Министерство науки и образования РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**ФГБОУ ВО Амурский государственный университет**

**С. В. Андросова, Е. Ю. Андросов**

# ЛИНГВИСТИКА И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

сборник учебно-методических материалов
для научной специальности 5.12.3 – Междисциплинарные исследования языка

Благовещенск
2022

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

*Андросова С. В., Андросов Е.Ю*

Лингвистика и информационные технологии: сборник учебно-методических материалов для научной специальности 5.12.3 – междисциплинарные исследования языка. – Благовещенск: Амурский гос. ун-т, 2022, - 46 с.

## ВВЕДЕНИЕ

В начале было слово...

Когда-то язык был слабым отражением вещей в сознании человека. Прозрачной зыбкой гранью, отделяющей человека от мира неживой материи. Сейчас язык развился настолько, что сам проявляет тенденцию к независимому движению и управляет развитием разума. Язык – самое древнее и самое загадочное приобретение человека, а лингвистика – наука о языке – древнейшая из наук. Развиваясь, язык раскрывает методы собственного познания.

Язык и мышление человека неразрывно связаны, поэтому разгадывая алгоритмы языка, человек тем самым пытается расшифровать алгоритмы разума. (по А. В. Анисимову). Эти многочисленные и настойчивые попытки привели в оформлению нового направления в лингвистики – прикладной лингвистики, неотъемлемой частью которой являются информационные технологии.

В наши дни владение информационными технологиями становится в один ряд с такими качествами, как умение читать и писать. Сегодня лингвист должен ориентироваться в мировом информационном пространстве и обладать необходимыми знаниями, умениями и навыками поиска, обработки и хранения информации с использованием информационных технологий, компьютерным систем и сетей.

В задачи настоящего курса входит изучение проблем алгоритмизации, моделирования лингвистических задач, современных языков программирования. Мы попытаемся дать вам представление о том, как ставится и решается лингвистическая задача с помощью компьютера: от ее словесной формулировке к алгоритму и компьютерной программе.

Для того, чтобы решать эти задачи знания высшей математики и построения сложных математических моделей не нужны, потому что компьютер – это языковая машина и основа его могущества заключается в способности манипулировать языковыми знаками – символами, которым приписывается некоторый смысл. Фактически, естественный язык в информатике занимает центральное место.

## АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

- 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, LibreOffice[1] 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, LibreOffice Calc) для обработки экономических и статистических данных;
9. Системы управления базами данных – СУБД (Access для Windows, MySQL[2] для Linux) позволяют осуществлять создание и модификацию больших совокупностей структурированных определенным образом данных (баз данных), а так же поиск в них информации.

К прикладным инструментальным средствам относят языки программирования и системы программирования. Рассмотрим этот аспект отдельно.

1 С 2010 года в дистрибутивы Linux включается ответвление OpenOffice под названием LibreOffice. LibreOffice с точки зрения пользователя ничем не отличается от OpenOffice, основная причина произведенного изменения – лицензионно-правовая.

2 С 2009 года в дистрибутивы Linux включается ответвление MySQL под названием MariaDB. Причина, как и в предыдущем случае, лицензионно-правовая.

## ПРИКЛАДНЫЕ ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА: ЯЗЫКИ ПРОГРАММИРОВАНИЯ И СИСТЕМЫ ПРОГРАММИРОВАНИЯ

*Когда деревья были большими, а компьютеры – ещё больше ...* Изначально никаких языков программирования не существовало. Программирование осуществлялось в машинных кодах процессора (любой набор цифр), и именно под этот процессор писалась программа. Для пользователя ПК 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 бурным развитием сети Интернет широкое распространение получили языки JAVA, PERL (Practical Extraction and Report Language – практический язык для извлечения данных и составления отчётов), PYTHON, PHP (Hypertext Preprocessor).

Помимо интерпретаторов и компиляторов был создан язык для оформления и динамического создания страниц **HTML** (hyper text markup language).

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++ (т. е. язык C с добавлением объектно-ориентированного программирования), а для обучения программированию – язык **PASCAL**.

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

1. Компилятор анализирует, какие внешние библиотеки 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.

**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 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 Pascal.

Далее посмотрим, как в общих чертах выглядит программа, написанная на алгоритмическом языке-компиляторе Pascal (версия Free Pascal). Это будет наипростейшая программа, которая выводит на экран надпись «Здравствуй, мир!»

## Общий вид программы на языке Паскаль

```
program Test;
begin
  WriteLn('Здравствуй, мир!');
end.
```

Слова (идентификаторы) **program**, **begin**, **end** – зарезервированные в языке слова. Напомним, что они могут использоваться только в определенном смысле. Эти слова не могут использоваться в качестве названий *процедур*, *функций*, *констант*, *типов*, *переменных* и т. п. Зарезервированные слова, как правило, в редакторах кода выделяются другим шрифтом, цветом и т. п. (напр., в Geany с настройками по умолчанию они пишутся темно-синим цветом и жирным шрифтом)[3].

Идентификатор **program** используется для указания имени программы. Какой-либо функциональности данная строка не несёт. Обратите внимание, что строка завершается символом точки с запятой «;». Символ «;» указывает компилятору языка, что строчка кода завершена. Язык паскаль ориентируется не на видимые переносы, а именно на символ точка с запятой; при её пропуске будет выдаваться ошибка компиляции[4].

Дальше идёт основной блок кода – собственно программа или тело программы, которое начинается с зарезервированного слова **begin** и заканчивается словом **end** за которым стоит точка. Эти слова называются операторными скобками. Словом **end** с точкой в паскале обозначается конец программы, в остальных случаях после **end** ставится точка с запятой.

Между **begin** и **end.** мы пишем текст нашей программы, которая в данном случае состоит из вызова одной библиотечной процедуры – WriteLn. Данная процедура занимается выводом на экран (или в файл) того, что ей дают в качестве аргументов. Аргументы процедуре или функции задают в круглых скобках. В данном случае в качестве аргументов выступает текстовая строка. О том, что это именно текстовая строка компилятору сигнализируют одинарные кавычки, которыми в начале и конце выделяется строка. На рисунке 1 показано, как выглядит данная программа в окне Geany.

После того, как вы написали несколько строк программного кода, файл необходимо сохранить с расширением **.pas** и тогда все зарезервированные слова будут выделяться тёмно-синим шрифтом, кроме этого, будут работать и другие полезные подсветки.

Поскольку Паскаль относится к компилируемым языкам, перед выполнением программу необходимо скомпилировать. Выбираем пункт меню Сборка-Скомпилировать или нажимаем клавишу F8. После этого Сборка-Выполнить (или клавишу F5).

---

3 Для данного форматирования файл с исходным текстом необходимо предварительно сохранить с правильным расширением – .pas

4 Из данного правила есть исключения. Точка с запятой не ставится после идентификаторов **begin**, и в конструкции if между **then** и **else**.

Рисунок 1. **Внешний вид программы «Hellow, world» в окне Geany**

## Константы, переменные и простейшая обработка строк и символов в Паскале

Введём несколько новых зарезервированных слов Паскаля:

**const** – описание *констант* (некоторых постоянных, которые мы не планируем изменять в ходе выполнения программы;

**var** – описание *переменных* – величин, которые изменяются в результате выполнения программы;

**string** – тип данных, называемых *строка;* это данные, которые содержат текстовую информацию – буквы, фразы или еще более длинные тексты.

Также познакомимся с оператором присваивания («:=») и конкатенации («+»).

Итак, пример программы.

```
program Test2;
const
  C1 = 'Здравствуй,';
  C2 = ' ';
  C3 = 'мир!';
var
  S1 : String;
begin
  S1 := '';
  WriteLn('S1=',S1);
  WriteLn('C1=',C1,'C2=',C2,'C3=',C3);
  S1 := C1+C2+C3;
  WriteLn('S1(C1+C2+C3)=',S1);
end.
```

В результате работы этой программы на экран выводится следующее:

```
S1=
C1=Здравствуй,C2= C3=мир!
S1(C1+C2+C3)=Здравствуй, мир!
```

Кратко разберём текст. За первой строкой с названием программы идёт блок описания констант. Блок начинается с зарезервированного слова **const** и продолжается до начала следующего блока (здесь до слова **var**). В блоке констант описываются три константы с именами C1, C2 и C3. Все константы текстовые, что компилятор понимает из того, что текстовые строки ограничены с двух сторон одинарными кавычками. Также не забываем, что конец строки программы обозначается точкой с запятой. После слова **const** точку с запятой ставить не надо – компилятор это воспринимает как ошибку – отсутствие имени и описания константы.

После блока констант идет блок *переменных* – обозначаемый зарезервированным словом **var**. Особенность переменных в том, что они, в отличие от констант, могут изменяться в ходе выполнения программы. В блоке переменных в данном случае описана одна переменная по имени S1 с типом **string**. Типом **string** в языке Паскаль обозначаются переменные типа строка – переменные, в которых хранятся и изменяются разные наборы символов.

Далее идет блок кода, первой строкой в котором идет оператор *присваивания* – команда вычислить значение выражения справа от символов «:=» и поместить результат этого в переменную слева. Присвоить переменной значение вычисленного выражения. В данном случае необходимо присвоить переменной S1 пустую строку. Данная операция называется *инициализацией* переменной – присвоение некоторого начального значения переменной. Данная операция необходима для того, чтобы задать начальное значение переменной. Без данной операции значение текстовых переменных не определено и может теоретически быть любым, что зависит от конкретной реализации компилятора. В любом случае перед использованием переменных рекомендуется задавать им какие-то начальные значения (для чисел это обычно 0, для строк – пустые строки), в противном случае программы могут вести себя непредсказуемо.

Следующей строкой (WriteLn('S1=',S1);) мы выводим на экран строку «S1=» и значение этой строки. Обратите внимание, что в процедуре WriteLn можно использовать несколько аргументов, отделяя их друг от друга запятыми. Далее (WriteLn('C1=',C1,'C2=',C2,'C3=',C3);) мы показываем на экране значения констант C1, C2 и C3, предваряя их соответствующим пояснительным текстом – «C1=» для C1 и т. д.

Строка «S1 := C1+C2+C3;» осуществляет вычисление *конкатенации* (сцепления) текстовых строк, заданных константами C1, C2 и C3 и *присваивает* это значение переменной S1.

Строка «WriteLn('S1(C1+C2+C3)=',S1);» выводит текст «S1(C1+C2+C3)=» и после него значение переменной S1, которое на предыдущем шаге стало «Здравствуй, мир!».

Строка «end.» сигнализирует о конце программы.

### Условный оператор (оператор ветвления), цикл со счетчиком, работа с отдельными символами строки

Рассмотрим два дополнительных элемента языка Паскаль. Начнем с условного оператора **if**. Форма его такова:

**if** (*Некоторое условие*)
  **then** *Некоторый код если истина*

**else** *Некоторый код, если ложь;*

Отступы сделаны для наглядности, для компилятора они не важны. Оператор **if** вычисляет выражение условия (значение может быть только двух типов – **true** (*истина*) либо **false** (*ложь*)). Вычисления идут по правилам булевой алгебры, используются выражения < (*меньше*), > (*больше*), =(*равно*), <> (*не равно*), а также >= (*больше либо равно*), <= (*меньше либо равно*) и операции – **and** (*логическое и*), **or** (*логическое или*), **xor** (*дизинъюкция, иначе исключающее или*) а также **not** (**одинарное отрицание**). Возможно, текстовое описание является слишком непонятным, поэтому ознакомьтесь с примерами отношений и получающимся значением.

| **Выражение** | **Значение** | **Выражение** | **Значение** |
|---|---|---|---|
| 2+2=4 | True | 2+2<>4 | False |
| 5>9-4 | False | 5>=9-4 | True |
| 7*5<>5*7 | False | 5+3=2+6 | True |

Строковые переменные и константы можно сравнивать только на равенство и неравенство.

| **Выражение** | **Значение** | **Выражение** | **Значение** |
|---|---|---|---|
| 'S1'='S1' | True | 'S1'<>'S1' | False |
| '#S#'='#s#' | False | 'Str'<>'sTR' | True |
| 'Some'='Some ' | False | '_Some='='_Some=' | True |

Для того, чтобы программа считала строки одинаковыми, они должны совпадать полностью, символ за символом и быть одинаковой длины. Во всех остальных случаях строки считаются различными.

Теперь ознакомимся с операторами **not**, **and** и **or**. Операцию **xor** не будем рассматривать, поскольку на практике она используется редко. Получившееся значение будет логическим – True (истина) или False (ложь). Значение операции **and** будет True тогда и только тогда, когда оба аргумента – True. Значение операции **or** будет True в том случае, если хотя бы один из аргументов True. Значение операции **not** будет обратным, т. е. True будет изменено на False и наоборот (операция **not** производится над одним аргументом).

| **Выражение** | **Значение** | **Выражение** | **Значение** |
|---|---|---|---|
| False **and** False | False | False **or** False | False |
| False **and** True | False | False **or** True | True |
| True **and** False | False | True **or** False | True |
| True **and** True | True | True **or** True | True |
| **not** True | False | **not** False | True |

Пример. Чтобы задать условие, что переменная X должна быть в промежутке от 5 до 15, включая 5 и 15, условие записывается как (X>=5) and (X<=15). Обратите внимание, что операции сравнения по умолчанию имеют более низкий приоритет при вычислении

выражений, чем операции булевой алгебры, поэтому операции сравнения необходимо брать в скобки, чтобы сначала вычислялись операции сравнения, а уж потом – операция **and**. Если необходимо обратное условие – т. е. либо меньше 5 либо больше 15, это можно записать двумя путями:

1. **not** ((X>=5) **and** (X<=15))
2. (X<5) **or** (X>15)

Выбор той или иной формы условия зависит только от личных предпочтений и должен диктоваться только удобством понимания заданного условия.
Следующим рассмотрим оператор цикла со счётчиком. Его форма такова:

```
for Переменная := Начальное значение to Конечное значение {step 1}
  do Некоторый код, повторяющийся от Начального значения до Конечного значения с
шагом , указанным в операторе step;
```

Пример. Чтобы вывести цифры от одного до десяти, можно использовать следующий код.

```
for i:=1 to 10
  do begin
    WriteLn(i);
  end;
```

Также цикл можно использовать для подсчёта в обратном направлении. Чтобы вывести цифры от десяти до одного, можно использовать следующий код.

```
for i:=10 downto 1
  do begin
    WriteLn(i);
  end;
```

Для понимания последующего программного кода необходимо познакомиться ещё с несколькими понятиями. Строки в Паскале хранятся в виде последовательности символов. К этим символам можно обращаться по *индексу* (порядковому номеру символа, начиная с 1). В качестве примера обратимся к константе С3 (С3 = 'мир!';) из примера, приведённого ранее. С3 = «мир!». С3[1] = «м», С3[2] = «и», С3[3] = «р», С3[4] = «!».
Каждый символ в памяти компьютера закодирован определенным цифровым кодом. Компьютер не оперирует символами, он оперирует числами. Для преобразования чисел в символы используются различные кодовые таблицы (одна из наиболее старых таблиц называется ASCII – 128 символов; последние Unicode-таблицы содержат десятки тысячи символов).
В языке Паскаль есть следующие функции для преобразования символа в число и числа в символ:
Chr(число) – преобразует аргумент-число в символ;
Ord(символ) – преобразует аргумент-символ в число.
Также есть функция Length – возвращает длину строки, переданной в качестве аргумента.

Представим следующую программу. Имеется строка «QUICK BROWN FOX JUMPS OVER THE LAZY DOG». Строка-тест, используемая для проверки печати букв латинского алфавита – шуточная фраза, в которую входят все буквы латинского алфавита.

Последовательно уберём из фразы все буквы от A до Z, последовательно заменяя их, к примеру, звездочкой.

```
program Test3;
const
  C1 = 'QUICK BROWN FOX JUMPS OVER THE LAZY DOG';
  var
  S1 : String;
  i1,i2 : Integer;
  c2 : Char;
begin
  S1 := C1;
  for i1 := Ord('A') to Ord('Z')
    do begin
      c2 := Chr(i1);
      WriteLn(S1,' remove ',c2);
      For i2 := 1 to Length(S1)
        do begin
          If S1[i2]=c2
            then S1[i2]:='*';
        end;
      WriteLn(S1);
    end;
  WriteLn('S1=',S1);
end.
```

Вывод на экран результата работы данной программы такой:

```
QUICK BROWN FOX JUMPS OVER THE LAZY DOG remove A
QUICK BROWN FOX JUMPS OVER THE L*ZY DOG
QUICK BROWN FOX JUMPS OVER THE L*ZY DOG remove B
QUICK *ROWN FOX JUMPS OVER THE L*ZY DOG
QUICK *ROWN FOX JUMPS OVER THE L*ZY DOG remove C
QUI*K *ROWN FOX JUMPS OVER THE L*ZY DOG
QUI*K *ROWN FOX JUMPS OVER THE L*ZY DOG remove D
QUI*K *ROWN FOX JUMPS OVER THE L*ZY *OG
QUI*K *ROWN FOX JUMPS OVER THE L*ZY *OG remove E
QUI*K *ROWN FOX JUMPS OV*R TH* L*ZY *OG
QUI*K *ROWN FOX JUMPS OV*R TH* L*ZY *OG remove F
QUI*K *ROWN *OX JUMPS OV*R TH* L*ZY *OG
QUI*K *ROWN *OX JUMPS OV*R TH* L*ZY *OG remove G
QUI*K *ROWN *OX JUMPS OV*R TH* L*ZY *O*
QUI*K *ROWN *OX JUMPS OV*R TH* L*ZY *O* remove H
QUI*K *ROWN *OX JUMPS OV*R T** L*ZY *O*
QUI*K *ROWN *OX JUMPS OV*R T** L*ZY *O* remove I
QU**K *ROWN *OX JUMPS OV*R T** L*ZY *O*
QU**K *ROWN *OX JUMPS OV*R T** L*ZY *O* remove J
QU**K *ROWN *OX *UMPS OV*R T** L*ZY *O*
QU**K *ROWN *OX *UMPS OV*R T** L*ZY *O* remove K
QU*** *ROWN *OX *UMPS OV*R T** L*ZY *O*
QU*** *ROWN *OX *UMPS OV*R T** L*ZY *O* remove L
```

```
QU*** *ROWN *OX *UMPS OV*R T** **ZY *O*
QU*** *ROWN *OX *UMPS OV*R T** **ZY *O* remove M
QU*** *ROWN *OX *U*PS OV*R T** **ZY *O*
QU*** *ROWN *OX *U*PS OV*R T** **ZY *O* remove N
QU*** *ROW* *OX *U*PS OV*R T** **ZY *O*
QU*** *ROW* *OX *U*PS OV*R T** **ZY *O* remove O
QU*** *R*W* **X *U*PS *V*R T** **ZY ***
QU*** *R*W* **X *U*PS *V*R T** **ZY *** remove P
QU*** *R*W* **X *U**S *V*R T** **ZY ***
QU*** *R*W* **X *U**S *V*R T** **ZY *** remove Q
*U*** *R*W* **X *U**S *V*R T** **ZY ***
*U*** *R*W* **X *U**S *V*R T** **ZY *** remove R
*U*** ***W* **X *U**S *V** T** **ZY ***
*U*** ***W* **X *U**S *V** T** **ZY *** remove S
*U*** ***W* **X *U*** *V** T** **ZY ***
*U*** ***W* **X *U*** *V** T** **ZY *** remove T
*U*** ***W* **X *U*** *V** *** **ZY ***
*U*** ***W* **X *U*** *V** *** **ZY *** remove U
***** ***W* **X ***** *V** *** **ZY ***
***** ***W* **X ***** *V** *** **ZY *** remove V
***** ***W* **X ***** **** *** **ZY ***
***** ***W* **X ***** **** *** **ZY *** remove W
***** ***** **X ***** **** *** **ZY ***
***** ***** **X ***** **** *** **ZY *** remove X
***** ***** *** ***** **** *** **ZY ***
***** ***** *** ***** **** *** **ZY *** remove Y
***** ***** *** ***** **** *** **Z* ***
***** ***** *** ***** **** *** **Z* *** remove Z
***** ***** *** ***** **** *** **** ***
S1=***** ***** *** ***** **** *** **** ***


------------------
(program exited with code: 0)
Press return to continue
```

Эти очень простые программы – первый шаг на пути решения сложных лингвистических задач, к которым относятся программирование автоматического транскриптора, автоматический морфологический, синтаксический и семантический анализ, анализ и синтез естественной речи и другие.

# АВТОМАТИЧЕСКИЙ ТРАНСКРИПТОР

Программирование автоматического транскриптора, который бы переводил буквенные символы в транскрипционные знаки – сама по себе достаточно сложная задача, если нам необходима не фонемная транскрипция, которую можно взять из словаря, а аллофонная (иными словами, акустическая или реальная), учитывающая явления связной речи. Такой транскриптор покажет пользователю, например, где в английском нужно придыхание, а где нет, где вместо канонического придыхательного [$t^h$] будет альвеолярный удар/скольжение или глоттализация и гортанный взрыв, а где звук вообще выпадет, как [d] из союза *and* и тому подобные «штучки», крайне полезные как для статистической обработки потенциальных позиций для явлений связной речи, так и для изучающего иностранный язык или свой этнический, который по ряду причин стал исчезающим, как, например, эвенкийский. Прежде чем создавать сложный фрейм в среде Lazarus, где будут учтены все переходы от буквы к звуку, необходимо научиться программировать алгоритмы обработки отдельных буквенных символов – одиночных и сочетаний, например, диграфов, трёх- и четырёхбуквенных. Ну а начать следует с самых простых случаев, когда буква имеет однозначное прочтение и её транскрипция совпадает с символами клавиатуры.

ЗАДАЧА 1

**Транскрибирование при совпадении со знаками на клавиатуре: «m»→[m]**

В этой программе имеется небольшая описательная часть, где дано три переменные: две из них – это переменные типа string (строка) – строка ввода и строка замены, а одна – целое число для определения места искомой буквы в строке ввода для последующей её замены на транскрипционный знак. В теле программы есть 1) процедура обнуления счётчиков для обеспечения бесперебойной работы программного кода, 2) процедура считывания с устройства ввода, 3) условный цикл замены одного символа (буквы алфавита) на другой (транскрипционный знак в определённой аранжировке) и внедрения его в строку ввода; 4) процедура вывода результата на экран.

```
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.
```

Теперь решим более сложную задачу, где букв не одна, а три, типов прочтения тоже не один, а три, и транскрипционные знаки в основном не совпадают с символами на клавиятуре.

# ЗАДАЧА 2

На языке Паскаль представить алгоритм выбора варианта прочтения буквенной последовательности «ere».

Вариант 1: [ɛə̯]: только для where, there

Вариант 2: [ɜ̯]: только для were.

Вариант 3: [iə̯] все остальные.

Сначала отработаем отдельные части этой задачи. Научим машину транскрибировать буквосочетание в одном конкретном слове (Вариант 2).

**Простейший вариант обработки строк на языке Pascal**

```
1. program Read_ere;
2. var
3.   sInp : String;
4. begin
5.  ReadLn(sInp);
6.   If sInp = ´were´
7.     then begin
8.       WriteLn ('w -[ɜ̯]') ;
9.     end;
10. end.
```

**1стр.** – ключевое слово **program**, за которым следует название программы (имя должно начинаться с буквы, состоять только из букв, цифр и знака нижнего подчеркивания; минус, плюс, точка и т. д. нельзя; различия между большими и маленькими буквами в зарезервированных словах и индентификаторах отсутствует), после названия программы точка с запятой (можно далее или enter, или пробел, или вообще ничего, но эстетических задач лучше enter).

**2 стр.** – секция определений, в данном случае – блок определения переменных: ключевое слово **var**, за которым через пустое место (whitespace – пробел или Enter, или Таб) следует описание переменных: имя, двоеточие, тип, далее точка с запятой. Если вводится несколько переменных одного типа, то можно записать из 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.

**3 стр.** – имя переменной – строки ввода, тип переменной – строка (string – зарезервированное слово)

***Типы переменных***

*Основные*: integer (целые числа), real (числа с плавающей запятой – вещественные, напр. $5.08\text{-}1.3\cdot10^{7}$), char (символ, напр. 'b' 'í'), 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»).

**7 стр.** – За условием следует зарезервированное слово **then**, за которым идет оператор Pascalя, за которым может следовать (а может и отсуттвовать 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)

**!!** Напомним, что зарезервированные слова – слова, которые используются в строго определенных случаях и не могут быть использованы в качестве имён 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

**Вариант обработки строк на языке Pascal: усложнение 1**

```
1.  program prog2;
2.  const
3.  count_ere = 2;
4.  Tarray_ere : array [1..count_ere] of string = ('where', 'there');
5.  var
6.    sInp : String;
7.  begin
8.   ReadLn(sInp);
9.    If sInp = 'were'
10.     then begin
11.       WriteLn ('w[ɜ]') ;
12.     end;
13.  If sInp = TArray_ere[1]
14.  then begin
15.    WriteLn ('w[ɛə]');
16.  end;
17.  If sInp = TArray_ere[2]
18.  then begin
19.    WriteLn ('th[ɛə]');
20.  end;
21.end.
```

Как нетрудно догадаться, такая запись неудобна по причине избыточности кода (в

реальности в такой программе слишком много почти одинаковых строк, например, если элементов в массиве не 2, а намного больше, а массивов не 1, а несколько).

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
```

Первая строка: У данной функции три переменных/аргумента на входе (указаны в скобках) – исходная строка (заданное слово), строка для поиска (буква или буквенная последовательность в заданном слове) и строка замены (на что заменить найденную букву или буквенную последовательность – в данном примере это один из транскрипционных знаков [ɜ˞], [ɛə˞], [iə˞]). Аргументов может и не быть, если это функция без аргументов. В скобках через двоеточие указывается тип переменной/аргумента, в данном случае – строковая переменная – string/string. После скобки двоеточие означает тип возвращаемого функцией результата, в данном случае – строка.

Вторая строка – 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 искомой буквенной последовательности. 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.

Третья строка – sTmp:String – временная переменная для хранения промежуточного значения строки. Имя произвольно. В нашем случае это то слово, в котором мы хотим прочесть буквенную последовательность -ere-.

Используются четыре библиотечные функции: *Pos, Delete, Insert, Length*.

*Функция Pos* возвращает индекс подстроки строке, напр. Для строки 'were' и подстроки 'ere' функция возвращает индекс 2 – искомая буквенная последовательность начинается со второго символа в строке.

*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, 2, 3), где s = were, строке s присваивается значение *w* (начиная со второго символа три удаляются). Обратите внимание, что в процедуру Delete можно передавать только строковые переменные. Если пытаться ввести напрямую Delete('were', 2, 3), то компилятор выдаст сообщение об ошибке.

*Функция Length* возвращает значение длины строки (количество символов в строке). Напр. Length ('were')=4. В нашем случае – длина искомой буквенной последовательности = 3 (три символа).

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

**Итоговый вариант программы**

```
1.  program Read_ere;
2.  const
3.  count_ere = 2;
4.  Tarray_ere : array[1..count_ere] of string=('there','where');
5.  Tstring_ere1 = 'were';
6.  var
7.  sinp,srpl: string;
8.  i: Integer;
9.  function Replace(sinput,ssearch,sreplace: string): string;
10. var i1:integer;
11. stmp:string;
12. begin
13. i1:=Pos(ssearch, sinput);
14. stmp:=sinput;
15. Delete(stmp, i1, length(ssearch));
16. Insert(sreplace, stmp,i1);
17. Replace:=stmp;
18. end;
19. begin
20.   Repeat
21.   srpl:='';
22.   readln(sinp);
23.   For i:=1 to count_ere do begin
24.     If sinp=Tarray_ere[i] then begin
25.       srpl:='-[ɛə]-';
26.     end;
27.   end;
28.   If sinp=Tstring_ere1
29.     then begin
30.       srpl:='-[ɜ]-';
31.     end;
32.   If(srpl='')And(Pos('ere',sinp)>0)
33.     then begin
34.       srpl:='-[iə]-';
35.     end;
36.   If Pos('ere',sinp)>0
37.     then begin
38.       sinp:=replace(sinp,'ere',srpl);
39.     end;
40.   If Pos('ere1',sinp)>0
41.     then begin
42.       sinp:=replace(sinp,'ere1',srpl);
43.     end;
44.   Writeln(sinp);
```

```
45.  Until sinp='end';
46. end.
```

## ЗАДАЧА 3

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

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**

```
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.
```

**Вариант обработки строк на языке Pascal: усложнение 1**

```
1.  program prog1;
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.  if sInp = 'create'
11.  then begin
12.  WriteLn ('cr -[iei] - te') ;
13.  end;
14. end.
```

По сравнению с предыдущим вариантом добавлено еще одно ветвление для слова «create».

**Вариант обработки строк на языке Pascal: усложнение 2**

```
1.  program prog2;
2.  const
3.  countEI = 3;
4.  TarrayEI : array [1..countEI] of string = ('great', 'break', 'steak');
5.  var
6.    sInp : String;
7.  begin
```

```
8.  ReadLn(sInp);
9.   If sInp = 'read'
10.    then begin
11.      WriteLn ('r -[i]- d, Infinitive and Present Simple, if other  - [e]') ;
12.    end;
13.  if sInp = 'create'
14.  then begin
15.  WriteLn ('cr -[iei] - te') ;
16.  end;
17.  If sInp = TArrayEI[1]
18.  then begin
19.    WriteLn ('gr -[ei]- t');
20.  end;
21.  If sInp = TArrayEI[2]
22.  then begin
23.    WriteLn ('br -[ei]- k');
24.  end;
25.  If sInp = TArrayEI[3]
26.  then begin
27.    WriteLn ('st -[ei]- k');
28.  end;
29. end.
```

По сравнению с предыдущим вариантом добавлена константа из трех составляющих, на каждую из которых добавлено ветвление. И вновь возникает то же неудобство записи, о котором говорилось при программировании алгоритма решения задачи 1 – избыточность кода. Оптимизируем наш алгорит аналогичным путём – через функцию по замене одних символов на другие.

```
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
```

Первая строка: три аргумента в скобках – исходная строка (заданное слово с диграфом), строка для поиска (дираф в заданном слове) и строка замены (на что заменить найденный дираф – один из транскрипционных знаков [i], [e], [ei]).

Вторая строка – var i1:Integer, где i1 – это индекс начала диграфа.

Третья строка – sTmp:String – в нашем случае это то слово, в котором мы хотим прочесть дираф -ea-.

Снова используются четыре библиотечные функции: *Pos, Delete, Insert, Length*.

*Функция Pos* Для строки 'read' и подстроки 'ea' функция возвращает индекс 2 – дираф начинается со второго символа в строке.

*Функция Delete* удаляет в нашем случае из строки create два символа, начиная с третьего

символа (см. аргументы в круглых скобках). Иными словами, для данного случая Delete(s, 3, 2), где s = create, строке s присваивается значение *crte* (начиная с третьего символа два удаляются).

*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 обратить внимание на отсутствие знака «;» после **end** в строке 51 презентации. Причина – препозиция к ключевому слову **else**.

**Итоговый вариант программы**

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.

```
1.  program ReadDigraf;
2.  const
3.    CountEI = 3;
4.    TArrayEI : array [1..CountEI] of string = ('great','break','steak');
5.    CountE = 20;
6.    TArrayE : array [1..CountE] of string =(
7.    'heavy','heaven','breast','meadow',
8.    'weapon','wealth','wealthy','ready',
9.    'dead','bread','death','breath','measure','pleasure', 'treasure',
10.   'head','thread','threat','meant', 'steady'
11.   );
12.   TStringI = 'read';
13.   TStringIEI = 'create';
14. var
15.   sInp, sRpl, sTail : String; // sTail -- строка примечаний в нашем случае для пояснения,
      что одно и тоже написание read может относиться и к инфинитиву, и к Past Simple
16.   i : Integer;
17. function Replace(sInput,sSearch,sReplace:String):String;
18. var i1:Integer;
19.     sTmp:String;
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.
28. begin
29.   sRpl := '';
30.   sTail := '';
31.   ReadLn(sInp);
32.   If sInp = TStringI
```

```
33.     then begin
34.       sRpl := ' -[i]- ';
35.       sTail := ', Infinitive and Present Simple, if other  - [e]';
36.     end;
37.   If sInp = TStringIEI
38.     then begin
39.       sRpl := ' -[iei]- ';
40.     end;
41.   For i := 1 to CountEI do begin
42.     If sInp = TArrayEI[i] then begin
43.       sRpl := ' -[ei]- ';
44.     end;
45.   end;
46.   For i := 1 to CountE do begin
47.     If sInp = TArrayE[i] then begin
48.       sRpl := ' -[e]- ';
49.     end;
50.   end;
51.   If sRpl ='' then sRpl := ' -[i]- ';
52.   i := Pos ('ea',sInp);
53.   If i>0 then begin
54.     Delete (sInp,i,2);
55.     Insert (sRpl, sInp, i);
56.     If Length(sTail)>0 then sInp := sInp + sTail;
57.   end
58.   else begin
59.     WriteLn ('WARNING!!! "ea" NOT FOUND!!!');
60.   end;
61.   WriteLn(sInp);
62. end.
```

ЗАДАЧА 4

Составить алгоритм чтения синонимичных диграфов -ou-, ow.

Условия:

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.

По сравнению с предыдущими задачами, сложность этой в наличии двух «синонимичных» диграфов – *-ou- / -ow-*, для которых применяются одинаковые правила прочтения.

**Программа**

```
1. program ReadDigraphsouow;
2. const
3. Countou = 7;
```

```
4.  TArrayou : array [1..Countou] of string = ('soul', 'poultry', 'shoulder', 'bowl', 'low', 'own',
    'owe');
5.  Counta = 4;
6.  TArraya : array [1..Counta] of string = ('country', 'cousin', 'young', 'touch');
7.  Countu1 = 4 ;
8.  TArrayu1 : array [1..Countu1] of string = ('should', 'would', 'could', 'wounded');
9.  Countu = 2;
10. TArrayu : array [1..Countu] of string = ('route', 'rouge');
11. Tstringou = 'bow';
12. Tstringu1 = 'wound';
13. var
14. sInp, SRpl, sTail : string;
15. i : Integer;
16. function Replace (sInput, sSearch, sReplace : string) : string ;
17. var i1 : Integer;
18. sTmp : string;
19. begin
20.  i1:= Pos(sSearch, sInput);
21.  sTmp:= sInput;
22.  delete(sTmp, i1, Length(sSearch));
23.  Insert(sReplace, sTmp, i1);
24.  Replace:= sTmp;
25.  end;
26.  begin
27.  sRpl := '';
28.  sTail := '';
29.  ReadLn(sInp);
30.  if sInp = Tstringou
31.  then begin
32.  sRpl := ' -[əʊ]- ';
33.  sTail := ', в значении "Лук", if other - [aʊ]';
34.  end;
35. if sInp = Tstringu1
36.  then begin
37.  sRpl := ' -[ʊ]- ';
38.  end;
39.  For i := 1 to Countou do begin
40.  if sInp = TArrayou[i] then begin
41.  sRpl := ' -[əʊ]- ';
42.  end;
43.  end;
44.  For i:= 1 to Counta do begin
45.  if sInp = TArraya[i] then begin
46.  sRpl := ' -[ʌ]- ';
47.  end;
48.  end;
49.  For i := 1 to Countu1 do begin
50.  if sInp = TArrayu1[i] then begin
51.  sRpl := ' -[ʊ]- ';
52.  end;
```

```
53.  end;
54. For i := 1 to Countu do begin
55.  if sInp = TArrayu[i] then begin
56.  sRpl := ' -[u]- ';
57.  end;
58.  end;
59.  If (sRpl ='') And (Pos('ou',sInp)>0)
60.    then begin
61.      sRpl := ' -[au]- ';
62.   End;
63.   If (sRpl ='') And (Pos('ow',sInp)>0)
64.    then begin
65.      sRpl := ' -[au]- ';
66.   End;
67.   If Pos('ou',sInp)>0
68.    then begin
69.      sInp:=Replace(sInp,'ou',sRpl)+sTail;
70.    end;
71.    If Pos('ow',sInp)>0
72.    then begin
73.      sInp:=Replace(sInp,'ow',sRpl)+sTail;
74.    end
75.   else begin
76.    WriteLn ('WARNING!!! "ou/ow" NOT FOUND!!!');
77.   end;
78.    WriteLn(sInp);
79. end.
```

Как видим, вышеуказанная сложность решается непосредственно в теле программы с помощью отдельных условий для каждого из двух рассматриваемых диграфов (см. строки 59–73). При описании исключений в виде констант (см. первый массив, объявленный в строке 4), слова с одинаковым типом прочтения – в данном случае как [əʊ] – собраны в один массив независимо от написания – через -ou- или через -ow-, а для единичных исключений просто использованы разные строковые константы (см. строки 11–12).

ЗАДАЧА 5

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)

По сравнению с предыдущими программами, 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 (диграф 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 (диграф занимает две позиции, потому что он диграф – 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 string = ('the', 'this', 'these', 'that', 'those', 'they', 'them',
    'their', 'there', 'then', 'than');
5.  CountTH2 = 2;
6.  TArrayTH2 : array [1..CountTH2] of string = ('Themes', 'Thomas');
7.  vowel : set of char = ['a', 'e', 'i', 'o', 'u', 'y'];
8.  var
9.  sInp, SRpl : string;
10. i : Integer;
11. cBefore, cAfter: char;
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.  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 := ' -[θ]- ';
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. else begin
49.     WriteLn ('WARNING!!! "th" NOT FOUND!!!');
50.   end;
51.  WriteLn(sInp);
52. end.
```

## АВТОМАТИЧЕСКАЯ ОБРАБОТКА ТЕКСТОВ НА ЕСТЕСТВЕННОМ ЯЗЫКЕ: ПОИСК ПО ТОЧНОЙ СЛОВОФОРМЕ

В качестве простого примера автоматической обработки грамматических показателей текста приведём программирование автоматического поиска нужных грамматических явлений.

ЗАДАЧА 6

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 блоке **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 блоке **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;
```

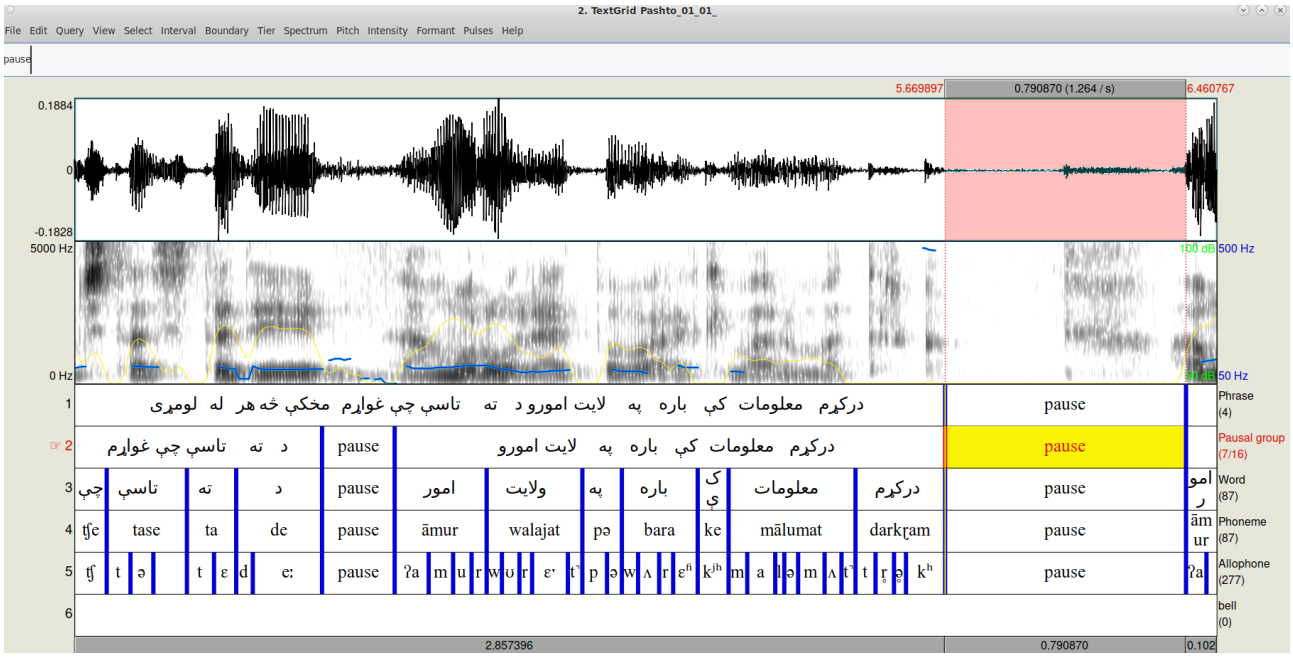
```
46. procedure CheckToBe;
47. var i : integer;
48. begin
49.   For i := 1 to Counttobe do begin
50.     if S = Arraytobe [i] then begin
51.       Tcounttobe [i] := Tcounttobe [i]+1;
52.     end;
53.    end;
54. end;
55. procedure final;
56. var i : integer;
57. begin
58.   close(F);
59.   Writeln ('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.
```

## ОБРАБОТКА ТЕКСТ-ГРИДОВ НА FREE PASCAL

Подготовленная поуровневая аннотация в Praat является хорошим материалом для программирования автоматического подсчета тех или иных явлений, например, количества пауз, их распределение по типам (если в аннотации паузы классифицированы по типам, напр., заполненная/незаполненная и т. д.), количества случаев выпадения сегментов, замен одних основных знаков на другие (что может говорить о заменах одних фонем на другие) и многого другого в зависимости от подробности разметки. Здесь продемонстрируем решение простейшей задачи подсчета количества пауз.

### ЗАДАЧА 7

Запрограммировать алгоритм автоматического поиска пауз между фраз и внутри фраз (в целом). Подсчёт будет осуществляться на втором уровне аннотации (Tire 2), который условно назван Pausal group (паузальная группа), на котором отмечены оба вида пауз (см. рисунок ниже).




При составлении аннотации в PRAAT образуется текстовый файл (TextGrid). Ниже показана начальная такого файла.

```
File type = "ooTextFile"
Object class = "TextGrid"

xmin = 0
xmax = 334.1009977324263
tiers? <exists>
size = 6
item []:
    item [1]:
        class = "IntervalTier"
        name = "Phrase"
        xmin = 0
```

```
        xmax = 334.1009977324263
        intervals: size = 4
        intervals [1]:
            xmin = 0
            xmax = 1.159214861425261
            text = ""
        intervals [2]:
            xmin = 1.159214861425261
            xmax = 5.66989698603239
            text = " لومړی له هر څه مخکې غواړم چې تاسې ته د امورو لایت په باره کې معلومات
درکړم"
        intervals [3]:
            xmin = 5.66989698603239
            xmax = 6.460767430750264
                  text = "pause"
    …
```

Формат образованного автоматически при аннотировании текстового файла – UTF 16 (на один юникодовский символ выделяется минимум 2 байта=16 бит).Язык фри паскаль предполагает, что каждая текстовая строка состоит из однобайтных символов: все встроенные функции обработки текста базируются на этом соглашении.

Например, слово File в кодировке UTF 16 (в шестнадцатиричном исчислении) выглядит следующим образом:

| F | i | l | e |
|---|---|---|---|
| 00 70 | 00 105 | 00 108 | 00 101 |

Нули используются для кодирования юникодовских символов. На паскале данная строка выглядит как 70 105 108 101 (без нулей). Чтобы строку можно было обрабатывать на фри паскале необходимо нули удалить. Строка sIntervals = 'intervals: size =' нужна для дальнейшего подсчета общего количества поставленных меток на данном уровне.

**Программа**

```
1.  program test;
2.  Const
3.    sBegin2 = 'item [2]';
4.    sEnd2 = 'item [3]';
5.    sIntervals = 'intervals: size =';
6.    sSearch = ' text = "pause"';
7.  var
8.    f:Text;
9.    S,S1, SIntervalsTot:String;
10.   IsTier2:Boolean; // Это вопрос: принадлежит ли читаемая строка к искомому уровню
      2?
11.   iCountPause:Integer;
12. Function ConvertUtf(sIn:String):String;
13. var
14.  i: Integer; // индекс для цикла «пробегания» по строке с целью поиска нулей
15.  sOut:String; // временная переменная для записи строки без нулей
16. begin
17.   sOut:='';
18.   For i:=1 to Length(sIn) do begin;
```

```
19.     If Ord(sIn[i])<>0 Then sOut:=sOut+sIn[i]; // Ord – это стандартная функция паскаля, преобразующая символ в код символа (напр., Ord('F')=70 – см. таблицу)
20.   end;
21.   ConvertUtf:=sOut;
22. end;
23. begin // начало выполнения программы
24.   IsTier2:=False;
25.   iCountPause:=0; // Эта и предыдущая строки – начальная инициализация переменных (значения сбрасываются на ноль)
26.   Assign(f,'text.txt');
27.   Reset(f); // Эта и предыдущая строки – открытие текстового файла для чтения (вместо text.txt можно вставлять нужное название файла)
28.   While not eof(f) do begin
29.     Readln(f,S1);
30.     S:=ConvertUtf(S1);
31.       If Pos(sBegin2, S) <> 0 // если Pos не равно нулю, то в строке S присутствует подстрока  sBegin2, что сигнализирует о начале искомого уровня 2
32.       Then begin
33.         IsTier2:=True;
34.       end;
35.     If Pos(sEnd2, S) <> 0
36.       Then begin
37.         IsTier2:=False;
38.       end;
39.     If (IsTier2) And (Pos(sIntervals,S)<>0)
40.       then begin
41.         SIntervalsTot := Copy(S,Pos(sIntervals,S)+Length(sIntervals)-1,10); //  SintervalsTot – временная переменная, в которой мы храния количество меток на уровне; функция Copy вырезает из строки S начиная с индекса, определенного вторым аргументом, количество символов, определенных третьим аргументом; функция Pos показывает, с какого символа начинается подстрока  sIntervals, прибавляя к ней длину самой подстроки  sIntervals и, отняв единицу, мы получаем индекс, с которого начинается количество меток; число десять – чтобы можно было прочитать максимально 10 символов после индекса, что заведомо превосходит количество возможных меток, исходя из того, что на данном уровне количество меток заведомо уложится в десятизначное число (напр., 1076 или 9999)
42.       end;
43.     If (IsTier2) And (Pos(sSearch,S)<>0)
44.       Then begin
45.         iCountPause:=iCountPause+1;
46.       end;
47.   end;
48.   Close(f);
49.   WriteLn('Total inervals in tier 2 is ',SIntervalsTot,' Pause count = ',iCountPause);
50. end.
```

**ПРОГРАММИРОВАНИЕ ТЕСТОВ**

# ЗАДАЧА 8

Запрограммировать многовыборный тест типа:

Choose 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

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.

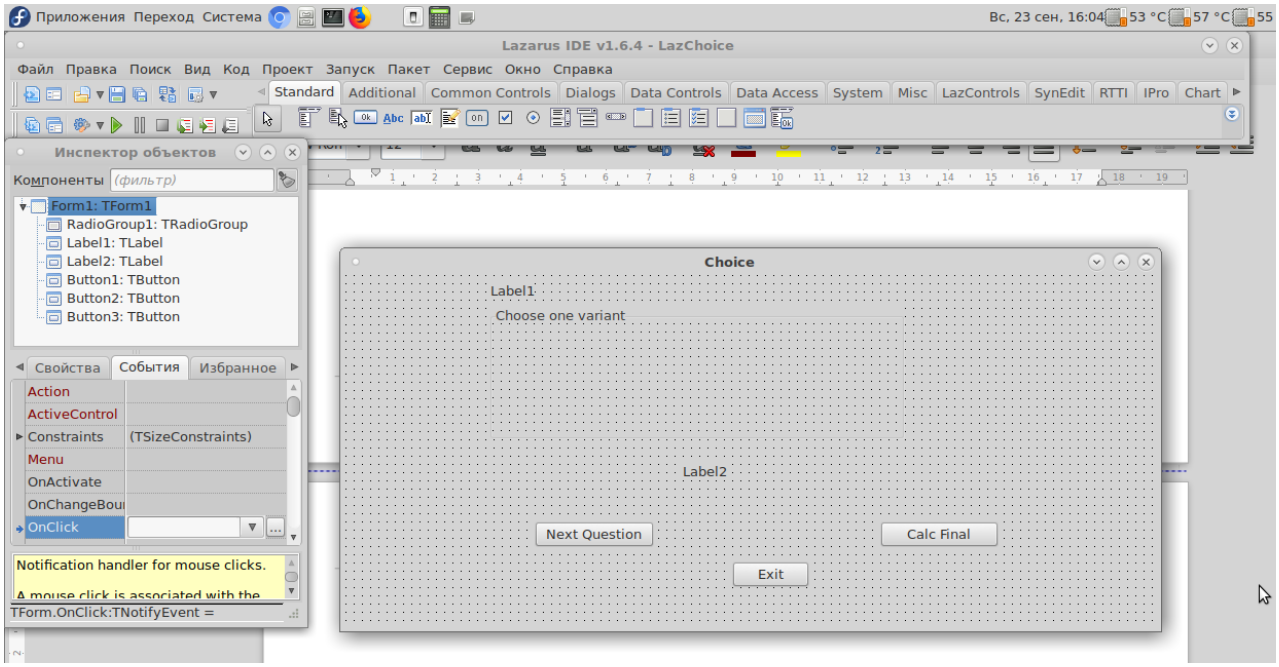
Программа

```
1. program MultipleChoice;
2. const
3. count = 5;
4. Questions:Array[1..count] of string = (
5. '1. Her eyes are red. She ... all night.',
6. '2. I cut myself when I ... .',
7. '3. I look ... hearing from you.',
8. '4. I ... to him before you arrived.',
9. '5. His father is a ... . He works on a big modern ship.');
10. AnswerVariants: Array [1..count] of string = (
11. 'a) cries; b) is crying; c) cried; d) has been crying.',
12. 'a) shaved; b)  was shaving; c) had shaven; d) had been shaving.',
13. 'a) after; b) for; c) at; d) forward to.',
14. 'a) was talking; b) talked; c) had talked; d) had been talking.',
15. 'a) sailor; b) seller; c) doctor; d) cop.');
16. CorrectVariant: Array [1..count] of char = (
17. 'd', 'b', 'd', 'c', 'a');
18. var
19. InpChar : char;
20. CorrectAnswer: integer;
21. Counter : integer;
22. begin
```

```
23. CorrectAnswer:= 0;
24. For Counter:= 1 to count do begin
25. Writeln('-------------------------------------------------');
26. Writeln('Question No ',Counter);
27. Writeln (Questions [Counter]);
28. 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.
```

Данная программа бесперебойно работает, но интерфейс пользователя в ней очень устаревший. Уже много лет в качестве стандарта приняты программы, реализующие оконный интерфейс взаимодействия с пользователем (использование мыши, нажатие экранных кнопок, чтение приглашений и реакция пользователя). В качестве инструмента, позволяющего создавать приложения с оконным интерфейсом разработчики Free Pascal предлагают программный продукт Lazarus. Среда программирования Lazarus позволяет в интерактивном режиме создавать экранные формы и программировать пользовательский интерфейс с помощью языка Pascal. Компилятором в данной среде по-прежнему является Free Pascal, но стандартные библиотеки значительно расширены добавлением классов и объектов, предназначенных для реализации пользовательского интерфейса. Программа, созданная в Lazarus, работает без радикальных изменений как в Windows, так и в Linux. Парадигма построения оконных приложении в значительной степени основана на объектно-ориентированном программировании, концепции событийно управляемых программ и вообще требует достаточно глубокого изучения теории, перед тем как начать писать программы в этой среде. Поскольку эти вопросы выходят далеко за пределы данного курса, мы покажем построение оконного интерфейса на примере программы тестирования, аналогичной приведённой выше.

# СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА С ПОМОЩЬЮ LAZARUS



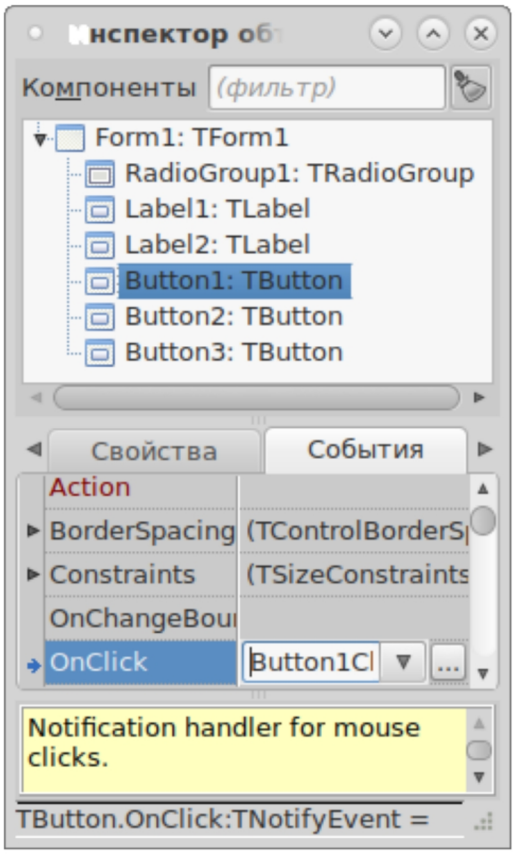

На форму поместим два компонента типа TLabel, компонент TRadioGroup, три кнопки (компонента) TButton. Изменим заголовки для объектов согласно картинки
– Form1 – Choice;
– RadioGroup1 – Choose one variant;
– Button1 – Next Question;
– Button2 – Calc Final;
– Button3 – Exit.
Код Unit1.pas выглядит следующим образом:

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs,
StdCtrls,
  ExtCtrls;

Const
  iMaxChoice = 5; // Кол-во тестов
  iMaxAnswer = 4; // Вариантов ответа на один вопрос

type
```

```
  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Label1: TLabel;
    Label2: TLabel;
    RadioGroup1: TRadioGroup;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure ChangeQuizDisplay(iQuiz:Integer);
    procedure FormCreate(Sender:TObject);
  private
    { private declarations }
    iCurQuest,iRightAns: Integer;
  public
    { public declarations }
  end;
   { Описание типа для тестов }
   TArrayOfAnswer = Array[1..iMaxAnswer] of String;
   TQuizRec = Record
      Question1 : String;
      Answer1 : TArrayOfAnswer;
      RightAnswer1 : Integer;
   end;
   TQuizArray = Array [1..iMaxChoice] of TQuizRec;

var
  Form1: TForm1;

implementation
{$R *.lfm}
Const
  QuizRec1 : TQuizArray = (
  (Question1:'Her eyes are red. She ... all night.';
   Answer1:('cries','is crying','cried','has been
crying');RightAnswer1:4),

   (Question1:'I cut myself when I ... .';
  Answer1:('shaved','was shaving','had shaven','had been
shaving');RightAnswer1:2),

  (Question1:'I look ... hearing from you.';
  Answer1:('after','for','at','forward to');RightAnswer1:4),

  (Question1:'I ... to him before you arrived.';
  Answer1:('was talking','talked','had talked','had been
talking');RightAnswer1:3),
```

```
  (Question1:'His father is a ... . He works on a big modern
ship.';
  Answer1:('sailor','seller','doctor','cop');RightAnswer1:1)
    );

Procedure TForm1.ChangeQuizDisplay(iQuiz:Integer);
var s1,s2:String;
  i1: Integer;
begin
  Str(iQuiz:1,s1);
  Str(iMaxChoice,s2);
  Label1.Caption:=QuizRec1[iQuiz].Question1;
  Label2.Caption:='Question #'+s1+' from '+s2;
  RadioGroup1.Items.Clear;
  for i1:=1 to iMaxAnswer do begin
     RadioGroup1.Items.Add(QuizRec1[iQuiz].Answer1[i1]);
  end;
  RadioGroup1.Update;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  If (QuizRec1[iCurQuest].RightAnswer1-1)=RadioGroup1.ItemIndex
    then iRightAns:=iRightAns+1;
  iCurQuest:=iCurQuest+1;
  If iCurQuest>iMaxChoice
    then begin
       Button1.Enabled:=False;
       RadioGroup1.Enabled:=False;
       Button2.Enabled:=True;
       Label2.Caption:='Finish. Please press Calc final';
      end
    else begin
      ChangeQuizDisplay(iCurQuest);
    end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var s1,s2:String;
  r: Real;
begin
  r:=iRightAns / iMaxChoice *100.0;
  Str(iRightAns:1,s1);
  Str(r:2:1,s2);
  Label2.Caption:='Right answer '+s1+' or '+s2+'%';
  Button2.Enabled:=False;
  Button3.Enabled:=True;
  If r>80 Then s1:='A'
    else if r>60 then s1:='B'
      else if r>50 then s1:='C'
      else s1:='D';
```

```
  Label1.Caption := 'The final estimate is '+s1;
  case s1 of
    'A' : Label1.Font.Color:=clGreen;
    'B' : Label1.Font.Color:=clBlue;
    'C' : Label1.Font.Color:=clYellow;
    else Label1.Font.Color:=clRed;
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;

Procedure TForm1.FormCreate(Sender:TObject);
begin
  iCurQuest:=1;
  iRightAns:=0;
  ChangeQuizDisplay(iCurQuest);
end;


end.
```



В инспекторе объектов на закладке События для компонентов типа кнопки (Button) укажем соответствующие обработчики событий типа OnClick. Для Button1 – выберем в выпадающем списке процедуру Button1Click, для Button2 – Button2Click и для Button3 – Button3Click.Рассмотрим эту программу пошагово. Первые строки не рассматриваем, это описание класса формы, которое Lazarus создает автоматически. Исключение – две процедуры и две переменных, которые мы добавим вручную:

```
    procedure ChangeQuizDisplay(iQuiz:Integer);
    procedure FormCreate(Sender:TObject);
  private
    { private declarations }
    iCurQuest,iRightAns: Integer;
```

Процедура ChangeQuizDisplay – будет менять вопрос и варианты ответа, согласно номеру iQuiz. Процедура FormCreate – процедура инициализации переменных, имя для FormCreate – стандартное имя в классе форм для того, чтобы делать некоторые действия при запуске программы.
Переменные iCurQuest и iRightAns – номер теста и номер правильного ответа в текущем тесте соответственно.
Далее рассмотрим вопрос по структуре данных, в которой хранятся вопросы, варианты ответа и номер правильного ответа.

```
Const
  iMaxChoice = 5; // Кол-во тестов
  iMaxAnswer = 4; // Вариантов ответа на один вопрос
Type
   TArrayOfAnswer = Array[1..iMaxAnswer] of String;
   TQuizRec = Record
      Question1 : String;
      Answer1 : TArrayOfAnswer;
      RightAnswer1 : Integer;
   end;
   TQuizArray = Array [1..iMaxChoice] of TQuizRec;
```

iMaxChoice и iMaxAnswer – константы, назначение которых понятно из комментария (5 наборов тестов, 4 варианта ответа в каждом тесте).
TArrayOfAnswer – массив из 4 (см. выше) строк с вариантами ответов. TQuizRec – структура типа запись (record) из разных типов для удобства работы. Содержит Question1 – строка с вопросом, Answer1 – массив строк с вариантами ответов, RightAnswer1 – число, соответствующие номеру правильного ответа (с 1 до 4, или, точнее, до iMaxAnswer).
Далее покажем, как в данную структуру можно вписывать данные. Для примера, данные впишем в программу как структура–константа.

```
Const
  QuizRec1 : TQuizArray = (
  (Question1:'Her eyes are red. She ... all night.';
   Answer1:('cries','is crying','cried','has been
crying');RightAnswer1:4),

   (Question1:'I cut myself when I ... .';
  Answer1:('shaved','was shaving','had shaven','had been
```

```
shaving');RightAnswer1:2),

  (Question1:'I look ... hearing from you.';
  Answer1:('after','for','at','forward to');RightAnswer1:4),

  (Question1:'I ... to him before you arrived.';
  Answer1:('was talking','talked','had talked','had been
talking');RightAnswer1:3),

  (Question1:'His father is a ... . He works on a big modern
ship.';
  Answer1:('sailor','seller','doctor','cop');RightAnswer1:1)
    );
  QuizRec1 : TQuizArray = (
  (Question1:'Her eyes are red. She ... all night.';
   Answer1:('cries','is crying','cried','has been
crying');RightAnswer1:4),

   (Question1:'I cut myself when I ... .';
  Answer1:('shaved','was shaving','had shaven','had been
shaving');RightAnswer1:2),

  (Question1:'I look ... hearing from you.';
  Answer1:('after','for','at','forward to');RightAnswer1:4),

  (Question1:'I ... to him before you arrived.';
  Answer1:('was talking','talked','had talked','had been
talking');RightAnswer1:3),

  (Question1:'His father is a ... . He works on a big modern
ship.';
  Answer1:('sailor','seller','doctor','cop');RightAnswer1:1)
    );
```

Элементы массива пишутся в круглых скобках, разделяются запятыми. Элементы записи предваряются именем, затем двоеточие, после которого пишется значение этого элемента, разделяются эти элементы точкой с запятой – в пределах одной записи.
Далее рассмотрим программный код (пойдем немного не последовательно, будем рассматривать процедуры в порядке их вызова в программе.

```
Procedure TForm1.FormCreate(Sender:TObject);
begin
  iCurQuest:=1;
  iRightAns:=0;
  ChangeQuizDisplay(iCurQuest);
end;
```

Процедура FormCreate вызывается из стандартных библиотек Lazarus, которые линкуются совместно с пользовательской программой. Если она не описана, вызывается стандартная процедура. Мы в данной процедуре инициализируем описанные приватные переменные iCurQuest (текущий номер теста), iRightAns (количество данных правильных ответов) и

вызываем процедуру ChangeQuizDisplay, которую мы рассмотрим далее.

```
Procedure TForm1.ChangeQuizDisplay(iQuiz:Integer);
var s1,s2:String;
  i1: Integer;
begin
  Str(iQuiz:1,s1);
  Str(iMaxChoice,s2);
  Label1.Caption:=QuizRec1[iQuiz].Question1;
  Label2.Caption:='Question #'+s1+' from '+s2;
  RadioGroup1.Items.Clear;
  for i1:=1 to iMaxAnswer do begin
     RadioGroup1.Items.Add(QuizRec1[iQuiz].Answer1[i1]);
  end;
  RadioGroup1.Update;
end;
```

Описывается три переменных – две типа строка и одна – целое число. Далее стандартными процедурами Str осуществляется преобразование чисел iQuiz и iMaxChoice в строки s1 и s2 соответственно.
Далее мы изменяем текст объекта Label1, изменяя его property Caption, присваиваяя этой property значение строки вопроса с номером iQuiz.
Текст объекта Label2 меняем на строку вида «Вопрос № *** из ***».
Далее сбрасываем значения вариантов ответа путем обращения к процедуре подобъекта RadioGroup1 – Items – процедуре Clear (RadioGroup1.Items.Clear;).
В цикле со счетчиком i1 добавляем варианты ответов в класс TStrings через процедуру – RadioGroup1.Items.Add.
После добавления всех вопросов производим перерисовку радиокнопок через процедуру RadioGroup1.Update.
Далее рассмотрим обработчики нажатий кнопок Button1 – Button2 – Button3.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  If (QuizRec1[iCurQuest].RightAnswer1-1)=RadioGroup1.ItemIndex
    then iRightAns:=iRightAns+1;
  iCurQuest:=iCurQuest+1;
  If iCurQuest>iMaxChoice
    then begin
       Button1.Enabled:=False;
       RadioGroup1.Enabled:=False;
       Button2.Enabled:=True;
       Label2.Caption:='Finish. Please press Calc final';
      end
    else begin
      ChangeQuizDisplay(iCurQuest);
    end;
end;
```

Кнопка 1 (с заголовком «Следующий вопрос») при нажатии на нее вызывает следующий код – сначала проверяем выбранный вариант с правильным – сравниваем значение RadioGroup1.IndexItem со значением правильного ответа. В данном случае необходимо

указать, что номер выбранного варианта хранится, начиная с нуля – т. е. то, что в массиве идет как 1–4, в данной property идет 0–3. Поэтому сравнение идет как (QuizRec1[iCurQuest].RightAnswer1-1)=RadioGroup1.ItemIndex. T.e. от RightAnswer мы отнимаем 1. Если выбранный вариант соответствует правильному ответу, добавляем единицу к счетчику правильных ответов iRightAns.
В следующей строке (iCurQuest:=iCurQuest+1;) добавляем единицу к счетчику тестов.
Далее проверяем, закончились ли у нас тесты (If iCurQuest>iMaxChoice). Если закончились, то переменная iCurQuest стала больше максимального количества тестов iMaxChoice. В этом случае делаем невыбираемой кнопку Button1 («Next Question») – (Button1.Enabled:=False;). Делаем неактивной группу радиокнопой RadioGroup1. Активируем кнопку 2 («Calc Final»). Пишем указание пользователю, чтобы он нажал кнопку Button2 путем изменения property Label2.Caption.
Если тесты не закончились, вызываем процедуру ChangeQuizDisplay (см. выше) с новым номером теста iCurQuest.

```
procedure TForm1.Button2Click(Sender: TObject);
var s1,s2:String;
  r: Real;
begin
  r:=iRightAns / iMaxChoice *100.0;
  Str(iRightAns:1,s1);
  Str(r:2:1,s2);
  Label2.Caption:='Right answer '+s1+' or '+s2+'%';
  Button2.Enabled:=False;
  Button3.Enabled:=True;
  If r>80 Then s1:='A'
    else if r>60 then s1:='B'
      else if r>50 then s1:='C'
      else s1:='D';
  Label1.Caption := 'The final estimate is '+s1;
  case s1 of
    'A' : Label1.Font.Color:=clGreen;
    'B' : Label1.Font.Color:=clBlue;
    'C' : Label1.Font.Color:=clYellow;
    else Label1.Font.Color:=clRed;
  end;
end;
```

Рассмотрим обработчик нажатия Button2. Описаны три переменные – две строки s1 и s2 и вещественное число r, в котором будем подсчитывать процент правильных ответов.
Сначала считаем процент правильных ответов – (r:=iRightAns / iMaxChoice *100.0;). Затем переводим в строковые переменные значения iRightAns и r – в s1 и s2 соответственно (код Str(iRightAns:1,s1); Str(r:2:1,s2);). Пишем в объекте Label2 текст типа «Правильных ответов *** или *** %» (Label2.Caption:='Right answer '+s1+' or '+s2+'%';). Запрещаем Button2 (Button2.Enabled:=False;), разрешаем Button3 (Button3.Enabled:=True;).
Далее ставим оценку – если процент правильных ответов свыше 80 – ставим A, от 60 до 80 – B, от 50 до 60 – C, если меньше 50% – D (несколько вложенных операторов if). Пишем оценку в Label1 (Label1.Caption := 'The final estimate is '+s1;).
Далее играем с цветом Label1 – для оценки A делаем текст зеленым, B – синим, C – желтым и в случае D – красным (оператор case of).

Обработчик Button3 предельно простой.

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;
```

Просто вызывается процедура Close, сигнализируя о необходимости закрыть форму и соответственно программу.

## ПРОГРАММИРОВАНИЕ РЕШЕНИЙ ЛИНГВИСТИЧЕСКИХ ЗАДАЧ СРЕДСТВАМИ ЯЗЫКА-ИНТЕРПРЕТАТОРА PYTHON

Для написания программы можно использовать тот же текстовый редактор, что и при программировании на FreePascal – Geany. Файл необходимо сохранить с расширением **.py** (от Python).
На примере трёх алгоритмов, ранее запрограммированных на FreePascal, изложим общие и специфические отличия языка-интерпретатора Python от языка-компилятора FreePascal в рамках решаемых задач автоматического транскрибирования.

### Программа 1. Автоматическое транскрибирование буквосочетания -ere-

```
sInput=""
sReplace=""
i=0
count_ere=2
tArray=["there","where"]
tString="were"
sInput=input()
# Примечание: В версии Python 2 используется функция raw_input(); в данной (третьей) версии питона используется input()
for i in range(count_ere):
        if sInput==tArray[i]:
                sReplace="-[ɛə˞]-"

if sInput==tString:
        sReplace="-[ɜ˞]-"

if len(sReplace)==0:
        sReplace="-[iə˞]-"

if "ere" in sInput:
        sInput=sInput.replace("ere",sReplace)

print(sInput)
```

### Программа 2. Автоматическое транскрибирование диграфа -ea- с пояснением для двойственного прочтения

```
sInput=""
sReplace=""
sTail=""
i=0
count_ei=3
tArray_ei=["great","break", "steak"]
count_e=20
tArray_e=['heavy','heaven','breast','meadow',
  'weapon','wealth','wealthy','ready',
```

```
  'dead','bread','death','breath','measure','pleasure', 'treasure',
  'head','thread','threat','meant', 'steady']
tString_iei="create"
tString_i_e="read"
sInput=input()
# Примечание: В версии Python 2 используется функция raw_input(); в данной (третьей) версии питона используется input()
for i in range(count_ei):
        if sInput==tArray_ei[i]:
                sReplace="-[ei]-"

for i in range(count_e):
        if sInput==tArray_e[i]:
           sReplace="-[ɛ]-"

if sInput==tString_iei:
        sReplace="-[iei]-"

if sInput==tString_i_e:
   sReplace="-[i]-"
   sTail=', Infinitive and Present Simple, if other  - [ɛ]'

if len(sReplace)==0:
        sReplace="-[i]-"

if "ea" in sInput:
        sInput=sInput.replace("ea",sReplace)+sTail
else:
        sInput='WARNING!!! "ea" NOT FOUND!!!'

print(sInput)
```

**Программа 3. Автоматическое транскрибирование диграфа -th-**

```
sInput=""
sReplace=""
cBefore="?"
cAfter="?"
i=0
count_TH1=12
tArray_TH1= ['the', 'this', 'these', 'that', 'those', 'they', 'them', 'their', 'there', 'then', 'than', 'though']
count_TH2=2
tArray_TH2 = ['Themes', 'Thomas']
vowel= 'aeiouy'
sInput=input()
for i in range(count_TH1):
        if sInput==tArray_TH1[i]:
           sReplace="-[ð]-"

for i in range(count_TH2):
```

```
        if sInput==tArray_TH2[i]:
           sReplace="-[t]-"
           sInput=sInput.replace("Th",sReplace)

if 'th' in sInput:
        i=sInput.find('th')

if 'th' in sInput and (i>1) and (i+2<len(sInput)) and (sReplace == ''):
  cBefore=(sInput[i-1])
  cAfter= (sInput[i+2])


if (cBefore in vowel) and (cAfter in vowel):
    sReplace = ' -[ð]- '

if len(sReplace)==0 and 'th' in sInput:
        sReplace="-[θ]-"
elif not ('th' in sInput.lower()) and len(sReplace)==0:
        sInput='WARNING!!! "th" NOT FOUND!!!'

if len(sReplace)!=0:
        sInput=sInput.replace("th",sReplace)

print(sInput)
```

Во-первых, обращает на себя внимание меньший объём программ на языке Питон. Это связано с 1) гораздо большим количеством библиотек разработанных для данного языка по сравнению с языком Паскаль; 2) отсутствием требований по обязательному предварительному описанию переменных и структур данных. С одной стороны, это значительно упрощает и ускоряет процесс программирования. С другой стороны, может приводить к ошибкам в логике работы программы, проблемам при разработке. Так, при ошибке в написании имени переменной Pascal выдаст ошибку, а Python «молча» создаст ещё одну переменную.

Во-вторых, основные различия в синтаксисе и пунктуации сводятся к следующим: 1) отсутствие в Python операторных скобок (begin/end), вместо которых используются отступы; 2) отсутствие точки с запятой при завершении строки программного кода, условный цикл и условие завершаются двоеточием; 3) для присвоения значения используется знак = без двоеточия; 4) в качестве оператора сравнения, дающего булево значение, используется == (двойное равно), оператор неравенства при этом выражается сочетанием != (восклицательный знак и равно).

Наблюдается разница в интерпретации больших и малых букв при написании зарезервированных слов, названий переменных, процедур и функций. Python требует строгого соответствия, Pascal интерпретирует большие и малые буквы как эквиваленты.

Обратим особое внимание на начальную часть программы 3 – вопросительные знаки в третьей и четвёртой строках. Вместо вопросительных знаков можно поставить любой текстовый символ, кроме гласных букв. Дело в том, что функция вхождения **in** считает, что пустая строка содержится в любой текстовой строке. Поэтому если при выполнении программы этим переменным не присвоится никакого значения (например, если -th- находится в начале или в конце слова), условный оператор вполне легально сработает на случае, когда он срабатывать не должен.