



Кафедра конструирования  
и производства радиоаппаратуры

## Алгоритмы и данные



**Кобрин Юрий Павлович**

**Алгоритмы и данные.** Методические указания к лабораторной работе и по организации самостоятельной работы по дисциплинам «Информатика» и «Информационные технологии» для студентов очного и заочного обучения специальностей 211000.62 (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет). - Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР), кафедра КИПР, 2017. – 32 с.

**Lazarus** — открытая бесплатная кроссплатформенная среда визуальной разработки программного обеспечения для компилятора *Free Pascal*, максимально приближённая к *Delphi*. Методические указания посвящены знакомству с алгоритмами и способами их представления, а также с основными видами элементов языков программирования.

Методические указания предназначены для помощи в подготовке бакалавров и магистрантов в Информатике, выполнения курсовых и дипломных проектов.

©Кафедра КИПР федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Томский государственный университет систем управления и радиоэлектроники (ТУСУР)», 2017.

© Кобрин Ю.П. 2017

## Оглавление

1	Цели и задачи работы .....	4
2	Порядок выполнения работы .....	4
3	Отчётность.....	4
4	Контрольные вопросы .....	4
5	Краткие теоретические сведения .....	5
5.1	Алгоритмы.....	5
5.2	Исторический обзор возникновения языка Free Pascal .....	9
5.3	Основные объекты алгоритмического языка Free Pascal.....	13
5.3.1	Набор символов языка Free Pascal .....	13
5.3.2	Лексемы (слова) языка Free Pascal .....	13
5.3.3	Специальные символы .....	14
5.3.4	Идентификаторы (имена).....	14
5.3.5	Зарезервированные (ключевые) слова.....	16
5.3.6	Комментарии.....	17
5.3.7	Ключи компиляции (директивы компилятора).....	18
5.4	Характеристики типов данных .....	18
5.4.1	Внешнее представление чисел.....	20
5.4.2	Целочисленные типы чисел .....	21
5.4.3	Тип-диапазон (интервальный).....	23
5.4.4	Перечисляемый тип .....	24
5.4.5	Шестнадцатеричные целые числа.....	24
5.4.6	Действительные (вещественные) десятичные числа .....	25
5.4.7	Символьный тип .....	26
5.4.7	Строки символов .....	28
5.4.8	Логический тип .....	32
6	Список литературы.....	33

## 1 Цели и задачи работы

Целью работы является знакомство с основными понятиями языков программирования.

В ходе выполнения работы следует:

- выяснить назначение алгоритма, его характеристики и формы представления;
- изучить назначение основных объектов алгоритмических языков.

## 2 Порядок выполнения работы

Перед выполнением этой работы следует:

- 1) ознакомиться с разделом «Что такое лексемы? Перечислите основные типы лексем.
- 2) Как записывают комментарии в Free Pascal?
- 3) Что такое «типы данных»? Перечислите основные типы данных
- 4) Охарактеризуйте целочисленные типы данных.
- 5) Охарактеризуйте вещественные типы данных.
- 6) Охарактеризуйте символьные типы данных.
- 1) Краткие теоретические сведения» (в качестве дополнительной литературы использовать [1,2,3,4,5,6,7,8]);
- 2) придумать и записать их в отчёт примеры изучаемых типов данных;
- 3) ответить на контрольные вопросы;
- 4) оформить отчёт в соответствии с [9] и защитить его у преподавателя.

## 3 Отчётность

Отчёт должен состоять из следующих разделов:

- 1) Тема и цель работы.
- 2) Ответы на контрольные вопросы.
- 3) Примеры записи рассмотренных типов данных.

При защите отчёта по работе для получения зачёта студент должен:

- уметь отвечать на контрольные вопросы;
- продемонстрировать знание основных объектов алгоритмических языков.

## 4 Контрольные вопросы

- 7) Что такое алгоритм? Перечислите основные свойства алгоритмов.
- 8) Каковы способы записи алгоритмов?
- 9) Охарактеризуйте словесно-пошаговый способ записи алгоритмов.
- 10) Что представляет собой графическая форма записи алгоритма?
- 11) Охарактеризуйте основные блоки блок-схем.
- 12) Охарактеризуйте основные составляющие алгоритмического языка: алфавит, синтаксис и семантика.

- 13) Что такое идентификатор? Перечислите правила формирования идентификаторов.
- 14) Что такое лексемы? Перечислите основные типы лексем.
- 15) Как записывают комментарии в Free Pascal?
- 16) Что такое «типы данных»? Перечислите основные типы данных
- 17) Охарактеризуйте целочисленные типы данных.
- 18) Охарактеризуйте вещественные типы данных.
- 19) Охарактеризуйте символьные типы данных.

## 5 Краткие теоретические сведения

### 5.1 Алгоритмы

Чтобы решить какую-либо задачу на компьютере необходимо сначала придумать *алгоритм* её решения.

**Алгоритм** – это строгая и чёткая конечная система правил, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к достижению поставленной цели.

Под **алгоритмизацией задачи** понимают процесс разработки (проектирования) алгоритма решения задачи с помощью компьютера на основе её условия и требований к ко-

#### Свойства алгоритмов:

- **массовость** - возможность один и тот же алгоритм многократно применять для решения не одной конкретной задачи, а некоторого множества однотипных задач;
- **детерминированность** - применение алгоритма к одним и тем же исходным данным всегда приводит к одному и тому же результату (в недетерминированном алгоритме может быть несколько путей обработки одних и тех же входных данных, - без какого-либо уточнения, какой именно вариант будет выбран);
- **результативность (конечность)** - алгоритм обязательно должен привести к решению поставленной задачи за конечное число шагов, либо к сообщению о том, что при заданных исходных величинах задачу решить невозможно;
- **дискретность (раздельность)** - алгоритм состоит из последовательности отдельных шагов - элементарных действий, причём каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего;
- **определённость** - каждое правило алгоритма должно быть чётким, однозначным, чтобы выполнение алгоритма носило механический характер и не требовало никаких дополнительных указаний или сведений о решаемой задаче;
- **корректность** - для всех исходных данных алгоритм должен всегда давать правильный результат решения определённой задачи и ни при каких исходных данных не будет получен неправильный результат.

нечному результату. На этапе постановки задачи описываются исходные данные и предпосылки, формируются правила начала и окончания решения задачи (достижения цели), т. е. разрабатывается информационная или эквивалентная ей математическая модель.

#### Порядок выполнения алгоритма:

- 1) действия в алгоритме выполняются в порядке их записи;
- 2) никакие два действия алгоритма нельзя менять местами;
- 3) не закончив одного действия нельзя переходить к следующему.

Для строгого задания различных структур данных и алгоритмов их обработки требуется иметь такую систему формальных обозначений и правил (языков описания алгоритмов), чтобы смысл всякого используемого предписания трактовался точно и однозначно.

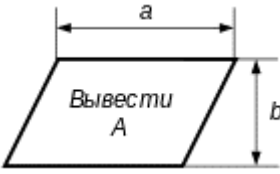



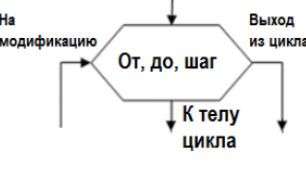
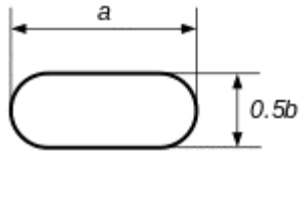
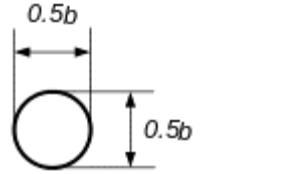

В случае **словесного описания алгоритма** описывается в виде последовательности шагов. На каждом шаге определяется состав выполняемых действий и направление дальнейших вычислений. При этом, если на текущем шаге не указывается какой шаг должен выполняться следующим, то осуществляется переход к следующему шагу. Особых правил составления словесного описания не существует. Достоинства: возможность подробного описания отдельных операций. Недостатки: многословность, отсутствие наглядности, недостаточная точность.

В случае **словесно-формульного описания алгоритма** кроме слов дополняется обычными математическими формулами.

**Блок-схемы** определяют последовательность передачи управления в алгоритмах и выполняются в соответствии с ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». Этот стандарт определяет способы построения схем и внешний вид используемых графических элементов (символов), внутри которых обычно указывают информацию о выполняемом действии (Таблица 5.1). При выполнении элементов рекомендуется придерживаться размеров, определяемых двумя значениями **a** и **b**, причём символы по возможности должны быть одного размера.

Таблица 5.1 - Основные элементы схем алгоритмов

Название символа	Начертание символа	Действие
Процесс		Символ отображает функцию обработки данных любого вида (выполнение определённой операции или группы операций, приводящее к изменению значения, формы или размещения информации). Внутри блока записываются формулы, обозначения и функции, не требующие вызова внешних функций.
Данные		Символ отображает преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Данный символ не определяет носителя данных (подробности ввода/вывода могут быть указаны в комментариях). <b>Ввод</b> подразумевает считывание значений, поступающих с клавиатуры, с диска или из порта ввода-вывода. <b>Вывод</b> означает запись информации на экран, на диск или в порт ввода-вывода.
Предопределённый процесс		Символ отображает предопределённый процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле). Например, вызов процедуры или функции.
Решение		Внутри символа записывают условия выбора направления действия алгоритма. Отображает решение или функцию переключательного типа с одним входом и двумя или более альтернативными выходами, из которых только один может быть выбран после вычисления условий, определённых внутри этого элемента. Вход в элемент обозначается линией, входящей обычно в верхнюю вершину элемента. Если выходов два или три, то обычно каждый выход обозначается линией, выходящей из оставшихся вершин (боковых и нижней). Если выходов больше трёх, то их следует показывать одной линией, выходящей из вершины (чаще нижней) элемента, которая затем разветвляется. Соответствующие результаты вычислений могут записываться рядом с линиями, отображающими эти пути.

<p><b>Цикл</b></p>		<p>Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.</p>
<p><b>Модификация</b></p>		<p>Символ начала цикла с известным числом повторений. Внутри символа записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра цикла для каждого повторения.</p>
<p><b>Терминатор</b></p>		<p>Терминатор обозначает выход во внешнюю среду или вход из внешней среды. Терминатором начинается и заканчивается любая функция или программа. Тип возвращаемого значения и аргументов функции обычно указывается в комментариях к блоку терминатора. Обычные описания терминаторов: начало/конец, запуск/останов, перезапуск (подразумевает перезапуск данной блок-схемы), ошибка (подразумевает завершение алгоритма с ошибкой).</p>
<p><b>Соединитель</b></p>		<p>Символ может использоваться, если по каким-либо причинам тянуть линию не удобно. Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения её в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.</p>
<p><b>Комментарий</b></p>		<p>Используется, если длина текста, помещаемого внутри некоего символа, превышает размер самого этого символа, а также для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Комментарии также применяют совместно с терминаторами для описания входных аргументов алгоритма при описании функций. Пунктирная линия используется для соединения символа с комментарием.</p>

Сплошные линии указывают потоки данных между процессами и/или носителями данных, а также потоки управления между процессами и могут снабжаться стрелкой. Стрелку можно не указывать при направлении дуги слева направо и сверху вниз. Линии должны подходить к символу слева, либо сверху, а исходить либо снизу, либо справа.



Блок-схемы дают большую наглядность и лучшую обзримость при отображении *общего хода решения задачи*. На низком уровне их использование нецелесообразно из-за большой трудоёмкости, громоздкости и излишне большой детализации.

**Программный способ** записи алгоритма – это запись алгоритма на языке программирования, позволяющем на основе строго определённых правил формировать последовательность инструкций, однозначно отражающих смысл и содержание алгоритма, с целью его последующего исполнения на компьютере. В итоге применения такого способа появляется **программа**.

## 5.2 Исторический обзор возникновения языка Free Pascal

В 1970 г. швейцарский учёный Никлаус Вирт (Nicklaus E. Wirth) (Рис. 5.1) специально для обучения студентов разработал новый язык программирования<sup>1</sup>, базирующийся на идеях уже существовавшего языка Алгол 60, и назвал этот язык в честь известного средневекового французского математика и инженера Блеза Паскаля — создателя одного из первых механических калькуляторов.

Язык Паскаль [1,3] выгодно отличался от других языков программирования более жёсткими правилами в описании и использовании данных различного типа. Чтобы избежать ошибок неверной обработки данных разных типов данных, в Паскале предусмотрено обязательное предварительное объявление конкретных типов данных всех используемых в программе переменных. Паскаль содержит набор операторов<sup>2</sup> описывающих структурированные данные, а также, позволяющие выполнять над ними различные операции. Структурированная программа на Паскале имеет иерархическую структуру, строится без использования операторов безусловного перехода из трёх базовых управляющих структур: *последовательность, ветвление, цикл*. Паскаль также называют **процедурным языком**, так как в программе могут использоваться *подпрограммы*<sup>3</sup> (процедуры и функции), позволяющие разбить большую монолитную программу на отдельные смысловые части в соответствии с модульным принципом программирования.

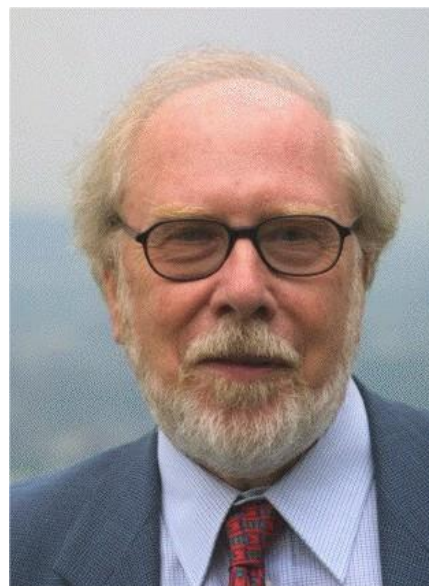


Рис. 5.1 - Никлаус Вирт — специалист в области информатики

<sup>1</sup> **Язык программирования** – это искусственный (формальный) язык, предназначенный для записи компьютерных программ, представляющих собой набор правил, позволяющих компьютеру выполнить тот или иной алгоритм, организовать управление различными объектами, и т. п.

<sup>2</sup> **Оператор (инструкция, директива)** (от англ. *Statement* - предложение) – это единое и неделимое предложение, выполняющее **в программе** какие-либо определённые алгоритмические действия.

<sup>3</sup> **Подпрограмма** - набор команд, который имеет имя и может быть неоднократно вызван из любого места программы по его имени с новым набором исходных данных.

Язык Паскаль был одним из первых языков программирования **высокого уровня**, т.е. языков наиболее приближенных к человеческому. Особенно большую популярность язык Паскаль (англ. *Pascal*) приобрёл в середине 80-х годов, когда американской фирмой *Borland International, Inc* была разработана система программирования *Turbo Pascal* [1,10] (более поздняя расширенная её версия имеет название *Borland Pascal*) для операционной системы *Microsoft DOS*. Интуитивно понятная интерактивная **интегрированная среда разработки** (англ. *Integrated development environment, IDE*) фирмы *Borland* объединяет:

- **Текстовый редактор** для создания и редактирования исходного текста программы на Паскале;
- **транслятор-компилятор**<sup>4</sup>, преобразующий исходные тексты основной программы и подпрограмм в отдельные промежуточные двоичные (бинарные) объектные файлы<sup>5</sup>, осуществляющий также определение местоположения синтаксических<sup>6</sup> ошибок во время компиляции в текстовом редакторе;
- **компоновщик** (англ. *link editor* - редактор связей), объединяющий объектные файлы подпрограмм для получения единого готового исполнимого модуля (файла с расширением *\*.exe*), либо библиотек с подпрограммами
- **отладчик** (англ. *debugger*), облегчающий тестирование, анализ и исправление семантических<sup>7</sup> ошибок в режиме интерактивного пошагового выполнения программы, во время которого производится наблюдение за изменением значений важных переменных или выражений;

---

<sup>4</sup> **Компилятор** (от англ. *compiler* - составитель, собиратель) читает всю программу целиком, делает ее перевод и создаёт законченный вариант программы на машинном языке, который затем и выполняется. Компиляция означает перевод исходной программы с алгоритмического языка высокого уровня в объектную программу - на язык компьютера.

<sup>5</sup> **Объектный файл** (англ. *object file*) — файл с промежуточным представлением отдельного модуля программы, полученный в результате обработки исходного кода компилятором.

<sup>6</sup> **Синтаксические ошибки** означают, что записываются операторы, которые не согласуются с правилами алгоритмического языка. Например, если забывают объявить переменную, пытаются присвоить целочисленной переменной значение действительного числа, передают неправильное количество параметров подпрограммы и т.п. После исправления синтаксической ошибки пользователь может начать процесс компиляции снова.

<sup>7</sup> **Семантические ошибки** - ошибки времени выполнения программы, связанные с неправильным содержанием действий и использованием недопустимых значений величин (нарушаются смысл и правила использования конструкций языка). Проявление ошибки в процессе ввода исходных данных или в процессе счёта обычно приводит к прерыванию счёта и выдаче диагностического сообщения рабочей программы. Семантические ошибки обнаруживаются, когда компилируется синтаксически правильная программа, которая делает то, что ей указали, но не так, как было задумано и что хотелось бы. Например, производит деление на 0, выполняет ошибочные вычисления, открывает несуществующий файл для ввода, изображения на экране выглядят неправильно и т.п.

– **справочную систему**, позволяющую изучать язык без обращения к внешним источникам.

Среди огромного числа реализаций языка Паскаль следует отметить *Object Pascal* от фирмы *Borland* - версию языка Паскаль, расширенную средствами объектно-ориентированного программирования, которая легла в основу *Delphi* (1994 г.), среды программирования для операционной системы *Microsoft Windows*. *Delphi* - среда быстрой разработки приложений (англ. **Rapid Application Development, RAD**), визуального создания программ, с помощью которой быстро и без особых усилий можно создать полноценно выглядящую программу. Возможности объектно-ориентированного подхода дали пользователям *Delphi* большое преимущество в скорости и качестве создания программ.

Появившаяся (2000 г.) версия компилятора **Free Pascal** (англ. *Free Pascal Compiler, FPC*) [4,6,7] языков *Pascal* и *Object Pascal*, лежит в основе кроссплатформенной<sup>8</sup> графической среды быстрой разработки программного обеспечения *Lazarus* [5,11,12], являющейся аналогом среды *Delphi*.

**Lazarus** - это **среда визуального программирования** для создания программ с графическим интерфейсом с открытым исходным кодом<sup>9</sup>. Интегрированная Среда Разработки программ *Lazarus* (IDE) (сокращение от англ. *Integrated Development Environment*) использует компилятор **FPC** (*Free Pascal Compiler*), редакторы кода, форм, Инспектор Объектов, отладчик и многие другие инструменты.

Технология визуального программирования *Lazarus* позволяет строить интерфейс будущей программы из специальных компонентов, реализующих нужные свойства. Количество таких компонентов достаточно велико. Каждый из компонентов содержит готовый программный код и все необходимые для работы данные, и это позволяет программисту максимально использовать то, что уже создано ранее.

Для создания графического интерфейса пользователя *Graphical User Interface* (GUI) обычно применяются методы **объектно-ориентированного программирования**.

**Объектно-ориентированное программирование (ООП)** — это метод программирования, основанный на использовании объектов.

**Объект** - совокупность свойств (структур данных, характерных для данного объекта), методов их обработки (подпрограмм изменения их свойств) и событий, на которые данный объект может реагировать и, которые приводят, как правило, к изменению свойств объекта. Для каждого объекта существует определённый набор действий (методов)<sup>10</sup>, которые с ним можно произвести.

---

<sup>8</sup> **Кроссплатформенная среда** – это программное обеспечение, работающее более чем на одной аппаратной платформе и/или операционной системе. Например компилятор *Free Pascal* может работать под операционными системами *Windows*, *Linux*, *Mac OS X*, *FreeBSD* и т.д.

<sup>9</sup> **Открытый исходный код программ** позволяет пользователю принять участие в разработке открытого (нелицензионного) программного обеспечения (от англ. *open-source software*), исправлять имеющиеся в нём ошибки, создавать на его основе новое программное обеспечение.

<sup>10</sup> В терминологии ООП понятия «метод», «сообщение» и «действие» синонимы. Поэтому выражения «вызвать метод объекта», «послать сообщение объекту для выполнения

Объединение данных и свойственных им процедур обработки (методов) в одном объекте, называется **инкапсуляцией** и является одним из важнейших принципов ООП. Инкапсуляция позволяет скрыть реализации класса и отделить его внутреннее представление от внешнего (интерфейса). При использовании ООП применять прямой доступ к свойствам какого-либо класса из методов других классов не принято. Для доступа к свойствам класса принято задействовать специальные методы этого класса для получения и изменения его свойств.

Ещё одним фундаментальным понятием ООП является *класс*. **Класс** - это шаблон, на основе которого может быть создан конкретный программный объект. Каждый конкретный объект, имеющий структуру этого класса, называется *экземпляром класса*. Все экземпляры одного класса (объекты, порождённые от одного класса) имеют один и тот же набор свойств и общее поведение, то есть одинаково реагируют на одинаковые сообщения.

Важнейшими принципами ООП также являются *наследование* и *полиморфизм*.

**Наследование** - предусматривает создание новых классов на базе существующих и позволяет классу-потомку иметь (наследовать) все свойства класса-родителя и добавлять к нему новые черты, характерные только для него.

**Полиморфизм** - означает, что рождённые объекты обладают информацией о том, какие методы они должны использовать в зависимости от того, в каком месте цепочки они находятся.

**Модульность** - свойство программ, при котором объекты заключают в себе полное определение их характеристик. Никакие определения методов и свойств не должны располагаться вне объекта, что делает возможным свободное копирование и внедрение одного объекта в другие.

Использование ООП позволяет:

- уменьшить сложность программного обеспечения и повысить его надёжность;
- переделывать отдельные компоненты программного обеспечения без изменения остальных его компонентов;
- повторно использовать отдельные компоненты программного обеспечения.

Работу над программой в среде визуального программирования условно можно разбить на две части:

- 1) создание интерфейса (внешнего вида) будущей программы в диалоговом режиме;
- 2) написание программного кода.

В отличие от дорогостоящей среды *Delphi*, *Free Pascal* и среда *Lazarus* относятся к категории **свободного (бесплатного)** программного обеспечения, не требующего лицензии. *Lazarus* и *Free Pascal* доступны в виде исходных текстов на сайтах [www.freepascal.ru](http://www.freepascal.ru) и [www.lazarus.freepascal.org](http://www.lazarus.freepascal.org).

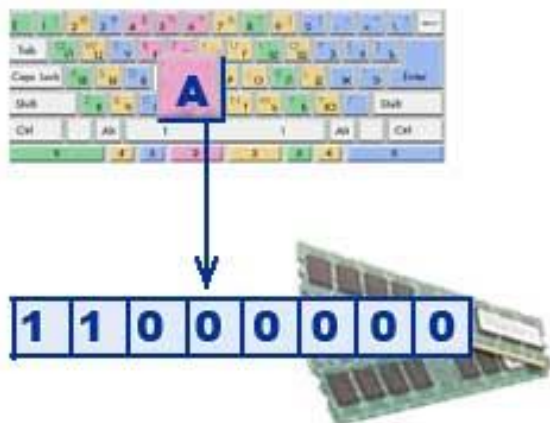
Существуют и кроссплатформенные версии *Free Pascal*, позволяющие на компьютере одного типа создавать приложения, предназначенные для работы на компьютерах другого типа.

---

какого-либо действия» и «выполнить действие над объектом» равносильны.

## 5.3 Основные объекты алгоритмического языка Free Pascal

### 5.3.1 Набор символов языка Free Pascal



**Символы языка** - это основные неделимые знаки, в терминах которых пишутся все тексты на языке.

**Кодировка символов** (иначе также кодовая страница) – это набор числовых значений, которые ставятся в соответствие группе алфавитно-цифровых символов, знаков пунктуации и специальных символов.

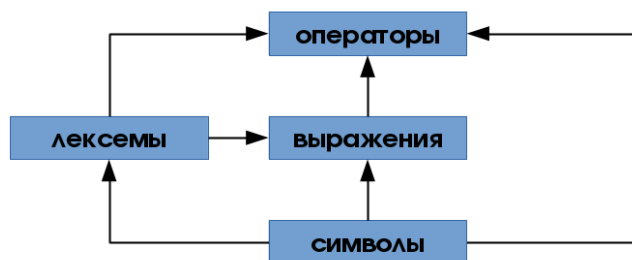
Любой символ, который мы вводим с клавиатуры, либо видим на экране монитора, закодирован определённой последовательностью битов (нулей и единиц).

Программа на языке Free Pascal формируется из *символов*:

прописных букв латинского алфавита	<b>A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</b>
строчных букв латинского алфавита	<b>a b c d e f g h i j k l m n o p q r s t u v w x y z</b>
символа подчёркивания	<b>_</b>
арабских десятичных цифр	<b>0 1 2 3 4 5 6 7 8 9</b>
специальных символов	<b>#, \$, %, ', *, /, +, -, ., :, ;, &lt;, =, &gt;, @, ^, (, ), [, ], {, }</b>
символ пробела	на экране не отображается
управляющие символы Free Pascal	на экране не отображаются

Остальные символы могут использоваться в символьных и строковых константах, а также в комментариях, что позволяет организовать диалог *пользователь-компьютер на русском языке*. Русские буквы в программе обязательно должны заключаться в апострофы, например **'Текст на русском языке в Free Pascal заключается в апострофы'**

### 5.3.2 Лексемы (слова) языка Free Pascal



**Лексемы** – минимальные значимые единицы текста программы.

К лексемам языка *Free Pascal* относят специальные символы, идентификаторы, служебные слова, комментарии, ключи компиляции.

В качестве **разделителей лексем** применяют *пробел, символ табуляции и составной символ перехода в начало следующей строки* (пара символов «Возврат каретки» и «перевод строки»).

Рассмотрим основные лексемы языка *Free Pascal*.

## 5.3.3 Специальные символы

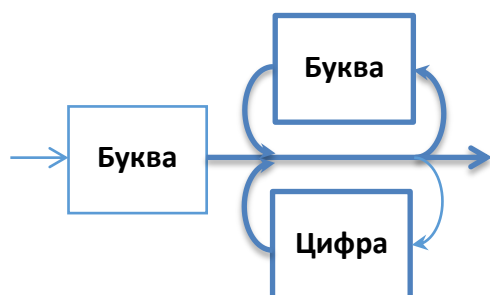
Составные символы (нельзя разделять пробелами)	<code>:=</code>	присвоить
	<code>..</code>	диапазон
	<code>&lt;=</code>	меньше или равно
	<code>&gt;=</code>	больше или равно
	<code>&lt;&gt;</code>	не равно
	<code>//{ } (* *) ( . .) [ . .]</code>	символы начала и конца комментариев

## 5.3.4 Идентификаторы (имена)

Чтобы программа решения задачи обладала свойством массовости, следует вместо конкретных значений величин использовать их обозначения, чтобы иметь возможность изменять их значения по ходу выполнения программы. Для обозначения переменных и других элементов программы используют **идентификаторы**.

**Идентификатор** (англ. *data name, identifier, ID* - опознаватель) – уникальный признак (имя) объекта, позволяющий отличать его от других объектов.

**Идентификаторы** представляют собой последовательности латинских букв, цифр и знаков подчёркивания, используемые для обращения к константам, типам данных, переменным, процедурам, функциям, модулям, меткам, полям в записях, программам, объектам, классам.



В идентификаторах не должно быть точек, запятых, самих скобок, а также пробелов, знаков операций и других специальных символов. Пробелы в идентификаторах обычно заменяют символом подчёркивания.

Идентификаторы могут иметь длину до 127 символов.

**Первый символ идентификатора – обязательно не цифра!**

## Правильные идентификаторы:

- *Summa*,
- *date\_11\_okt\_1947*,
- *Alpha*,
- *\_1\_2\_3\_*,
- *MyConst*,
- *MyProgram*



### Неправильные идентификаторы:

- **Сумма** – ошибка, идентификатор содержит буквы русского алфавита
- **MyConst#** - содержит недопустимый символ #
- **My-variable** - содержит недопустимый символ
- **1\_2\_3** – начинается с цифры
- **Lab 3** – пробел между частями идентификатора

**В языке Free Pascal не делается различие между символами на верхнем и нижнем регистрах (прописными и строчными)!**

Например, следующие идентификаторы будут обозначать одно и то же:

***Summa summa SUMMA sUMMA SuMma***

<p><b>Стандартные идентификаторы</b></p>	<p>Служат для обозначения заранее определённых разработчиками языка типов данных, констант, процедур и функций, например: <i>integer, Sin, Cos, Ln, Sqr, Sqrt, Read, Readln, Write, Writeln</i> и др.</p> <p>В этом примере стандартный идентификатор <i>Sin</i> вызывает функцию, вычисляющую синус заданного угла. <i>Read, Readln</i> вызывают процедуру, организующую ввод данных, <i>Write, Writeln</i> вызывают процедуру, организующую вывод данных.</p> <p>Любой из стандартных идентификаторов, в отличие от зарезервированных слов, можно переопределить, но это чаще всего приводит к ошибкам.</p> <p><b>Не переопределяйте стандартные идентификаторы,</b> чтобы не запутать себя и компьютер!</p>
<p><b>Пользовательские идентификаторы</b></p>	<p>Применяются для обозначения меток, констант, переменных, процедур и функций, определённых самим программистом. При этом идентификаторы в программе должны быть уникальными, т. е. в каждом блоке программы один идентификатор не может использоваться для обозначения более чем одной переменной или постоянной величины, и т. д.</p>

Иногда в разных модулях (например, **Unit1, Unit2, Unit3**) описаны *одинаковые* имена объектов (например, **Error**).

Для уточнения, из какого именно модуля используется объект, используются **квалифицируемые (уточнённые) идентификаторы**, в которых перед именем переменной ставится модуль, содержащего этот идентификатор. При этом оба идентификатора разделяются точкой:

***Unit1.Error Unit2.Error Unit3.Error***

Уточняются могут: идентификаторы **меток, констант, переменных, типов данных, а также имена процедур и функций**.

Заметим, что обращение к полям записей и свойств объектов, методам (подпрограммам) объектов также выполняются с помощью уточнённого (составного) имени.

### 5.3.5 Зарезервированные (ключевые) слова

**Зарезервированные слова** имеют строго определённое назначение и используются только так, как они определены при описании языка. Они не могут быть переопределены или использованы в качестве идентификаторов:



**Зарезервированные (ключевые) слова (Таблица 4.2)** – это зарезервированные слова, имеющие специальное значение, являющиеся составной частью языка программирования

- Их можно использовать только в том смысле, в котором они определены.
- Имена, создаваемые программистом, не должны совпадать с ключевыми словами.
- Использование идентификатора, который совпадает с ключевым словом, приведет к ошибке компиляции.

Таблица 5.2 - Зарезервированные слова *Free Pascal*

Слово	Смысл слова	Слово	Смысл слова
<i>as</i>	как	<i>mod</i>	остаток от деления
<i>absolute</i>	абсолютный	<i>new</i>	новый
<i>alias</i>	псевдоним	<i>name</i>	имя
<i>and</i>	логическое «и»	<i>near</i>	вблизи
<i>array</i>	массив	<i>nil</i>	отсутствие
<i>asm</i>	ассемблер	<i>not</i>	логическое "не"
<i>begin</i>	начало блока	<i>nostackframe</i>	не кадр стека
<i>case</i>	вариант	<i>object</i>	объект
<i>const</i>	константа	<i>of</i>	из
<i>constructor</i>	конструктор	<i>on</i>	на
<i>class</i>	класс	<i>operator</i>	оператор
<i>default</i>	по умолчанию	<i>or</i>	логическое «или»
<i>destructor</i>	деструктор	<i>out</i>	вне
<i>dispinterface</i>	диспетчерский интерфейс	<i>packed</i>	упакованный
<i>div</i>	деление нацело	<i>property</i>	свойство
<i>dispose</i>	удалять	<i>private</i>	частные
<i>do</i>	выполнять	<i>procedure</i>	процедура
<i>downto</i>	уменьшить до	<i>program</i>	программа
<i>except</i>	за исключением	<i>protected</i>	защита
<i>export</i>	экспорт	<i>raise</i>	поднять
<i>end</i>	конец блока	<i>record</i>	запись



Слово	Смысл слова	Слово	Смысл слова
<i>external</i>	внешний	<i>reintroduce</i>	вновь
<i>else</i>	иначе	<i>resourcestring</i>	строка ресурса
<i>exit</i>	выход	<i>repeat</i>	повторять
<i>file</i>	файл	<i>self</i>	себя, сам, само
<i>finally</i>	и наконец	<i>set</i>	множество
<i>finalization</i>	завершение работы	<i>shl</i>	сдвиг битов влево
<i>for</i>	для	<i>shr</i>	сдвиг битов вправо
<i>function</i>	функция	<i>string</i>	строка
<i>forward</i>	опережающий	<i>then</i>	то
<i>goto</i>	переход на	<i>to</i>	до
<i>if</i>	если	<i>type</i>	тип
<i>implementation</i>	реализация	<i>threadvar</i>	переменная потока
<i>in</i>	в (входит в)	<i>try</i>	проба
<i>initialization</i>	инициализация	<i>unit</i>	модуль
<i>Inherited</i>	унаследованный	<i>until</i>	до
<i>inline</i>	основной	<i>uses</i>	использовать
<i>interface</i>	интерфейс	<i>var</i>	переменная
<i>interrupt</i>	прерывание	<i>virtual</i>	виртуальный
<i>is</i>	это	<i>while</i>	пока
<i>label</i>	метка	<i>with</i>	с
<i>library</i>	библиотека	<i>xor</i>	исключающее "или"

### 5.3.6 Комментарии

**Комментарии** в программе выполняют чисто информационную функцию и служат для пояснений к программе, описания работы алгоритма программы, отдельных подпрограмм, назначения типов, переменных и т.п. Их можно записывать в любом месте программы, где разрешён пробел.

**Комментарии компилятор игнорирует.**

Во время отладки часто возникает необходимость временно убрать из программы некоторую группу операторов, сохранив, тем не менее, их запись. Закрывая эту группу операторов в символы `{ }` можно временно исключить эту группу из программы, поскольку она станет текстом комментария и будет полностью игнорироваться компилятором.

**Однорочные комментарии** начинаются с двух символов «косая черта» `//` и заканчиваются символом перехода на новую строку

- `//` каждая строка должна начинаться
- `//` с двух символов «косая черта»

**Многострочные комментарии** заключают либо в фигурные скобки { }, либо между парами составных символов (\* \*).

- {Комментарием может быть любая последовательность символов, ограниченная с двух сторон фигурными скобками, независимо от того, сколько строк она будет занимать!}
- (\* Последовательность символов в круглых скобках со звёздочками также является комментарием \*)

### 5.3.7 Ключи компиляции (директивы компилятора)

Если непосредственно после открывающего символа комментария { следует знак \$, то комментарий воспринимается как *директива компилятору* - **ключ компиляции**, специальная инструкция, управляющая процессом компиляции. Ключ компиляции с последующим после \$ знаком «+» включает некоторый режим компиляции, а со знаком «-» - выключает. С помощью таких директив можно включать или выключать контроль ошибок, совершать или нет проверку диапазонов массивов, изменять распределение памяти и т.д.

Ключи компиляции вставляются прямо в исходный код программы, например:

**{\$I+}** - включает обработку ошибок ввода-вывода;

**{\$R-}** - отключает проверку диапазонов индексов массивов;

**{\$I My\_file.pas}** - включает в программу внешний текстовый файл *My\_file.pas* с фрагментом программы на Паскале.

## 5.4 Характеристики типов данных

Для решения задач в любой программе выполняется обработка каких-либо данных.

**Данные** (англ. *data*) - это объекты любой формы, выступающие в качестве средства представления информации. Другими словами, данные – это информация, зафиксированная в определённой форме: константы, переменные и структуры, содержащие числа (целые и вещественные), текст (символы и строки) или адреса (переменных и структур).

Данные хранятся в памяти компьютера и могут быть самых различных типов: целые и вещественные числа, символы, строки, массивы.

**Типы данных** определяют способ хранения чисел, символов или других видов информации в памяти компьютера. Они задают размеры ячейки, в которую будет записано то или иное значение, определяя тем самым его максимальную величину или точность задания.

Для того, чтобы использовать компьютер для обработки данных, необходимо располагать некоторым способом представления данных. Способ представления данных будет зависеть от того, для кого эти данные предназначены: для человека (**внешнее представление**) или для компьютера (**внутреннее представление**).

Компьютерная программа выполняет обработку данных определённого типа адекватными им методами и создаёт новый продукт – **информацию**.

Типы данных (англ. *data type*) характеризуются:

- способом их внутреннего (машинного) представления (кодирования)
- множеством допустимых значений, которые могут принимать данные, принадлежащие к этому типу
- набором операций, которые могут выполняться над величинами, принадлежащими к данному типу;
- размерами ячейки памяти, в которую можно поместить значение, соответствующее этому типу данных

Хранящиеся в памяти компьютера данные могут быть самых различных типов: целые и действительные (вещественные) числа, символы и строки текста, массивы, мультимедийные (графические объекты, фотографии и чертежи, звуковые и видеосигналы) и т.п. Но в любом случае предварительно эти данные соответствующим для этого типа данных стандартным образом должны быть преобразованы (кодированы) из внешнего представления во внутреннее машинное двоичное представление, занимающее ячейку памяти из  $n$ -разрядов (или байт) (Рис. 5.2 - Ячейка памяти).



Рис. 5.2 - Ячейка памяти

Компьютер обрабатывает двоичные данные без учёта их смыслового содержания, используя лишь набор своих машинных команд, позволяющий работать с этим типом данных. Чтобы представить результаты на устройствах ввода-вывода в привычной для человека форме – их нужно адекватно преобразовать из внутреннего во внешнее представление. Оценить смысловое содержание данных пока может только человек.

Язык Free Pascal является строго-типизированным языком программирования. Это означает, что для всех переменных, обрабатываемых программой, заранее должен быть установлен их тип (Рис. 5.3).

Кроме **стандартных типов данных** (определённых в самом языке), в Free Pascal предусмотрен механизм создания **пользовательских типов данных**, заданных самим программистом. Благодаря этому общее количество типов, используемых в программе, может быть сколь угодно большим.

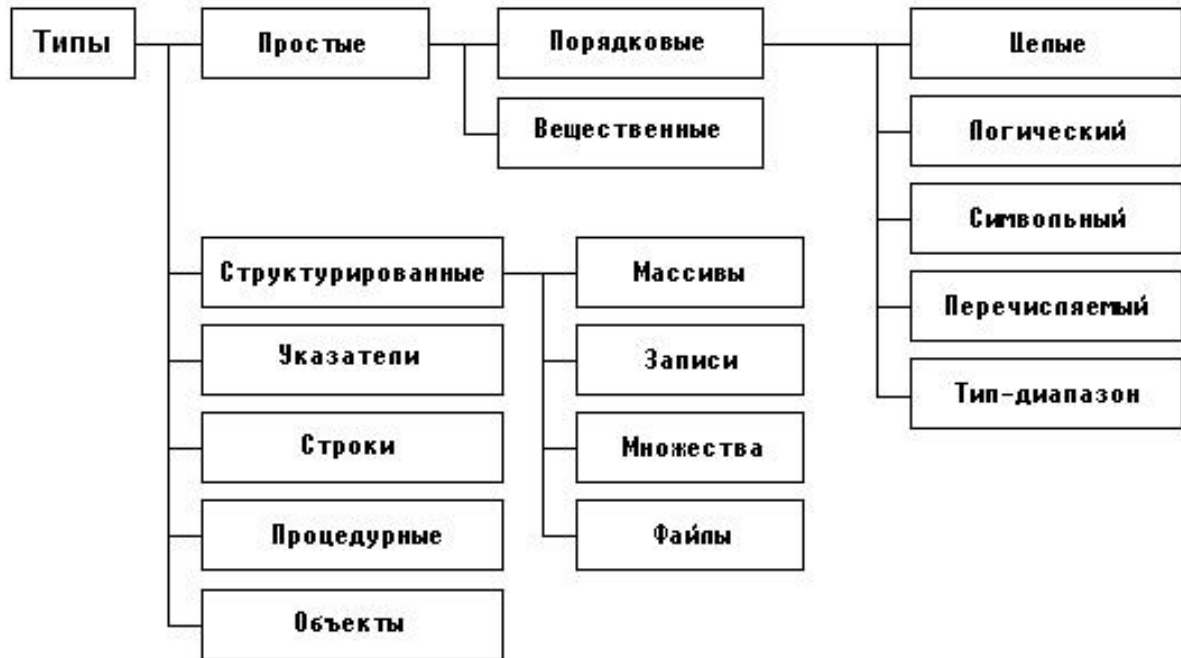


Рис. 5.3 – Стандартные типы данных в *Free Pascal*

Пользовательские типы данных обязательно следует описать в разделе описаний перед блоком, в котором они используются с помощью ключевого (зарезервированного) слова **type**:

```

type
  ИмяТипа = НовыйТип;

```

Здесь **ИмяТипа** - это уникальный идентификатор Вашего типа данных, не совпадающий с идентификаторами стандартных predefined типов и с зарезервированными словами. **НовыйТип** - это либо predefined стандартный тип Free Pascal, либо Ваш собственный тип данных.

Рассмотрим базовые типы данных, используемые в программировании.

#### 5.4.1 Внешнее представление чисел

Внешнее представление чисел - способ их записи в текстах программ, при наборе чисел, вводимых в компьютер по запросу программы, при отображении результатов на экране дисплея или на принтере.



В Free Pascal используются числа:

- **целые десятичные,**
- **вещественные десятичные,**
- **целые шестнадцатеричные.**



Таблица 5.3 представляет диапазоны возможных значений стандартных целочисленных типов данных в Free Pascal.

Таблица 5.3 - Стандартные целочисленные типы данных в Free Pascal

Тип	Диапазон возможных значений	Формат
<i>Byte</i>	0 .. 255	1 байт без знака
<i>Shortint</i>	-128 .. 127	1 байт со знаком
<i>Smallint</i>	-35768 .. 32767	2 байта со знаком
<i>Word</i>	0 .. 65535	2 байта без знака
<i>Integer</i>	Зависит от режима компиляции	2 или 4 байта со знаком
<i>Cardinal</i>	0 .. 4294967295	4 байта без знака
<i>Longint</i>	-2147483648 .. 2147483647	4 байта со знаком
<i>Longword</i>	0 .. 4294967295	4 байта без знака
<i>Int64</i>	-9223372036854775808 .. 9223372036854775807	8 байт со знаком
<i>QWord</i>	0 .. 18446744073709551615	8 байт без знака

Максимальное значение числа зависит от количества байт, которые выделены для него. При попытке записать в ячейку значение больше максимального возникает **переполнение**. В этом случае в ячейку будет записано искажённое значение (меньше того значения, которое Вы пытались присвоить, так как старшие разряды числа, которые не вмещающиеся будут попросту отброшены).

#### 5.4.3 Тип-диапазон (интервальный)

**Тип-диапазон** – это *подмножество* своего базового типа, в качестве которого может выступать любой порядковый тип, кроме типа-диапазона.

Тип-диапазон задаётся границами своих значений внутри базового типа:

**<МИН.ЗНАЧ.> .. <МАКС.ЗНАЧ.>**

Например:

```

type
  digit = '0' .. '9';    {символы цифр могут быть только от '0' до '9'}
  MonthDay = 1 .. 31;  {дни месяца могут быть только в диапазоне от 1 до 31}

```

Чаще всего тип-диапазон употребляется для указания допустимых границ изменения индексов у элементов массивов.

#### 5.4.4 Перечисляемый тип

**Перечисляемый тип** состоит из тех значений, которые он может получать

Этот тип, определяемый пользователем, делает программы нагляднее. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамлённом

*type*

*tColors = (red, green, blue);*

*tMonth = (jan, feb, mar, may, jun, jul, aug, sep, oct, nov, dec);*

круглыми скобками, например:

Если теперь мы создадим переменную типа *tColors*, то присваивать ей можно только значения *red*, *green* или *blue*.

Соответствие между значениями перечисляемого типа и порядковыми номерами этих значений устанавливается порядком перечисления: первое значение в списке получает порядковый номер 0, второе - 1 и т.д.

#### 5.4.5 Шестнадцатеричные целые числа

Шестнадцатеричные числа удобно использовать для краткой и безошибочной записи двоичного кода – каждая четвёрка (тетрада) двоичных цифр заменяется одной шестнадцатеричной. В один байт могут быть записаны две шестнадцатеричные цифры.

1	2	3	C
4	5	6	D
7	8	9	E
A	0	B	F

Перед *шестнадцатеричными целыми числами* ставится *символ \$*.

Знак шестнадцатеричного числа (плюс или минус) определяется самой формой записи и *зависит от значения старшего разряда* (бита) двоичного представления числа.

*\$0*  
*\$4E*  
*\$FFFFFF*  
*\$A10*  
*\$CBADF9*

Перед шестнадцатеричными целыми числами ставится символ *\$*. Знак шестнадцатеричного числа (плюс или минус) определяется самой формой записи и *зависит от значения старшего разряда* (бита) двоичного представления числа.

*\$0 \$4E \$FFFFFF \$A10 \$CBADF9*



## 5.4.6 Действительные (вещественные) десятичные числа

**Действительными** или **вещественными** числами называются все положительные и отрицательные числа (в том числе и дробные), а также нуль.

**Действительные (вещественные) десятичные числа** записываются либо в виде обычной десятичной дроби, либо в показательной (инженерной) форме с основанием 10.

Вместо основания 10 ставится строчная *e* или прописная *E* буква, за которой непосредственно указывается показатель степени.

**Разделительный символ – обязательно точка!**

6.28  
-234.567  
+33.5  
-1E6  
2.83e14  
-1E+27  
0.4569e-10



Вещественные типы в *Free Pascal* различаются диапазоном и точностью связанных с ними значений (Таблица 5.4).

Таблица 5.4 - Диапазоны представления и точность представления (числа значащих десятичных цифр) для вещественных типов данных

Тип	Название	Диапазон возможных значений	Число значащих цифр	Размер в байтах
<i>Real</i>	вещественное		от 15 до 16	8
<i>Real48</i>	вещественное	$2.9 \cdot 10^{-39} .. 1.7 \cdot 10^{38}$	от 11 до 12	6
<i>Single</i>	с одинарной точностью	$1.5 \cdot 10^{-45} .. 3.4 \cdot 10^{38}$	От 7 до 8	4
<i>Double</i>	с двойной точностью	$5.0 \cdot 10^{-324} .. 1.7 \cdot 10^{308}$	от 15 до 16	8
<i>Extended</i>	с повышенной точностью	$1.9 \cdot 10^{-4951} .. 1.1 \cdot 10^{4932}$	от 19 до 20	10
<i>Comp</i>	сложный тип (целочисленные значения)	$-2^{63} + 1 .. 2^{63} - 1$	от 19 до 20	8
<i>Currency</i>	Число с фиксированной точкой	-922337203685477.5808 .. 922337203685477.5807	от 19 до 20	8



Таблица 5.5 - Кодировка символов в соответствии со стандартом **ANSI**

Код	Символ	Код	Символ	Код	Символ	Код	Символ
0	<i>NUL</i>	32	<i>BL</i>	64	®	96	'
1	<i>SOH</i>	33	!	65	A	97	a
2	<i>STX</i>	34	"	66	B	98	b
3	<i>ETX</i>	35	#	67	C	99	c
4	<i>EOT</i>	36	\$	68	D	100	d
5	<i>ENQ</i>	37	%	69	E	101	e
6	<i>ACK</i>	38	&	70	F	102	f
7	<i>BEL</i>	39	'	71	G	103	g
8	<i>BS</i>	40	(	72	H	104	h
9	<i>HT</i>	41	)	73	I	105	i
10	<i>LF</i>	42	*	74	J	106	j
11	<i>VT</i>	43	+	75	k	107	k
12	<i>FF</i>	44	,	76	L	108	l
13	<i>CR</i>	45	-	77	M	109	m
14	<i>SO</i>	46	.	78	N	110	n
15	<i>SI</i>	47	/	79	O	111	o
16	<i>DEL</i>	48	0	80	p	112	p
17	<i>DC1</i>	49	1	81	Q	113	q
18	<i>DC2</i>	50	2	82	R	114	r
19	<i>DC3</i>	51	3	83	S	115	s
20	<i>DC4</i>	52	4	84	T	116	t
21	<i>NAK</i>	53	5	85	U	117	u
22	<i>SYN</i>	54	6	86	V	118	v
23	<i>ETB</i>	55	7	87	w	119	w
24	<i>CAN</i>	56	8	88	X	120	x
25	<i>EM</i>	57	9	89	y	121	y
26	<i>SUB</i>	58	:	90	z	122	z
27	<i>ESC</i>	59	/	91	[	123	{
28	<i>FS</i>	60	<	92	\	124	
29	<i>GS</i>	61	=	93	]	125	}
30	<i>RS</i>	62	>	94	^	126	~
31	<i>US</i>	63	?	95	—	127	~

В Windows популярна кодовая таблица стандарта *Unicode*, в которой для представления каждого символа используют два байта. В 65536 позициях новой таблицы умещаются все современные алфавиты (в том числе буквы греческий, латинский и кириллица), математические символы, специальные знаки (вроде интегралов и нот), даже китайские иероглифы. *Free Pascal* поддерживает и кодировку *Unicode*.

Стандарт состоит из двух основных разделов: универсальный набор символов (**UCS**, *Universal Character Set*) и семейство кодировок (**UTF**, *Unicode Transformation Format*). Универсальный набор символов задаёт однозначное соответствие символов кодам — элементам кодового пространства, представляющим неотрицательные целые числа. Семейство кодировок определяет машинное представление последовательности кодов **UCS**. Коды в стандарте Юникод разделены на несколько областей. Область с кодами от U+0000 до

U+007F содержит символы набора **ASCII** с соответствующими кодами. Далее расположены области знаков различных письменностей, знаки пунктуации и технические символы.

По умолчанию в *Lazarus* используется кодировка *UTF-8*, в которой все символы разделены на несколько групп. Символы с кодами менее 128 кодируются *одним байтом*, первый бит которого равен нулю, а последующие 7 бит в точности соответствуют первым 128 символам 7-битной таблицы **ANSI**. Следующие 1920 символов кодируются двумя байтами.

Символы кириллицы кодируются в *UTF-8* в точности двумя байтами!

Последующие символы кодируются тремя и четырьмя байтами.

Таблица 5.6 представляет функции для работы с символьными переменными.

Таблица 5.6 - Функции для работы с символами

Функция	Действие
<b>Chr(X)</b>	Возвращает значение символа по его коду
<b>Ord(Ch)</b>	Возвращает код символа ch
<b>Pred(Ch)</b>	Возвращает предыдущий символ из кодовой таблицы
<b>Succ(Ch)</b>	Возвращает следующий символ кодовой таблицы
<b>Uppcase(Ch)</b>	Преобразует строчную букву в заглавную. Работает только для букв английского алфавита.

#### 5.4.7 Строки символов

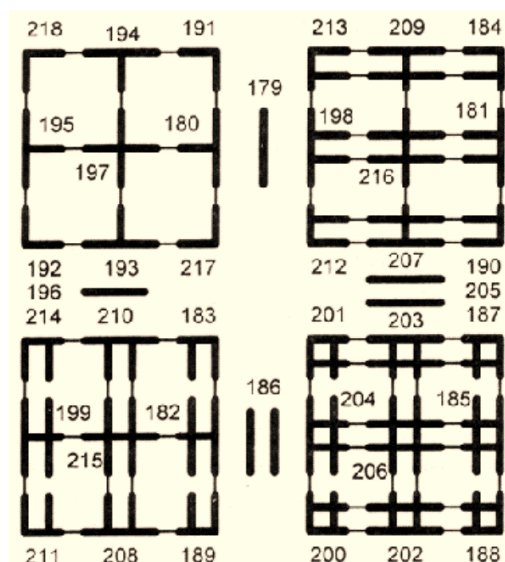
Для представления *текстовой информации* в **Pascal** используются *строки символов*, которые представляют собой последовательности символов, заключённые в одинарные кавычки, например:

**'Free Pascal – очередная версия Pascal'**  
**'Автоматизированное проектирование РЭС'**  
**'Автор алгоритмического языка Pascal - Н. Вирт'.**

Если между кавычками нет ни одного символа, то такая строка считается пустой (нулевой длины):

**" - пустая строка**

Кроме того, любой символ **ASCII** в строке (в том числе и те символы, которых нет на клавиатуре) может быть представлен в виде целого десятичного числа (от 0 до 255), перед которым ставится символ #.



**#182** Псевдографические символы используются в основном для графического оформления программ с текстовым интерфейсом пользователя — отображения в них окон, меню, кнопок и прочих элементов интерфейса. Первоначально предназначались для изображения рамок и таблиц на алфавитно-цифровых терминалах. Например, символ псевдографики «`├─┤`» - одинарное ответвление справа в таблице с двойной рамкой, выполненной в текстовом режиме.



**#7** Символ «короткий гудок» - позволяет при выводе чего-либо на экран дать сигнал

Перевод  
| строки

**#10** Управляющий символ «перевод строки» - перевод курсора на строку ниже



**#13** Управляющий символ «возврат каретки» - перевод курсора в начало строки



**#9** Символ табуляции используется для выравнивания начала текста в строках. Встретив этот символ, терминал перемещает курсор (или каретку принтера) вправо на ближайшую позицию табуляции.

Тип строковой переменной указывается соответствующими зарезервированными служебными словами (Таблица 5.7).

Таблица 5.7 – Символьные типы в Free Pascal

Тип строки	Максимальная длина	Есть ли нулевой символ в конце	Назначение
<i>AnsiString</i>	2 Гб	есть	Для хранения символов <i>ANSI</i> (1 байт на символ);
<i>String</i>	255 байт/2 Гб	нет/есть	По умолчанию воспринимается компилятором как <i>AnsiString</i>
<i>ShortString</i>	255 байт	нет	Аналогична <i>String</i>
<i>WideString</i>	1 Гб	есть	Для хранения символа <i>Unicode</i> (2 байта на символ)

Таблица 5.8 и Таблица 5.9 приводят традиционные для *Pascal* встроенные процедуры и функции для работы со строками символов, а также примеры их использования. Их можно применять и в *Free Pascal*.

Таблица 5.8 – Встроенные процедуры *Free Pascal* для работы со строками

Процедура	Описание	Примеры использования		
		Значение	Выражение	Результат
<b>Delete(St, Poz, N)</b>	Удаляет <b>N</b> символов из строки <b>St</b> , начиная с позиции <b>Poz</b>	<b>St = '1234567'</b>	<b>Delete(St, 4, 2)</b>	<b>St = '12367'</b>
<b>Insert(St1, St2, Poz)</b>	Добавляет в строку <b>St2</b> подстроку <b>St1</b> , начиная с позиции <b>Poz</b>	<b>St1 = 'аб'</b> <b>St2 = '1234'</b>	<b>Insert(St1, St2, 3)</b>	<b>St1 = 'аб'</b> <b>St2 = '12аб34'</b>

Таблица 5.9 – Встроенные функции *Free Pascal* для работы со строками

Функция	Описание	Примеры использования		
		Значение	Выражение	Результат
<b>Length(St)</b>	Возвращает динамическую длину строки в байтах (не в символах!).	<b>St = '1234abc'</b>	<b>L := Length(St)</b>	<b>L = 8</b>
<b>Copy(St, Poz, N)</b>	Выделяет из строки <b>St</b> подстроку длиной <b>N</b> символов, начиная с позиции <b>Poz</b>	<b>St = '1234567'</b>	<b>S := Copy(St, 2, 4)</b>	<b>S = '2345'</b>
<b>Pos(St1, St2)</b>	Обнаруживает первое появление в строке <b>St2</b> подстроки <b>St1</b> . Результат – номер той позиции, где находится первый символ подстроки <b>St1</b> . Если в <b>St2</b> подстроки <b>St1</b> не найдено, результат равен нулю.	<b>St1 = 'эд'</b> <b>St2 = 'абвгдеж'</b>	<b>N := Pos(St1, St2)</b>	<b>N = 4</b>
		<b>St1 = '12'</b> <b>St2 = 'абвгдеж'</b>	<b>N := Pos(St1, St2)</b>	<b>N = 0</b>
<b>UpCase(Ch)</b>	Преобразует строчную латинскую букву <b>Ch</b> в заглавную.	<b>Ch = 'a'</b>	<b>S := UpCase(Ch)</b>	<b>S = 'A'</b>
		<b>Ch = 'ы'</b>	<b>S := UpCase(Ch)</b>	<b>S = 'Ы'</b>
<b>Concat(St1, St2, ..., StN)</b>	Выполняет конкатенацию (сцепление) последовательности строк <b>St1, St2, ..., StN</b>	<b>St1 = 'з.'</b> <b>St2 = 'УУ'</b> <b>St2 = 'Томск'</b>	<b>S := Concat(St1, St2, St3)</b>	<b>S = 'з.УУТомск'</b>
<b>St1 + St2 + ... + StN</b>	Выполняет конкатенацию (сцепление) последовательности строк <b>St1, St2, ..., StN</b>	<b>St1 = 'з.'</b> <b>St2 = 'УУ'</b> <b>St2 = 'Томск'</b>	<b>S := St1 + St2 + St3</b>	<b>S = 'з.УУТомск'</b>

При выводе числовой информации на внешние устройства системные программы осуществляют перевод данных из машинного представления в строковый вид. Ввод числовых данных с клавиатуры тоже сопровождается преобразованием внешнего символьного представления чисел в их внутренний машинный формат. Средства языка позволяют делать аналогичные преобразования в оперативной памяти по соответствующим обращениям из программы (Таблица 5.10).

Таблица 5.10 - Прямые и обратные преобразования числовых данных в *Free Pascal*

Процедура	Описание	Примеры использования		
		Значение	Выражение	Результат
<i>Str(X[:M][:N], St)</i>	Преобразует вещественное <i>X</i> или целое <i>I</i> численное значение в его строковое представление <i>St</i> . <i>M, N</i> – необязательные параметры форматирования вывода: <i>M</i> - длина строки <i>St</i> , <i>N</i> - количество знаков после запятой	<i>Целое</i> <i>I = 2567</i>	<i>Str(I:6, St)</i>	<i>St = '002567'</i>
		<i>Веществ.</i> <i>X = 3.7e+03</i>	<i>Str(X:10, St)</i>	<i>St = '03.7e+0003'</i>
		<i>Веществ.</i> <i>X = 3.7e+03</i>	<i>Str(X:10:2, St)</i>	<i>St = '0003700.00'</i>
<i>Val(St, X, Cod)</i>	Преобразует строковое значение <i>St</i> в его численное представление <i>X</i> (переменная вещественного или целого типа). В вспомогательной переменной <i>Cod</i> (целого типа) возвращается номер позиции первого ошибочного символа. <i>Cod = 0</i> , если преобразование произошло успешно.	<i>St = '1234'</i>	<i>Val(St, I, Cod)</i>	<i>Целое</i> <i>I = 1234</i> <i>Cod = 0</i>
		<i>St = '3.7e+03'</i>	<i>Val(St, X, Cod)</i>	<i>Веществ.</i> <i>X = 3700.0</i> <i>Cod = 0</i>
		<i>St = '3.7+03'</i>	<i>Val(St, X, Cod)</i>	<i>Веществ.</i> <i>X = ?</i> <i>Cod = 4</i>

Во *Free Pascal* появилось много дополнительных функций по прямому и обратному преобразованию числовых данных (Таблица 5.11).

Таблица 5.11 - Новые функции *Free Pascal* для преобразования числовых данных

Формат обращения	Выполняемые действия
<i>St := IntToBin(Num, k)</i>	Преобразование целочисленного значения <i>Num</i> из машинного формата в символьное представление, содержащее <i>k</i> двоичных разрядов
<i>St := IntToHex(Num, k)</i>	Преобразование целочисленного значения <i>Num</i> из машинного формата в символьное представление, содержащее <i>k</i> шестнадцатеричных разрядов



Формат обращения	Выполняемые действия
<i>St := IntToStr(Num)</i>	Преобразование целочисленного значения <i>Num</i> из машинного формата в символьное представление десятичного числа
<i>V := StrToInt(St)</i>	Преобразование целочисленного значения из символьного представления в машинный формат типа <i>Integer</i>
<i>V := StrToInt64(St)</i>	Преобразование целочисленного значения из символьного представления в машинный формат типа <i>Int64</i>
<i>V := StrToQWord(St);</i>	Преобразование целочисленного значения из символьного представления в машинный формат типа <i>QWord</i>
<i>St := FloatToStr(Num);</i>	Преобразование вещественного значения <i>Num</i> в символьное представление
<i>V := StrToFloat(St);</i>	Преобразование символьного представления вещественного числа в машинный формат типа <i>Extended</i>
<i>V := FloatToCurr(Num);</i>	Преобразование вещественного значения <i>Num</i> в машинный формат представления денежных единиц
<i>V := StrToCurr(St);</i>	Преобразование символьного представления денежных единиц в машинный формат типа <i>Currency</i>

**Примечание.** Функция преобразования вещественного значения в символьный формат допускает задание дополнительных аргументов, управляющих форматом результата:

*St := FloatToStr(Num, Format[, Precision, m]);*

Параметр *Format* может принимать одно из следующих значений:

- *ffCurrency* — перевод в символьный формат денежных единиц;
- *ffExponent* — перевод в символьный формат с плавающей запятой;
- *ffFixed* — перевод в символьный формат с фиксированной запятой;
- *ffGeneral* — перевод в формат с плавающей или фиксированной запятой;
- *ffNumber* — перевод в формат десятичного числа со вставкой символа разделителя «тысяч».

Параметр *Precision* с последующим числовым аргументом *m* управляет количеством значащих цифр.

Параметр *Format* с последующим числовым аргументом *m* управляет количеством значащих цифр.

Выбор того или иного представления в формате *ffGeneral* определяется системой по величине преобразуемого значения.

#### 5.4.8 Логический тип



Логическое высказывание (выражение) может быть либо истинно, либо ложно. Значениями логического типа **Boolean** может быть одна из предварительно объявленных констант **False** (Ложь) или **True** (Истина). Причём логическому **True** соответствует порядковое число 1, а **False** 0. Следовательно **True** «больше» **False**!

Переменные типа **Boolean** занимают 1 байт.



## 6 Список литературы

1. Фаронов В.В. Turbo Pascal. Наиболее полное руководство (в подлиннике). — СПб.: БХВ-Петербург, 2004. — 1056 с.
2. Попов В.Б. Паскаль и Дельфи. Самоучитель. — СПб.: Питер, 2004. — 544 с.
3. Макарова Н.В. Информатика: Учебник для вузов/ Н.В. Макарова, В.Б. Волков. - СПб.: Питер, 2012. - 576 с. (Стандарт третьего поколения). [51 экз].
4. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В.. Самоучитель по программированию на Free Pascal и Lazarus.. - Донецк: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. - 503 с.
5. Мансуров К.Т. Основы программирования в среде Lazarus. - М.: Нобель пресс, 2013. – 772 с.
6. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Free Pascal и Lazarus: Учебник по программированию / Е.Р. Алексеев, О.В. Чеснокова, Т.В. Кучер. - М.: Издательский дом ДМК-пресс, 2010. - 440 с.
7. Кетков Ю.Л. Свободное программное обеспечение. FREE PASCAL для студентов и школьников / Ю.Л. Кетков, А.Ю. Кетков. — СПб.: БХВ-Петербург, 2011. — 384 с.
8. Фленов М.Е. Библия Delphi. — 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2008. - 800 с.
9. ОС ТУСУР 01-2013 (СТО 02069326.1.01-2013). Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления. - Томск: ТУСУР, 2013. – 57 с.
10. Рапаков Г.Г., Ржеуцкая С.Ю. Программирование на языке Pascal. — СПб.: БХВ-Петербург, 2004. - 480 с.
11. Lazarus Tutorial/ru [Электронный ресурс] // База знаний о Free Pascal, Lazarus и родственных проектах: [сайт]. URL: [http://wiki.freepascal.org/Lazarus\\_Tutorial/ru](http://wiki.freepascal.org/Lazarus_Tutorial/ru)
12. Программирование на Lazarus [Электронный ресурс] // «ИНТУИТ» Национальный открытый университет: [сайт]. URL: <http://www.intuit.ru/studies/courses/13745/1221/lecture/23276?page=1>