[1) Разработка программ, язык программирования Pascal.](#_e5gfzhuit3vz)

[2) Выражения и операции во Free Pascal](#_xowz933ee142)

[3) Простые операторы.](#_ulyz1wtuv8dl)

[4) Операторы ветвлений и циклов.](#_sor9rlsu2c1n)

[5) Процедуры и функции.](#_bgplbptjj1kq)

[6) Модули.](#_9vp6o828ht6x)

[7) Типы данных и переменные](#_savw0yrpe85q)

[8) Простые типы данных.](#_fv60f457k6w2)

[9) Тип данных массив.](#_3e96gt4y2zyy)

[10) Строковый тип.](#_q78z1mw3ga93)

[11) Тип данных множество.](#_7rhnrzfr7zha)

[12) Тип данных запись.](#_ewlukcixwqa5)

[13) Тип данных файл.](#_8gqsihh3d6b6)

[14) Тип данных указатель.](#_bew3tvcr3od6)

[15) Вариантный тип данных.](#_skqxxiuzp0i2)

[16) Хелперы.](#_tkbj21mvumrr)

# Разработка программ, язык программирования Pascal.

Инструментальные средства разработки программ: система (среда) программирования, редактор текста, транслятор, компилятор, интерпретатор, компоновщик, отладчик. История развития языка Pascal, распространенные диалекты и компиляторы Pascal: Turbo Pascal, Delphi, PascalABC.NET, Free Pascal. Кроссплатформенный компилятор Free Pascal. Состав системы программирования Free Pascal, утилиты. Структура программы Free Pascal. Режимы совместимости. Подготовка текста программы, компиляция, запуск программы.

Выражения и операции во Free Pascal.

Среда программирования - продвинутый текстовый редактор, где кроме редактора есть другие инструменты. Компиляция, сборка и запуск. Дебагинг (включены по умолчанию). Удобно работать над большим проектом

Редактор кода - легковесный редактор, где пишут код без доп. функций (но их можно установить), они туда не интегрированы по умолчанию

Транслятор - программное средство, которое переводит исходный код программы(понятный человеку) в бинарный код (понятный для процессора)

Компилятор — это вид транслятора, преобразующий исходный код с какого-либо языка программирования на бинарный код.

* Препроцессор (удалить коммент, выполнить текстовые замены, макросы, подключение библиотек)
* компиляция (трансляция, объектный файл(машинный код), компоновка(подключение библиотек))

**История** Язык программирования Pascal был разработан в 1970 году Никлаусом Виртом как средство обучения программированию и обеспечения строгой типизации.

**Распространённые диалекты и компиляторы Pascal**:

- Turbo Pascal: Выпущен в 1983 году компанией Borland, Turbo Pascal стал популярным благодаря высокой скорости компиляции и интегрированной среде разработки (IDE).

- Delphi: Разработан на основе Object Pascal, Delphi предоставляет мощные средства для разработки приложений с графическим интерфейсом и поддерживает объектно-ориентированное программирование.

- PascalABC.NET: Современный компилятор Pascal, ориентированный на .NET-платформу, поддерживает объектно-ориентированное программирование и интеграцию с .NET Framework.

- Free Pascal: Кроссплатформенный компилятор с открытым исходным кодом, поддерживающий различные архитектуры и операционные системы, включая Windows, Linux, macOS и другие.

**Кроссплатформенный компилятор Free Pascal**:

Free Pascal Compiler (FPC) — это бесплатный компилятор, предназначенный для языка программирования Pascal. Он предоставляет возможность создания кроссплатформенных приложений, что означает возможность написания программы один раз и компиляции её для различных операционных систем, таких как Windows, Linux, macOS и даже для некоторых встраиваемых платформ.

**Состав системы программирования Free Pascal**:

Система программирования Free Pascal включает в себя компилятор, который преобразует исходный код в машинный код, а также набор утилит и библиотек, обеспечивающих поддержку различных функций и платформ.

**Структура программы Free Pascal**:

Программа на Free Pascal состоит из следующих частей:

1. Модульная часть: определяет используемые модули и библиотеки.

2. Объявления: включают объявления типов, переменных и констант.

3. Основная программа: содержит блок `begin...end`, в котором размещается исполняемый код.

**Режимы совместимости**:

Free Pascal поддерживает различные режимы совместимости, позволяя адаптировать код под особенности различных диалектов Pascal, таких как Turbo Pascal, Delphi и другие.

**Подготовка текста программы, компиляция, запуск программы**:

1. Подготовка текста программы: Создайте исходный код программы в текстовом редакторе с расширением `.pas`.

2. Компиляция: Запустите компилятор Free Pascal, указав имя исходного файла.

3. Запуск программы: После успешной компиляции выполните полученный исполняемый файл.

**Выражения и операции во Free Pascal**:

Free Pascal поддерживает стандартные арифметические, логические и строковые операции, а также операторы сравнения и присваивания. Особенности синтаксиса и доступные операции могут зависеть от выбранного режима совместимости.

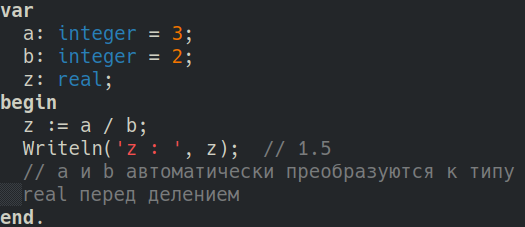
# Выражения и операции во Free Pascal

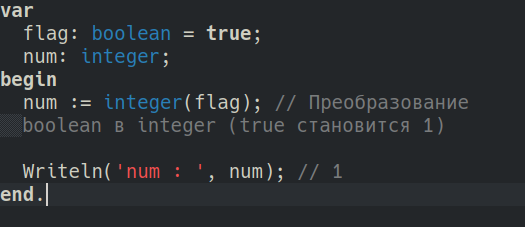
Структура выражений. Унарные и бинарные операции, типы аргументов и результата, автоматическое преобразование типа во время выполнения операций. Типы операций: арифметические, логические, строковые, операции над множеством, отношения, операция @. Приоритет операций. Перегрузка операций, разработка перечислителей для операции in.

**Унарные операции**: Операции, которые работают с одним операндом. Пример: not, унарный минус -.

Бинарные операции: Операции, работающие с двумя операндами. Пример: x + y, a > b.

**Автоматическое преобразование типа во время выполнения операций**





# Простые операторы.

Понятие оператора, отличия от операции. Составной оператор. Простые операторы. Оператор присваивания. Особенности оператора присваивания. Перегрузка оператора присваивания. Оператор вызова процедуры, операция вызова функции. Оператор перехода goto. Ограничения в применения оператора перехода. Причины нежелательности применения оператора перехода.

**Оператор присваивания** - оператор с помощью которого задается значение в переменную, заменяя то, которое было.

Сначала вычисляется значение выражения, указанного в правой части оператора, а затем его результат записывается в область памяти (переменную), имя которой указано слева.

**Оператор вызова процедуры, функции** - имя функции или процедуры.

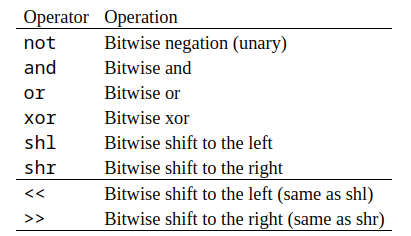
**Операторы** — это конструкции языка программирования с определенными функциями. Отличие от операции состоит в том, что операции выполняются над операндами. Операции могут быть унарными и бинарными.

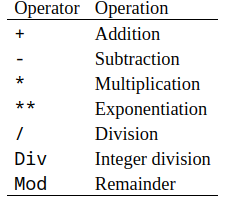
**Простой оператор**: оператор, который выполняет одну простую инструкцию, действие. Не содержат вложенных инструкций. Например, оператор присваивания, инкрементация, readln.

**Составной оператор**: оператор который состоит из нескольких инструкций, заключенных между begin - end.

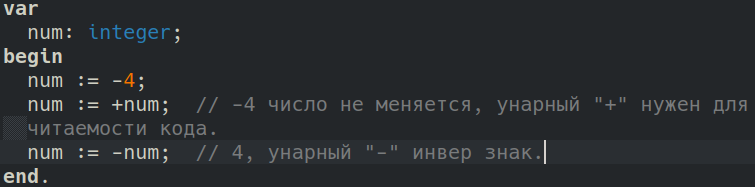
(циклы, условия и тд)

Пример логический операторов(побитовые и, побитовое или):



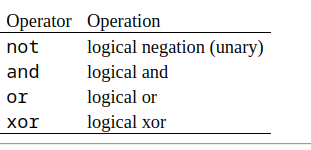
Арифметические (бинарные):  


Унарные арифм (всего два: +, -):



Другие унарные: @, \*\*

Булевые операторы (однобайтовые)



goto - прыжки внутрь(извне) циклов могут привести к неопределенному поведению; неудобно читать, отлаживать код.

# Операторы ветвлений и циклов.

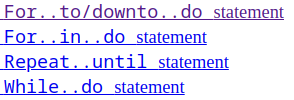
Условный оператор if. Полная и сокращенная запись. Вложенные операторы if. Оператор выбора сase. Взаимозамена условного оператора и оператора выбора. Виды циклов. Цикл со счетчиком for, Ограничения на переменную-счетчик. Цикл for in. Цикл с предусловием while. Цикл с постусловием repeat. Области применения каждого вида цикла. Вложенные циклы одинаковых и разных видов. Прерывание и переход на следующую итерацию цикла: операторы break и continue. Выход из вложенных циклов с помощью оператора goto и процедуры exit.

**Условный оператор if** - оператор, с помощью которого можно реализовывать логические условия, ветвления.

If..then..else - полная

If..then.. - сокращенная

**Циклы**



**Ограничения на переменную-счетчик**

Должна быть порядкового типа (ordinal type)

Нельзя изменять значение переменной-счетчика внутри цикла

После завершения цикла или если цикл не выполнился ни разу, значение переменной-счетчика становится неопределенным

Для вложенных процедур переменная-счетчик должна быть локальной переменной

**For in …**  
На основании предоставленного текста, расскажу про цикл for..in в Free Pascal:

Цикл for..in используется для перебора элементов в перечисляемых типах данных. Существует 5 возможных вариантов использования:

1. С типом-перечислением (enumeration):

- Перебирает все элементы перечисляемого типа

- Переменная цикла должна быть того же типа, что и перечисление

2. С множеством (set):

- Перебирает все элементы множества

- Переменная цикла должна быть базового типа множества

3. С массивом:

- Перебирает все элементы массива

- Переменная цикла должна иметь тот же тип, что и элементы массива

- Строки рассматриваются как массив символов

- Поддерживаются многомерные массивы (перебор слева направо)

4. С классами/объектами/записями, поддерживающими интерфейсы IEnumerator и IEnumerable:

- Тип переменной цикла должен соответствовать возвращаемому типу IEnumerator.GetCurrent

- Многие стандартные классы (TFPList, TList, TCollection, TStringList, TComponent) поддерживают эту функциональность

5. С любым типом, для которого определен оператор enumerator:

- Оператор должен возвращать структурированный тип, реализующий интерфейс IEnumerator

- Тип переменной цикла должен соответствовать возвращаемому типу GetCurrent

Важное ограничение: как и в обычном цикле for, нельзя изменять значение управляющей переменной внутри цикла.

# Процедуры и функции.

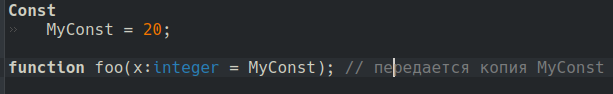
Понятие функции. Синтаксис и семантика определения функции. **Формальные параметры**. Параметры-значения, параметры-переменные, параметры - нетипированные переменные. Возврат результата из функции. Вызов функции, фактические параметры. Использование функций в выражениях. Процедуры: отличия от функций. Процедурные типы данных. Перегрузка процедур и функций. Рекурсия.

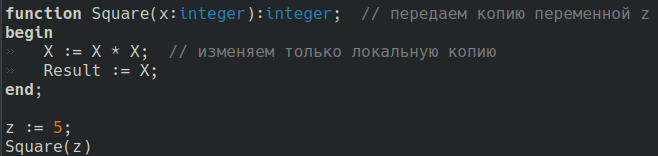
**Функция** - именованный блок кода, который используется в программе. Функция имеет имя(идентификатор) параметры и тип возвращаемого результата.

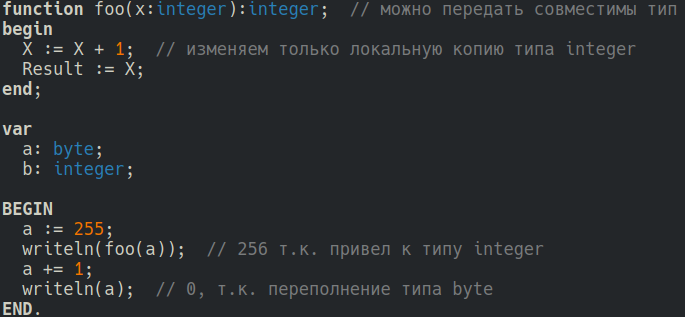
**Определение функции** происходит с помощью ключевого слова function.

**Параметры-значения**:

Когда срабатывает оператор вызова функции, то в функцию передается копия переданного параметра.



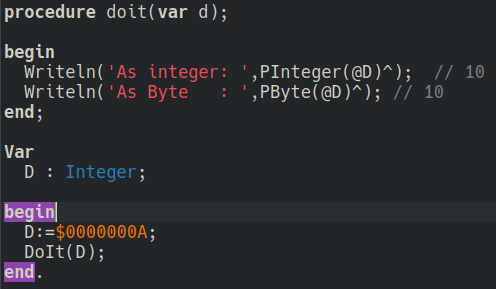




**Параметры-переменные:**

Когда срабатывает оператор вызова функции, то в функцию передается ссылка переданного параметра.

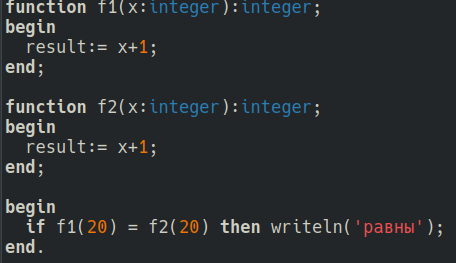
**параметры - нетипизированные переменные:**



В этом случае переменная не имеет типа в процедуре функции и, следовательно, несовместима со всеми остальными типами: Компилятор просто передает адрес переданной переменной в процедуру, поэтому все, что доступно в вызываемой процедуре, - это адрес, без какой-либо информации о типе

**Вызов функции, фактические параметры. Использование функций в выражениях:**

**Вызов функции** осуществляется с помощью оператора вызова(имя функции). Если у функции есть параметры(фактические), вызывать данную функцию необходимо с данными параметрами (их количество должно совпадать и они должны быть совместными). Для параметров-ссылок типы должны совпадать



**Процедурные типы данных** позволяют ссылаться на процедуры или функции как на переменные. Это дает возможность передавать процедуры в качестве параметров или хранить их в переменных.

Рекурсия - вызов самого себя.

# 6) Модули.

Понятие модуля, модульного программирования. Модульные программы. Модули в различных языках программирования. Объектные файлы, динамические и статические библиотеки. Компоновка программы из набора модулей. Синтаксис и семантика определения модулей. Подключение модулей. Структура модуля: интерфейсная секция, секция реализации, секции инициализации и финализации. Области видимости в модуле. Подключение модулей в модулях: прямые и косвенные ссылки. Решение проблемы циклических ссылок.

**Модуль** - файл, который содержит набор функций, процедур, объявлений, которые могут использоваться в разных программах или других модулях.

**Модульное программирование** - метод программирования, где программа организуется посредством объединения независимых частей(модулей). Модули разработаны отдельно и независимо друг от друга.

**Модули в различных языках программирования**: Си, C++, Python, JavaScript:

C++: Модульность реализуется через заголовочные файлы и исходные файлы. Заголовочные файлы содержат объявления функций и классов, а исходные файлы — их реализации.

Python: модули - файлы с расширением .py. Модули могут содержать функции, классы и переменные. Python также предлагает пакеты — группы связанных модулей. import название модуля

JS: отдельные файлы .js import/export

Как видно из синтаксической диаграммы, модуль всегда состоит из частей **interface** (интерфейса) и **implementation** (реализации). И необязательных блоков initialization (инициализации) и finalization (финализации), содержащие код, выполняющийся соответственно при запуске программы, и при её остановке.

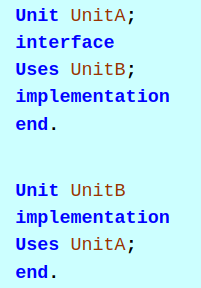
Интерфейсная часть объявляет все идентификаторы (имена), экспортируемые из модуля. Они могут быть объявлением константы, типы, переменные, процедуры и функции. Интерфейсная часть не может содержать исполняемый код: допускается **только** объявления

Все функции и методы, объявленные в *интерфейсной* части, должны быть реализованы в части *реализации* модул

Часть *реализации* в основном предназначена для реализации процедур и функций объявленных в *интерфейсной* части. Тем не менее, она тоже может содержать собственные объявления: объявленные внутри части реализации *недоступны* за пределами модуля.

Все, что объявлено в *implementation*, доступно только внутри модуля.

Модули могут быть взаимно *зависимы*, то есть они могут ссылаться друг на друга при использовании (uses). Это допускается, при условии, что по крайней мере одна из ссылок сделана в разделе *implementation* (реализации) модуля.



# 7) Типы данных и переменные

Понятие типа данных. Дихотомия типов данных и переменных. Объявление переменных. Глобальные, локальные, статические переменные, ограничения на размер переменных. Совмещение переменных различного типа в оперативной памяти с помощью модификатора absolute. Стандартные типы данных. Простые и структурные типы. Статические и динамические типы. Пользовательские типы данных. Преобразования типов, прямое и косвенное, перегрузка прямого преобразования типов. Принципы описания характеристик типа данных: объем, байтовая последовательность расположения в оперативной памяти, битовая структура, диапазоны значения, точность (для вещественных типов данных), операции и функции для работы с типом данных. Функции для получения информации о размере типа данных, границах типа, модификации байтовой структуры, перемещения и заполнения больших объемов данных в оперативной памяти.

**Тип данных** - определяет множество значений, которое может принимать переменная.

**Дихотомия типов данных и переменных**: типы данных бывает различное количество, переменные могут принимать разные типы данных.

Переменные **объявляются** в секции var и требуют указания типа:

**Простые типы**: Integer, Real, Boolean, Char.

**Составные типы**: массивы (array), множества (set), записи (record), файлы (file).

**Указатели**: ссылки на другие объекты.

**Глобальные переменные**: объявлены вне процедур и функций, доступны во всей программе.

**Локальные переменные**: объявлены внутри процедуры или функции и доступны только в их пределах.

**Статическая переменная** сохраняет свое значение даже после выхода из блока(фукнции), в котором она определена.

**Глобально** инициализированные переменные инициализируются однажды, *при запуске программы*.

**Локально** инициализированные переменные инициализируются *каждый раз*, когда вызывается процедура (функция).

**Статические типы** требуют объявления типа данных и этот тип не меняется

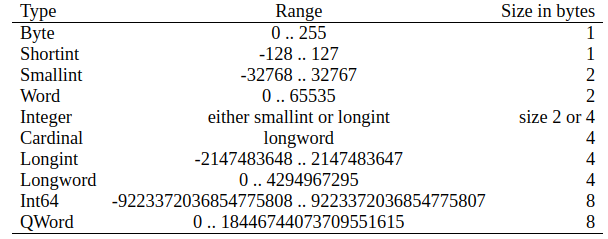
**Динамические типы**: Тип значения, хранящегося в переменной variant определяется во время выполнения: он зависит от того, что было ей присвоено

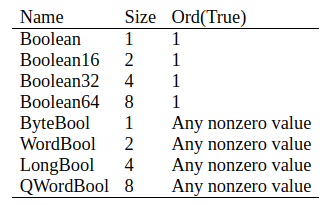
# 8) Простые типы данных.

Порядковые типы. Встроенный порядковый тип: целочисленные, логические, символьные типы. Характеристики встроенных порядковых типов. Производные порядковые типы: перечислимый тип и тип поддиапазона. Вещественные типы. Использование математического сопроцессора для выполнения операций над вещественными типами. Характеристики вещественных типов данных.

Все базовые типы, кроме вещественных, являются порядковыми

**целочисленные**:



**логические:**

Существует 2 **символьных** типа:

AnsiChar

Это 1-байтовый символ. Интерпретация символа зависит от кодовой страницы.

WideChar

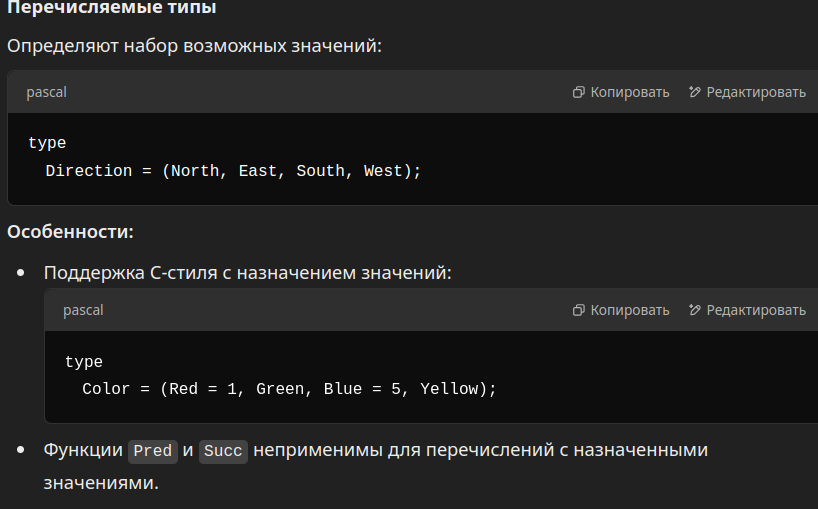
Это 2-байтовый символ. Интерпретация символа зависит от кодовой страницы.

Символы можно использовать в цикле

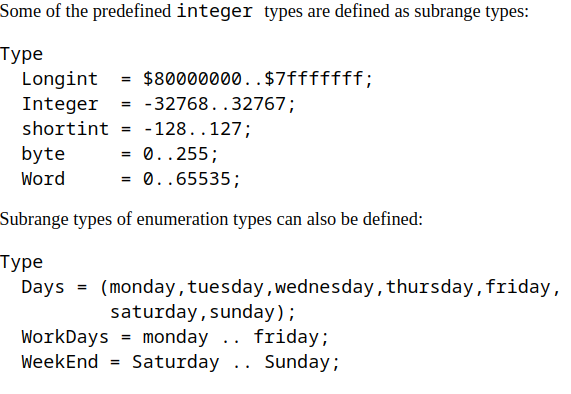
**Характеристики** встроенных порядковых типов:

* являются считаемыми(countable) и упорядоченными (ordered). Можно считать по одному с определенного порядка. Работают функции Inc, dec
* попытка уменьшить наименьшее число или увеличить наибольшее приведет к "оборачиванию" в пределах определённого модуля. Так же работают функции pred, succ

**Производные порядковые типы**:



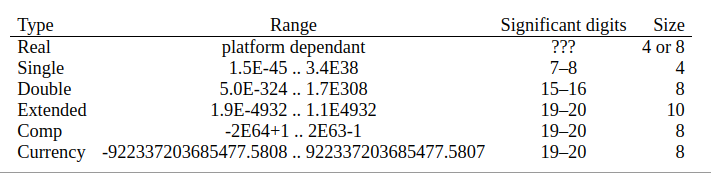
**Тип поддиапазона** - это диапазон значений порядкового типа (основного типа). Чтобы определить тип поддиапазона, необходимо указать его предельные значения: наибольшее и наименьшее значение типа.



Free Pascal использует **математический сопроцессор** (или эмуляцию) для всех вычислений с плавающей точкой. Родной тип Real зависим от процессора, но он является либо типом Single либо типом Double. Поддерживаются только типы плавающей точки стандарта IEEE

Современные процессоры (например, Intel, AMD) имеют **встроенный математический сопроцессор** (FPU — Floating Point Unit). Когда программа выполняет операции с типами Single, Double, Extended, сопроцессор автоматически берет на себя их обработку.

Вещественные типы:



# 9) Тип данных массив.

Понятие массива. Одномерные и многомерные массивы. Типы индекса. Литералы массива. Обращение к элементу массива. Итерация по массиву. Заполнение и обработка массива циклами. Индуктивные алгоритмы на массивах. Платформо-зависимые ограничения на объем массива. Битовые массивы. Динамические массивы, автоматическое выделение и освобождение памяти при манипуляциях с динамическими массивами. Характеристики массива как типа данных.

**Массив** - непрерывный участок памяти, содержащие последовательность объектов одного типа.

**Одномерный массив** - имеет одну размерность

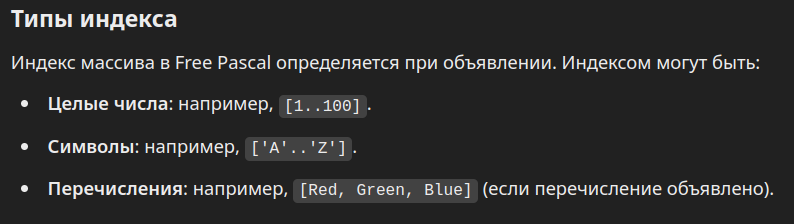
**многомерный массив** - массив внутри которого есть другой массив:

можно объединить два массива в один многомерный массив:

**двумерные массивы**:

b: array[1..3, 1..4] of real;

APoints = array[1..100] of Array[1..3] of Real;

**Тип индекса:**  


**Литерал массива** — это синтаксическая конструкция, с помощью которой можно инициализировать элементы массива при инициализации самого массива

var

numbers: array[1..5] of integer = (10, 20, 30, 40, 50);

К элементу массива **обращаются** по индексу

итерация по массиву осуществляется через цикл for

Var

A: Array[1..5] of Integer;

I: Integer;

begin

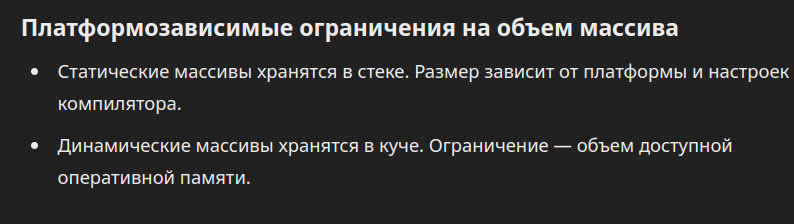
For I := Low(A) to High(A) do

Writeln(A[I]);

end;

**Индуктивные алгоритмы** подразумевают использование текущего и предыдущего значения.

**Платформо-зависимые ограничения на объем массива**



Однако с помощью процедуры pack можно преобразовать обычный массив в **битовый**. Возможна и обратная операция: битовый массив можно преобразовать в нормально упакованный с помощью процедуры распаковки, как показано в следующем примере:

Var

foo : array [ 'a'..'f' ] of Boolean

= ( false, false, true, false, false, false );

bar : bitpacked array [ 42..47 ] of Boolean;

baz : array [ '0'..'5' ] of Boolean;

begin

pack(foo,'a',bar);

unpack(bar,baz,'0');

end.

Операции pack и unpack используются для преобразования обычных массивов в битовые и обратно.

**Динамические массивы:**

При объявлении переменной типа динамический массив начальная длина массива равна нулю. Фактическая длина массива должна быть установлена стандартной функцией SetLength, которая выделит память необходимую для размещения элементов массива в куче. Следующий пример установит длину массива равную 1000:

Var

A : TByteArray;

begin

SetLength(A,1000);

Обратите внимание, что **длина массива задается в элементах**, а не в байтах выделяемой памяти (хотя они могут быть и одинаковыми). Объем выделенной памяти равняется размеру массива умноженном на размер одного элемента в массиве. **Память будет освобождена** при выходе из текущей процедуры или функции.

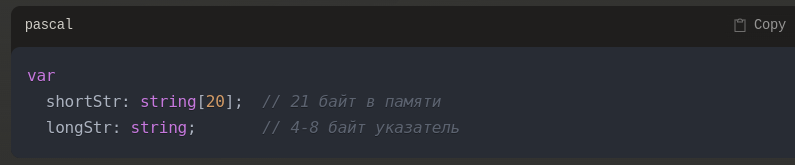
**Также есть возможность изменить размер массива**: в этом случае будет сохранено настолько много элементов сколько поместится в новом размере массива. Размер массива может быть **установлен в ноль**, это эффективный способ сбросить переменную.

Для динамических массивов ведется подсчет **ссылок**: присвоение одной переменной типа динамический массив, другой переменной, позволит **обеим** переменным **указывать на тот же самый массив.**

Также возможно скопировать и/или изменить размер массива с помощью стандартной функции **Copy**, которая действует как функция копирования для строк

# 10) Строковый тип.

Понятие строки в языках программирования. Литералы строк. Обращение к отдельным символам строки. Строка статичного максимального объема, с хранением актуальной длины (короткие строки). Строка, как последовательность символов, заканчивающаяся нулем (pchar). Гибридные (длинные) строки. Реализация всех типов строк в языке программирования. Переключение между длинными и короткими строками опциями компилятора. Сравнительные характеристики коротких и длинных строк. Автоматическое выделение и освобождение памяти при манипуляциях с длинными строками. Характеристики всех типов строк как типа данных.



На основе предоставленного документа, я расскажу о различных типах строк в Free Pascal:

1. Короткие строки (ShortString):

- Максимальная длина 255 символов

- Фактическая длина хранится в нулевом индексе строки(ord(s[0]))

- Память выделяется на стеке

- Объявляются при выключенном переключателе {$H-} или при явном указании размера в квадратных скобках

- Размер в памяти: объявленная длина + 1 байт

2. PChar (строки в стиле C):

- Строки, заканчивающиеся нулевым символом

- Представляют собой указатель на массив символов

- Поддерживают арифметику указателей

- Размер в памяти: размер указателя на стеке + (длина + 1) байт в куче

- Можно выполнять преобразования между PChar и другими типами строк

3. AnsiString (длинные строки):

- Без ограничения длины

- Хранятся в куче (heap)

- Имеют счетчик ссылок (reference counting)

- Автоматическое управление памятью

- Всегда завершаются нулем

- Поддерживают работу с кодовыми страницами

- Размер в памяти: размер указателя на стеке + (длина + 1 + 16) байт в куче

Особенности управления памятью для AnsiString:

- При присваивании увеличивается счетчик ссылок

- Копирование данных происходит только при модификации (copy-on-write)

- Автоматическое освобождение памяти при обнулении счетчика ссылок

- Пустая строка представлена как nil-указатель

Переключение между типами строк:

- Директива компилятора {$H+} делает тип String эквивалентным AnsiString

- Директива {$H-} делает тип String эквивалентным ShortString

- При указании размера в квадратных скобках всегда создается ShortString независимо от директивы

Сравнительные характеристики:

1. ShortString:

- Преимущества: быстрые операции, меньше накладных расходов

- Недостатки: ограниченная длина, нет автоматического управления памятью

2. AnsiString:

- Преимущества: неограниченная длина, автоматическое управление памятью, эффективное копирование

- Недостатки: больше накладных расходов, дополнительное использование кучи

3. PChar:

- Преимущества: совместимость с C, прямой доступ к памяти

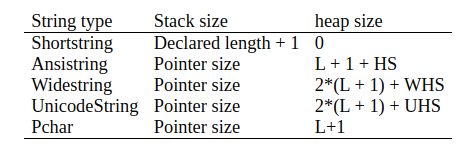
- Недостатки: ручное управление памятью, риск переполнения буфера

Рекомендации по использованию:

- Для коротких строк фиксированной длины: ShortString

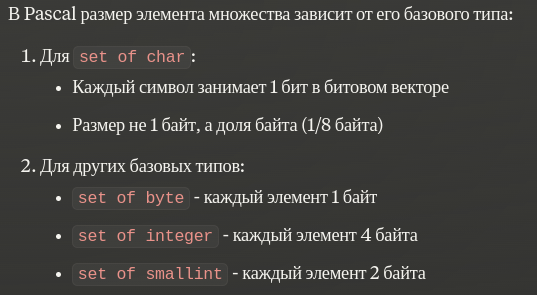
- Для длинных строк или когда требуется автоматическое управление памятью: AnsiString

- Для взаимодействия с C-кодом или низкоуровневых операций: PChar

размер:  


# 11) Тип данных множество.

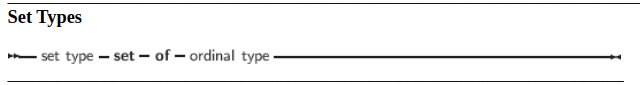
Множество как математическая абстракция. Подходы к реализации типа множества в различных языках программирование. Встроенный тип множества в языке программирования. Литералы множества. Доступ к элементам множества. Визуализация содержимого множества. Ограничения на диапазоны значений типа множества. Типичные задачи, решаемые использованием множеств. Характеристики множества как типа данных.



**Множество** - совокупность элементов(неупорядоченная), которые обладают общим свойством.

В **Python** для работы с множествами используется встроенный тип данных set. Это неупорядоченная коллекция, которая не допускает дубликатов.

Литералы:

  
Type

Junk = Set of Char;

Days = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

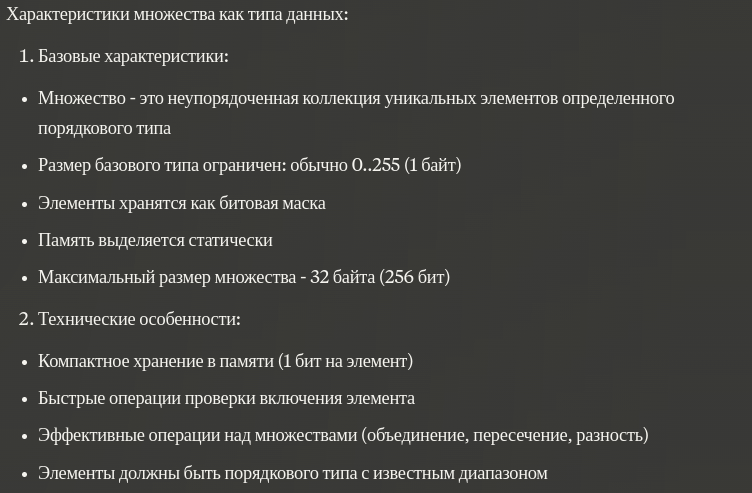
Var

WorkDays : Set of days;

**Доступ к элементам множества** осуществляется через цикл и in.  
можно сделать через побитовый сдвиг вправо.

**Ограничения на диапазоны значений типа множества:**

Множество должно иметь порядковый тип.  
множество может содержать максимум 256 элементов



Фильтрация и проверка символов:

var

Digits: set of char;

c: char;

begin

Digits := ['0'..'9'];

if c in Digits then

// обработка цифры

end;

# 12) Тип данных запись.

Понятие структуры (составного типа, типа запись). Реализация записи в языке программирования. Литералы записи. Обращение к полям записи. Битовые поля записи. Базовая и вариантная часть записи. Разделение одной области памяти различными типами переменных с помощью вариантной части записи. Использование особенностей вариантной части для исследования типов данных. Характеристики записи как типа данных.

**Тип запись** - это составной тип данных, который объединяет в себе разные типы(поля). Данные поля могут быть доступны через имена.

пример:

type

TPerson = record

Name: string;

Age: integer;

end;

var

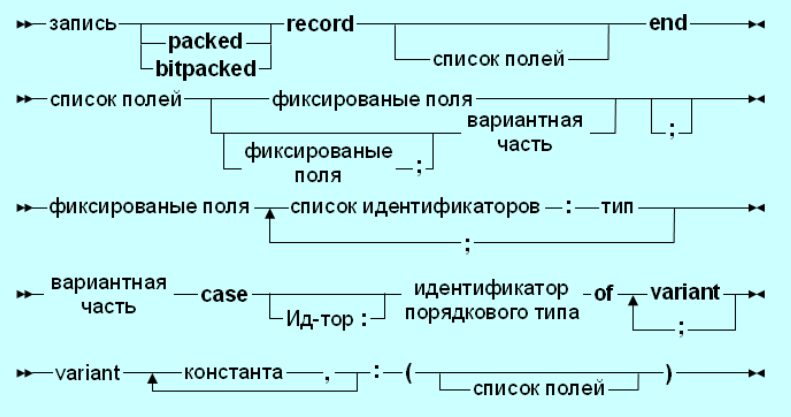
Person: TPerson;

begin

Person.Name := 'John';

Person.Age := 30;

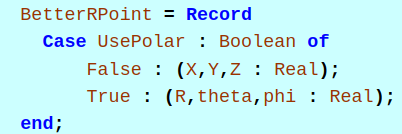
end;

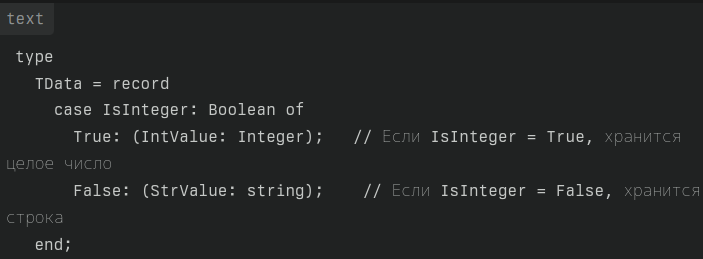


**Обращения** к полям записи происходит через их имена.

**Битовые поля** позволяют эффективно использовать память, выделяя конкретное число битов для хранения данных в полях записи. Однако Free Pascal непосредственно не поддерживает битовые поля, в отличие от таких языков, как C/C++. В Free Pascal возможно реализовать подобные механизмы через побитовые операции.

**Вариантная часть записи:**

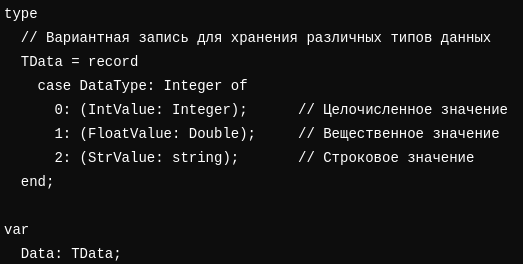




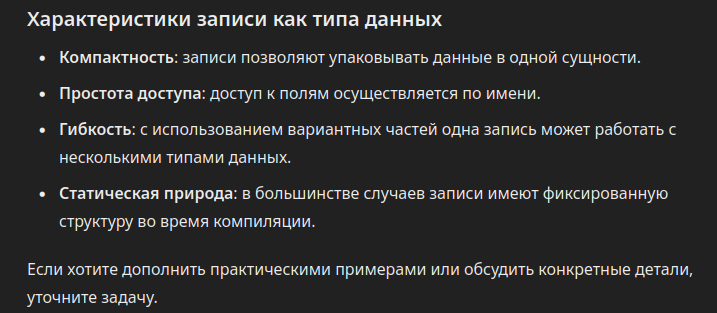
**Вариантная часть** должна быть последней в записи. Опциональный идентификатор в операторе **case** служит для того, чтобы получить доступ к значению поля, которое иначе было бы невидимо для программиста. Оно может использоваться, чтобы видеть, какая вариантная часть является активной в определенное время. (Это нужно для поддержания этого поля) В действительности оно представляет новое поле в записи.

**Вариантная часть** структуры позволяет использовать **одну область памяти** для представления разных типов данных.

**Исследование разных типов:**



**Характеристики:**



# 13) Тип данных файл.

Файл, как последовательность данных во внешней памяти. Файлы последовательного и прямого доступа. Специфика понятия «файл» на различных платформах. Понятие файлового указателя. Типы файлов в языке программирования: текстовые, типированные, нетипированные. Последовательность обработки файлов. Стандартные потоки ввода-вывода как текстовые файлы, перенаправление и конвееризация. Режимы открытия текстовых файлов, последовательные запись-чтение, механизмы определения конца файла. Файлы прямого доступа: режимы открытия, чтение-запись, перемещение по файлу, определение текущей позиции, размера файла. Обработка ошибок доступа к файлу. Характеристики файлов как типа данных.

**Файловые типы** - типы, который хранит последовательность некоторого базового типа. Файловый тип используются обычно, чтобы хранить данные на диске, долговременной памяти.

**Последовательный** **доступ**: Данные читаются или записываются последовательно, от начала файла до конца. Этот метод подходит для файлов, где информация обрабатывается в порядке её поступления. Примером могут служить текстовые файлы, где строки читаются одна за другой.

**Прямой** **доступ**: Дает возможность обращаться к произвольной позиции в файле без необходимости считывать данные последовательно.

**Unix и Linux**: В этих системах реализована концепция "всё есть файл", что означает, что не только данные, но и устройства (например, принтеры) рассматриваются как файлы. Это позволяет использовать единый интерфейс для работы с различными ресурсами.

**Windows**: В Windows файлы организованы в иерархическую структуру каталогов, и акцент делается на удобство пользовательского интерфейса.

FilePos(f) возвращает текущую позицию в открытом файле.

FileSize(f) возвращает общий размер файла.

EOF - end of file

# 14) Тип данных указатель.

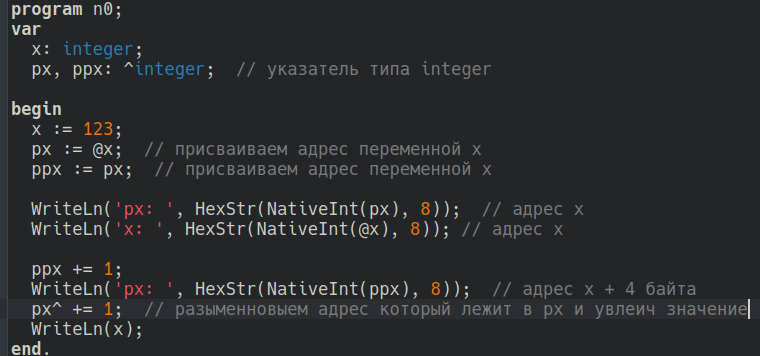
Понятие указателя. Типы указателей, преобразование типов указателя. Получение адреса переменной. Выделение и освобождение памяти для указателей. Понятие «утечка памяти». Системы сборки мусора в различных языках программирования, их достоинства и недостатки. Последовательное перемещение по оперативной памяти с помощью указателей. Характеристики указателя как типа данных.

**Указатель** - тип данных, который хранит адрес ячейки памяти, где хранятся данные какой-то переменной.

Указатели **типизированы**, это означает, что они указывают на определенный вид данных. Тип этих данных должен быть известен во время компиляции.

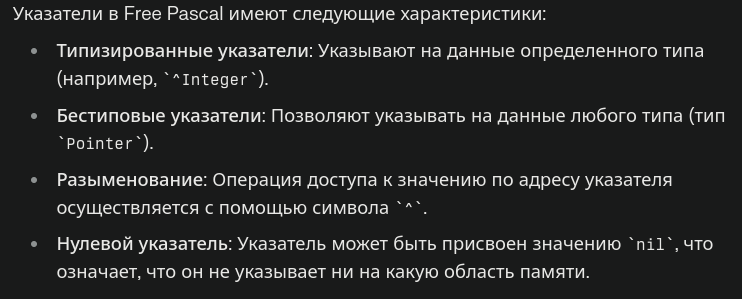
**Получение адреса** переменной - @

**Разыменование** - ^



**Утечка памяти** — это ситуация, когда программа выделяет память в динамической области, но не освобождает её после использования.

**Характеристика**:

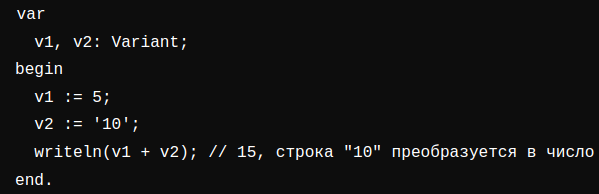


# 15) Вариантный тип данных.

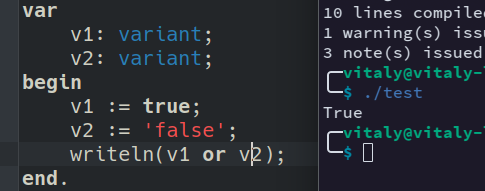
Понятие вариантного типа. Принципы работ с вариантным типом. Операции над вариантным типом, принципы формирования типа результата. Ограничения вариантных типов. Определение типа данных, сохраненного в вариантной переменной. Модуль variants. Вариантные массивы, области их применения. Создание вариантных массивов заранее неизвестной размерности. Манипуляции над элементами вариантных массивов.

**Вариантный тип** - универсальный тип данных, способный хранить значения разных типов(строки, числа). Способен изменять свой тип в ходе выполнения программы. Весит больше, операции за меньшее время.

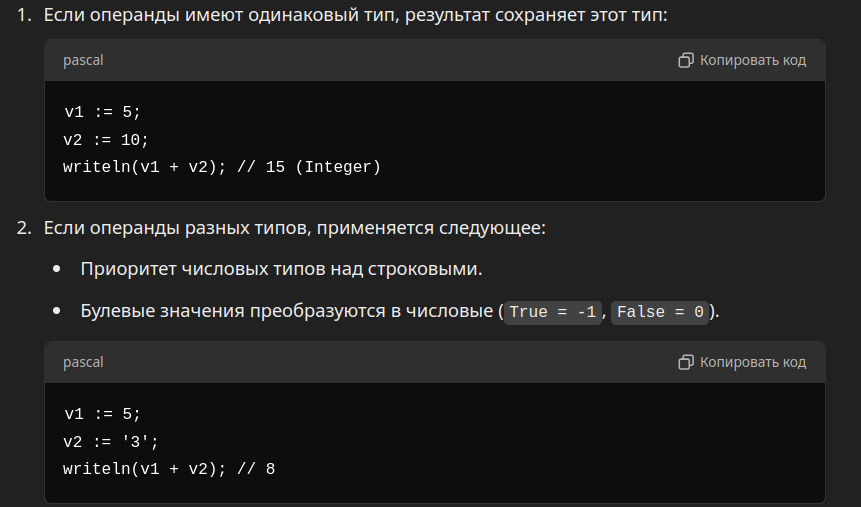
**Операции над вариантным типом:** Сложение, вычитание, умножение, деление и другие операции поддерживаются, если оба операнда могут быть преобразованы к числовым типам.

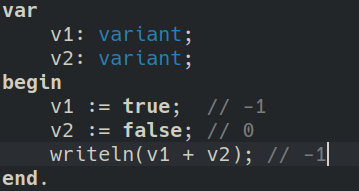


Поддерживаются and, or, not, если операнды булевы или могут быть преобразованы в булевы



**Принципы формирования типа результата:**

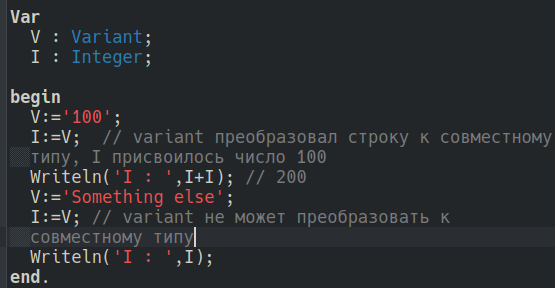
****

****

**Ограничения вариантных типов:**

Если типы несовместимы, выбрасывается ошибка

(или: byte to integer, integer to double, char to string)



Variant не может напрямую хранить сложные типы(записи)

для этого нужно создавать массив

// Создаем Variant с отдельными двумя значениями(0, 1)

V := VarArrayCreate([0, 1], varVariant);

…

Тип variant занимает больше памяти(16-24 байта) и требует больше времени для вычисления

**Определение типа данных, сохраненного в вариантной переменной:**

writeln(VarType(v)); // Выводит тип данных (например, 19 — varInteger)

**Модуль variants**

Модуль variants предоставляет функции для работы с Variant

VarType

VarToStr

VarArrayCreate

**Вариантные массивы**

Массив, который может хранить разные типы данных./  
может хранить сложные типы (записи)

VarArrayLowBound — нижняя граница индекса.

VarArrayHighBound — верхняя граница индекса.

# 16) Хелперы.

Понятие хелпера, области применения хелперов. Структура хелпера, способы привязки хелпера к обычным типам данных, записям и классам. Наследование хелперов. Штатные хелперы модуля sysutils. Разработка собственного хелпера.

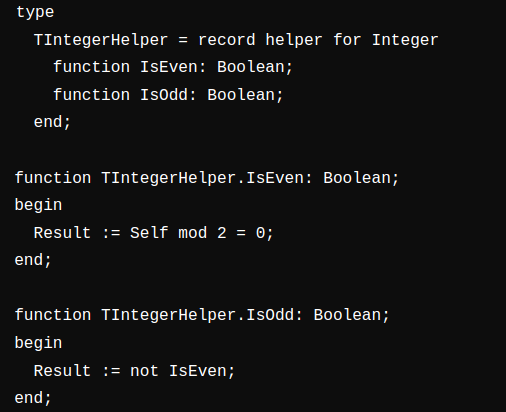
**Хелперы** — это инструмент языка с помощью которого можно расширять функциональность существующих типов данных (включая классы, записи и примитивы), не изменяя их исходного кода.

Хелперы позволяют расширить данный класс, интерфейс, запись или примитивный тип дополнительной функциональностью, не делая наследование класса и без повторного объявления записи.

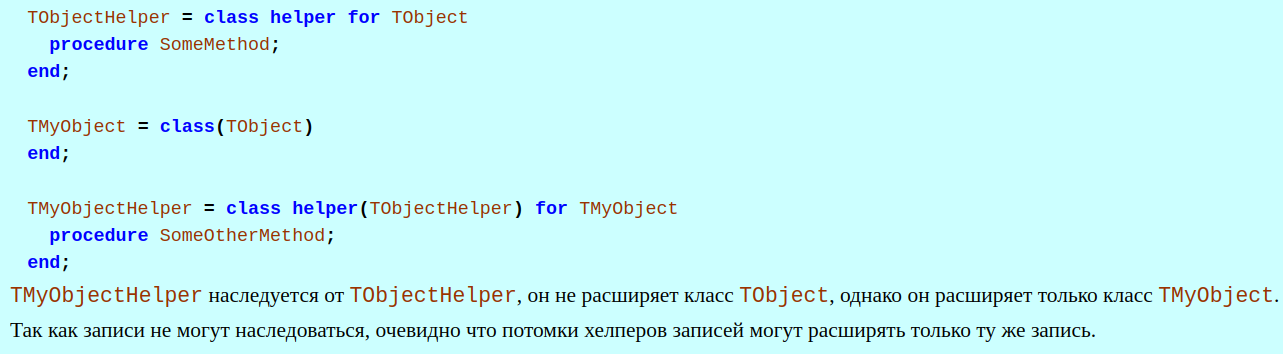
Внутри методов, **Self** будет ссылаться на значение записи или простого типа.

**Структура:**

Хелпер описывается с помощью ключевого слова type и должен быть объявлен как record helper, class helper или for для конкретного типа данных.



**Наследование:**  
Потомок хелпера класса может расширять другой класс, чем его родитель.



## Хелперы в Free Pascal и их использование

\*\*Хелперы\*\* (Helpers) в Free Pascal представляют собой механизм, позволяющий расширять функциональность существующих типов данных, таких как классы, записи и базовые типы. Это особенно полезно, когда необходимо добавить методы или свойства к уже существующим типам без создания новых классов или записей.

### Способы привязки хелперов к типам данных

1. \*\*Классовые хелперы\*\*: Используются для добавления методов к классам. Объявляются с помощью синтаксиса `type <ИмяХелпера> = class helper for <ИмяКласса>`. Например:

```pascal

type

TMyClassHelper = class helper for TMyClass

public

function NewMethod: Integer;

end;

```

2. \*\***Хелперы для записей**\*\*: Позволяют добавлять методы к записям. Синтаксис аналогичен классовым хелперам, но используется `record` вместо `class`.

3. \*\***Типовые хелперы**\*\*: Могут быть использованы для расширения функциональности базовых типов данных, таких как строки или числовые типы. Например, можно создать хелпер для типа `Integer`.

### **Наследование хелперов**

Хелперы могут наследоваться друг от друга. Это позволяет создавать более специализированные хелперы на основе уже существующих. Синтаксис наследования аналогичен обычному классовому наследованию:

```pascal

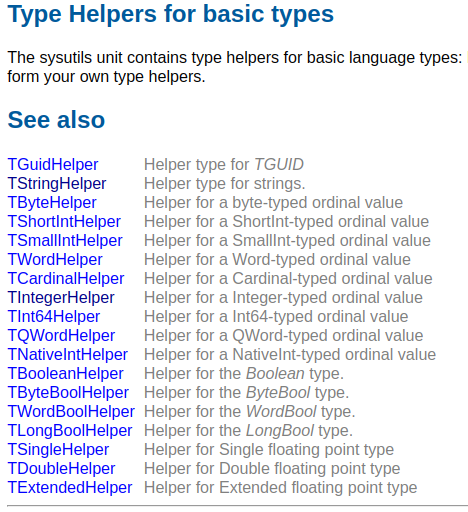
type

TBaseHelper = class helper for TBaseClass;

TDerivedHelper = class helper for TDerivedClass(TBaseHelper);

```

### **Штатные хелперы модуля sysutils**



- Хелпер для строк (`String`)

- Хелпер для целых чисел (`Integer`)

- Хелпер для логических значений (`Boolean`)

Эти хелперы могут использоваться напрямую или могут служить основой для создания собственных хелперов.

### **Разработка собственного хелпера**

Для разработки собственного хелпера необходимо:

1. Определить тип, к которому будет привязан хелпер (класс, запись или базовый тип).

2. Объявить хелпер с использованием соответствующего синтаксиса.

3. Реализовать методы и свойства, которые вы хотите добавить.