№1. Алгоритм и его свойства. Способы описания алгоритмов. Пример.

Алгоритмизация - сведение задачи к последовательности этапов, выполняемых друг за другом так, что результаты предыдущих этапов используются при выполнении следующих.

Алгоритм - система правил, четко описывающая последовательность действий, которые необходимо выполнить для решения задачи.

Правильно разработанный алгоритм обладает следующими свойствами.

1) Дискретность – значения величин в каждый следующий момент времени должны получаться по определенным правилам из значений величин, имевшихся в предшествующий момент времени. Иными словами, каждый шаг алгоритма имеет четкие границы.

2) Определенность (детерминированность) – каждое правило алгоритма должно быть однозначным. Значения величин, получаемых в какой-то момент времени, однозначно связаны со значениями величин, вычисленных ранее. Иными словами, для всех возможных случаев есть определенное действие

3) Результативность (конечность) – алгоритм должен приводить к решению задачи за конечное число шагов.

4) Массовость – алгоритм должен разрабатываться в общем виде так, чтобы его можно было применить для класса задач, различающихся лишь исходными данными (т е для всех входных) значений.

Существуют следующие способы описания алгоритмов:

1) запись на естественном языке (словесное описание по пунктам);

2) изображение в виде схемы (графическое описание с помощью схемы ГОСТ 19.701-90 – единая система программной документации – Схемы алгоритмов, программ, данных, систем – Условное обозначения и правила выполнения.);

3) запись на алгоритмическом языке (составление программы, исходный код программы тоже является законченным описанием алгоритма).

Пример: Нахождение максимального элемента каждой строки в двумерном массиве и вывести их сумму.

№2. Графическое представление алгоритмов по ГОСТ 19.701-90. Пример

Схема алгоритма – графическое представление алгоритма, в котором этапы процесса обработки информации и носители информации представлены в виде геометрических символов, а последовательность процесса отражена направлением линий.

ГОСТ 19.701-90 – Единая система программной документации – Схемы алгоритмов, программ, данных и систем – Условные обозначения и правила выполнения.

Виды схем:

1. Схема данных.
2. Схема программы (практически равно схеме алгоритма, только разный уровень детализации).
3. Схема работы системы.
4. Схема взаимодействия программ.
5. Схема ресурсов системы.

Символы, регламентируемые стандартом, подразделяются на следующие группы:

1) символы данных (например, ввод в виде параллелограмма, когда неважно, как мы вводим элементы);

2) символы процесса, отображающие функцию обработки данных любого вида (например, символы процесс и решение, а также символ граница цикла);

3) символы линий (обычная линия, линия со стрелкой, пунктирная линия);

4) специальные символы (соединитель, комментарий, пропуск).

Каждая из трех первых групп в свою очередь подразделяется на две подгруппы:

• Основные символы;

• Специфические символы.

Нужно помнить, что символы должны располагаться равномерно, должно содержаться минимальное число линий. Нельзя изменять форму фигур, а только их размер. Следует помещать внутрь символа как можно меньше текста, если текст не помещается, использовать комментарии. Линии должны входить слева и сверху, а выходить снизу и справа.

Пример: Нахождение максимального элемента каждой строки в двумерном массиве и вывести их сумму.

№ 3. Разновидности структур алгоритмов. Виды циклов. Пример.

Различают следующие структуры алгоритмов:

1. Линейные процессы - это процесс, в котором направление вычислений является единственным.

Таковым является вычисление формулы или, например, многочлена от x. То есть ход вычислений не зависит от условий.

1. Разветвляющиеся процесс – это процесс, в котором направление вычислений определяется некоторыми условиями. Например, если путь, по которому нужно пойти зависит от знака переменной, которую вводит пользователь, то можно использовать оператор принятия двоичного решения if
2. Циклический вычислительный процесс – процесс, в котором отдельные участки вычислений выполняются многократно. Этот участок называется циклом.

В соответствии со взаимным расположением циклов в теле программы или алгоритма различают следующие циклы:

1) простые – циклы, не содержащие внутри себя других циклов;

2) сложные – циклы, содержащие внутри себя другие циклы;

3) вложенные (внутренние) – циклы, входящие в состав других циклов (цикл в цикле);

4) внешние – циклы, не являющиеся составной частью других циклов, но содержащие в своем составе внутренние циклы.

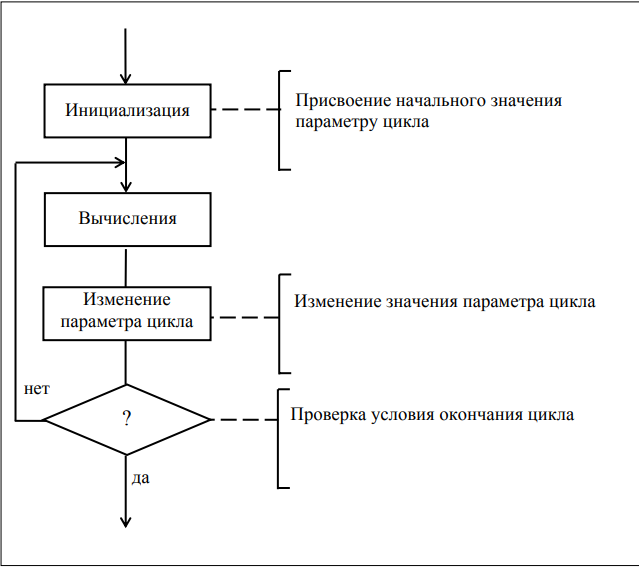
В зависимости от местоположения условия выполнения цикла различают следующие циклы:

1) циклы с предусловием (инициализация, проверка выполнения цикла, если да, то тело цикла);

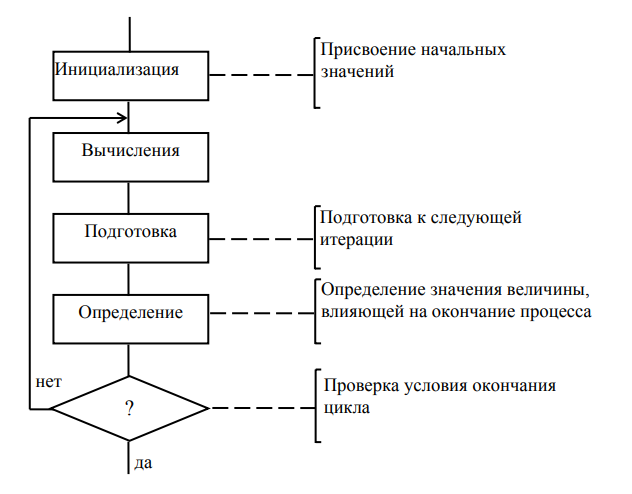
2) циклы с постусловием (инициализация, тело цикла: вычисления и изменение параметра цикла, проверка условия окончания цикла).

В соответствии с видом условия выполнения циклы делятся на следующие виды:

1) циклы с параметром (также называется счетчиком цикла);



2) итерационные циклы (циклы, где заранее не определено число повторений, а количество этих повторений зависит от результатов вычислений. Примером может быть сходящийся ряд).



Часть цикла, которая выполняется многократно – тело цикла.

Пример: Нахождение максимального элемента каждой строки в двумерном массиве и вывести их сумму.

№ 4. Основные положения теории структурного программирования. Пример.

К концепциям структурного программирования относится отказ от использования оператора безусловного перехода (GoTo), замена его рядом других структурированных операторов и использование идей нисходящего проектирования программ. Нисходящее проектирование предполагает конструирование общей структуры алгоритма, без детальной проработки, а только на следующих шагах детально описывать недетализированные блоки. В основу структурного программирования положено требование, чтобы каждый модуль алгоритма (программы) проектировался с единственным входом и единственным выходом. Программа представляется в виде множества вложенных модулей, каждый из которых имеет один вход и один выход.

Базой для реализации структурированных программ является принцип Бома и Джакопини, в соответствии с которым всякая реальная программа может быть построена с использованием лишь двух управляющих конструкций.

По Бому и Джакопини логическая структура программы может быть выражена комбинациями трех базовых структур:

1) Функциональный блок (отдельный вычислительный оператор или последовательность вычислений с 1 входом и 1 выходом, которая изображается с помощью символа «процесс»)

2) Конструкция принятия двоичного (дихотомического) решения (обычно называется элементом If-Then-Else (если-то-иначе), разветвлением или ветвлением – структура, обеспечивающая выбор между двумя альтернативными путями вычислительного процесса в зависимости от выполнения некоторого условия. Изображается с помощью блока решения и двух блоков «процесс»).

3) Конструкции обобщенного цикла (в качестве базовой конструкции структурного программирования используется цикл с предусловием, называемый циклом «Пока» (Do-While). Изображается с помощью блока решение и блока «процесс» идущий вниз в ветви да, также содержащий тело цикла).

Конструкция следования(несколько блоков «процесс») может быть также объеденена в 1 блок «процесс». Таким образом всякая программа, состоящая из функциональных блоков, конструкций обобщенных циклов и конструкций принятия двоичного решения может быть сведена к 1 функциональному блоку. Эти преобразования носят названия преобразования Бома-Джакопини и являются доказательством структурированности программ.

Пример: выделение функциональных блоков, нахождение максимального элемента каждой строки в двумерном массиве и вывести их сумму.

№ 5. Реализация теоретических основ структурного программирования в современных языках программирования. Достоинства структурного программирования. Пример.

Реализация теоретических основ структурного программирования базируется на следующих правилах. Все операции в программе должны представлять собой либо непосредственно исполняемые в линейном порядке выражения, либо одну из следующих управляющих конструкций:

1) вызовы подпрограмм – любое допустимое на конкретном языке программирования обращение к замкнутой подпрограмме с одним входом и одним выходом;

2) вложенные на произвольную глубину операторы If-Then-Else;

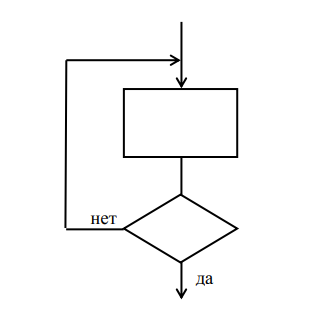
3) циклические операторы (цикл с предусловием, называемый циклом «Пока»).

Этих средств достаточно для составления структурированных программ. Однако иногда допускаются их некоторые расширения:

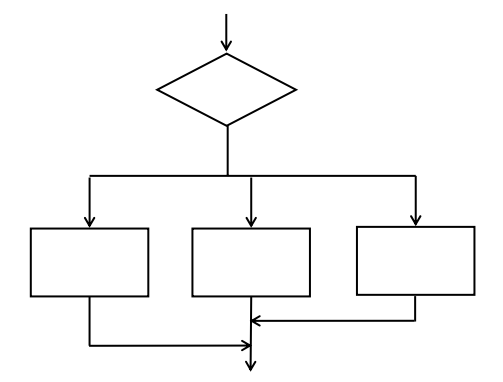
1. дополнительные конструкции организации цикла:

• цикл с параметром как вариант цикла с предусловием;

• цикл с постусловием, называемый в структурном программировании циклом «До», – для него характерно то, что тело цикла выполняется до проверки условия выхода из цикла;



1. подпрограммы с несколькими входами и несколькими выходами (например, один выход нормальный, второй – по ошибке);
2. применение оператора GoTo с жёсткими ограничениями (например, передача управления не далее, чем на десять операторов, или только вперёд по программе);
3. использование оператора Case как расширения конструкции If-ThenElse; в структурном проектировании программ конструкция Case представляется так:



Достоинства структурного программирования по сравнению с интуитивным неструктурным программированием: 1) Уменьшение трудностей тестирования программ. 2) Более высокая производительность программистов. 3) Ясность и читаемость программ, что упрощает их сопровождение. 4) Эффективность программ.

№ 6. Преобразование неструктурированных программ в структурированные. Метод дублирования кодов. Достоинства и недостатки метода. Пример.

В общем случае произвольная программа не может быть преобразована в структурированную программу, которая реализует тот же алгоритм, построена с применением тех же конструкций и не использует дополнительных переменных. Такое преобразование возможно при использовании трех известных методов:

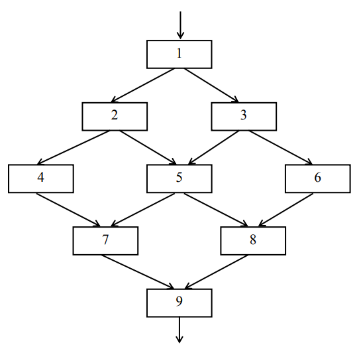
• дублирование кодов программы;

• введение переменной состояния;

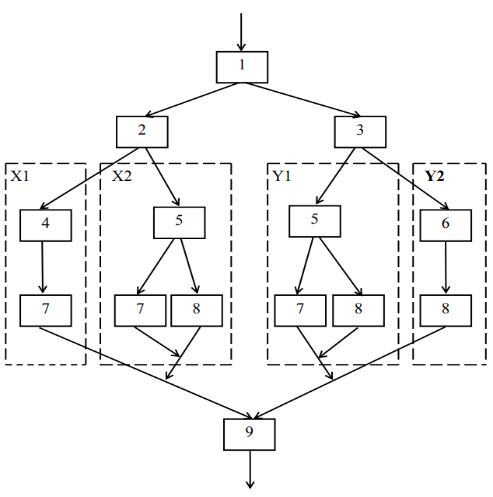
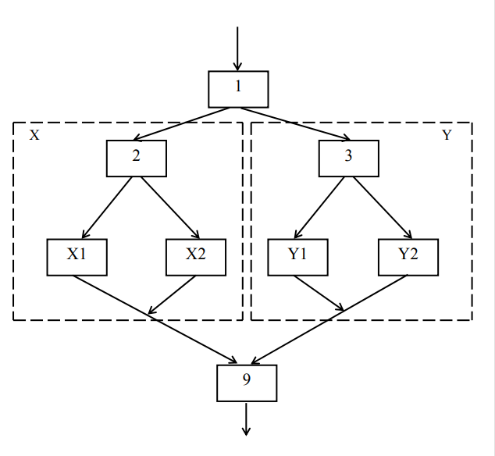
• метод булевых признаков.

Рассмотрим применение данного метода на примере неструктурированной программы типа «решетка».

Программа, реализующая данный алгоритм, не является структурированной, так как не удовлетворяет условию «один вход – один выход».

Сущность метода дублирования кодов: дублируются те модули исходного алгоритма или программы, в которые можно войти из нескольких мест (кроме последнего блока). В соответствии с этим в исходной схеме необходимо дублировать модули 5 (в него можно войти из модулей 2 и 3), 7 (в него можно войти из модулей 4 и 5) и 8 (в него можно войти из модулей 5 и 6). Может понадобиться не дублировать, а даже утроить какой-то блок (например блок 8 и 7). Нужно также передвинуть стрелки, которые имеют 1 и тот же выход в 1 направление(см. рисунок).

На вышеприведенной схеме модули 4 и 7 представляют собой конструкцию следования. В соответствии с преобразованиями БомаДжакопини они могут быть сведены к одному функциональному блоку Х1. Модули 5, 7, 8 представляют собой конструкцию If-Then-Else с одним входом и одним выходом. Они также могут быть преобразованы к одному функциональному блоку (Х2 и Y1). Аналогичные рассуждения справедливы для конструкции следования, состоящей из модулей 6, 8 – она сводится к функциональному блоку Y2.

 В рузкльтате получаем схему, где, с помощью преобразований Бома-Джакопини можно получить 1 функциональный блок

Достоинством метода дублирования кодов является то, что его удобно использовать при нисходящем проектировании программ. Исходную задачу укрупненно можно представить в виде одного функционального блока, а затем ее постепенно разукрупнять через промежуточные схемы алгоритма к результирующей структурированной схеме. Недостатки метода дублирования кодов: 1) неприменимость к программам с циклами; 2) дополнительные затраты памяти для хранения дублируемых модулей. Поэтому метод используется, если дублируемые модули содержат незначительное число операторов. Если модули велики, то вместо дублирования кодов необходимо использовать вызываемые подпрограммы с формальными параметрами.

№ 7. Преобразование неструктурированных программ в структурированные. Метод введения переменной состояния. Достоинства и недостатки метода. Пример.

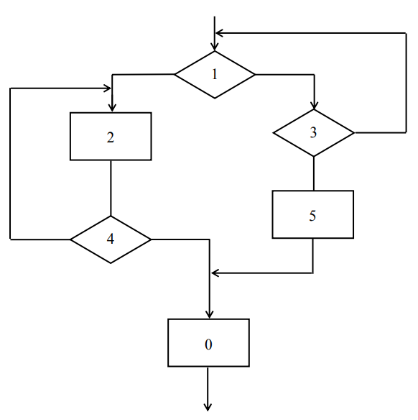
В общем случае произвольная программа не может быть преобразована в структурированную программу, которая реализует тот же алгоритм, построена с применением тех же конструкций и не использует дополнительных переменных. Такое преобразование возможно при использовании трех известных методов:

• дублирование кодов программы;

• введение переменной состояния;

• метод булевых признаков.

Рассмотрим применение данного метода на примере неструктурированной программы, алгоритм которой схематично представляет рисунок:

Данная схема не является структурированной, так как из цикла, состоящего из блоков 1 и 3, существует два выхода. Таким образом, нарушено условие «один вход – один выход», которому должны удовлетворять структурированные схемы.

Процесс преобразования программы в структурированную состоит из следующей последовательности шагов.

1) Каждому блоку неструктурированной схемы приписывается номер. Обычно первому блоку присваивается 1, последнему – 0. 1 2 0 5 3 4 59

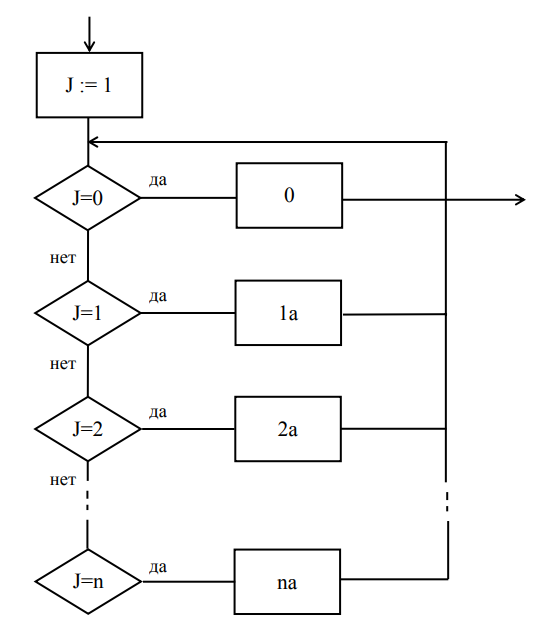
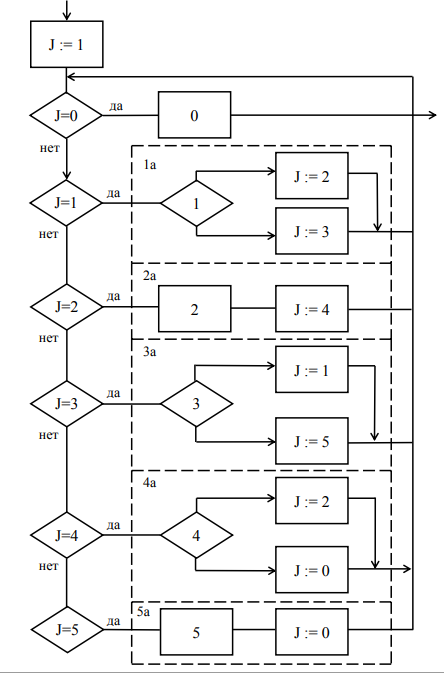
2) В программу вводится дополнительная переменная целого типа (например, J), называемая переменной состояния.

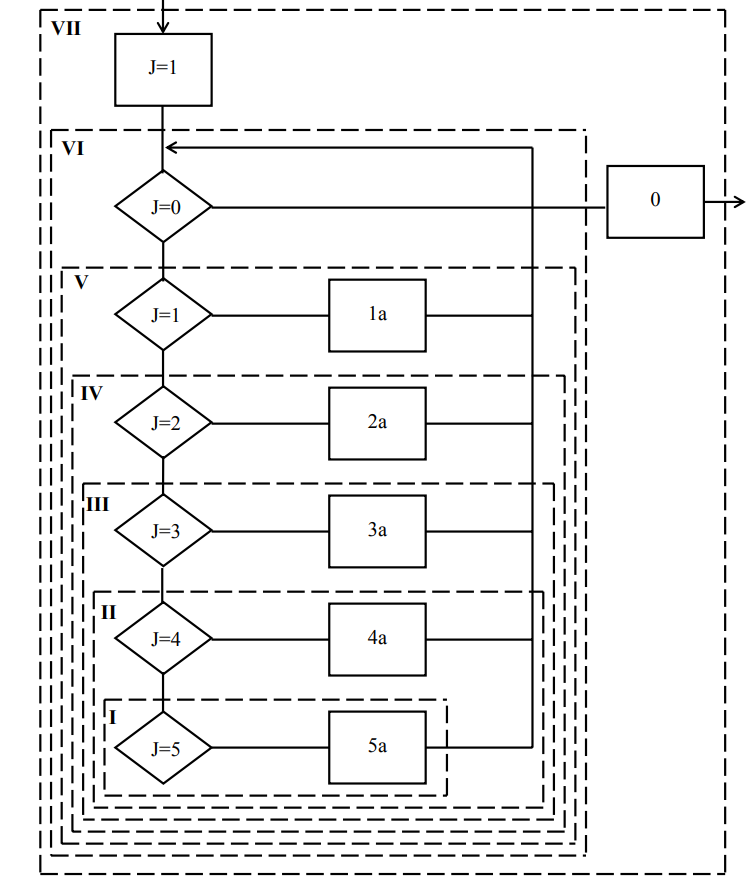
3) Функциональные блоки исходной схемы заменяются блоками, выполняющими помимо основных функций преобразование переменной J: переменной J присваивается значение, равное номеру блока-приёмника в исходной схеме.

4) Аналогично преобразуются логические блоки. При этом, если в логическом блоке условие истинно, то это соответствует одному значению J, если ложно – другому.

5) Исходная схема перестраивается к виду, предложенному АшкрофтомМанной.

При выполнении алгоритма, реализованного по методу АшкрофтаМанны, переменная состояния J устанавливается в начальное значение, равное номеру первого блока непреобразованной схемы (как правило, это единица). Затем осуществляется последовательный опрос переменной J, начиная с нуля и заканчивая максимальным номером блока исходной схемы (в нашем примере он равен пяти). Выполняется тот блок исходной схемы, номер которого соответствует текущему значению J. Помимо этого в J заносится значение, равное номеру того блока исходной схемы, который должен выполняться за текущим блоком. Когда значение J станет равно нулю, выполняется последний блок непреобразованной схемы (блок с номером ноль) и осуществляется выход из алгоритма.

Полученная по методу Ашкрофта-Манны схема алгоритма является структурированной. Для доказательства этого достаточно последовательно преобразовать данную схему к одному функциональному блоку.



Достоинства метода введения переменной состояния:

1) процесс преобразования программы отличается наглядностью и чёткостью; 2) любому блоку исходной схемы соответствует определённое состояние программы, что помогает выполнять тестирование и отладку программы; 3) метод применим к программам любой структуры (разветвляющимся и циклическим); 4) возможно автоматическое применение данного метода. Недостатки метода:

1) структурированная форма схемы алгоритма сильно отличается от топологии исходной схемы, что затрудняет ее понимание; 2) дополнительные затраты времени на анализ и установку значений переменной сосотояния; 3) громоздкость результирующей схемы.

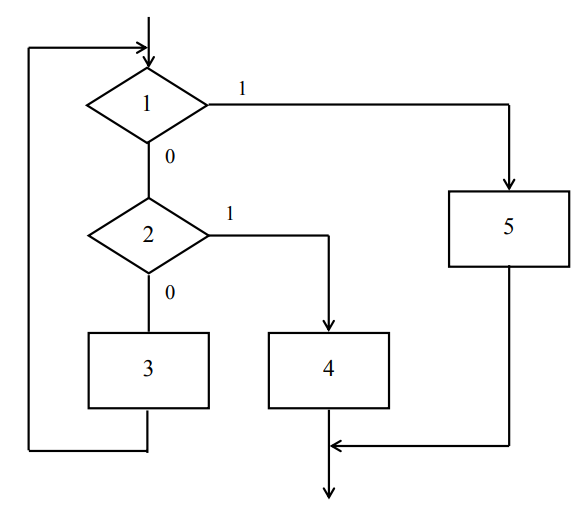
№ 8. Преобразование неструктурированных программ в структурированные. Метод булевого признака. Достоинства и недостатки метода. Пример.  
В общем случае произвольная программа не может быть преобразована в структурированную программу, которая реализует тот же алгоритм, построена с применением тех же конструкций и не использует дополнительных переменных. Такое преобразование возможно при использовании трех известных методов:

• дублирование кодов программы;

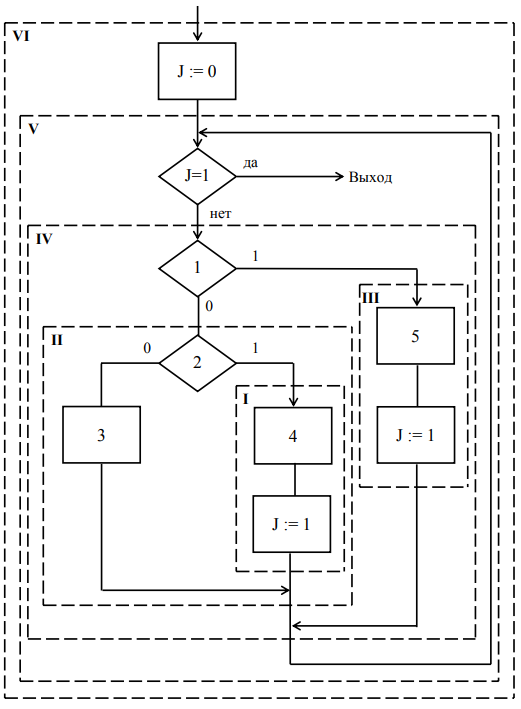
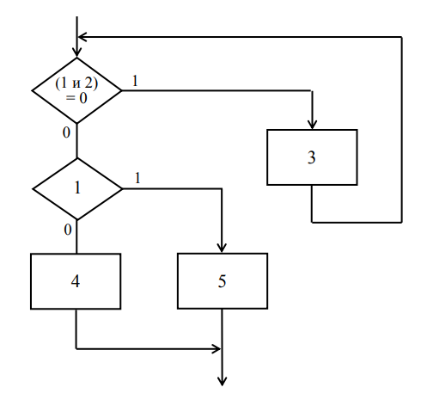
• введение переменной состояния;

• метод булевых признаков.

Сущность метода булевого признака заключается в следующем. В программу, содержащую циклы, вводится некоторый признак. Начальное значение признака задаётся до цикла. Цикл выполняется, пока признак сохраняет своё исходное значение. Значение признака изменяется при наличии некоторых условий внутри цикла. Рассмотрим применение метода булевого признака на примере преобразования неструктурированной схемы, которую представляет рисунок

Схема алгоритма не является структурированной, поскольку входящий в нее цикл (блоки 1, 2, 3) содержит один вход и два выхода.

В соответствии с рассматриваемым методом в исходную схему вводится признак (например, J). Схема приобретает структурированный вид и легко реализуется конструкциями обобщенного цикла и принятия двоичного решения

Иногда можно обойтись без специального признака, используя те условия, которые уже есть в исходной схеме.

Результирующая схема содержит обобщенный цикл с одним входом и одним выходом и конструкцию If-Then-Else, то есть является структурированной.

Следует иметь в виду, что преобразование схемы к структурированному виду без дополнительного признака возможно и эффективно только при небольшом количестве условий.

Достоинства метода булевого признака: 1) компактность, экономичность; 2) топология исходной схемы изменяется незначительно.

Недостаток метода булевого признака: метод предназначен для использования только в циклах.

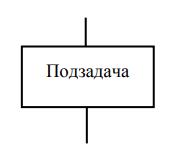
№ 9. Метод Дамке представления схем алгоритмов. Пример.

Многие специалисты в области теории программирования считают, что графическое представление алгоритмов в соответствии с ГОСТ 19.701-90 (ISO 5807-85, см. п. 2.2.2) скрывают структуру структурированной программы. Поэтому для представления структурированных схем алгоритмов были предложены специальные методы графических обозначений.

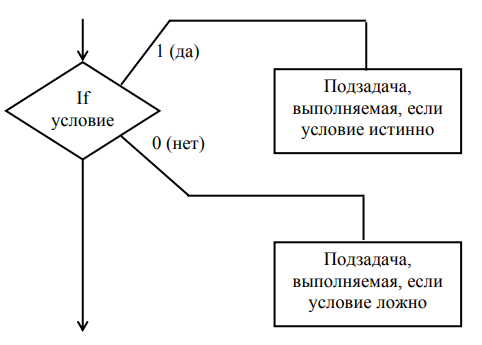
Основным принципом при разработке структурированных схем алгоритмов по методу Дамке является принцип декомпозиции. Он означает, что любой элемент, представляющий собой задачу, можно разделить на несколько элементов, образующих необходимые подзадачи. Элементы в самой левой части схемы представляют укрупнённую структуру алгоритма. Затем элементы расширяются вправо по мере разделения каждого элемента на подзадачи. Чтобы исследовать любую подзадачу, достаточно анализировать только те элементы и управляющие структуры, которые находятся справа от данной подзадачи.

Три основных конструкции структурного программирования изображаются следующим образом.

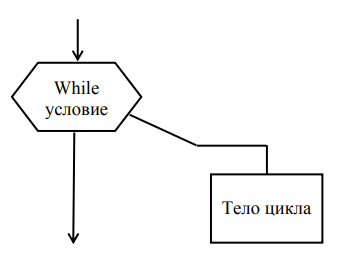
1. Функциональный блок по-прежнему обозначается прямоугольником



1. Конструкция If-Then-Else изображается так

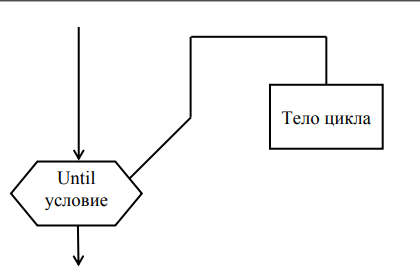


1. Конструкция Do-While (цикл с предусловием “Пока”) изображается так

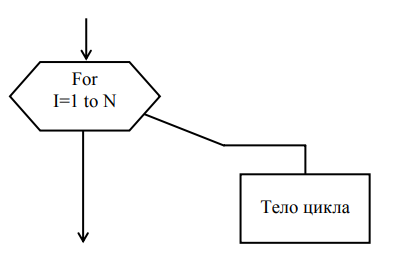


Помимо трех основных конструкций в структурном программировании допускаются дополнительные конструкции. В методе Дамке они изображаются следующим образом.

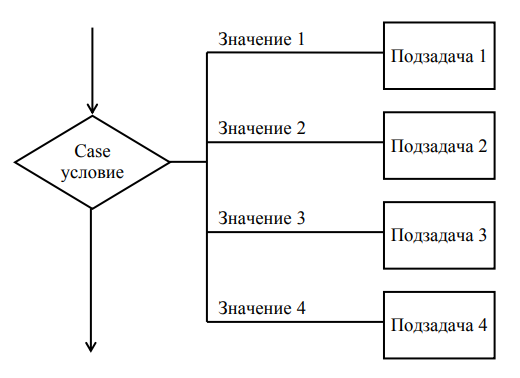
1. Конструкция Repeat-Until (цикл с постусловием “До”) изображается так



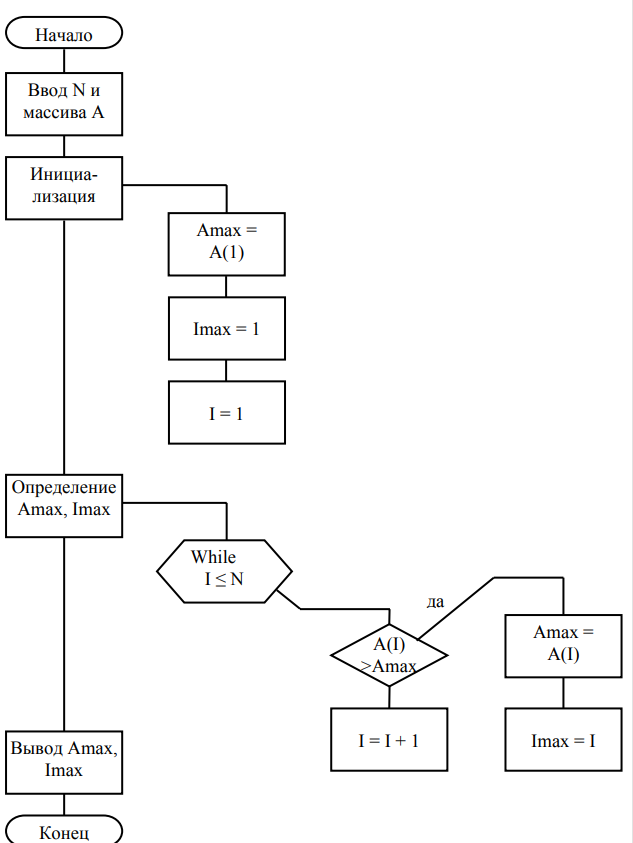
1. Конструкция цикла с параметром изображается так



1. Конструкция Case изображается так



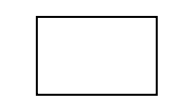
Пример: Дан массив А, состоящий из N элементов. Найти наибольший из элементов массива (Amax) и его номер (Imax).



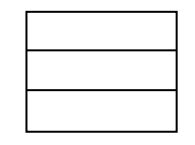
№ 10. Схемы Насси-Шнейдермана. Пример.

Схемы Насси-Шнейдермана – это схемы, иллюстрирующие структуру передач управления внутри модуля с помощью вложенных друг в друга блоков. Схемы используются для изображения структурированных схем и позволяют уменьшить громоздкость схем за счёт отсутствия явного указания линий перехода по управлению. Изображение основных элементов структурного программирования в схемах Насси-Шнейдермана организовано следующим образом. Каждый блок имеет форму прямоугольника и может быть вписан в любой внутренний прямоугольник любого другого блока.

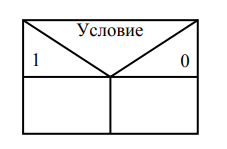
1. Функциональный блок (блок обработки) изображается так



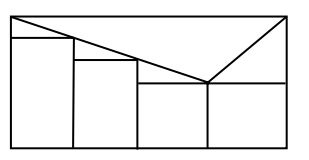
1. Блок следования изображается так



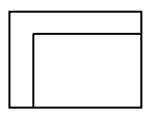
1. Блок решения изображается так (вместо 1 и 0 могут быть слова да и нет)



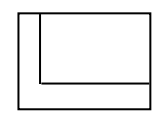
1. Блок Case изображается так(правая ветка – ветка else)



1. Цикл “Пока” изображается так

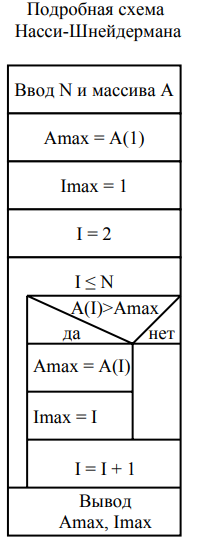


1. Цикл “До” изображается так



Из данных рисунков видно, что схемы Насси-Шнейдермана являются существенно более компактными по сравнению с аналогичными схемами алгоритмов, представленными в соответствии с требованиями ГОСТ 19.701-90 (см. п. 2.2.2). Очевидно, что связано это с отсутствием линий, отображающих потоки управления (линии перехода по управлению) между блоками. Нужно также отметить, что параметр(для цикла с параметром) инициализируется функциональным блоком, а затем только записывается сам цикл(как для цикла while). Приветствуется также выравнивание текста в блоках слева.

Пример: Дан массив А, состоящий из N элементов. Найти наибольший из элементов массива (Amax) и его номер (Imax).



№ 11. Сложность алгоритма. Оценка сложности алгоритмов. Пример.

Одна и та же задача может быть решена несколькими способами. В таком случае может появится и несколько алгоритмов, которое будут отличаться друг от друга.

Алгоритм может быть оценен:

1. По количеству выполняемых операций
2. По объему потребляемой памяти
3. По использованию других ресурсов

Оценка может быть:

1. Минимальной (лучший случай)
2. Средней
3. Максимальной (худший случай)

Примеры:

О(n) – линейная сложность

Такой сложностью обладает, например, алгоритм поиска минимального или максимального элемента в неотсортированном массиве. Нам придется пройтись по всем n элементам массива, чтобы понять, какой элемент является искомым

О(log n) - логарифмическая сложность

Простейший пример – бинарный поиск. Если массив отсортирован, мы можем проверить, есть ли в нем конкретное значение делением пополам. Проверяем средний элемент, если он больше искомого, отбрасываем всю правую часть, аналогично и для левой части. И так будем делить пополам, в итоге проверим log n элементов.

О(n^2) – квадратичная сложность

Такую сложность имеет, например, алгоритм сортировки вставками. В обычной его реализации имеем 2 цикла. Один служит, чтобы проходить по всему массиву элементов, а второй, чтобы находить место элементу и сдвигать элементы влево или вправо. Таким образом количество операций будет зависеть от размера массива n\*n, то есть n^2.

№ 12. Этапы постановки и решения задачи на компьютере. Методы автоматизации программирования. Назначение и классификация языков программирования.

Можно выделить следующие этапы постановки и решения задачи на компьютере:

1) четкая формулировка задачи, выделение исходных данных и формы представления результатов;

2) формальная (математическая) постановка задачи – представление ее в виде уравнений, соотношений, ограничений;

3) выбор метода решения; метод решения определяется решаемой задачей;

4) разработка алгоритма решения задачи;

5) выбор структуры данных; от выбора способа представления данных зависит способ их обработки; поэтому этапы 4) и 5) взаимосвязаны;

6) собственно программирование (запись разработанного алгоритма на языке программирования);

7) тестирование и отладка программы (проверка правильности работы программы и исправление обнаруженных ошибок);

8) выполнение программы на компьютере.

Для того, чтобы программа была понятна компьютеру, она должна быть составленной из последовательности элементарных операций, представленных на машинном языке – в виде совокупности нулей и единиц, т.е. в так называемых машинных кодах.

К основным методам автоматизации программирования можно отнести следующие.

1) Использование языков высокого уровня, близких к естественному человеческому языку, позволяющих автоматически однозначно преобразовывать написанную на них программу в программу на языке машины, т.е. в машинные коды.

2) Создание и использование библиотек стандартных программ и подпрограмм, предназначенных для реализации часто используемых задач.

3) Использование современных технологий программирования.

4) Использование Case-средств, предназначенных для автоматизации процесса разработки программ.

Основная идея автоматизации программирования заключается в отказе от написания программ непосредственно в машинных кодах. Программа пишется на некотором входном языке. Входные языки делятся на следующие группы:

1) машинно-ориентированные языки (Языки низкого уровня, одной команде кода равна 1 машинная команда) ;

2) процедурно-ориентированные языки (предназначались для решения конкретного класса задач – инженерных, научно-технических, обработки экономической информации, обработки списков, моделирования и т.д. Си, фортран, паскаль, Бейсик и тд);

3) объектно-ориентированные языки (Впервые введено понятие объекта. Си++ и ряд других) ;

4) проблемно-ориентированные языки (Исключают работу программиста по разработке алгоритмов решаемых задач. Report Program Generator – генератор отчетов);

5) языки четвертого поколения (4GL) (Языки визуального программирования. Автоматически генерируют исходный текст программ целиком или в виде отдельных фрагментов. К данной группе языков можно отнести, например, визуальные среды программирования Delphi, Builder C++)

№ 13. Структура программного обеспечения. Системы программирования. Назначение, состав

Под системой программного обеспечения (СПО) подразумевают совокупность специальных программ, облегчающих процесс изготовления программ и обеспечивающих процесс их выполнения на компьютере, и связанную с ними документацию. Программное обеспечение (ПО) по функциям и задачам, выполняемым его программами, можно разделить на две группы: 1) специализированное ПО; 2) стандартное ПО. Специализированное ПО состоит из прикладных программ, предназначенных для решения некоторых самостоятельных задач, достаточно часто встречающихся (например, задачи начисления заработной платы, решения систем уравнений и т.п.). К стандартному ПО относятся системы программирования и операционные системы. Система программирования – совокупность программ, описаний и инструкций, предназначенных для автоматизации процесса разработки программ. Операционная система – совокупность программ, описаний и инструкций, предназначенных для организации и контроля выполнения программ на компьютере. Программы, входящие в стандартное программное обеспечение, называются системными программами.

Основное назначение систем программирования – максимально облегчить процесс общения программиста с компьютером, освободить его от необходимости описания алгоритма на машинном языке, предоставить возможность использования языка высокого уровня. Состав системы программирования:

1) входной язык системы (Язык, на котором пишется исходный код программы);

2) транслятор с входного языка на машинный язык (Компиляторы и интерпретаторы);

3) редактор связей (В современных ЯП, программа состоит из модулей);

4) библиотеки программ( Библиотека стандартных программ содержит готовые решения определенных задач);

5) средства отладки (Основная цель этапа отладки – выявление и исправление ошибок в программе. Процесс отладки состоит из многократных попыток выполнения программы на компьютере и анализа получившихся результатов);

6) обслуживающие (сервисные) программы (Вспомогательные программы);

7) документация.

№ 14. Общая характеристика языка Delphi. Достоинства языка Delphi.

Delphi - структурированный, объектно-ориентированный язык программирования со строгой типизацией переменных.

История Delphi начинается с 60-х гг., когда профессор Н.Вирт разработал язык высокого уровня Pascal. Это был лучший язык для изучения программирования, и для создания программ для операционной системы MS-DOS. Затем, в 1983 г., А. Хейлсберг совместно с другими программистами, которые только что организовали компанию Borland, разработал компилятор Turbo Pascal, который стал следующим шагом в эволюции Delphi.  Затем появился Object Pascal, который уже использовал Объектно-Ориентированный подход к программированию.

В основе среды Delphi лежит одноименный язык программирования — Delphi, ранее известный как Object Pascal. При разработке программы среда Delphi выполняет свою часть работы — создает пользовательский интерфейс согласно вашему дизайну, а вы выполняете свою часть — пишите обработчики событий на языке Delphi.

Достоинства языка :

1) относительная простота (т.к. разрабатывался с целью обучения программированию);

2) идеология языка близка к современным методикам и технологиям программирования, в частности, к структурному программированию и нисходящему проектированию (методу пошаговой детализации) программ. Delphi может использоваться для записи программы на различных уровнях ее детализации, не прибегая к помощи схем алгоритмов;

3) гибкие возможности в отношении используемых структур данных;

4) высокая эффективность программ;

5) наличие средств повышения надежности программ, включающих контроль правильности использования данных различных типов и программных элементов на этапах трансляции, редактирования и выполнения.

№ 15. Алфавит языка Delphi. Классификация символов. Пример.

Алфавит языка делится на три группы:

1. Буквы:

В данную группу входят прописные (заглавные) буквы латинского алфавита от A до Z, строчные буквы латинского алфавита от a до z и символ подчеркивания ( \_ ). В программах строчные латинские буквы эквивалентны прописным везде, за исключением литералов (строковых констант).

1. Цифры:

В данную группу входят десять десятичных цифр от 0 до 9. При записи программы на бумаге принято перечеркивать цифру 0, чтобы отличить ее от буквы O, которая не перечеркивается.

1. Специальные символы:

Специальные символы делятся на две подгруппы.

**Первую подгруппу** составляют простые и составные специальные символы.

Простые:

* символы ограничители:

1. Знаки арифметических операций(+,-,\*,/)
2. Знаки операций сравнения(<,>,=)
3. Знаки разделители (точка, запятая, точка с запятой, двоеточие, апостроф)
4. Знаки скобок( (),{},[] )

* Остальные символы(@,#,$,^(знак карата)), пробел)

Составные специальные: комбинация символов с опр. значением. Также являются символами ограничителями

<= меньше или равно;

>= больше или равно;

<> не равно;

:= операция присваивания;

.. обозначение диапазона.

Кроме того, имеются составные символы, являющиеся аналогами специальных символов скобок: (\* \*) аналоги { }; (. .) аналоги [ ].

В языке Паскаль используются также синтаксическое понятие «шестнадцатеричная цифра». Шестнадцатеричными цифрами являются десятичные цифры и буквы A, B, C, D, E, F (или a, b, c, d, e, f). Обозначения шестнадцатеричных констант начинаются со знака $. Например, $F – это число 15 в десятичной системе счисления. Шестнадцатеричные цифры относятся к составным символам.

**Вторая подгруппа** специальных символов состоит из служебных слов. Служебные слова – это определенные комбинации латинских букв, имеющие в языке строго фиксированный смысл. Эти слова нельзя применять для других целей. Поэтому их еще называют зарезервированными словами. В языке Паскаль имеется большое количество служебных слов.

К ним относятся: Absolute And Asm Array Begin Break Case Const Constructor Continue Destructor Div Do Downto Else End Exit External File For Forward Function Goto If Implementation In Inline Interface Interrupt Label Mod Nil Not Object Of Or Packed Procedure Program Record Repeat Set Shl Shr String Then To Type Unit Until Uses Var Virtual While With Xor.

№ 16. Основные понятия языка Delphi. Лексемы и их типы. Идентификаторы. Комментарии. Понятие оператора. Типы операторов. Пример.

Текст программы, написанной на языке Паскаль, состоит из лексем, комментариев и пробелов. В программировании лексемой (или лексической единицей) называют неделимую последовательность знаков алфавита, имеющую в программе определенный смысл. В Паскале имеются следующие типы лексем:

1) простые и составные специальные символы;

2) идентификаторы;

3) литералы (строковые и символьные константы);

4) числовые константы;

Последовательность символов, используемая для обозначения какого-либо элемента программы или его атрибутов, называется идентификатором. В Делфи идентификатором является любая последовательность букв, цифр и знаков подчеркивания, начинающаяся буквой или знаком подчеркивания. Никакие другие символы в идентификаторах использовать нельзя. Идентификаторы, обозначающие элементы внутри программы, называются внутренними. В Делфи максимальная длина таких идентификаторов не должна превышать 126 символов, т.е. ограничена только длиной программной строки. Идентификаторы могут использоваться также для связи независимо транслированных частей программы между собой и с операционной системой. Такие идентификаторы называются внешними. В Делфи внешние идентификаторы не должны иметь более 8 символов (лишние символы справа обрежутся).

Идентификаторы делятся на две группы: 1) предопределенные идентификаторы; 2) идентификаторы, определенные программистом. Предопределенный (стандартный) идентификатор – это идентификатор, имеющий стандартный смысл и входящий в описание языка. Предопределенные идентификаторы не являются служебными словами (например integer).

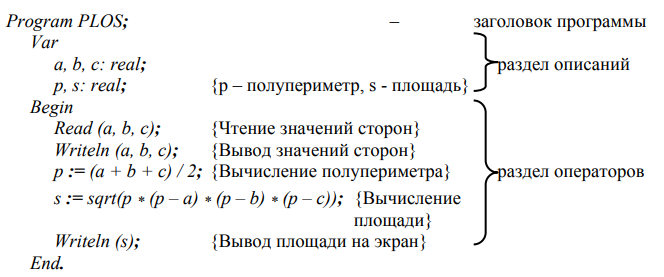
Идентификатор, определенный программистом – это идентификатор, смысл которого определен непосредственно в программе. Такие идентификаторы обычно задают имена некоторых элементов программы. Например, Х – имя переменной.

Комментарий служит для внесения пояснений в программу и может размещаться везде, где допускается пробел. В качестве комментария может использоваться любая последовательность символов, включая русские буквы, за исключением символов-ограничителей комментариев. В языке Паскаль комментарий ограничен специальными символами. Комментарии игнорируются транслятором и не оказывают влияния на решение задачи

К основным видам комментариев относятся: • вводный комментарий – это комментарий, записываемый перед текстом программы, в котором даются общие сведения о программе (например, назначение программы, используемые методы вычислений, длительность работы программы, необходимые ресурсы памяти, даты разработки и последнего обновления, авторы и т.д.); • комментарии-заголовки – это комментарии, записываемые перед подпрограммами (или большими блоками программы), в которых описывается назначение подпрограмм (блоков программы), выполняемые в них действия, методы, положенные в основу их работы и т.д.; нормы комментариев-заголовков: 4 – 5 строк на подпрограмму; • строчные комментарии – это комментарии, описывающие мелкие фрагменты программ.

Программа, написанная на языке Делфи, состоит из трех основных разделов: 1) заголовок программы; 2) раздел описаний (объявлений); 3) раздел операторов. Второй и третий разделы составляют тело программы (блок).

Одним из основных компонентов программы является оператор. Оператор – это команда, указывающая, какие действия на данном этапе должны быть выполнены машиной. Операторы в языке Делфи могут быть разделены на две группы: основные (простые) и производные. Основные операторы – операторы, не содержащие в своем составе других операторов. Например, оператор присваивания. Производные операторы – операторы, в состав которых входят другие операторы, например, операторы цикла, составной оператор. Признаком конца оператора в последовательности операторов является точка с запятой. Признаком конца тела программы является точка.



№ 17. Способы описания синтаксиса языков программирования. РБНФ. Пример.

Синтаксис – это набор правил и соглашений, описывающих правильные предложения языка. Для записи правил синтаксиса языков программирования применяют различные формализованные системы обозначений, называемые метаязыками. Мы рассмотрим два метаязыка, широко используемых для описания синтаксиса языка Делфи: 1) язык металингвистических формул. Одна из его разновидностей – это расширенная форма Бэкуса-Наура (РБНФ); 2) синтаксические диаграммы.

В метаязыках, описывающих синтаксис языка программирования, используются следующие понятия: метапеременная, метаконстанта, синтаксическая единица и метасимвол.

Метапеременная – это определенная синтаксисом конструкция языка (исключая основные символы). Для записи метапеременных используются последовательности слов русского языка и служебных слов, между которыми находится символ подчеркивания. Метапеременные при записи заключаются в угловые скобки.

Метаконстанта – это лексема языка программирования. В программе метаконстанте соответствует она сама. В РБНФ метаконстанты заключаются в кавычки.



Метасимволы – специальные символы, используемые в метаязыках для описания синтаксиса языков программирования. В РБНФ используется следующий набор метасимволов;

а) = (или ::=) имеет смысл «определяется как», «по определению есть»; справа от знака ::= записывается синтаксическая единица, слева – метапеременная;

б) . точка; обозначает конец определения;

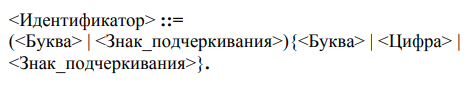
в) | вертикальная черта; обозначает выбор, альтернативу (смысл эквивалентен словам «либо», «или»);

г) { } фигурные скобки; означают возможность повторения заключенной в них конструкции ноль, один или более раз; например, {};

д) [ ] квадратные скобки; обозначают необязательную часть конструкции, т.е. возможность повторения заключенной в них конструкции ноль или ровно один раз; например, запись [“+”] означает, что знак + перед числом может писаться или нет;

е) ( | ) круглые скобки вместе с используемой внутри них вертикальной чертой; означают альтернативы внутри определения, заключенного в скобки; например, запись (“X”|“Y”|...|“Z”) означает вхождение в конструкцию элемента “X” или “Y” или ... “Z”.

Синтаксическая единица – это строка, описывающая состав и порядок следования элементов конструкций языка программирования. Синтаксическая единица состоит из метапеременных, метаконстант и метасимволов.



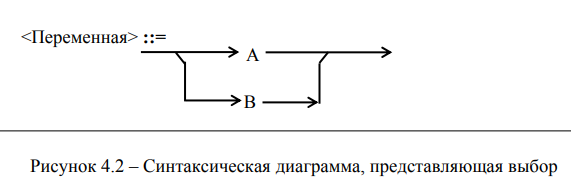


№ 18. Способы описания синтаксиса языков программирования. Синтаксические диаграммы. Пример

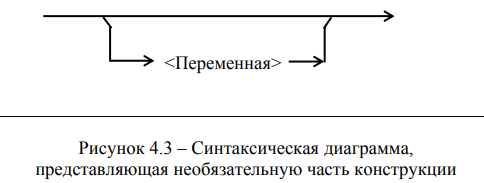
Синтаксис – это набор правил и соглашений, описывающих правильные предложения языка. Для записи правил синтаксиса языков программирования применяют различные формализованные системы обозначений, называемые метаязыками. Мы рассмотрим два метаязыка, широко используемых для описания синтаксиса языка Делфи: 1) язык металингвистических формул. Одна из его разновидностей – это расширенная форма Бэкуса-Наура (РБНФ); 2) синтаксические диаграммы.

Синтаксическая диаграмма представляет собой ориентированный граф с размеченными ребрами. Для разметки ребер используются метапеременные и метаконстанты. Метасимволы на синтаксической диаграмме не используются. Поэтому метаконстанты в синтаксических диаграммах в кавычки не заключаются. Метапеременные заключаются в угловые скобки < >. Для отделения имени синтаксической диаграммы от графа используется метасимвол ::=. Символы < > и ::= являются единственными используемыми метасимволами.

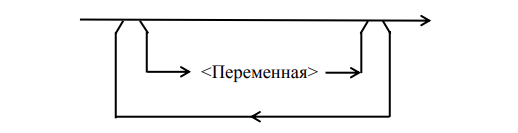
1) Выбору, альтернативе (метасимволу | (Или)) соответствует разветвление в синтаксической диаграмме с последующим объединением.

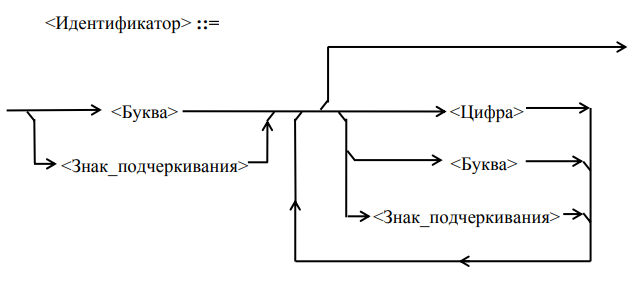


2) Необязательной части конструкции (повторению ноль или один раз, т.е. метасимволам [ ]) соответствует диаграмма, которую представляет рисунок



1. Возможности повторения конструкций ноль, один или более раз (метасимволам { }) соответствует фрагмент синтаксической диаграммы, который представляет рисунок





№ 19. Классификация данных в языке Delphi. Пример.

В программировании данными называется любой набор знаков, над которыми можно выполнять определенные действия. Данные в программе представляются в виде констант и переменных.

Константа – это элемент данных, имеющий фиксированное значение, определяемое записью константы в программе. Значение константы не может быть изменено в ходе выполнения программы. Обозначением константы в программе является она сама либо некоторый идентификатор при соответствующем объявлении.

Переменная – элемент данных, который обозначается в программе идентификатором и может в процессе выполнения программы изменять свое значение.

Под типом значения в Паскале в общем случае подразумевается число отдельных данных, входящих в структуру, тип каждого из этих данных и способ их объединения в структуру. Тип значений определяет:

1) множество различных значений, которые могут принимать программные элементы данного типа;

2) свойства данных значений;

3) операции, которые могут выполняться над этими значениями.

Все типы значений в Паскале делятся на две группы:

1) основные (простые);

2) производные (структурные).

Основные типы – это типы, состоящие из единственного элемента данных (так называемые тривиальные структуры данных).

Основные типы делятся на:

– ссылочный тип;

– скалярные типы.

Скалярные типы в свою очередь делятся на:

1) стандартные типы - это типы, зафиксированные в языке (предопределенные типы); например, типы Integer, Real, Char;

2) описанные типы – это типы, определенные программистом; например, перечислимый тип. В языке Паскаль большинство скалярных типов обладает свойством перенумерованности (упорядоченности). Оно обозначает, что среди элементов соответствующего типа данных установлен линейный порядок. Таким образом, относительно любых двух элементов данного типа определено, какой из них предшествует другому.

Производные (структурные) типы – это типы, которые образуются на основании других типов (как основных, так и производных). В общем случае производный тип состоит из нескольких элементов данных. В Паскале производные типы данных делятся на: – стандартные (предопределенные) – например, типы String, Text и др. – описанные – например, массивы, записи и др.

№ 20. Целочисленные типы данных. Форматы. Диапазоны представления данных. Представление в памяти. Пример.

В языке Паскаль существует пять целочисленных типов:

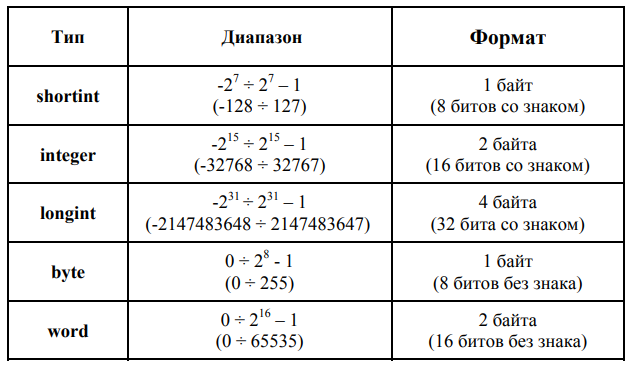
• shortint (короткое целое);

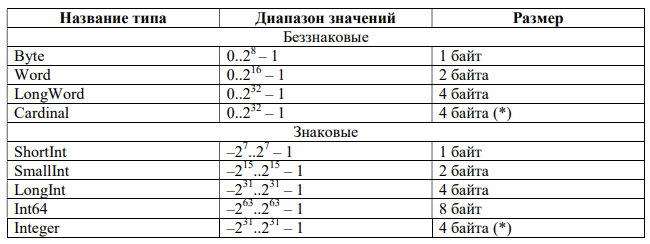
• integer (целое);

• longint (длинное целое);

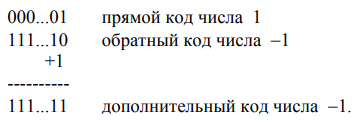
• byte (байт);

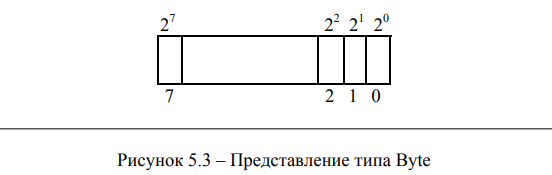
• word (слово).



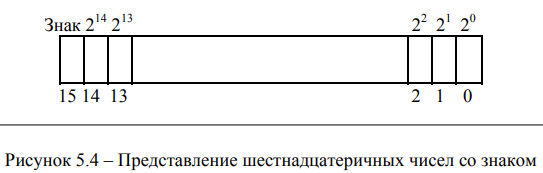


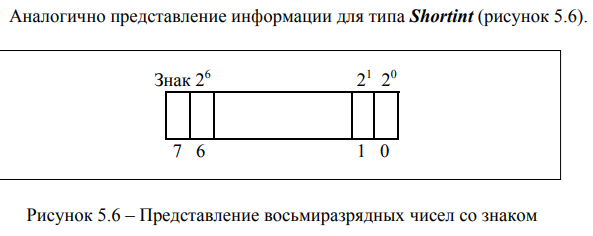
Отрицательные числа в памяти машины хранятся в дополнительном коде. Дополнительный код (ДК) числа i образуется путем инвертирования (замены значения на противоположное) всех значащих разрядов прямого кода числа и прибавления 1 к самому младшему разряду. Аналогично производится преобразование из дополнительного кода в прямой код.

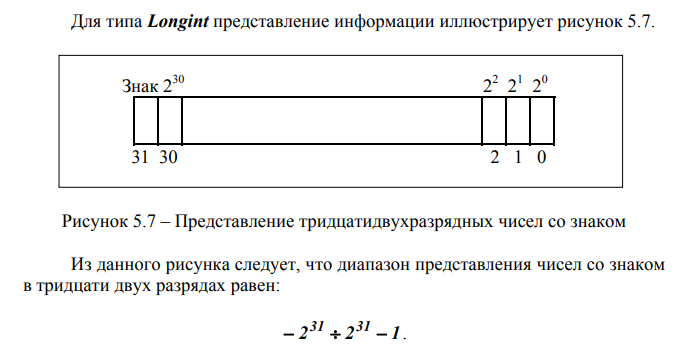




Для чисел со знаком старший бит соответствующего поля памяти, отведенного для хранения числа, считается знаковым (рисунок 5.4). Если в нем 0 – число положительное (знак +), 1 – отрицательное (знак –).



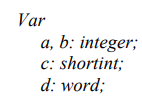




№ 21. Целочисленные константы. Операции над целочисленными данными. Пример.

В программах целые числа обычно используются в качестве счетчиков (для управления количеством повторений цикла) и в качестве индексов.

Множество целочисленных значений является перенумерованным, порядковым номером каждого значения целочисленного типа является само это значение.



Перед выполнением операции оба операнда преобразуются к их общему типу. Общим типом является целочисленный тип с наименьшим диапазоном, включающим все возможные значения обоих типов. Например, общим типом для целого (Integer) и длинного целого (Longint) является Longint. Общим типом для типов Integer и Word является Longint.

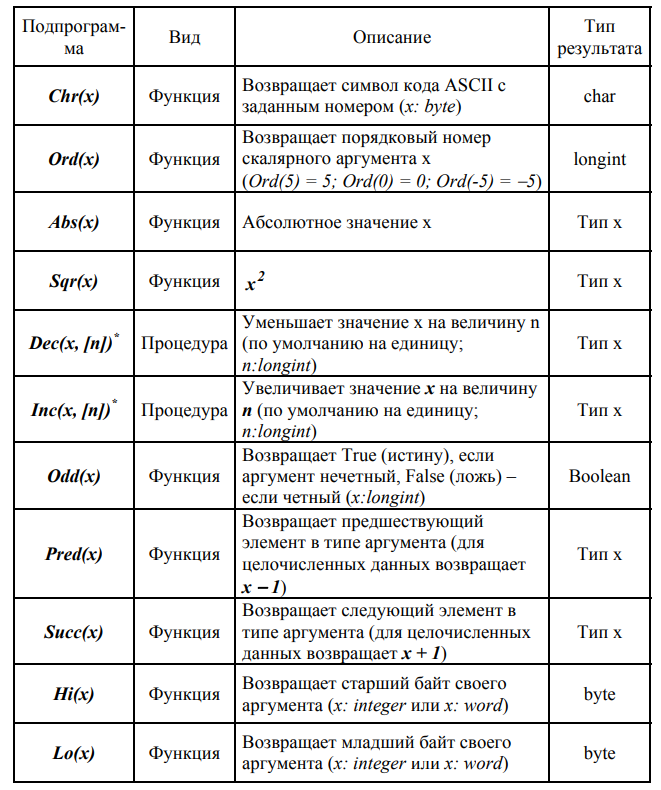
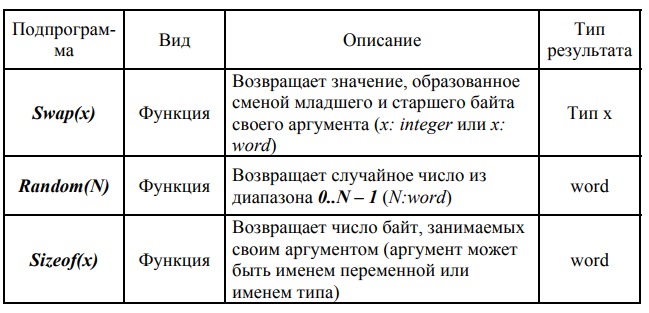
Операции:



№ 22. Встроенные процедуры и функции, определенные над целочисленными данными. Пример.

Встроенные процедуры и функции – это процедуры и функции, которые определены в компиляторе языка программирования. Имена встроенных процедур и функций являются предопределенными идентификаторами.

Процедуры и функции:



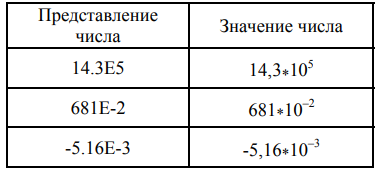
№ 23. Вещественные типы данных. Форматы. Диапазоны представления данных. Представление в памяти. Вещественные константы и способы их записи в программе. Пример.

Значениями вещественных типов являются числа с плавающей точкой (ПТ), представленные в виде мантиссы и порядка. В общем случае числа с ПТ представляются неточно, операции над ними выполняются по правилам действий над приближенными числами.

Вещественные константы в программе могут быть записаны в форме числа с ФТ или числа с ПТ. В форме с ФТ запись константы состоит из целой и дробной частей, разделенных точкой. Знак необязателен. Примеры записи вещественных чисел в форме с ФТ: 0.25 −2.48 +31.0

Наличие точки в вещественной константе обязательно.

В форме с ПТ константа задается в виде десятичной мантиссы m и десятичного порядка p с необязательными знаками, между которыми помещается символ E (или e): mEp или mep Мантисса m представляется целой или вещественной константой с ФТ. Порядок – целое десятичное число, определяющее истинное положение запятой в константе: m ∗ 10^p .



Существует пять вещественных типов для представления переменных:

• Real (вещественный);

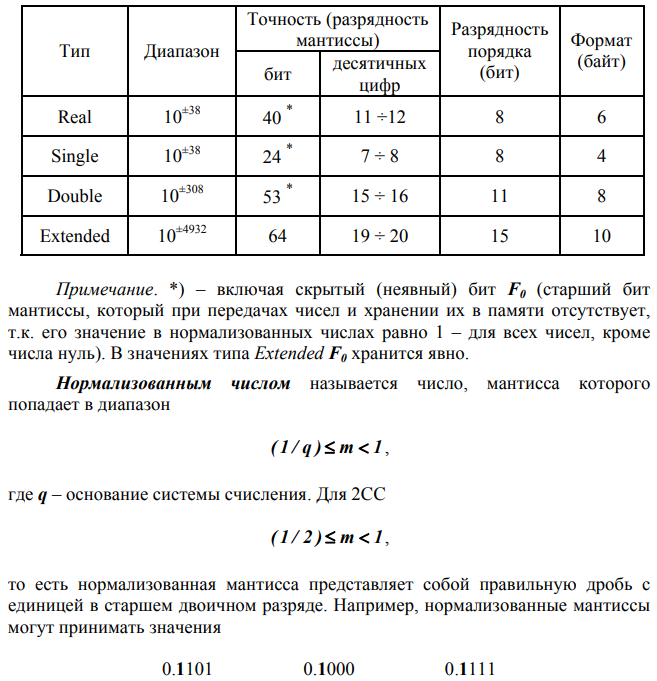
• Single (с одинарной точностью);

• Double (с двойной точностью);

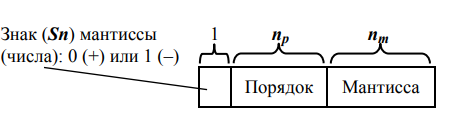
• Extended (с повышенной (расширенной) точностью);

• Comp (сложный тип).

Тип Extended еще называется временным, т.к. при вычислениях в сопроцессоре математических операций все остальные типы преобразуются к нему.



Представление в памяти:



Значение числа равно m(двоич)\* 2^p . Здесь m(дв) – значение двоичного кода мантиссы, р – значение двоичного кода порядка.

Вещественные типы различаются разрядностью поля мантиссы (nm) и поля порядка (np). Мантисса, независимо от знака, хранится в прямом двоичном коде. Порядок хранится в виде характеристики – это двоичный код порядка, представленный с избытком (смещением): 2 1 n 1 p − − . Например, для типа Single смещение равно 127, для типа Double – 1023, для типа Extended – 16383. Характеристика определяется выражением: Характеристика = порядок + смещение. Таким образом, формат вещественного числа в битах (занимаемая им область памяти) определяется выражением: Sn + n(p) + n(m) .

№ 24. Операции над вещественными данными. Встроенные функции, определенные над вещественными данными. Пример

Для значений вещественных типов определены следующие операции.

1) Арифметические операции:

а) одноместные

+ (сохранение знака);

– (изменение знака);

б) двухместные

+ (сложение);

– (вычитание);

\* (умножение);

/ (деление).

2) Операции сравнения:

= (равно);

<> (не равно);

>= (больше или равно);

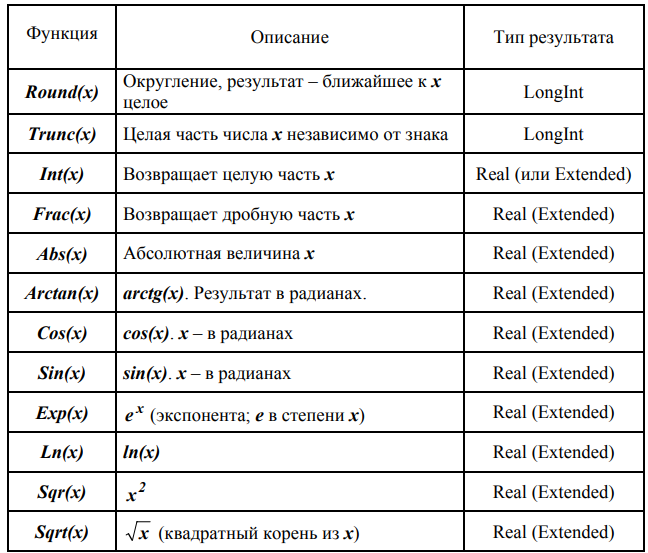
> (больше);

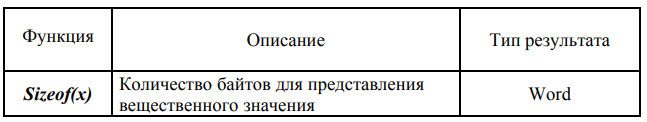
<= (меньше или равно);

< (меньше).

Операции сравнения нужно выполнять осторожно.

Встроенные операции и функции:





№ 25. Символьный тип данных. Способ упорядоченности. Представление в памяти. Операции и встроенные функции, определенные над символьными данными. Пример.

Символьный (литерный) тип Char – это скалярный тип. Значениями этого типа являются элементы расширенного набора символов (литер) кода ASCII. Для представления значений типа Char отводится один байт памяти. Элементы множества значений типа Char считаются перенумерованными (упорядоченными), т.е. каждому значению типа Char поставлен в соответствие свой порядковый номер. Порядковый номер символа равен его коду ASCII.

Переменные типа Char объявляются в разделе описания переменных.

Например, Var a, b, c, x: char. Константой типа Char является один из допустимых символов, взятый в апострофы. Если значением константы является сам апостроф, то он записывается дважды.

Над значениями типа Char определены только операции сравнения:

= (равно);

<> (не равно);

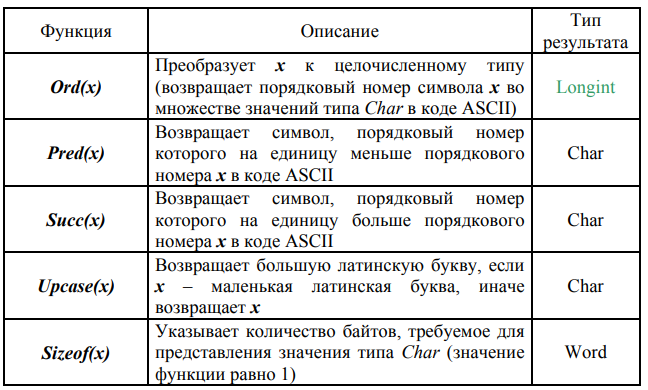
>= (больше или равно);

> (больше);

<= (меньше или равно);

< (меньше).

Над значениями типа Char определены встроенные функции, основные их которых представляет таблица



№ 26. Логический тип данных. Способ упорядоченности. Представление в памяти. Операции и встроенные функции, определенные над логическими данными. Пример.

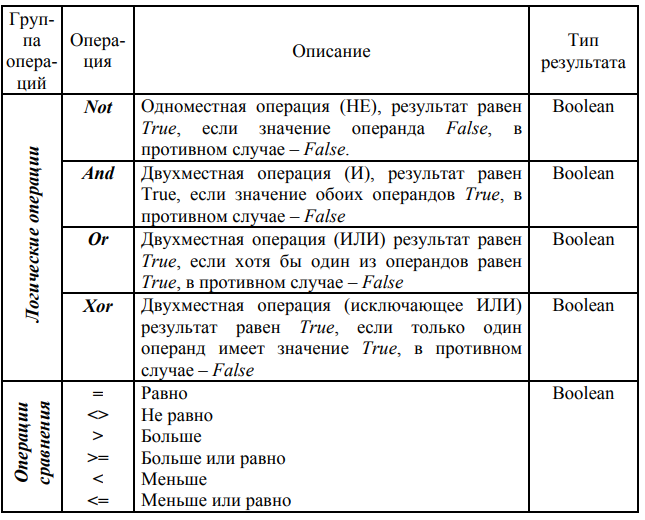
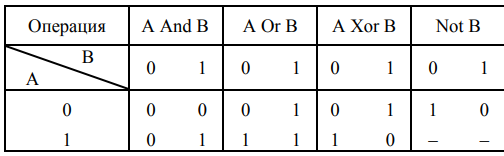
Логический тип определяется как скалярный тип, множество значений которого состоит всего из двух значений: False (ложь) и True (истина). Значения логического типа упорядочены: значение False имеет порядковый номер 0, значение True имеет порядковый номер 1.

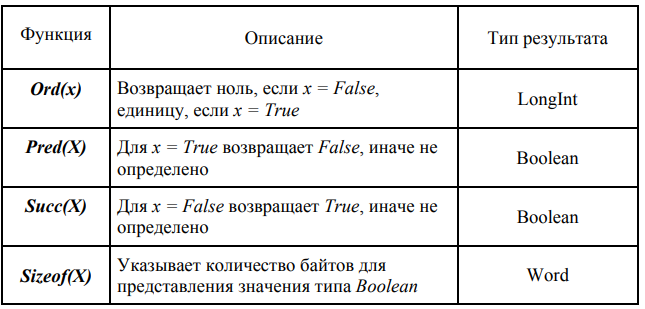
Логические переменные объявляются в разделе описания переменных.

Например Var X, Y, Z: Boolean; Логическими константами является предопределенные в языке Паскаль идентификаторы True и False.

Над значениями типа Boolean определены операции, которые представляет таблица

Таблица построена по правилам алгебры логики (булевой алгебры), где 0 соответствует значению False (ложь), 1 – значению True (истина).





№ 27. Выражения и их типы. Правила написания и вычисления выражений. Приоритет операций. Пример.

Выражение – формула для вычисления некоторого значения, состоящая из операндов, знаков операций и круглых скобок. Операндами могут быть константы, переменные и указатели функций (вызовы функций, функции).

Ниже даны общие правила написания и вычисления выражений:

1) Первая группа операций имеет самый высокий приоритет, т.е. операции данной группы выполняются в первую очередь.

2) В каждой из групп операции имеют одинаковый приоритет.

3) Операции одного приоритета обычно вычисляются в порядке их следования в выражении (слева направо). Иногда для оптимизации вычислений транслятор может переупорядочить их.

4) Операнд, находящийся между двумя операциями с различными приоритетами, связывается с операцией, имеющей более высокий приоритет.

5) Операнд, находящийся между двумя операциями с равными приоритетами, связывается с той операцией, которая находится слева от него.

6) Для уточнения последовательности действий применяются круглые скобки. Действия в скобках выполняются в первую очередь.

7) Последовательная запись знаков двух операций запрещена. Исключение – операция Not.

8) Многоэтажные формулы в выражении записываются в одну строку.

9) Знак умножения \* опускать нельзя.

В порядке убывания приоритета операции в Паскале делятся на четыре группы:

1) операции Not, @ (взятие адреса объекта)

2) операции группы умножения: \*, /, Div, Mod, And, Shl, Shr

3) операции группы сложения +, –, Or, Xor

4) операции группы сравнения In, =, <>, >, >=, <=

В Делфи каждое выражение определяет значение какого-то определенного типа, в нем могут фигурировать операнды тоже определенных типов. Поэтому в Делфи выражения также делятся на несколько типов. Здесь мы рассмотрим выражения скалярных стандартных типов.

Скалярные стандартные выражения делятся на три типа: — арифметические; — логические; — символьные. Тип выражения определяется типом результата его вычисления.

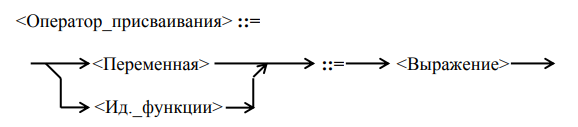
Арифметическое выражение – это выражение, результатом вычисления которого является целое или вещественное значение. Операнды арифметического выражения должны иметь арифметический (целый или вещественный) тип. В качестве операций могут использоваться арифметические операции, определенные для этих типов операндов. В обычном случае в Делфи не определена операция возведения в степень, для нее используется экспонента.

Символьное выражение – это выражение, результатом вычисления которого является символьный тип данных Char (т.е. отдельные символы). В Паскале отсутствуют операции, результатом выполнения которых являются данные типа Char. Поэтому символьным выражением может быть только символьная константа, символьная переменная или функция, дающая результат типа Char.

Логическое выражение – это выражение, результатом которого является логическое значение (True или False). Операнды логического выражения должны иметь тип Boolean (операндами могут быть переменные, константы, имена функций).

№ 28. Оператор присваивания и его типы. Составной оператор. Пустой оператор. Назначение. Формат. Пример

Многие программы или фрагменты программ являются линейными – т.е. такими, в которых операторы выполняются строго последовательно, в порядке записи в тексте программы. Наиболее часто используемым оператором линейной программы является оператор присваивания. В общем случае оператор присваивания имеет формат, который представляет рисунок

  
Оператор присваивания предписывает вычислить значение выражения, записанного в его правой части, и присвоить его переменной, имя которой записано в левой части.

Поэтому, с учетом классификации скалярных стандартных выражений, существует три типа скалярных стандартных операторов присваивания: арифметический, логический, символьный.

Арифметический оператор присваивания. Служит для присваивания значения переменной арифметического типа (вещественного или целочисленного). В правой части оператора должно быть записано арифметическое выражение. Примеры арифметических операторов присваивания: X := 0; Y := 2 \* a / b; Z := sin(c \* 2 + a \* a); Все переменные должны иметь арифметический тип.

Логический оператор присваивания. Это оператор присваивания, в левой части которого указана переменная типа Boolean. В правой части оператора должно быть логическое выражение. Примеры логических операторов присваивания: A := False; B := G > L; C := (E <> F) Or Odd(X); D := Y = Z; Здесь: A, B, C, D – переменные логического типа, Х – переменная целого типа, E, F, G, L, Y, Z – переменные любых скалярных типов (совместимых между собой).

Символьный оператор присваивания. Это оператор присваивания, в левой части которого указана переменная типа Char. В правой части оператора должно быть задано символьное выражение. Примеры литерных операторов присваивания: A := ‘A’; B := C; D := Pred(B); Здесь A, B, C, D – переменные типа Char.

Составной оператор – это последовательность операторов программы, заключенных в операторные скобки begin и end (цикл, if…)

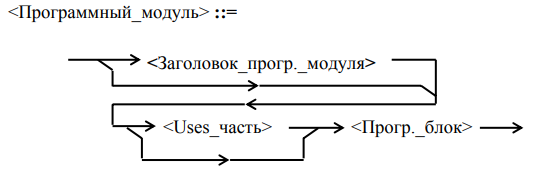
Пустой оператор – оператор, состоящий из точки с запятой и не содержащий других элементов. Поскольку end является закрывающей операторной скобкой, то указывать точку с запятой в конце модуля после выполнения последней операции не обязательно. Наличие точки с запятой перед end означает, что перед end находится еще 1 пустой оператор, который не совершает никаких действий, поэтому он в основном служит для передачи управления в конец составного оператора.

№ 29. Структура программы на языке Delphi. Синтаксис. Назначение стандартных модулей UNIT. Виды объявлений. Раздел операторов. Пример.

Делфи-программа состоит из одного или нескольких независимо компилируемых модулей. Различают два вида модулей: • программный модуль; • модуль Unit.

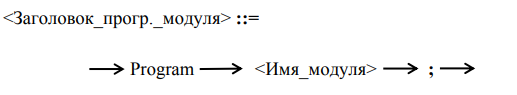
Программный модуль определяет программу, получающую управление от операционной системы. Модуль Unit имеет вспомогательный характер, используется для создания библиотек подпрограмм и может применяться лишь вместе с программным модулем. Каждый модуль состоит из заголовка, Uses-части (предложения использования) и тела модуля (блока). Признаком конца модуля является точка.

Программный модуль имеет следующую структуру

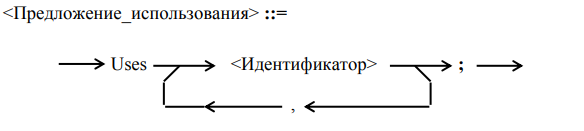


Заголовок программного модуля начинается служебным словом Program, за которым записывается имя, присваиваемое модулю.

Заголовок программного модуля имеет вид

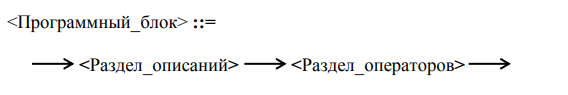


Предложение использования Uses определяет все модули Unit, которые будут использоваться в данной программе



Модули Unit могут быть стандартными и модулями, созданными программистом. Существует восемь стандартных модулей Unit: System, Dos, Crt, Printer, Graph, Overlay, Turbo3, Graph3.

Тело программного модуля (программный блок) состоит из раздела описаний (объявлений) и раздела операторов. Формат программного блока представляет рисунок



Раздел описаний следует вначале тела модуля. В языке Паскаль предусмотрено пять видов объявлений (описаний):

1) объявление меток;

2) объявление констант;

3) объявление типов; объявлений

4) объявление переменных;

5) объявление процедур и функций.

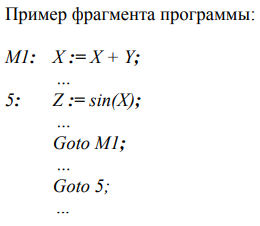
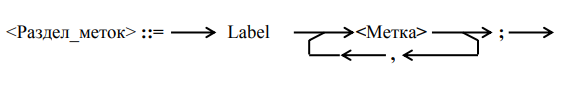
Раздел операторов служит для описаний действий, необходимых для решения задачи

Раздел операторов имеет вид последовательности операторов, заключенных в операторные скобки.

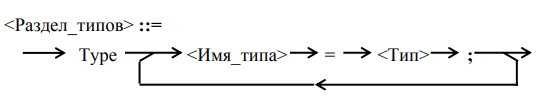
№ 30. Разделы меток, типов, переменных. Назначение, синтаксис. Пример.

Раздел меток:

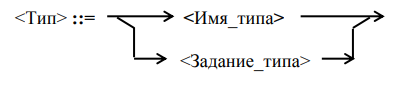
Метка – это идентификатор или целочисленная константа в диапазоне 0 ÷ 9999, стоящая перед оператором и отделенная от него двоеточием. Метка служит для выделения оператора и играет роль имени данного оператора. Оператор с меткой называется помеченным. Метки используются, чтобы перейти к выполнению определенного оператора, нарушая естественный последовательный порядок выполнения операторов. Все метки операторов должны быть различны. Передача управления оператору, помеченному меткой, осуществляется с помощью управляющего оператора Goto. Все метки, используемые в разделе операторов модуля, должны быть обязательно предварительно описаны в разделе меток.

  
Раздел типов:

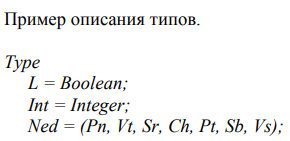
Язык Паскаль позволяет программистам наряду со стандартными типами значений вводить в употребление свои типы данных. Каждый вводимый тип необходимо описать и присвоить ему имя. Это выполняется в разделе типов.



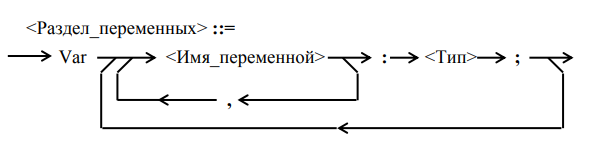
За словом Type следуют описания типов, разделенные точкой с запятой (;). Описание типа задает имя типа и присваиваемый ему тип. Тип определяется в соответствии с диаграммой, которую иллюстрирует рисунок

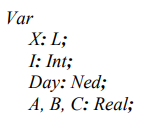


Таким образом, в описании типа может быть использовано имя описанного ранее или стандартного типа (верхнее ребро синтаксической диаграммы) или задан нестандартный тип данных (нижнее ребро диаграммы).



Раздел переменных:

Каждая переменная, используемая в программе, обязательно должна быть объявлена (описана). Это значит, что переменной должно быть присвоено имя и указан тип значений, которые может принимать данная переменная. Описание переменных содержатся в разделе переменных. Раздел начинается служебным словом Var и имеет вид

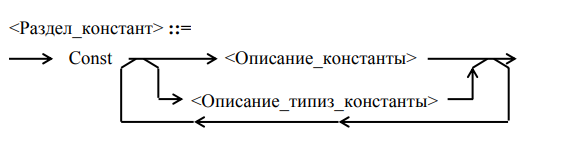
Соответственно описать переменную можно, используя имя стандартного(пользовательского) типа, так и явным заданием типа.

№ 31. Раздел констант. Типизованные и нетипизованные константы. Назначение, синтаксис. Типизованные скалярные константы. Пример.

Раздел констант:

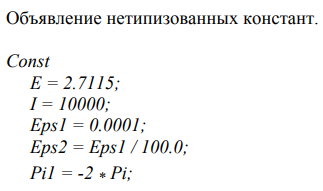
Если в программе некоторые константы используются многократно, то удобно присвоить им имя.

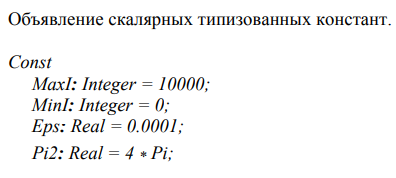
Присвоение константе имени выполняется в разделе описания констант. Раздел начинается служебным словом Const и имеет формат



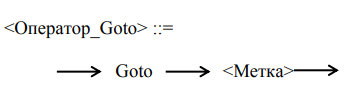
Таким образом, в разделе констант могут быть объявлены обычные (нетипизованные) константы и типизованные константы. Описание константы вводит идентификатор ( на синтаксической диаграмме), под которым подразумевается некоторое константное выражение. Тип константы определяется типом данного выражения. Константное выражение – это выражение, вычисляемое на этапе компиляции.



  
Типизованные константы аналогичны инициализированным переменным. Инициализированные переменные – это переменные, которым при входе в блок присваиваются начальные значения. Различают скалярные и структурные типизованные константы.

Типизованные константы можно использовать аналогично переменным того же типа. Их значения можно изменять в процессе вычислений. Они могут использоваться в левой части оператора присваивания.   


№ 32. Оператор GOTO. Формат. Назначение. Ограничения использования. Недостатки. Пример.

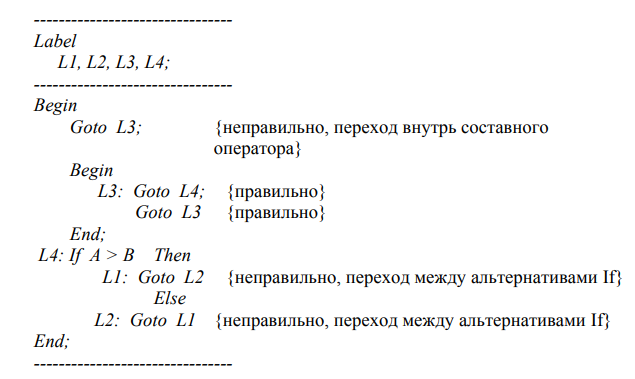
Оператор Goto еще называют оператором безусловного перехода. Он служит для безусловной передачи управления помеченному оператору. Формат Goto: 

Оператор Goto указывает, какой оператор должен выполняться следующим. В языке имеется ряд ограничений на использование операторов Goto:

1) с помощью Goto нельзя переходить внутрь производных операторов, не содержащих данный оператор Goto (составного оператора, операторов For, Repeat, While, If, Case, With);

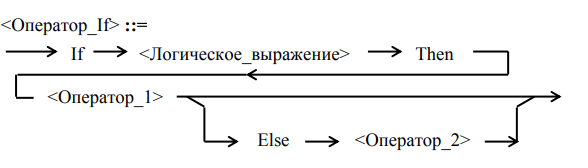
2) с помощью оператора Goto запрещен переход из одной альтернативы в другую в выбирающих операторах (If, Case);

3) с помощью оператора Goto нельзя входить в подпрограмму или выходить из нее.

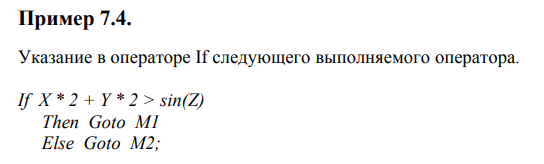


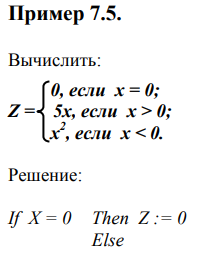
№ 33. Оператор IF. Формат. Назначение. Полная и сокращенная форма. Пример

Оператор If является производным оператором. Относится к группе выбирающих операторов. Используется в разветвляющихся программах для выбора того или иного участка вычислений в зависимости от выполнения некоторого условия. Поэтому его еще называют оператором условного перехода или оператором принятия двоичного(дихотомического) решения.



Вообще ветка else в операторе if может и не указываться. В таком случае, в зависимости от условия, выполнится оператор 1 или вообще ничего не произойдет. Операторы if могут быть вложены и друг в друга.

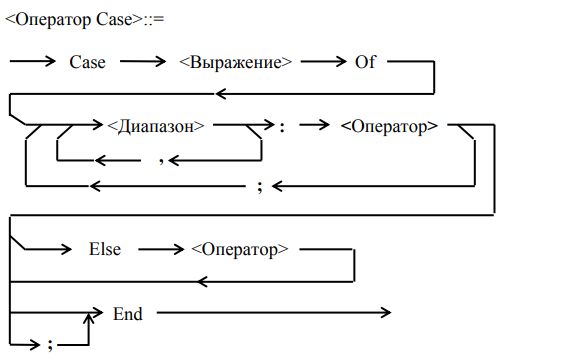




№ 34. Оператор CASE. Формат. Назначение. Правила выполнения. Пример.

Оператор Case является производным оператором. И так же как и if, относится к выбирающим операторам. Используется, если вычисления нужно разветвить более, чем на 2 направления.

Формат оператора Case представляет рисунок

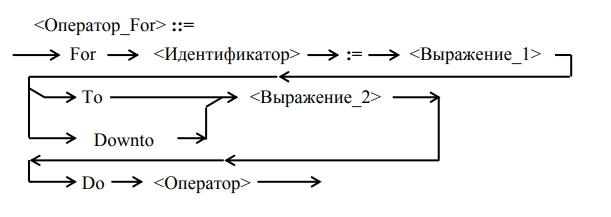
Выражение также называется селектором и может принимать любой перенумерованный тип, кроме Longint, Word и строкового типа.

Каждый оператор помечен списком констант и(или) диапазоном.

Также последний оператор может помечаться веткой else. Сначала выполняется та ветка, где выражение совпало с диапазоном. Если совпадений нет, то либо выполнится ветка else, либо ничего не произойдет.

№ 35. Оператор цикла с параметром. Формат. Назначение. Правила выполнения. Пример.

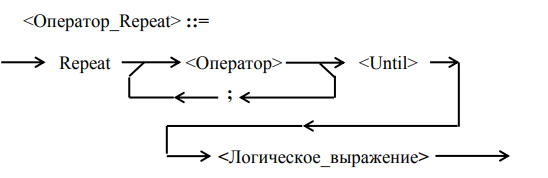
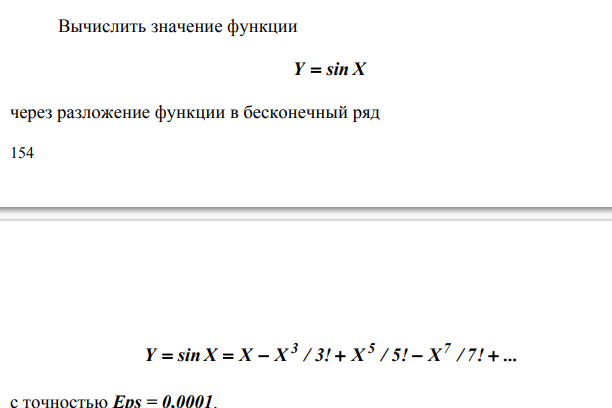
Оператор цикла с параметром используется для циклов, где нам заранее известно число повторений. Параметром цикла служит переменная, которая так же называется счетчиком, служит также для проверки условия. Оператор имеет вид:



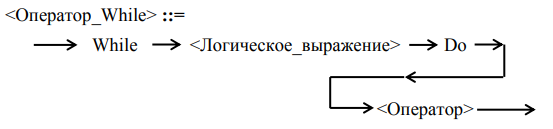
На синтаксической диаграмме Идентификатор = параметр цикла. Это может быть переменная перенумерованного скалярного типа. Телом цикла является Оператор. Если в цикле используется служебное слово To, то значение счетчика в конце выполнения тела цикла автоматически увеличивается на 1(Вычисляется с помощью функции succ!). Если используется Downto, то значение уменьшится на 1. Перед каждым выполнением тела цикла, происходит сравнение текущего значения идентификатора с предельным его значением. Если он меньше(для To) или больше(для Downto), то тело цикла начинает выполнятся. Не следаует также изменять значение идентификатора в теле цикла.

№ 36. Оператор цикла с постусловием. Формат. Назначение. Правила выполнения. Пример.

В общем случае используется для программирования цикла с заранее неизвестным числом повторений. Окончание цикла зависит от выполнения условия в самом цикле. Тело цикла выполняется до тех пор, пока логическое выражение принимает значение False. Проверка условия выполняется как раз после выполнения тела цикла.

  
  
№ 37. Оператор цикла с предусловием. Формат. Назначение. Правила выполнения. Пример.

Используется в общем случае для программирования цикла, где число повторений заранее известно. Однако можно использовать и для итерационных циклов (но это зачастую не очень удобно и наглядно).



Выполнение цикла прекращается, когда 157 впервые станет равным False. Если к началу выполнения цикла значение < Логического выражения> равно False, то тело цикла не выполняется ни разу.

Если в теле цикла необходимо выполнить несколько операторов, используется составной оператор. Оператор While является наиболее универсальным из операторов цикла. Его можно использовать и вместо операторов For и Repeat.

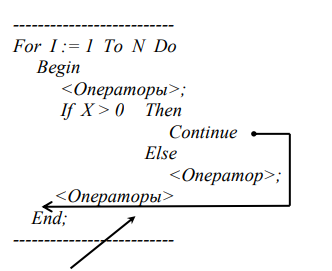
№ 38. Сравнительная характеристика операторов цикла. Пример.

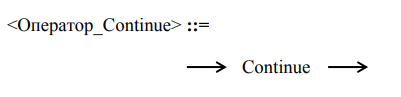
Описать все операторы цикла….

№ 39. Операторы Continue и Break. Пример.

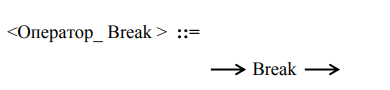
Данные операторы предназначены для гибкого управления операторами циклов For, While, Repeat.

Оператор Continue осуществляет передачу управления на конец тела цикла (к End). Данный оператор используется, если при некоторых условиях тело цикла или его часть выполнять не нужно. Осуществляет переход на конец тела цикла, после чего выполняется анализ условия дальнейшего выполнения цикла.



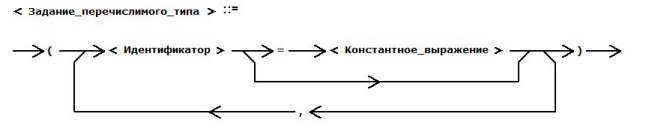


Оператор Break служит для безусловного выхода из операторов For, Repeat и While.



Если в примере вместо оператора Continue записать оператор Break, то по отмеченной ветви осуществится выход из цикла: цикл при следующих значениях параметра выполняться не будет, осуществляется передача управления оператору, следующему за оператором цикла. Использование операторов Continue и Break позволяет практически во всех программах обходиться без оператора безусловного перехода Goto.

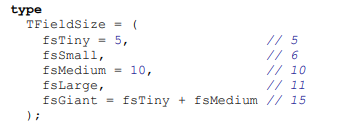
№ 40. Перечислимый тип данных. Формат задания. Способ упорядоченности. Представление в памяти. Операции и встроенные функции, определенные над данными перечислимого типа. Пример.

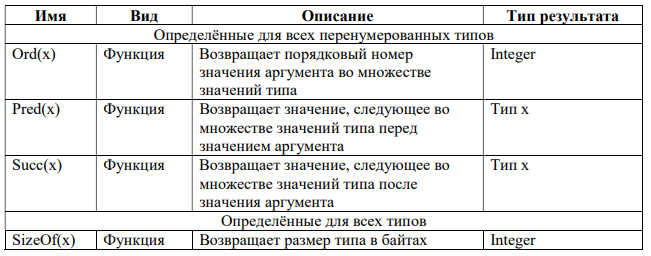
Перечислимый тип задается перечислением всех своих значений. При этом переменные, относящиеся к типу (перечислимому типу) смогут принимать только те значения, которые были перечислены в скобках

В памяти значения перечислимых типов хранятся как целые числа, причём, если ничего не предпринимать, то литералам, использованным при объявлении, эти числа назначаются последовательно, начиная с 0

Количество памяти, выделяемое под переменную перечислимого типа, определяется двумя факторами: − количеством и значениями этого типа; − настройками компилятора.

Ещё одна полезная возможность — явное задание значений, соответствующих литералам перечислимого типа.

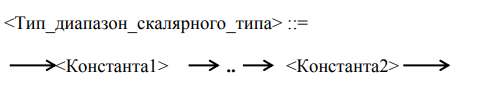
Где каждое новое явное задание становится новой точкой отсчета  
Над значениями перечислимого типа определены только операции сравнения. Большим считается то значение, которое в памяти представлено большим числом. Кроме того, доступны следующие функции:



Как нетрудно догадаться по первым трём функциям, значения перечислимых типов обладают свойством перенумерованности. Порядковый номер значения, возвращаемый функцией Ord, совпадает с числовым представлением этого значения в памяти.

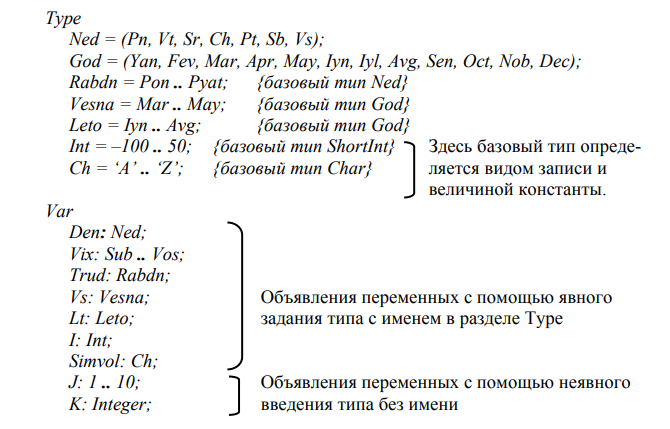
№ 41. Тип данных диапазон. Формат задания. Базовый тип. Способ упорядоченности. Операции и встроенные функции, определенные над данными типа диапазон. Пример.

Этот тип данных называют еще ограниченным типом или интервальным типом. Тип диапазон задается путем накладывания ограничений на уже заданный или стандартный тип, называемый базовым типом. В качестве базового типа могут быть использованы скалярные типы, обладающие свойством перенумерованности (т.е. все скалярные типы, кроме вещественных). Тип диапазон определяется в программе так, как представляет рисунок



Значение должно быть не более значения . Тип диапазон является перенумерованным. Порядковый номер элемента совпадает с порядковым номером этого элемента в базовом типе.

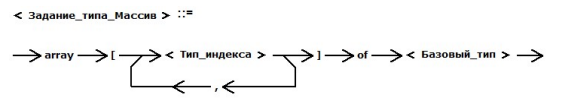
Тип диапазон может быть определен либо в разделе типов, либо с помощью описания переменных в разделе Var.

Таким образом, множество значений типа диапазон принадлежит множеству значений базового типа. К значениям типа диапазон применимы все операции и функции, определенные над значениями базового типа. Тип диапазон совместим со своим базовым типом, т.е. значения типа диапазон могут использоваться везде, где могут использоваться значения базового типа.

Достоинства типа диапазон: 1) позволяет транслятору экономнее использовать память при представлении значений переменных; 2) обеспечивает возможность контроля как на этапе трансляции, так и во время выполнения программы за корректностью присваиваний, что помогает исправлять ошибки в программе; 3) обеспечивает наглядную форму представления решаемой задачи. Тип диапазон широко применяется в комплексе с производными типами, в частности, в задачах обработки массивов.

№ 42. Массивы. Формат задания. Представление в памяти. Типы индекса. Полная и сокращенная формы задания многомерных массивов. Подмассивы. Пример.

Массив – упорядоченная совокупность однотипных элементов, имеющих общее имя. Массив относится к структурным типам данных. Тип элементов массива называется базовым типом массива.

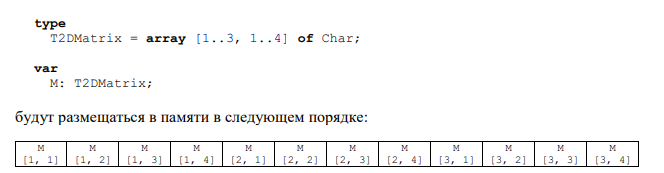


Количество элементов в массиве определяется типами индексов. Например, такое объявление

type TExample = array [Byte] of Boolean;

Задаёт массив из 256 элементов типа Boolean, причём индексы элементов будут принимать значения от 0 до 255 включительно. Но наиболее часто встречается использование типа диапазон. Кстати, использование произвольной нумерации элементов массива – почти уникальная возможность языка Delphi. В индексации массива важно, чтобы индексация (тип индексации) обладал свойством перенумерованности. Поэтому элементы массивов также называют индексированными переменными.

В памяти компьютера массив представляет собой последовательно расположенные значения каждого из его элементов. Размер каждого элемента в памяти определяет базовый тип. Аналогично устроены и многомерные массивы. Как мы уже обсудили ранее, многомерный массив представляет собой массив из массивов меньшей размерности, поэтому, например, элементы двухмерной матрицы(см. рисунок). То есть, чтобы перейти к какому-либо не первому элементу, нужно отступить место от самого первого элемента.



Массивы в языке Delphi могут иметь и несколько индексов. Такие массивы будут называться многомерными. Рассмотрим на примере двумерного массива(матрицы) использование полной и сокращенной формы задания массива.

type

TGameDesk = array [1..3, 1..3] of TCellState;

TGameDesk = array [1..3] of array [1..3] of TCellState;

То есть во втором случае конкретно указывается, что каждый элемент массива также является массивом. Аналогично и при обращении к элементам.

WriteLn(Desk[1, 3]);

WriteLn(Desk[1][3]);

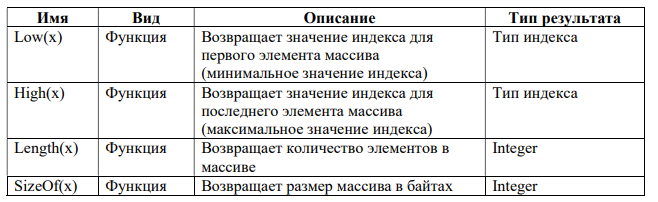
Возможно указание меньшего количества индексов, чем использовано при объявлении. В этом случае подразумевается обращение к подмассиву — массиву меньшей размерности, являющемуся частью многомерного массива.

№ 43. Действия над массивами и над их элементами. Одномерная и многомерная константа-массив. Формат задания. Назначение. Пример.

Над индексированными переменными (элементами массива) определены те же самые операции, процедуры и функции, что и для базового типа массива. Другими словами, элемент массива ведёт себя так же, как и обычная переменная того же самого типа.

Над полной переменной массива не определено никаких операций, однако полные переменные могут появляться в операторе присваивания. Например, если массивы одинакового размера и одинакового базового типа, то простой операцией присваивания одного массива другому, можно скопировать значения одного массива в другой.

Также над массивом определены следующие функции:



Массивы в языке Delphi могут иметь и несколько индексов. Такие массивы будут называться многомерными. Рассмотрим на примере двумерного массива(матрицы) использование полной и сокращенной формы задания массива.

type

TGameDesk = array [1..3, 1..3] of TCellState;

TGameDesk = array [1..3] of array [1..3] of TCellState;

То есть во втором случае конкретно указывается, что каждый элемент массива также является массивом. Аналогично и при обращении к элементам.

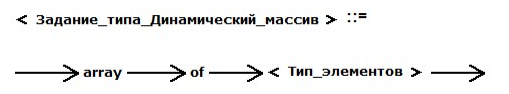
WriteLn(Desk[1, 3]);

WriteLn(Desk[1][3]);

Возможно указание меньшего количества индексов, чем использовано при объявлении. В этом случае подразумевается обращение к подмассиву — массиву меньшей размерности, являющемуся частью многомерного массива.

№ 44. Динамические массивы. Синтаксис объявления. Представление в памяти. Основные операции над динамическими массивами. Подсчёт ссылок.

Динамический массив — это массив, количество элементов в котором может быть изменено во время работы программы.

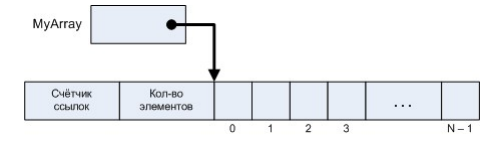
Изначально такой массив содержит 0 элементов.

Разумеется, пользы от такого массива немного, поэтому, когда становится понятно, сколько элементов нужно для работы, следует воспользоваться процедурой SetLength, например, так: SetLength(Numbers, 10); После такого вызова массив Numbers будет содержать 10 элементов.

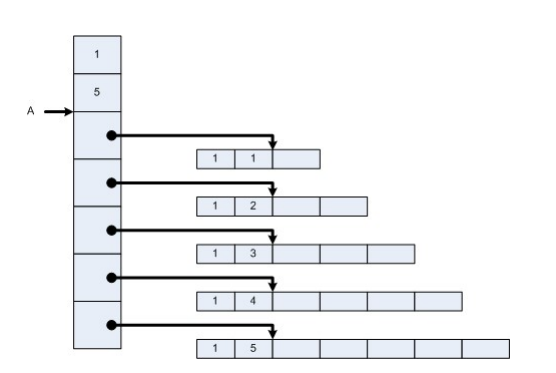
Чтобы проверить длину массива, можно воспользоваться функцией Length. Индекс последнего элемента на 1 меньше, чем длина массива, т к нумерация идет с 0.

Важно, что элементы динамического массива имеют целочисленные индексы, начинающиеся с 0. Это означает, что для динамического массива с 10 элементами допустимы значения индексов от 0 до 9 включительно.

Представление динамических массивов в памяти:

Как можно видеть на рисунке, по внутреннему устройству динамические массивы похожи на динамические строки. Как и в случае с динамическими строками, переменная типа динамический массив содержит в себе не сам массив, а лишь информацию о том, где в памяти он находится. При попытке присвоить массиву B значение массива А, будет создан не новый массив, а всего лишь переменная, которая ссылается на тот же экземпляр массива. Чтобы создать именно копию, следует использовать функцию Copy.

То же самое касается и многомерных динамических массивов. В памяти каждый элемент такого массива будет содержать ссылку на экземпляр динамического массива:



Чтобы очистить память, занимаемую экземпляром массива, нужно использовать присвоение полной переменной значения nil. В результате счетчик ссылок обнулится и память очистится.

№ 45. Строковые константы. Правила их записи в программе. Строковые переменные постоянной длины. Формат задания. Представление в памяти. Операции, определенные над строковыми данными постоянной длины. Пример.

Строковая константа – это последовательность любых символов, допускаемых для представления в компьютере, заключенная в апострофы. Например, ‘TEKST’.

Правила записи строковых констант в программе:

1) Если в строке необходимо поместить апостроф, то его повторяют дважды. При подсчете длины строки два рядом стоящих апострофа считаются одним символом.

2) При подсчете длины строки учитываются пробелы.

3) Допускаются пустые символьные константы, т.е. константы, не содержащие ни одного символа

4) Паскаль разрешает вставлять в строку символов управляющие символы. Символ # с целой константой без знака (от 0 до 255) обозначает соответствующий этому значению символ в коде ASCII. Между # и целой константой не должно быть никаких разделителей. Если несколько управляющих символов входят в строковую константу, то между ними не должно быть разделителей.

Строки постоянной длины — это не что иное, как массивы типа Char. Пример объявления:

var S:

array [1..14] of Char;

И да, причина, почему длина остаётся неизменной, заключается в том, что она уже прописана прямо в тексте программы как количество элементов массива.

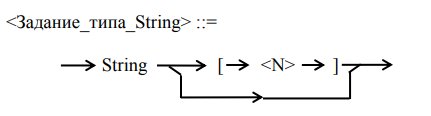
Действительно, строки постоянной длины имеют все те же свойства, что и другие массивы. В частности, можно выполнять присваивание полных переменных, если размеры массивов одинаковы, можно обращаться к отдельным элементам (в данном случае это будут элементы типа Char, т.е. символы) и т.д.

Тем не менее, есть и два важных отличия. Во-первых, переменной типа «строка переменной длины» можно присваивать строковые литералы той же длины: S := '7123456A001PB5';

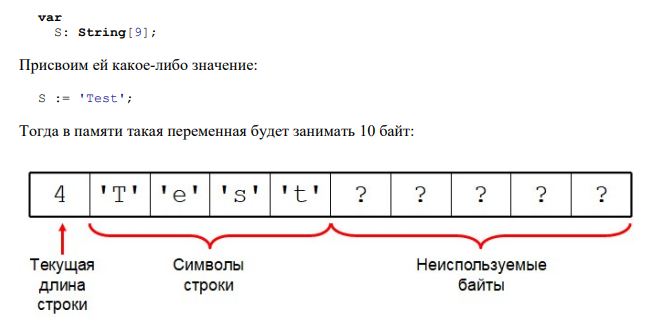
Во-вторых, можно сравнивать строковые переменные постоянной длины. Сравнение происходит слева направо. Большей считается та строка, где первый не совпавший символ больше(по коду АСКИ).

№ 46. Строки переменной длины. Формат задания. Представление в памяти. Операции, определенные над строковыми данными переменной длины. Пример.

Вторым способом задания строковых переменных является использование типа String. С помощью типа String определяются строки переменной длины.

Такие строки могут хранить максимум 255 значений. Если присваиваемая строка больше по размеру, то она обрезается до максимальной длины справа.

Представление в памяти:



Над типом String определены только операции сравнения(по коду первого не совпавшего символа) и операция конкатенации(склеивания). Операция конкатенации двух строк эквивалентна записи S1+S2. Если результирующая строка больше(например 255), то она обрезается. Дело в том, что 255 – это максимальное значение, которое можно хранить в 1 байте отведенном для текущей длины строки.

№ 47. Встроенные процедуры и функции, определенные над строками переменной длины. Пример.

Функции:

1. Copy (St, Poz, N) – копирует откуда, с какой позиции, сколько символов
2. Concat (St1[, St2, ..., StN] ) - Выполняет сцепление строк в том порядке, в каком они указаны в списке параметров
3. Length (St) - Возвращает текущую длину строки St
4. Pos (St1, St2) – возвращает первое найденное вхождение строки St1 в строке St2, возвращает номер позиции.
5. UpCase (Ch) – Возвращает заглавную букву аргумента типа char(для преобразования всей стрки используется цикл).

Процедуры:

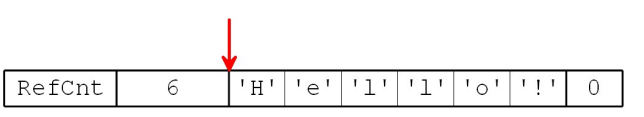
1. Delete (St, Poz, N) – удаляет из строки St с позиции Poz N символов.
2. Insert (St1, St2, Poz) – вставляет строку St1 в строку St2 в позицию Poz.
3. Str (I, St) – Преобразует число I в строку St.
4. Val(St,I,Cod) – переводит строку St в числовое значение I. Cod – содержит целочисленное значение позиции первого ошибочного символа.

№ 48. Динамические строки (Delphi-строки). Синтаксис объявления. Представление в памяти. Отличия в реализации от динамических массивов. Подсчёт ссылок.

Для так называемых коротких строк, по сути, ситуация аналогична: размер такой строковой переменной всегда на 1 байт больше, чем максимальная длина строки. Однако есть и проблема: var S: String[200]; ... S := 'Вася'; Обратите внимание, что переменная занимает 201 байт, хотя реально для записи строки было бы достаточно 4 байт. Казалось бы, две сотни байт в наше время — совсем мелочь, но посмотрите на это с другой стороны: мы использовали для хранения строки в 50 с лишним раз больше, чем требовалось! Решение этой проблемы, конечно же, заключается в том, чтобы выделять память по мере необходимости.

var S: String; // Скорее всего, это Delphi-строка

Объявление string с явным заданием размера – паскаль строка.



Счетчик перед количеством символов указывает сколько переменных ссылаются на данный участок памяти, содержащий строку. При попытке изменения строки, если счетчик ссылок равен 1, то изменение происходит по месту. В противном случае для данной переменной создается копия строки, с которой и выполняются преобразования.

Главным отличием от динамических массивов является то, что в массивах не нужно было обеспечивать совместимость. То есть динамический массив хранит только счетчик ссылок и количество элементов.