn(sInp);
  6. If sInp = ´read´
    7. then begin
      8. WriteLn ('r -[i]- d, Infinitive and Present Simple, if other  - [e]') ;
    9. end;
10. end.
```

1стр. – ключевое слово **program**, за которым через пустое место (whitespace – пробел или энтер или таб) следует название (имя) программы.

!! Имя должно начинаться с буквы, состоять только из букв, цифр и знака нижнего подчеркивания; минус, плюс, точка и т. д. нельзя.

!! В данном алгоритмическом языке различия между большими и маленькими буквами в зарезервированных словах и индентификаторах отсутствует), после названия программы точка с запятой, после которой можно поставить или enter, или пробел, или вообще ничего (точка с запятой сама по себе является разделителем и не является ни к ключевым словом, ни идентификатором). Но для эстетических задач лучше поставить enter, поскольку эстетичность – одно из требований к разрабатываемым моделям, нарушение которого, однако, не мешает корректной сборке программы и правильному ее выполнению.

!! К случаям обязательного использования пустого места относятся:

- использование после ключевых (зарезервированных слов);
- использование после идентификаторов.

2 стр. – секция определений, в данном случае – блок определения переменных: ключевое слово **var**, за которым через пустое место (это ключевое слово) следует описание

переменных (**3 стр.**): имя, двоеточие, тип, далее точка с запятой. Если вводится несколько переменных одного типа, то можно записать их в одной строке, разделив запятыми, затем двоеточие, тип и точка с запятой.

!! Имеются разные области видимости переменных: глобальные (видимые для всей программы) и локальные (видимые в пределах конкретного *программного блока – в Pascal программные блоки – это процедуры и функции*). Если **var** стоит после слова **program**, то это глобальная область, а если после слова **function** или **procedure**, то это локальная область.

Типы переменных

Основные: integer (целые числа), real (числа с плавающей запятой — вещественные, напр. $5.08\text{-}1.3\cdot10^{7}$), char (символ, напр. 'b' 'ї'), string (строка)

Более сложные: array (массив одномерный, двухмерный), record (запись: если в массиве можно задавать элементы только одного типа, то в записи – разных), object (используется в объектно-ориентированном программировании (ОП) наряду с данными разных типов содержит процедуры и функции), file (внешние файл на диске).

4 стр. – **begin** и пустое место – начало программы.

5 стр. – **Readln** – стандартная процедура чтения с устройства ввода (напр. с клавиатуры), а в скобках указываются параметры, куда заносится результат чтения (в данном случае строковая переменная **sInp**).

6 стр. – условный оператор. Начинается с зарезервированного слова **if**, за которым следует условие (выражение на языке Pascal, вычисляемое по правилам Булевой алгебры – то есть с результатом «Истина или ложь»). За условием следует (**7 стр.**) зарезервированное слово **then**, за которым идет оператор Pascal, за которым может следовать (а может и отсутствовать вторая часть – **else**). Если необходимо ввести более одного оператора, используется так называемые операторные скобки – пара **begin end**, между членами которой можно использовать любое количество операторов (несколько командных строк).

8 стр. – **Writeln** – стандартная процедура вывода (в файл или на экран). В скобках дается, что именно выводить.

9 стр. – конец блока (того, что начинался с **then**), точка с запятой.

10 стр. – конец программы: **end.** (с точкой без пробелов)

!! Зарезервированные слова – слова, которые используются в строго определенных случаях и не могут быть использованы в качестве имен переменных, функций, процедур.

Список зарезервированных слов:

absolute, and, array, asm, begin, case, const, constructor, destructor, div, do, downto, else, end, file, for, function, goto, if, implementation, in, inherited, inline, interface, label, mod, nil, not, object, of, on, operator, or, packed, procedure, program, record, reintroduce, repeat, self, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor

Простейший вариант обработки предложен только для одного слова read с двумя способами прочтения – через [i] и через [e]. Это лишь малая часть, предполагаемая условиями задачи. Чтобы выполнить все автоматическое транскрибирование необходимы дальнейшие усложнения.

Вариант обработки строк на языке Pascal: усложнение 1

```
program prog1;
var
  sInp : String;
begin
```

```
 ReadLn(sInp);
  If sInp = 'read'
    then begin
      WriteLn ('r -[i]- d, Infinitive and Present Simple, if other  - [e]') ;
    end;
  if sInp = 'create'
  then begin
  WriteLn ('cr -[iei] - te') ;
  end;
end.
```

По сравнению с предыдущим простейшим вариантом добавлено еще одно ветвление для слова *create* (см. условия задачи и выделенный голубым цветом фрагмент данной программы).

Вариант обработки строк на языке Pascal: усложнение 2

```
program prog2;
const
countEI = 3;
TarrayEI : array [1..countEI] of string = ('great', 'break', 'steak');
var
  sInp : String;
begin
 ReadLn(sInp);
  If sInp = 'read'
    then begin
      WriteLn ('r -[i]- d, Infinitive and Present Simple, if other  - [e]') ;
    end;
  if sInp = 'create'
  then begin
  WriteLn ('cr -[iei] - te') ;
  end;
  If sInp = TArrayEI[1]
  then begin
    WriteLn ('gr -[ei]- t');
  end;
  If sInp = TArrayEI[2]
  then begin
    WriteLn ('br -[ei]- k');
  end;
  If sInp = TArrayEI[3]
  then begin
    WriteLn ('st -[ei]- k');
  end;
end.
```

По сравнению с предыдущим вариантом добавлена константа из трех составляющих (константы удобно применять для исключений из правил чтения, поскольку их количество всегда ограничено), на каждую из которых добавлено ветвление (см. голубую заливку).

Такая запись неудобна по причине избыточности кода (слишком много почти одинаковых строк).

Чтобы сделать запись более экономной, напишем функцию по замене одних символов в строке на другие символы.

```
1. function Replace(sInput,sSearch,sReplace:String):String;
2. var i1:Integer;
   3. sTmp:String;
4. begin
  5. i1:=Pos(sSearch,sInput);
  6. sTmp:=sInput;
  7. Delete(sTmp,i1,Length(sSearch));
  8. Insert(sReplace,sTmp,i1);
   9. Replace:=sTmp; // функции Replace присвоить значение временной переменной sTmp
10. end; // end Replace function
```

1 стр.: У данной функции три переменных/аргумента на входе (указаны в круглых скобках) – исходная строка (заданное слово), строка для поиска (дираф в заданном слове) и строка замены (на что заменить найденный дираф – один из транскрипционных знаков [i], [e], [ei]). Аргументов может и не быть, если это функция без аргументов. В скобках через двоеточие указывается тип переменной/аргумента, в данном случае – строковая переменная – **string**. После скобки через двоеточие указан тип возвращаемого функцией результата, в данном случае это тоже строка. Поэтому у нас дважды употребляется слово **string**: для типа переменной и для типа результата, возвращаемого функцией.

2 стр. – **var i1:Integer** – это описание переменных, используемых внутри данной функции. Ключевое слово **var** показывает, что наступает начало блока, описывающего переменные. **i1** – произвольно выбранное имя, которое используется для названия переменной. В нашем случае это индекс начала диграфа. Через двоеточие указан тип переменной – **integer** – целое число, потому что номером символа в строке может быть только целое число. Другие примеры: z8 – номер ряда в кинотеатре, тип – integer – целое число; m22 – цена продукта, тип – real – вещественная переменная с плавающей запятой и т. д.

3 стр. – **sTmp:String** – временная переменная для хранения промежуточного значения строки. Имя произвольно. В нашем случае это то слово, в котором мы хотим прочесть дираф -ea-.

4 стр. и **10 стр.** – операторные скобки.

В программе используются четыре библиотечные функции: *Pos, Delete, Insert, Length* (**5–9 стр.**).

Функция Pos возвращает индекс подстроки строке, напр. Для строки 'read' и подстроки 'ea' функция возвращает индекс 2 – дираф начинается со второго символа в строке (в слове).

Функция Delete удаляет из строки определенное количество символов. Строка определяется первым аргументов, с какого символа начинать удалять — вторым аргументов, сколько символов удалять – третьим аргументом (аргументы в примере даны в скобках). Например, для случая Delete(s, 3, 2), где s = create, строке s присваивается значение *crte* (начиная с третьего символа два удаляются). Обратите внимание, что в процедуру *Delete* можно передавать только строковые переменные. Если пытаться ввести напрямую Delete('create', 3, 2), то компилятор выдаст сообщение об ошибке.

Функция Length возвращает значение длины строки (количество символов в строке). Напр. Length ('break')=5. В нашем случае – длина диграфа = 2 (два символа).

Функция Insert вставляет первый аргумент во второй аргумент, начиная с символа, который определяется третьим аргументом. Напр. s1=rd, s2=ea, тогда после выполнения процедуры Insert (s2, s1, 2), s1 = read, а s2 не меняется.

!! В итоговом варианте программы нужно обратить внимание на отсутствие знака «;» после **end** в строке 51 презентации. Причина – препозиция к ключевому слову **else**.

Итоговый вариант программы

```
1. program ReadDigraf;
2. const
  3. CountEI = 3;
  4. TArrayEI : array [1..CountEI] of string = ('great','break','steak');
  5. CountE = 19;
  6. TArrayE : array [1..CountE] of string =('heavy', 'heaven', 'breast', 'meadow', 'weapon', 'wealth',
'wealthy', 'ready',  'dead', 'bread', 'death', 'breath', 'measure', 'pleasure', 'treasure', 'head', 'thread',
'threat','meant');
  7. TStringI = 'read';
  8. TStringIEI = 'create';
9. var
  10. sInp, sRpl, sTail : String;
  11. i : Integer;
12. function Replace(sInput,sSearch,sReplace:String):String;
13. var i1:Integer;
    14. sTmp:String;
15. begin
  16. i1:=Pos(sSearch,sInput);
  17. sTmp:=sInput;
  18. Delete(sTmp,i1,Length(sSearch));
  19. Insert(sReplace,sTmp,i1);
  20. Replace:=sTmp;
21. end;
22. begin
  23. sRpl := '';
  24. sTail := '';
  25. ReadLn(sInp);
  26. If sInp = TStringI
    27. then begin
      28. sRpl := ' -[i]- ';
      29. sTail := ', Infinitive and Present Simple, if other  - [e]';
    30. end;
31. If sInp = TStringIEI
    32. then begin
      33. sRpl := ' -[iei]- ';
    34. end;
  35. For i := 1 to CountEI do begin
    36. If sInp = TArrayEI[i] then begin
      37. sRpl := ' -[ei]- ';
    38. end;
  39. end;
  40. For i := 1 to CountE do begin
    41. If sInp = TArrayE[i] then begin
```

```
    42. sRpl := ' -[e]- ';
   43. end;
  44. end;
  45. If sRpl ='' then sRpl := ' -[i]- ';
  46. i := Pos ('ea',sInp);
  47. If i>0 then begin
   48. Delete (sInp,i,2);
   49. Insert (sRpl, sInp, i);
   50. If Length(sTail)>0 then sInp := sInp + sTail;
  51. end
  52. else begin
   53. WriteLn ('WARNING!!! "ea" NOT FOUND!!!');
  54. end;
  55. WriteLn(sInp);
56. end.
```

Примечания к программе

Добавлено: Tail — для комментария про Infinitive and Present Simple глагола *read*.

Строки с 47 по 53: знак больше нуля вводится для того, чтобы отсечь случаи, где искомого диграфа нет.

Программирование алгоритма решения задачи 1 – довольно сложный случай. Самый простой случай – когда транскрипционные знаки совпадают со знаками клавиатуры и когда имеется только один вариант прочтения той или иной буквы. Программирование таких случаев на примере английской буквы «m» представлено ниже.

Автоматическое транскрибирование буквы «m»

```
1.program Read_m;
2.var
3.sInp, SRpl: string;
4.i : Integer;
5. begin
6. sRpl := '';
7. ReadLn(sInp);
8. for i:=1 to Length(sInp) do begin
9.  If (sInp[i] ='m')
10.    then begin
11.      sRpl := sRpl+' -[m]- ';
12.  End else begin
13.    sRpl := sRpl+sInp[i];
14.  end;
15.  end;
16.   WriteLn(sRpl);
17.end.
```

В данной программе не представлены константы, только область переменных, поскольку имеется только одно правило прочтения и исключения отсутствуют. Соответственно, программа маленькая – состоит только из 17 строк. Схожей простотой программирования характеризуются очень немного букв для английского языка (вспомним

известную поговорку: «Пишем – Манчестер, читаем – Ливерпуль»), пожалуй, только «l», «v», «r» (последний – для американского и канадского вариантов). С остальными возникают более или менее серьезные сложности. Для каких-то букв их меньше, а для каких-то больше. Например, для «b», «n», в дополнение к приведенному примеру,

ЗАДАЧА

Составить алгоритм чтения синонимичных диграфов -ou-, ow.

Условия задачи

Основной тип [aʊ];

Исключение 1: [əʊ]: soul, poultry, shoulder, low, bow (лук), bowl, owe, own.

Исключение 2: [ʌ]: country, cousin, young, touch.

Исключение 3: [ʊ]: should, would, could, wounded;

Исключение 4: [u]: route, rouge.

В отличие от предыдущего алгоритма (см. задачу 1), в данном, согласно условиям задачи, предполагается использование знаков, не имеющихся на клавиатуре – транскрипционных знаков (кроме исключения 4). Это требует введения строк с поддержкой юникода. Для этого вместо типа данных string везде в программе необходимо использовать ansistring. Во 2 стр. через две косы черты (так можно обозначить любое примечание) дан соответствующий комментарий:
// uses AnsiString;
В области констант описаны четыре массива для работы с исключениями из правила. Перед описанием массива указывается количество элементов в нем (4, 6, 8 и 10 стр.). Затем следует описание массива: его имя (идентификатор), указание на тип данных (массив) и счетчик элементов [от первого элемента и до конца массива], тип каждого элемента (строка со знаками Юникода) и после знака равенства в круглых скобках и одинарных кавычках через запятую все эти элементы перечисляются (см. 5, 7, 9, 11 стр.). Такое описание легко можно модифицировать – убрать или добавить элементы, – не забыв при этом поменять указание на количество элементов.
Помимо четырех констант типа массив, имеются две строковые константы для задания одиночных уникальных исключений (12–13 строки).

Программа

```
1.  program ReadDigraphsouow;
2.  // uses AnsiString;
3.  const
4.  Countou = 7;
5.  TArrayou : array [1..Countou] of AnsiString = ('soul', 'poultry', 'shoulder', 'bowl', 'low',
    'own', 'owe');
6.  Counta = 4;
7.  TArraya : array [1..Counta] of AnsiString = ('country', 'cousin', 'young', 'touch');
8.  Countu1 = 4 ;
9.  TArrayu1 : array [1..Countu1] of AnsiString = ('should', 'would', 'could', 'wounded');
10. Countu = 2;
11. TArrayu : array [1..Countu] of AnsiString = ('route', 'rouge');
12. TAnsiStringou = 'bow';
13. TAnsiStringu1 = 'wound';
```

```
14. var
15. sInp, SRpl, sTail : AnsiString;
16. i : Integer;
17. function Replace (sInput, sSearch, sReplace : AnsiString) : AnsiString ;
18. var i1 : Integer;
19. sTmp : AnsiString;
20. begin
21.  i1:= Pos(sSearch, sInput);
22.  sTmp:= sInput;
23.  delete(sTmp, i1, Length(sSearch));
24.  Insert(sReplace, sTmp, i1);
25.  Replace:= sTmp;
26.  end;
27.  begin
28.  sRpl := '';
29.  sTail := '';
30.  ReadLn(sInp);
31.  if sInp = TAnsiStringou
32.  then begin
33.  sRpl := ' -[əu]- ';
34.  sTail := ', в значении "Лук", if other - [au]';
35.  end;
36. if sInp = TAnsiStringu1
37.  then begin
38.  sRpl := ' -[u]- ';
39.  end;
40.  For i := 1 to Countou do begin
41.  if sInp = TArrayou[i] then begin
42.  sRpl := ' -[əu]- ';
43.  end;
44.  end;
45.  For i:= 1 to Counta do begin
46.  if sInp = TArraya[i] then begin
47.  sRpl := ' -[□]- ';
48.  end;
49.  end;
50.  For i := 1 to Countu1 do begin
51.  if sInp = TArrayu1[i] then begin
52.  sRpl := ' -[u]- ';
53.  end;
54.  end;
55. For i := 1 to Countu do begin
56.  if sInp = TArrayu[i] then begin
57.  sRpl := ' -[u]- ';
58.  end;
59.  end;
60.  If (sRpl ='') And (Pos('ou',sInp)>0)
61.     then begin
62.       sRpl := ' -[au]- ';
63.   End;
```

```
64.  If (sRpl ='') And (Pos('ow',sInp)>0)
65.    then begin
66.      sRpl := ' -[au]- ';
67.  End;
68.  If Pos('ou',sInp)>0
69.    then begin
70.      sInp:=Replace(sInp,'ou',sRpl)+sTail;
71.    end;
72.    If Pos('ow',sInp)>0
73.    then begin
74.      sInp:=Replace(sInp,'ow',sRpl)+sTail;
75.    end;
76.    WriteLn(sInp);
77. end.
```

ЗАДАЧА

Составить алгоритм прочтения диграфа -th-

Условия задачи

1. **/θ/:** thin, thaw, teeth (at the beginning and at the end of the words)

!/ð/: the, this, these, that, those, they, them, their, there, then, than, ***with***

!/t/: Themes, Thomas

2. **/ð/:** breathe, wither, teethe (between vowel graphemes)

Дополнительная сложность – необходимость учитывать положение в слове (начало, конец, середина) и положение между гласными (задать, какие графемы являются гласными).

Задаем гласные через константу, тип которой — набор символов: **set of char ['a',]** и т.д. -- в квадратных скобках, в одинарных кавычках.

Далее будем проверять оператором **In** нахождение символа до и после диграфа в этом наборе гласных.

Проверим, чтобы перед и после -th- была хотя бы одна буква (дираф не в начале и не в конце строки).

Для этого позиция диграфа должна быть больше единицы (не в начале слова) и одновременно с этим (оператор булевой алгебры **and**) позиция должна быть как минимум на два меньше, чем длина слова (дираф занимает две позиции, потому что он дираф — состоит из двух символов). Еще одно **and** для того, чтобы проверить, что **sRpl** по-прежнему пустая строка (sRpl = ' ').

Строки 35-38: в переменной **i** находится позиция -th-, которая засылается туда функцией **Pos.**

i-1 — это позиция перед диграфом

i+2 — это позиция после диграфа. Предположим, что позиция **i=3**. Это означает, что буква -t- третья, а буква -h- четвертая, позиция за диграфом, таким образом, будет пятой.

Выполнение через **консоль** с вызовом **mc** (midnight commander) — аналог Norton Commander в Dos. Быстрый вызов списка папок через **CTRL верхняя косая.** Просмотр результатов через **CTRL O**.

Программа

```
1. program ReadDigraphTH;
2. const
```

```
3.  CountTH1 =  11;
4.  TArrayTH1 : array [1..CountTH1] of AnsiString = ('the', 'this', 'these', 'that', 'those', 'they',
    'them', 'their', 'there', 'then', 'than');
5.  CountTH2 = 2;
6.  TArrayTH2 : array [1..CountTH2] of AnsiString = ('Themes', 'Thomas');
7.  vowel : set of char = ['a', 'e', 'i', 'o', 'u', 'y'];
8.  var
9.  sInp, SRpl : AnsiString;
10. i : Integer;
11. cBefore, cAfter: char;
12. function Replace (sInput, sSearch, sReplace : AnsiString) : AnsiString ;
13. var i1 : Integer;
14. sTmp : AnsiString;
15. begin
16.  i1:= Pos(sSearch, sInput);
17.  sTmp:= sInput;
18.  delete(sTmp, i1, Length(sSearch));
19.  Insert(sReplace, sTmp, i1);
20.  Replace:= sTmp;
21. end;
22. begin
23.  sRpl := '';
24.  ReadLn(sInp);
25.  For i := 1 to CountTH1 do begin
26.    if sInp = TArrayTH1[i] then begin
27.      sRpl := ' -[ð]- ';
28.    end;
29.  end;
30.  For i:= 1 to CountTH2 do begin
31.    if sInp = TArrayTH2[i] then begin
32.      sRpl := ' -[t]- ';
33.    end;
34.  end;
35.  i := Pos('th', sInp);
36.  if (i>1) and (i<length(sInp)-1) and (sRpl = '') then begin
37.    cBefore := Char(sInp[i-1]);
38.    cAfter := Char (sInp[i+2]);
39.    if (cBefore In vowel) and (cAfter In vowel) then begin
40.      sRpl := ' -[ð]- ';
41.    end;
42.  end;
43.  if sRpl = '' then sRpl := ' -[θ]- ';
44.  If Pos('th',sInp)>0
45.   then begin
46.     sInp:=Replace(sInp,'th',sRpl);
47.  end;
48.  WriteLn(sInp);
49. end.
```

ТЕМА 4. АВТОМАТИЗИРОВАННОЕ ОБУЧЕНИЕ И ПРОГРАММИРОВАНИЕ ТЕСТОВ

Существует два основных подхода к методам автоматизированного обучения: бихевиористский и когнитивно-интеллектуальный.

При **бихевиористском** подходе:
1) программирование учебной деятельности обучаемого;
2) тестирование;
3) информирование.

Первый из этих методов обучения характерен тем, что управляющие воздействия на обучаемого полностью определяются обучающей программой. В такой программе каждому обучаемому в зависимости от его уровня знаний полностью задается последовательность учебных или контрольных заданий.
При тестировании компьютер по специальным программам выявляет индивидуальные профессиональные и психологические характеристики обучаемых и достигнутые ими уровни знаний. При этом обучаемый лишь отвечает на вопросы, но оценку за знания не Получает. Этот метод достаточно часто используется при оценке различных аспектов знания иностранных языков (тестирование словарного запаса, способности к изучению иностранных языков и т.п.).
Суть метода информирования заключается в том, что в память компьютера помещаются некоторые справочно-информационные данные (грамматический справочник, орфографический словарь, двуязычный словарь и т.п.), которые обучаемый может использовать при подготовке к занятиям или непосредственно в процессе занятий.

При **когнитивно-интеллектуальном** подходе осуществляется:
1) моделирование учебной среды;
2) свободное обучение.
Суть процесса моделирования учебной среды сводится к созданию компьютерных программ, которые моделируют структуру некоторого объекта или принципы его действия. При этом, как и при бихевиористском подходе, управляющие воздействия также полностью определяются обучающей программой. Обучаемый может выбирать учебные задания из некоторого конечного множества заданий, включенного в универсальную учебную среду.

Различают два типа моделей учебной среды:
1) модели объективного (предметного) типа;
2) модели мыслительного типа.

Модели первого типа служат для познания некоторого объекта или приобретения навыков работы с таким объектом. Примером такой модели является компьютерная программа, обучающая студента работе на клавиатуре.
Модели мыслительного типа — это уже упоминавшиеся выше воспроизводящие инженерно-лингвистические модели. Их задача — изучение некоторых процессов, явлений или же динамического поведения некоторого объекта. Чаще всего они используются в процессе преподавания радиотехники, электроники, физики, биологии и других дисциплин. Широко используются подобные модели и в лингвистике.
Метод свободного обучения характерен тем, что он дает возможность самому выбрать тематику обучения и способ работы с компьютером. Компьютер при этом не только

предъявляет обучающие кадры по указанию обучаемого, но и уточняет его действия, предлагая обучаемому наиболее эффективный способ использования учебного материала.

При этом выделяют три вида обучения:
1) структурно-управляемое обучение;
2) обучение принятию решения;
3) генеративное обучение.

Учебный материал, используемый при структурно-управляемом обучении, представляется в виде некоторой иерархической структуры данных. Для каждого структурного уровня в компьютере заранее определены цели обучения и условия для успешного достижения такой цели. Элементарным примером такого вида обучения может служить задача обучения формообразованию русских глаголов. Здесь можно выделить четыре структурных уровня, вводящих по принципу «от простого к сложному» основные понятия, связанные с формообразованием русского глагола.
При структурно-управляемом свободном обучении обучаемый сам может выбирать начальный уровень обучения. Однако локальная цель высшего уровня должна совпадать с конечной целью обучения по данной программе.

При обучении принятию решения учебная информация, заложенная в компьютер, представляется в виде некоторого набора специальным образом записанных ситуаций (например, в виде фреймов). При таком виде свободного обучения обучаемый, исходя из задаваемой компьютером исходной ситуации и ориентируясь на его ответы, должен прийти к некоторой конечной ситуации.
При обучении иностранному языку, например, можно дать задание компьютеру выступить в роли продавца в магазине, где обучаемый, ведя диалог с продавцом, хочет купить некоторую вещь. При этом обучающая программа может корректировать действия обучаемого или после каждого его шага принятия решения, либо она может комментировать его действия после окончательного завершения всей программы.

Генеративное обучение как вид свободного обучения пока возможно лишь теоретически.

Задачи автоматизированного учебного курса
1) проектирование состава курса и его содержания;
2) методическая проработка учебного материала каждой отдельной задачи,
входящей в состав курса, и создание ее обучающего сценария;
3) создание обучающей программы по данной задаче и ее
экспериментальная проверка (тестирование);
4) объединение обучающих программ всех задач курса в едином автоматизированном учебном курсе.

Требования к компьютерным программам индивидуализированного обучения
1) совмещать в себе обучающую, контрольную и поисковую функции;
2) опираться на сценарии, приближенные к обычному традиционному обучению;
3) максимально использовать принцип наглядности и доступности, т. е. выводить на экран компьютера не только текст, но и звук, иллюстрации, видео и т.п.;
4) иметь средства быстрой и объективной оценки знаний обучаемых даже в тех случаях, когда ответ обучаемого далек от наиболее ожидаемого;
5) содержать возможность настройки на конкретного обучаемого (выбор способа подачи нового материала, типа упражнений, скорости ответа и т.п.).

Процесс создания мультимедийных обучающих программ включает следующие **этапы**:
1) разделение всего курса, который будет предложен для обучения, на определенное число тем и подтем;
2) отбор для каждой темы или подтемы определенного лексического и грамматического материала;
3) создание для каждой темы или подтемы набора сценариев, в рамках которых будут закрепляться лексический материал и грамматические правила. Такие сценарии должны включать звук, графику и движение;
4)подбор в соответствии со сценариями необходимых текстов, аудио- и видеоматериалов; программирование сценариев.

Программа строится на следующих дидактических параметрах:
1) тип пользователя;
2)назначение программы;
3)количество разделов, тем и подтем в программе;
4)язык комментариев и подсказок;
5)количество единиц словаря;
6)способы представления единиц словаря;
7)логический объем курса в часах (продолжительность последовательного прослушивания всего звукового материала программы без повторов);
8)число иллюстраций.

Помимо этого, каждую программу можно охарактеризовать рядом технических параметров:
1) объем учебного текста в Кб;
2) объем памяти в Мб, занятый звуковыми фрагментами;
3) объем видеофрагментов в Мб;
4) формат видео;
5) формат аудио;
6) формат иллюстраций.

По типу пользователей различают следующие мультимедийные программы обучения иностранным языкам: 1) для детей; 2) для молодежи и взрослых; 3) для бизнес-применений; 4) специализированные программы.
По назначению такие программы делят на следующие виды: 1) для игр; 2) для начального обучения языку; 3) для совершенствования знаний языка; 4) для сдачи различных экзаменов; 5) для работы с деловыми текстами.

Программирование тестов

ЗАДАЧА

Запрограммировать многовыборный тест.

Условия задачи
Имеется некоторое количество заданий с четырьмя вариантами ответов на каждое. Только один из этих вариантов – правильный. По окончании выполнения теста необходимо дать результаты автоматического подсчета количества правильных ответов. Задания и варианты представлены в следующем виде.

Choose only one correct variant

1. Her eyes are red. She ... all night.
a) cries; b) is crying; c) cried; **d)** has been crying.
2. I cut myself when I ... .
a) shaved; **b)** was shaving; c) had shaven; d) had been shaving.
3. I look ... hearing from you.
a) after; b) for; c) at; **d)** forward to.
4. I ... to him before you arrived.
a) was talking; b) talked; **c)** had talked; d) had been talking.
5. His father is a ... . He works on a big modern ship.
a) sailor; b) seller; c) doctor; d) cop.

Язык программирования – Free Pascal

Пояснения
Имеются две строки: первая – вопроса, вторая – вариантов ответа. На каждый вопрос требуется ввести один символ – букву правильного ответа.

Задачу решаем через введение ***трех массивов***: первый массив – вопрос, тип – строка; второй массив – варианты ответов, тип – строка; третий массив – правильный ответ, тип – символ.

Программа

```
program MultipleChoice;
const
count = 5;
Questions:Array[1..count] of string = (
'1. Her eyes are red. She ... all night.',
'2. I cut myself when I ... .',
'3. I look ... hearing from you.',
'4. I ... to him before you arrived.',
'5. His father is a ... . He works on a big modern ship.');
AnswerVariants: Array [1..count] of string = (
'a) cries; b) is crying; c) cried; d) has been crying.',
'a) shaved; b)  was shaving; c) had shaven; d) had been shaving.',
'a) after; b) for; c) at; d) forward to.',
'a) was talking; b) talked; c) had talked; d) had been talking.',
'a) sailor; b) seller; c) doctor; d) cop.');
CorrectVariant: Array [1..count] of char = (
'd', 'b', 'd', 'c', 'a');
var
InpChar : char;
CorrectAnswer: integer;
Counter : integer;
begin
CorrectAnswer:= 0;
For Counter:= 1 to count do begin
Writeln('-------------------------------------------------');
Writeln('Question No ',Counter);
Writeln (Questions [Counter]);
Writeln (AnswerVariants [Counter]);
```

```
29. repeat
30. Readln (InpChar);
31. if not (InpChar in ['a'..'d']) then begin
32. Writeln('Input error. Please enter character a, b, c or d')
33. end;
34. until InpChar in ['a'..'d'];
35. if InpChar = CorrectVariant [Counter] then begin
36. CorrectAnswer:= CorrectAnswer + 1;
37. end;
38. end;
39. writeln ('Correct answers ',CorrectAnswer,' or ',(CorrectAnswer/count*100):7:3,'%');
40. end.
```