**8. Основные операторы языка Pascal. Организация циклических процессов.**

**Цикл** - в программировании - оператор языка программирования, позволяющий многократно повторять одну и ту же последовательность команд (тело цикла).

Если цикл имеет известное заранее число повторений, то он называется *циклом со счётчиком*, или *циклом с заданным числом шагов*. Такие циклы в языке программирования Pascal организуются с помощью **оператора цикла с параметром**, имеющего следующий формат:

**FOR <переменная> := n1 TO n2 DO  
BEGIN  
<тело цикла>;  
END;**

Здесь **<переменная>** должна быть перечисляемого типа (целого или символьного), **n1** - её начальное значение, **n2** – конечное значение, и **n1 < n2**. Параметр цикла (**<переменная>**) пробегает последовательно все значения от **n1** до **n2**, возрастая с каждым шагом на 1 (принимая следующее значение). **<тело цикла>** – это набор операторов, которые циклически выполняются. Если **<тело цикла>** состоит только из одного оператора, то операторные скобки **BEGIN - END** не нужны.

Оператор цикла с предварительным условием

**Предусловие** - условие, истинность которого проверяется в начале выполнения тела цикла или процедуры.  
**Оператор цикла с предусловием** применяется, как правило, для организации итерационных циклов и имеет следующий формат:

**WHILE <условие> DO  
BEGIN  
<тело цикла>;  
END;**

Здесь **<тело цикла>** выполняется, если **<условие>**, являющееся логическим выражением, верное (принимает значение **True**). В противном случае тело цикла не выполняется.

Оператор цикла с последующим условием

**Постусловие** - условие, истинность которого проверяется после выполнения тела цикла или процедуры.  
В операторе цикла с постусловием условие следует после тела цикла, а значит, операторы тела цикла выполняются по крайней мере один раз. Формат **оператора цикла с постусловием** следующий:

**REPEAT  
<тело цикла>;  
UNTIL <условие>;**

Работает такой оператор следующим образом: **<тело цикла>** (один или несколько операторов) выполняется до тех пор, пока **<условие>** неверное (имеет значение **False**). Как только **<условие>** станет верным (**True**), происходит выход из цикла и выполняется оператор, стоящий следом за циклом.

**9. Работа со структурированными типами данных. Понятие массив, размерность массива, индекс элемента массива.**

**Массив — структурированный тип данных, состоящий из фиксированного числа однотипных элементов, объединённых одним именем, где каждый элемент имеет свой номер (индекс).**

**Элемент массива —** отдельная переменная, входящая в массив.

**Размерность массива** — количество индексов, по которым определяется положение элемента в массиве.

**Индексы элемента массива** — совокупность номеров, определяющих его местоположение в массиве

**Если непонятно что такое размерность массива, то читайте следующее:**

(Одномерный массив сравним с таблицей, состоящей из одной строки. Для определения положения элемента в строке достаточно знать порядковый номер ячейки, в которой находится элемент. Поэтому в одномерном массиве один индекс — порядковый номер элемента.

Двумерный массив — прямоугольная таблица. Для определения положения элемента в прямоугольной таблице нужно знать порядковый номер строки и столбца, на пересечении которых находится ячейка. Поэтому в двумерном массиве — два индекса, номер строки и номер столбца.

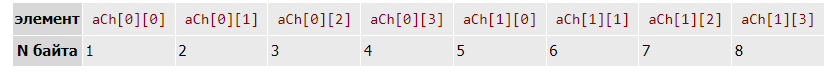
И так, потребность использовать массив возникает всякий раз, когда при решении задачи приходится иметь дело с большим, но конечным количеством однотипных данных, которые необходимо хранить в памяти.)

**10. Работа со структурированными типами данных. Размещение массива в памяти. Способы инициализации массивов. Открытый массив.**

**Размещение элементов массива в памяти выполняется**

При создании массива память под все его элементы выделяется последовательно для каждого элемента в зависимости от типа массива. Для многомерных массивов в первую очередь изменяются значения самого правого индекса.

Например, для массива charaCh[2][4] будет выделено восемь байтов памяти, в которых в следующем порядке будут размещены элементы массива:

****

Двухмерные массивы можно рассматривать как матрицу, в которой первый индекс определяет строку, а второй индекс - столбец. Порядок расположения элементов матрицы в памяти - по строкам.

**Способы инициализации массивов**

Рассмотрим три способа инициализации элементов массива в языке Паскаль.

1 способ

Ввод с клавиатуры абсолютно каждого элемента массива. Например, мы описали массив, состоящий из 10 элементов целочисленного типа данных (допустим, Integer). Затем запускаем цикл по всем элементам и запрашиваем ввод значения с клавиатуры от пользователя.

2 способ

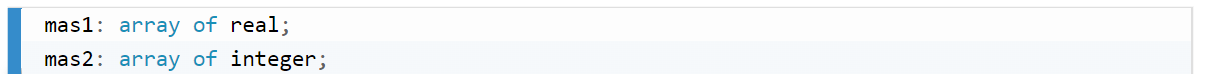
Инициализация массивов Паскаль при помощи генератора случайных чисел. Например, мы описали **массив Pascal**, состоящий из 8 элементов целочисленного типа данных (допустим, Word). Затем запускаем арифметический цикл for-to-do, который сканирует все элементы массива от первого до последнего и случайным образом присваивает число в диапазоне от 1 до 99.

3 способ

Если значения элементов массива определены до начала работы программы. То есть записаны в массив.

**Открытые массивы в Pascal**

При описании *открытого массива* (в разделе type или var) указывается тип элементов, из которых он состоит (например, real, char), но не указываются границы индексов. Например:



В результате получаются как бы безразмерные массивы. Их размер может задаваться и меняться в программе при ее выполнении. Это так называемое динамическое распределение памяти, а переменные открытых массивов представляют собой ничто иное, как указатели на динамически выделяемую область памяти. То есть в переменных открытых массивов будут содержаться адреса начала массива, а не сам массив.

**Особенностью** открытых массивов является то, что их индексы всегда начинаются с нуля (а не с единицы, которая чаще всего используется для обычных массивов).

Доп. инфо.:

Чтобы в программе выделить память под открытый массив, следует воспользоваться процедурой setlength, которая принимает два фактических параметра – имя открытого массива и устанавливаемое количество элементов в нем. В результате работы setlength в памяти выделяется столько байт, сколько необходимо для хранения n-го количества элементов определенного типа. Так, если массив ранее описан как real и задано 5 элементов, то процедура setlength выделит под него 40 байт, т.к. для хранения каждого числа типа real требуется 8 байт памяти (это может зависеть от компилятора).

**11. Работа со структурированными типами данных. Вычисление индекса элемента массива. Удаление и вставка элемента в массив.**

**Вычисление индекса элемента массива**

При нахождении индекса элемента массива, он не должен выходить за границы массива.

**Пример программы с ошибкой массива Паскаля**

**Program** primer \_ error;  
**Type**  
vector=**array** [1..80] **of** word;  
**var**  
n: integer;  
a: vector;  
**begin**  
n:=45;  
a[n\*2]:=25;  
**end**.

Хотя данная программа полностью соответствует синтаксису языка, и транслятор «пропустит» ее, на стадии выполнения произойдет ошибка выхода за пределы массива Паскаля. При n =45 выражение n \*2=90, компьютер сделает попытку обратиться к элементу массива a [90], но такого элемента нет, поскольку описан массив размерностью 80.

**Удаление и вставка элемента в массив**

Для удаления элемента массива по значению нужно выполнить поиск удаляемого элемента (индекса) и выполнить удаление элемента по индексу. В данном случае рассмотрено удаление элемента по индексу.

В результате удаления элемента из массива, при условии, что на вход подано корректное значение индекса элемента (больше нуля и не больше размера массива), должен получиться массив, размер которого на единицу меньше старого. Причем, в новом массиве должен исчезнуть элемент с заданным номером. Такая операция выполняться с помощью сдвига всех элементов, расположенных правее удаляемого на одну позицию влево.

Также можно пользоваться:

(Для расширения массива путем вставки какого-либо элемента, необходимо предусмотреть место в памяти. По этой причине для данных случаев проще использовать списки:  
Списки — List — это тоже динамический массив, который может «расширяться» и «сужаться» по ходу программы (вставка и удаление элементов)).

**Вставка элемента в массив**

Алгоритм решения задачи:

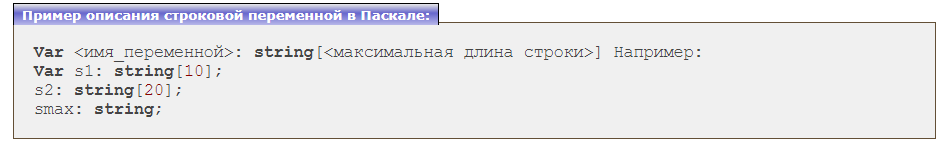
1. Задаем длину массива на один элемент больше, чем он будет заполнен в начале.
2. Выясняем значение и позицию добавляемого элемента
3. Все элементы до указанной позиции сдвигаем на один назад.
4. Присваиваем по указанному индексу (позиции) значение.
5. Остальная (передняя) часть массива не изменяется.



**12.** **Организация работы со строками. Типы строковых переменных в Pascal. Хранение строк в памяти. Строки как частный случай одномерного массива.**

Строковые переменные в Паскале имеют тип String.

В общем виде описание строковой переменной будет выглядеть следующим образом:

****

**Хранение строк в памяти**

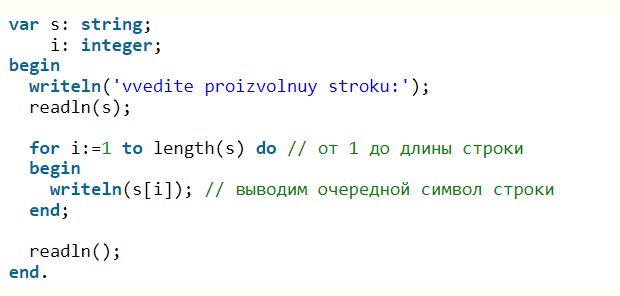
Они используются для хранения последовательностей символов. В Паскале длина стандартной строки ограничена 255 символами. Под каждый символ отводится по одному байту, в котором хранится код символа. Кроме того, каждая строка содержит еще дополнительный байт, в котором хранится длина строки.

**Строки как частный случай одномерного массива**

Обход символов строки в цикле основная идея

Основная идея состоит в том, что строку можно читать посимвольно, обходя её [как одномерный массив](http://fkn.ktu10.com/?q=node/8614) символов.

Сразу же рассмотрим пример, в котором символы полученной от пользователя строки мы выведем по одному, например в столбик:



- здесь следует обратить внимание на то, что символы строки **нумеруются с единицы**, потому что нумерация string начинается с 0. Для определения длины строки (в символах) мы использовали стандартную функцию length(). Строку можно использовать, как одномерный массив.  К любому символу в строке можно обратиться точно так же, как к элементу одномерного массива.

**13. Организация работы со строками. Функции работы со строками на Pascal.**

**Функции для работы со строками в Паскале**

**Copy (S, poz, n)** выделяет из строки S, начиная с позиции poz, подстроку из n символов. Здесь S – любое строковое выражение, poz, n – целочисленные выражения.

**Concat (s1, s2,...,sn)** выполняет слияние строк s1, s2,...,sn в одну строку.

**Length(S)** определяет текущую длину строкового выражения S. Результат – значение целого типа.

**Pos(subS, S)** определяет позицию первого вхождения подстроки subS в строку S. Результат – целое число, равное номеру позиции, где находится первый символ искомой подстроки. Если вхождение подстроки не обнаружено, то результат функции будет равен 0.

**Процедуры для работы со строками в Паскале (Дополнительно)**

**Str(x, S)** преобразует число x в строковый формат. Здесь x – любое числовое выражение, S – строковая переменная. В процедуре есть возможность задавать формат числа x. Например, str(x: 8: 3, S), где 8 – общее число знаков в числе x, а 3 – число знаков после запятой.

**Insert(subS, S, poz)** вставляет в строку S, начиная с позиции poz, подстроку subS. Здесь subS – любое строковое выражение, S – строковая переменная (именно ей будет присвоен результат выполнения процедуры), **poz** – целочисленное выражение.

**Delete (S, poz, n)** удаляет из строки S, начиная с позиции poz, подстроку из n символов. Здесь S – строковая переменная (в данном случае нельзя записать никакое другое строковое выражение, кроме имени строковой переменной, т.к. только с именем переменной связана область памяти, куда будет помещен результат выполнения процедуры); poz, n – любые целочисленные выражения.

**Val(S, x, kod)** преобразует строку символов S в число x. Здесь S – строковое выражение, x – числовая переменная *(именно туда будет помещен результат)*, kod – целочисленная переменная *(типа integer)*, которая равна номеру позиции в строке S, начиная с которой произошла ошибка преобразования, если преобразование прошло без ошибок, то переменная kod равна 0.

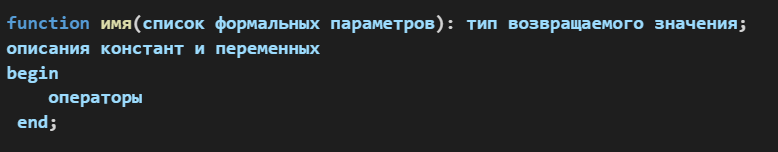
**14. Подпрограммы. Понятие процедуры и функции.**

**Процедура** или функция представляет собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей программы.

***Функция* - подпрограмма, имеющая единственный результат, записываемый в ячейку памяти, имя которой совпадает с именем функции.**

Синтаксис:

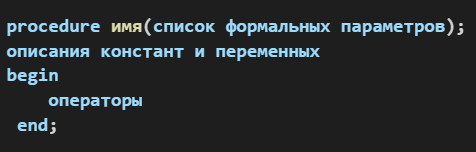
заголовок начинается служебным словом function



Функция ­– подпрограмма, имеющая единственный результат, записываемый в ячейку памяти, имя которой совпадает с именем функции.

Синтаксис:

заголовок начинается служебным словом procedure



Процедура – подпрограмма, имеющая произвольное количество входных и выходных данных.

**15. Подпрограммы. Локальные и глобальные переменные.**

**Локальные** **переменные** **—** **это** **те,** **которые** **описаны** **в** **процедурах** **и** **функциях.** Эти переменные помещаются в стек при вызове подпрограммы и удаляются из него при возвращении в точку вызова. Таким образом, локальные переменные существуют только тогда, когда работает подпрограмма, появляются при вызове и исчезают при завершении работы.

**Глобальная переменная –** это переменная, которая объявляется в главной секции программы. Глобальными переменными называются переменные, которые описаны в самой программе, а используются как в программе, так и в процедуре (функции).