# **1. Этапы и особенности решения задач с помощью ЭВМ.**

1. Постановка задачи. Точная формулировка задачи.

2. Разработка математической модели. Математическая модель − приближенное описание некоторого класса явлений внешнего мира, выраженное с помощью математической символики, то есть конкретное явление переводится на язык математических формул и дальше рассматривается как математическая проблема.

3. Выбор способа решения задачи и спецификация алгоритма. Должны быть установлены выходные и входные данные алгоритма, выделены основные отношения между ними. Необходимо определить, может ли задача быть решена и при каких исходных данных. Результат этапа - спецификация алгоритма, то есть формулировка в общем виде того, что должен делать алгоритм, чтобы переработать входные данные в выходные. Причем должен быть описан класс входных данных со всеми ограничениями.

4. Разработка алгоритма и его запись. От разработки алгоритма зависит качество программы.

5. Кодирование. Кодирование − запись алгоритма на языке программирования.

6. Отладка. Отладка заключается в поиске и исправлении ошибок. Существует два метода отладки: ручная проверка, не требующая применения ЭВМ, и тестирование. Тестирование заключается в прогонах полностью или частично завершенной программы на ЭВМ. Оба метода требуют тестовых данных. Тестовые данные − наборы исходных данных и ожидаемых результатов. Подбор тестовых данных − важная часть работы программиста. Тесты рекомендуется готовить параллельно с разработкой алгоритма.

7. Получение и анализ результатов. Организация прогона программы на ЭВМ зависит от конкретного типа ЭВМ

# **2. Алгоритмы, их свойства, способы описания алгоритмов.**

Алгоритм - это конечный набор команд, который определяет последовательность действий для переработки входных данных в выходные.

**Свойства алгоритмов:**

1. **Детерминированность (определенность)** −ориентированность на определенного исполнителя, исключающая неоднозначность понимания.
2. **Массовость** − пригодность для решения задач определенного класса при любых допустимых значениях исходных данных.
3. **Дискретность** − пошаговый характер получения результата.
4. **Результативность** − свойство алгоритма приводить к результату за конечное время.

**Способы описания алгоритмов:**

1. **Словесно-формульный.**

Алгоритм в словесно-формульном виде представляет собой перенумерованную последовательность действий, описанных обычным языком с использованием математической символики.

1. **Графический:** 
   1. **блок-схемы;**

Блок-схема алгоритма представляет собой набор геометрических фигур (блоков), соединенных линиями или линиями со стрелками для указания направления перехода от блока к блоку

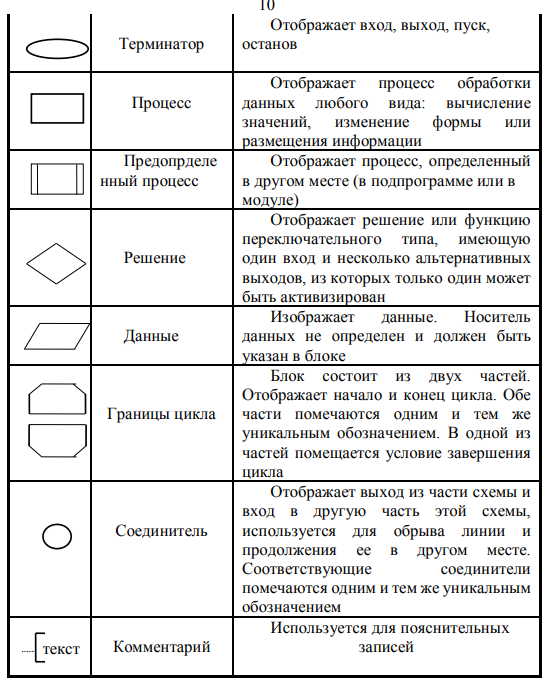
* 1. **структурограммы (диаграммы Насси − Шнейдермана).** Структурограммы изображают последовательность действий не с помощью линий перехода от блока к блоку, а в виде вложенных друг в друга фигур.

1. **Псевдокод.**

Он позволяет описывать логику программы на естественном языке, но включать типовые конструкции языка программирования, не заботясь о синтаксических тонкостях.

1. **Программный (запись на языке программирования).**

# **3. Блок-схемы алгоритмов.**

Блок-схемы алгоритмов Графическая форма записи алгоритма более наглядна, позволяет отчетливо представить все логические связи между частями алгоритма. Блок-схема алгоритма представляет собой набор геометрических фигур (блоков), соединенных линиями или линиями со стрелками для указания направления перехода от блока к блоку. Движение от блока к блоку сверху вниз или слева направо считается стандартным. В этом случае стрелки можно не указывать. Если же направление отлично от стандартного, то стрелки обязательны. Необходимая для выполнения очередного действия информация помещается в блок в виде текста или математических обозначений. Перечень блоков, их форма и отображаемые функции установлены ГОСТ 19.701-90 ЕСПД. В таблице приведены основные блоки. 

# **4. Структурное программирование. Основные принципы. Теорема о структурировании.**

**Структу́рное** **программи́рование** — парадигма **программирования**, в основе которой лежит представление программы в виде иерархической **структуры** блоков.

Принципы структурного программирования

1. Разработка алгоритма «сверху вниз» (метод пошаговой детализации). Начиная со спецификации, полученной в результате анализа задачи, выделяют небольшое число достаточно самостоятельных подзадач и описывают спецификации для каждой. Это будет первый шаг детализации. С каждой из выделенных подзадач поступают так же (второй шаг детализации) и т.д. Таким образом получается последовательность все более детальных спецификаций, приближающаяся к окончательной версии программы.

2. Модульность. Метод пошаговой детализации дает возможность разбить алгоритм на части (модули), каждая из которых решает самостоятельную подзадачу. Размеры модулей должны быть небольшими, а инструкции, входящие в состав модуля, должны давать исчерпывающее представление о действиях, выполняемых модулем. Связи по управлению между модулями осуществляются посредством обращений к ним, а обмен информацией − через параметры и глобальные переменные.

3. Каждый модуль должен иметь один вход и один выход. Это позволяет упростить стыковку модулей в сложной программе.

4. Логика алгоритма должна опираться на небольшое число достаточно простых базовых управляющих структур:

1. Следование

2. Развилка

3. Цикл: а) цикл с предусловием б) цикл с постусловием

4. Выбор из нескольких альтернатив (переключатель)

Фундаментом структурного программирования является теорема о структурировании. Она утверждает, что как бы ни сложна была задача, схема алгоритма может быть представлена с использованием ограниченного числа элементарных управляющих структур. Теорема о полноте. Базовые элементарные структуры: следование, разветвление и цикл − обладают функциональной полнотой, то есть любой алгоритм может быть реализован в виде композиции этих конструкций.

# **5. Классификация языков программирования.**

**Их создание шло по двум направлениям:**

1. **машинно ориентированные**
2. **машинно независимые (алгоритмические) языки.**

**Машинно ориентированные языки (ассемблер)** − языки низкого уровня, требующие указания мелких деталей процесса обработки данных. **Алгоритмических языков** − языков высокого уровня − в настоящее время более 500.

Каждый язык имеет свои особенности, но есть **ряд общих черт, отличающих алгоритмические языки от языков низкого уровня:**

1. Алгоритмические языки обладают большими выразительными возможностями, имея широкий алфавит, что повышает наглядность текста программ.

2. Набор операций не зависит от машинных операций, а выбирается для удовлетворения потребностей конкретной прикладной области.

3. Операции задаются в удобном виде.

4. Одним предложением можно задать значительный этап обработки данных.

5. Программным объектам, над которыми выполняются действия, присваиваются имена, и обращение к ним происходит по имени.

6. В алгоритмических языках предусмотрен широкий набор типов данных.

При использовании алгоритмических языков мы идем на издержки. Во-первых, программа должна быть оттранслирована (переведена на язык машинных кодов программой-транслятором), на это требуется время. Во-вторых, полученная после трансляции программа может быть менее эффективной, чем составленная опытным программистом на языке низкогоуровня с учетом специфики системы команд и организации памяти. Но эти недостатки окупаются удобствами для программиста.

# **6. Свойства языков программирования, характеризующие качество программ.**

1. Простота.

2. Надежность (некоторая мера отсутствия ошибок в программе).

3. Быстрота трансляции.

4. Эффективность (характеризует быстродействие и объем используемой памяти).

5. Удобочитаемость.

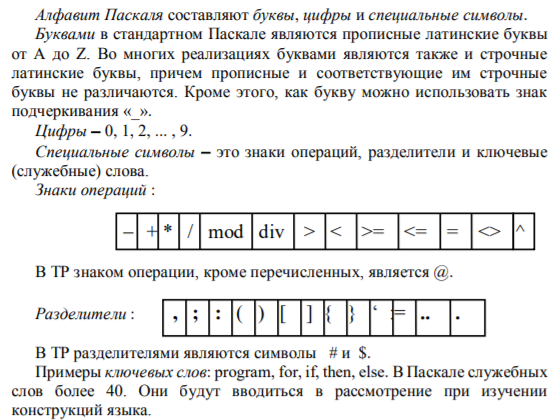
6. Модульность (возможность независимой обработки отдельных частей программы и последующего их связывания в единую систему).

# **7.** **Понятие об алфавите, синтаксисе и семантике языка программирования. Алфавит языка Паскаль.**

**Алфавит** − это фиксированный набор символов, из которых состоит текст на данном языке. Текст на языке программирования − программа.

**Синтаксис** языка программирования определяет правила построения из символов алфавита специальных конструкций, с помощью которых можно составлять алгоритмы решения задач.

**Семантика** − система правил истолкования конструкций языка



# **8. Способы описания синтаксиса**.

**Способы описания синтаксиса:**

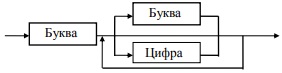
Для описания синтаксиса широко применяются:

1) **синтаксические диаграммы;**

2) **металингвистические формулы Бэкуса − Науэра.**

**Синтаксические диаграммы** напоминают структурные схемы, но с их помощью описываются не процессы обработки, а структуры данных.

В синтаксических диаграммах используются блоки двух видов: овальные и прямоугольные. В овальные блоки помещают символы, которые без изменений входят в описываемую конструкцию, а в прямоугольные помещают понятия, требующие определения или определенные ранее. Блоки соединяются стрелками. Чтобы получить правильные грамматические конструкции, нужно идти по путям, указанным стрелками, от одного блока к другому, пока не придем к выходу. Если предусмотрено более одного направления движения, можно выбрать любое. В качестве примера приведем синтаксическую диаграмму одного из важных понятий программирования − идентификатора.



Словесно это определение можно записать следующим образом. Идентификатором является последовательность букв и цифр, начинающаяся с буквы. В стандартном Паскале допускается длина идентификатора не более 8 символов, а в ТР − до 63 символов включительно.

**Метод описания синтаксиса с помощью формул Бэкуса − Науэра** заключается в использовании специальных обозначений :

::= − читается: «по определению есть»;

( вертикальная черта) − выбор, альтернатива;

{ } (фигурные скобки) − возможность повторения, что соответствует замкнутому циклу в синтаксических диаграммах;

[ ] (квадратные скобки) − необязательная часть синтаксической конструкции;

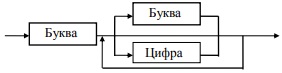
(....) (круглые скобки с вертикальной чертой) − альтернатива внутри определения;

< > в угловые скобки заключают понятия, требующие определения (метапеременные), которые в синтаксических диаграммах записывают в прямоугольных блоках. **На языке формул Бэкуса − Науэра (ЯБНФ)** определение идентификатора имеет вид <буква>{<буква><цифра>}

# **9. Понятие идентификатора в языках Паскаль и Турбо Паскаль. Определение идентификатора с помощью синтаксической диаграммы и языка БНФ**

**Идентификатор** − имя программного объекта. Программными объектами являются программы, переменные, константы, процедуры, функции, типы и метки. Каждый идентификатор является либо стандартным, либо определяемым программистом. Стандартные − это встроенные в язык идентификаторы и идентификаторы, описанные в библиотеках Паскаля. Смысл этих идентификаторов уже определен. Например, integer − имя целого типа, read − имя процедуры ввода, cos − имя функции, возвращающей косинус своего аргумента.

**Синтаксическая диаграмма:**



**На языке формул Бэкуса − Науэра (ЯБНФ)** определение идентификатора имеет вид <буква>{<буква><цифра>}

# **10. Основные характеристики языка Паскаль.**

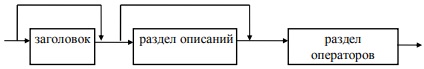
Pascal является традиционным алгоритмическим языком программирования

*Во-первых*, по своей идеологии Паскаль наиболее близок к современной методике и технологии программирования. В частности, он достаточно полно отражает идеи структурного программирования, что довольно хорошо видно даже из основных управляющих структур языка.

*Во-вторых*, Паскаль хорошо приспособлен для применения технологии разработки программ сверху-вниз (пошаговой детализации).

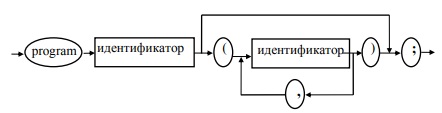
*В-третьих*, Паскаль содержит большое разнообразие различных структур данных, что обеспечивает простоту алгоритмов, а следовательно снижение трудоемкости при разработке программ.

# **11. Структура программы на языке Паскаль**.



Раздел описаний вместе с разделом операторов называют **блоком**.

**Заголовок**

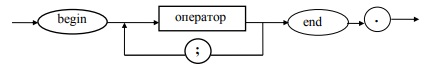
****

Идентификатор после ключевого слова **program** − имя программы. В стандартном Паскале заголовок обязателен, а также обязателен список идентификаторов в круглых скобках.

**Раздел описаний**

Все программные объекты, которые вводятся в рассмотрение программистом, должны быть описаны в разделе описаний. Нельзя использовать программный объект до его описания. В стандартном Паскале программные объекты должны описываться в следующей последовательности: метки, константы, типы, переменные, подпрограммы. В ТР они могут описываться в произвольном порядке.

**Раздел операторов**



В разделе операторов записывается алгоритм решения задачи в виде последовательности операторов. **Оператор** − законченная фраза языка, которая определяет некоторый этап обработки данных.

**Операторы можно разделить** **на две группы:**

1. простые
2. производные.

В Паскале 4 вида простых операторов:

1. оператор присваивания
2. оператор процедуры
3. оператор перехода
4. пустой оператор

а также 4 вида производных операторов:

1. составной оператор
2. выбирающий оператор
3. оператор цикла
4. оператор присоединения.

# **12. Понятие типа данных.**

*Тип данных* характеризует *множество значений,* к которым относится константа и которые может принимать переменная или выражение. Например, если переменная в некотором алгоритме может принимать только значения из множества целых чисел, то ей ставится в соответствие целый тип данных.

Типы данных принято делить на простые *(базовые)* и *структурированные.*

К основным базовым типам относятся:

• *целый* (INTEGER) — определяет подмножество допустимых значений из множества целых чисел;

• *вещественный* (REAL) — определяет подмножество допустимых значений из множества вещественных чисел;

• *логический* (BOOLEAN) — множество допустимых значений — истина и ложь;

• *символьный* (CHAR) — цифры, буквы, знаки препинания и пр.

*Структурированные типы* описывают наборы однотипных или разнотипных данных, с которыми алгоритм должен работать как с од­ной именованной переменной.

*Массив.*

*Запись.*



**13. Классификация типов данных в Паскале.**

|  |
| --- |
| К *простым типам* относятся данные, значения которых являются неделимыми. Данные *структурированных типов* представляют собой совокупности компонентов ранее определенных типов.  **Константы**  *Константа* - это программный объект, не изменяющий своего значения. Каждая константа является либо *литералом*, либо *именованной константой*.  ***Литералы*** - те константы, тип которых определяется по их виду. Литералами являются целые и вещественные числа, символы, символьные строки.  ***Именованная константа***-это фиксированное значение, которому в разделе описаний присваивается имя.  Описание констант:  Идентификатор - имя константы. Имена констант должны быть осмысленными. Использование именованных констант делает программу удобной для понимания и внесения изменений.  **Переменные**  Переменные - объекты, способные изменять свои значения.  Описание переменных:  Идентификатор - имя переменной. Каждая переменная в Паскале имеет тип, определяемый при описании, в соответствии с которым этой переменной выделяется определенный объём памяти. Секцией в разделе описания переменных называется список перечисленных через запятую переменных с указанием их общего типа. |

# **14. Классификация операторов в Паскале.**

Операторы можно разделить на две группы:

1.простые

2.производные.

В Паскале 4 вида простых операторов:

1.оператор присваивания

2.оператор процедуры

3.оператор перехода

4.пустой оператор

а также 4 вида производных операторов:

1.составной оператор

2.выбирающий оператор

3.оператор цикла

4.оператор присоединения.

**ОПЕРАТОР ПРИСВАИВАНИЯ**

****

Действие оператора присваивания заключается в вычислении значения выражения и присваивании этого значения переменной. Тип переменной в левой части оператора присваивания должен быть совместим по присваиванию с типом выражения в правой части.

**ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА**

Оператор безусловного перехода goto позволяет нарушить естественный порядок выполнения действий и передать управление из одной части программы в другую: Выполнение этого оператора заключается в передаче управления оператору, помеченному меткой. Метка − целое неотрицательное число от 0 до 9999, а в ТР метка может быть и идентификатором. Оператор, помеченный меткой, имеет вид Оператор безусловного перехода в структурном программировании не используется.

**ПУСТОЙ ОПЕРАТОР**

Пустому оператору синтаксически не соответствуют никакие символы. Например, Между оператором ввода и оператором присваивания находится пустой оператор. Синтаксис не требует разделять точкой с запятой оператор и ключевое слово еnd; если она есть, это значит, что перед еnd стоит пустой оператор.

**СТРУКТУРИРОВАННЫЕ ОПЕРАТОРЫ**

Cтруктурированные операторы − операторы, в состав которых входят другие операторы.

**Составной оператор**

В некоторых случаях синтаксис языка требует размещения одного оператора в какой-либо части конструкции, а по алгоритму там должно быть несколько операторов. Составной оператор позволяет рассматривать группу операторов как один оператор: метка : оператор ; begin оператор end ; Ключевые слова begin и end называются операторными скобками.

**Выбирающий оператор**

В Паскале два вида выбирающего оператора: условный оператор и оператор переключатель.

1. Условный оператор используется при кодировании развилок, то есть для организации бинарного ветвления. При отсутствии ветви еlse условный оператор будем называть неполным.При отсутствии операторных скобок неполным будет внешний оператор.

2. Оператор переключатель предназначен для организации множественного ветвления.

**Оператор цикла**

В Паскале 3 вида оператора цикла: цикл с предусловием, цикл с постусловием и цикл с фиксированным числом шагов.

# **15. Числовые типы Паскаля.**

**Целые типы TP**

1) **byte** − целое без знака[0; 255], занимает 1 байт;

2) **shortint** − короткое целое [−128; 127], занимает 1 байт;

3) **word** − целое, без знака, занимает машинное слово;

4) **integer** − целое со знаком, занимает машинное слово;

5) **longint** − целое со знаком размера больше или равного integer.

Все целые типы являются упорядоченными типами.

**Вещественные типы ТР**

Вещественное число, не равное нулю, можно представить в виде r= a \* 10^n , где а − мантисса, удовлетворяющая условию 1<=а<10 при r не= 0, а n − порядок. Значения вещественных типов хранятся в памяти в виде мантиссы и порядка.

1. **single** вещественный одинарной точности 4 байта; 7−8 мантиса; n[−39; 38]

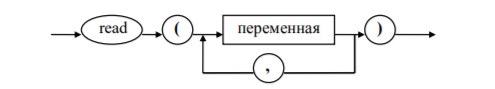
2. **real** вещественный 6 байта; 11−12 мантиса; n[−39; 38]

3. **double** вещественный двойной точности 8 байта; 15−16 мантиса; n[−324; 308]

4. **extended** вещественный расширенный 10 байта; 19−20 мантиса; n[−4932; 4932]

5. **comp** сложный 8 байта; 19−20 мантиса; n[0; 18] В

# 16. Стандартный ввод в Паскале.

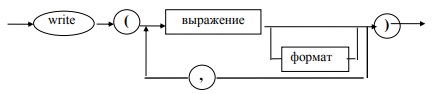
Процедуры ввода read и readLn позволяют инициализировать переменные значениями, вводимыми с клавиатуры. Синтаксическая диаграмма процедуры read

Процедура read позволяет ввести значения числовых типов, символьного и строкового. При выполнении процедуры ввода программа приостанавливает работу и ждет, пока с клавиатуры не будут введены значения для всех переменных, перечисленных в списке параметров. Набираемые на клавиатуре символы хранятся в некоторой области памяти, называемой буфером ввода. Считывание данных происходит из буфера ввода.

При вводе значений типы переменных должны быть совместимы по присваиванию с соответствующими им вводимыми значениями

# **17. Стандартный вывод в Паскале, в том числе и форматный.**

Вывод на дисплей выполняют процедуры write и writeLn. Синтаксическая диаграмма вызова процедуры write:



Процедура write вычисляет и выводит на экран значения выражений, перечисленных в качестве параметров. Выражения могут быть числовыми, символьными, логическими и строковыми. Вывод начинается с текущей позиции курсора на экране. После вывода значения курсор помещается за последним выведенным символом, и вывод следующего значения начнется с этой позиции. Никаких разделителей между выводимыми значениями не предусмотрено. Значения типа real выводятся в форме с плавающей точкой в следующем виде: (−|)<цифра>.<цифра>{<цифра>}E(+|−)<цифра><цифра> Процедура writeLn выполняется так же, как и write, но после вывода значений переводит курсор в начало новой строки. WriteLn можно использовать и без параметров для перехода к новой строке.

**Форматный вывод**

Необязательная часть конструкции оператора вывода − формат − предназначена для управления формой вывода. Формат имеет вид



Первое целое определяет ширину поля вывода − количество выделяемых значению позиций. Второе целое в формате можно использовать только при выводе вещественных значений. Если значение занимает меньше позиций, чем ширина поля вывода, то оно прижимается к правому краю поля вывода, а свободные позиции слева заполняются пробелами. Если для вывода целого, логического, символьного или строкового значения ширины поля вывода недостаточно, то поле вывода расширяется, и значение выводится без искажения.

**Вывод значений типа real**

1. Вывод в форме с плавающей точкой в виде (−|)<цифра>.<цифра>{<цифра>}E(+|−)<цифра><цифра> происходит по умолчанию (без указания формата) и в формате с одним целым. Максимальная длина выводимого значения − 17. Если в формате указано больше, то число будет выведено в 17 позициях, а свободные позиции слева заполнятся пробелами. Значение формата, меньшее 17, используется для уменьшения количества цифр после десятичной точки. Их минимальное число − 1, то есть минимальная длина выводимого значения − 8. Если целое в формате меньше 8, то считается, что оно равно 8.

# **18. Стандартные арифметические функции. Арифметические выражения. Порядок действий в арифметических выражениях.**

Стандартная функция возвращат своё значение если задано значение её аргумента.

sin(x)

cos(x)

tan(x)

arctn(x)

sqrt(x) – корень

abst(x) – модуль

exp(x) – ex

Арифметические выражения. Образуются арифметических операций соединённых знаками арифметических действий и круглыми сковками.

Приоритеты операций в арифметических выражениях (в порядке убывания):

1. Унарные (+ и −).

2. Умножение, деление, деление нацело, нахождение остатка от деления.

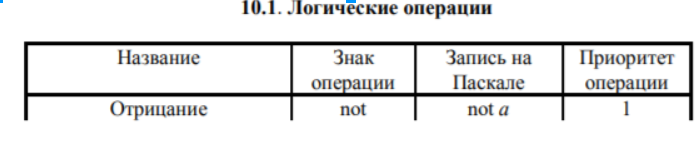
3. Сложение, вычитание.

# **19. Логический тип. Логические выражения. Порядок действий в логических выражениях.**

Имя логического типа − boolean. Множество значений: false и true.

Значение логического типа занимает 1 байт. False (ложь) и true (истина)

хранятся в памяти как 0 и 1 соответственно. Тип упорядочен: false<true



# **20. Символьный тип и символьные выражения**.

**СИМВОЛЬНЫЙ ТИП**

Имя символьного типа − char. Значениями символьного типа являются символы из определенного набора символов, который зависит от конкретной реализации языка. В этот набор входят некоторые символы алфавита и, возможно, другие символы. Все символы пронумерованы. Их номера - коды. Множество значений символьного типа является упорядоченным множеством (чем больше код, тем больше символ). В различных реализациях символы могут быть упорядочены по-разному, но обязательно выполнение следующих условий:

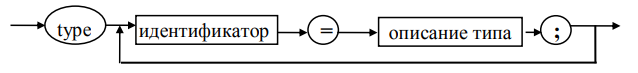
1. Коды цифр − последовательные числа и '0' < '1'< '2' < ...< '9'.

2. Прописные и строчные буквы должны быть упорядочены по алфавиту, но не требуется, чтобы коды были последовательными числами. В ТР множеством значений символьного типа является множество символов из расширенной таблицы ASCII, состоящей из 256 символов. Первая половина таблицы (с кодами от 0 до 127) − неизменяемая, вторая (с кодами от 128 до 255) − альтернативная. Символы с кодами меньше 32 являются управляющими, пробел имеет код 32. Цифры, латинские буквы, некоторые знаки операций, разделители находятся в первой половине таблицы. Важная особенность: строчные латинские и прописные латинские буквы имеют последовательные коды. Альтернативная часть таблицы содержит буквы русского алфавита, символы псевдографики. Над данными символьного типа не определены никакие операции, кроме операций сравнения. Поэтому выражениями символьного типа являются константы, переменные и функции, возвращающие значения типа char. **Стандартные функции**, используемые при работе с символами: chr(i) возвращает символ по его коду i ord(c) возвращает код символа с Примеры:

Значением выражения chr(ord(‘A’)+5) является шестая буква латинского алфавита.

# **21. Скалярные типы, определяемые программистом.**

Любой тип, вводимый в рассмотрение пользователем, можно описать непосредственно при описании переменных или этому типу можно присвоить имя в разделе описания типов, а затем использовать имя типа при описании переменных. Для того чтобы отличать имена типов от имен других программных объектов, можно начинать имя типа с префикса t\_.

Раздел описания типов: 

Здесь индентификатор − имя типа. Описание типа определяется правилами языка для каждого вида типов. Из простых типов, определяемых пользователем, рассмотрим интервальные и перечисляемые. Указательный тип будет рассмотрен ниже.

11.1. Интервальный тип (тип диапазон)

Описание интервального типа:



Константные выражения (границы диапазона) должны принадлежать одному и тому же упорядоченному типу, который является базовым для вводимого типа. Значение первого константного выражения не может превышать значения второго константного выражения. Символ алфавита «..» называется индикатором диапазона. Множество значений интервального типа составляют значения базового типа, принадлежащие диапазону. Интервальные типы можно использовать всюду, где допустим соответствующий базовый тип.

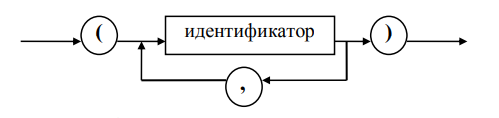
Примеры: type t\_month = 1..12; {Интервальный тип на базе целого}

t\_letter = ‘A’..’Z’; {Интервальный тип на базе символьного}

var m1, m2: t\_month; {Переменные интервального типа}

digit: ‘0’..‘9’; {Переменная интервального типа на базе символьного}

11.2. Перечисляемый тип

Описание перечисляемого типа:

Каждый идентификатор в описании перечисляемого типа является константой определяемого типа.

Например, Type t\_season = (Winter, Spring, Summer, Autumn);

Определен перечисляемый тип из четырех значений. Перечисляемый тип является упорядоченным, порядок определяется перечислением. В памяти значения представлены кодами, нумерация начинается с нуля. Данные перечисляемых типов, как и всех рассмотренных выше типов, можно сравнивать. Никакие другие операции над данными перечисляемых типов не определены. Для ввода и вывода данных перечисляемых типов, определяемых программистом, нельзя использовать стандартные процедуры read и write. Ввод и вывод таких данных осуществляется программно.

Использование перечисляемых типов улучшает смысловую читаемость программы.

В стандартной библиотеке есть ряд функций для работы с любыми упорядоченными типами, в том числе и перечисляемыми. Р

ассмотрим некоторые из них:

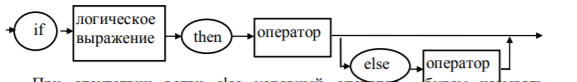
**ord(n)** − возвращает код значения аргумента n (для целочисленных типов код значения − само число).

**pred(n) и succ(n)** − возвращают соответственно значение, предшествующее аргументу и следующее за аргументом в упорядоченной последовательности значений типа аргумента.

Если соответствующего элемента в последовательности не окажется, то произойдет ошибка времени выполнения.

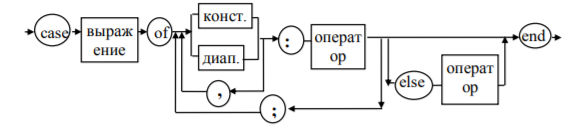
Будем называть перечисляемыми типами упорядоченные типы, нумерация элементов которых начинается с нуля. Согласно такому определению, стандартные типы boolean, char, byte, word являются перечисляемыми.

# **22. Организация ветвлений в Паскале.**

1. Условный оператор используется при кодировании развилок, то есть для организации бинарного ветвления. 

При отсутствии ветви еlse условный оператор будем называть неполным.

После ключевых слов then и еlse синтаксис требует наличия только одного оператора. Если же по какой-либо ветви нужно выполнить несколько операторов, то следует использовать составной оператор. Перед еlse точка с запятой недопустима, так как в этом случае между then и else будут два оператора, один из которых пустой.

2. Оператор переключатель предназначен для организации множественного ветвления. Синтаксическая диаграмма переключателя:

Выражение после ключевого слова case называется селектором. Селектор должен иметь упорядоченный тип, кроме типа longint. Константы называются метками случаев. они должны относиться к тому же типу, что и переключатель. Работа оператора сase заключается в следующем. Вычисляется значение селектора. Если это значение совпадает с одной из меток случаев, то выполняется оператор, записанный после нее. Если значение селектора не совпало ни с одной из меток случаев, то выполняется оператор, следующий за еlse. Если ветвь еlse отсутствует, то управление передается оператору, следующему за переключателем. Диапазоны и метки случаев не должны пересекаться.

# **23. Организация циклов в Паскале**

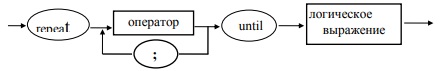
В Паскале 3 вида оператора цикла: цикл с предусловием, цикл с постусловием и цикл с фиксированным числом шагов.

1. **Цикл с предусловием:**



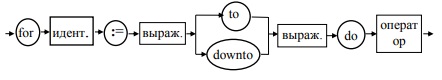
Логическое выражение в заголовке цикла является условием возобновления цикла, то есть, пока оно истинно, выполняется оператор − тело цикла. Если при входе в цикл логическое выражение ложно, то тело цикла не выполнится ни разу.

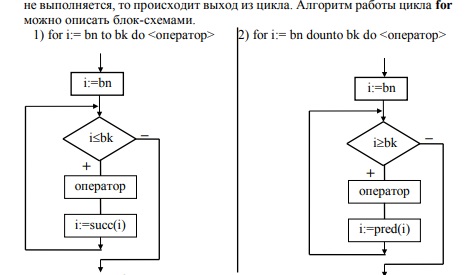
2. **Цикл с постусловием:**



Телом цикла с постусловием является последовательность операторов между ключевыми словами repeat и until. Логическое выражение представляет собой условие выхода из цикла, то есть тело цикла выполняется до тех пор, пока значение логического выражения ложно. Как только оно станет истинным, происходит выход из цикла. Так как условие выхода из цикла проверяется после выполнения тела цикла, тело цикла с постусловием выполняется хотя бы один раз.

3. **Цикл с фиксированным числом шагов:**



− В приведенной диаграмме оператор представляет собой тело цикла. Перед телом цикла − заголовок цикла. Идентификатор после ключевого слова for называется **управляющей переменной или параметром цикла.** Управляющая переменная должна иметь упорядоченный тип. Значения начального и конечного выражений вычисляются один раз при входе в цикл. 

В теле цикла for нельзя изменять параметр цикла. Так как начальное и конечное выражения вычисляются только один раз при входе в цикл, то изменение bn и bk на количество повторений не влияет. После выхода из цикла for значение параметра цикла считается неопределенным.

# **24. Одномерные массивы, их описание и использование**.

Если указан только один тип индекса, массив является одномерным.

Примеры описаний одномерных массивов:

const MAXLEN=100;{константа удобна для изменения размера массива}

type t\_vect =array[1.. MAXLEN] of integer;

var a : t\_vect; {для хранения целочисленных последовательностей}

b : t\_vect;

c, d : array[1.. MAXLEN] of integer;

point : array[(x, y, z)] of real; {для хранения координат точки или вектора} flag : array[‘A’..‘Z’] of boolen;

Массивы a, b, c, d имеют одинаковые размеры (=100). Размер массива point равен 3, flag − 26.

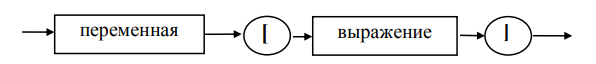
Компоненты (элементы) массива занимают последовательные ячейки памяти. Объем памяти, выделяемой массиву, равен произведению размера массива на объем, занимаемый элементом. Переменная а занимает 100\*2=200 байтов, point − 3\*6=18 байтов, flag − 26\*1=26 байтов.

Следует обратить внимание на то, что типы переменных a и с компилятор считает разными. Важным понятием в Паскале является понятие тождественности типов. Переменные имеют тождественные типы, если они определены в одной секции или через одно и то же имя типа.

Согласно приведенным выше описаниям, типы переменных a и b тождественны, так как переменные описаны с использованием имени типа t\_vect. Переменные с и d тоже имеют тождественные типы, так как описаны в одной секции.

Над массивами любой размерности как над едиными целыми не определены никакие операции. Разрешено присваивание переменной типа массив значения переменной тождественного типа, то есть для совместимости массивов по присваиванию требуется тождественность типов.

Если массивы a и d инициализированы, то допустимы операторы b:=a и c:=d.

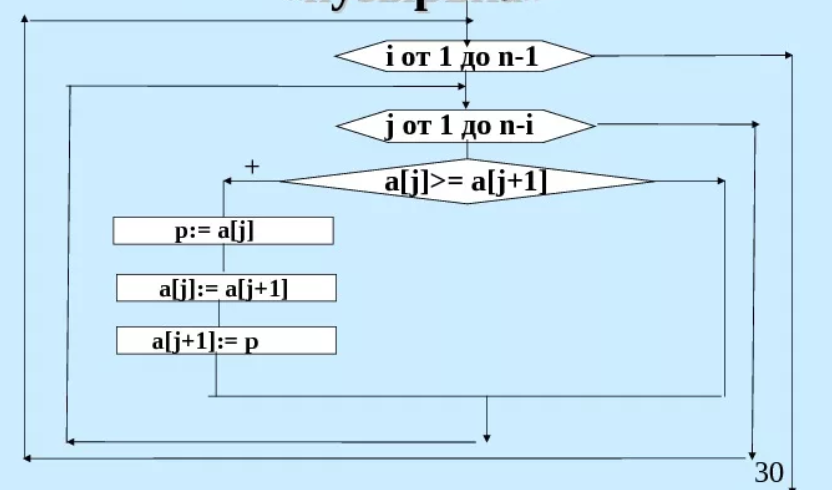
Имя массива является общим именем совокупности компонентов (элементов) массива. Обращение к отдельным компонентам массива осуществляется с помощью переменных с индексами: 

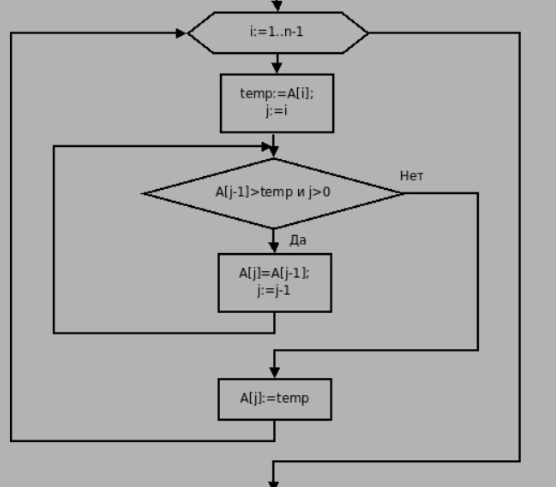
Выражение в квадратных скобках должно иметь тип индекса. Например, a[1], d[MAXLEN div 2], point[x], point[y], flag[‘X’], flag[succ(‘G’)].

Переменные с индексами можно использовать в любом месте программы, где допустим базовый тип. Ввод и вывод массива производится покомпонентно, обычно в цикле for.

# **25. Алгоритмы сортировки массивов.**

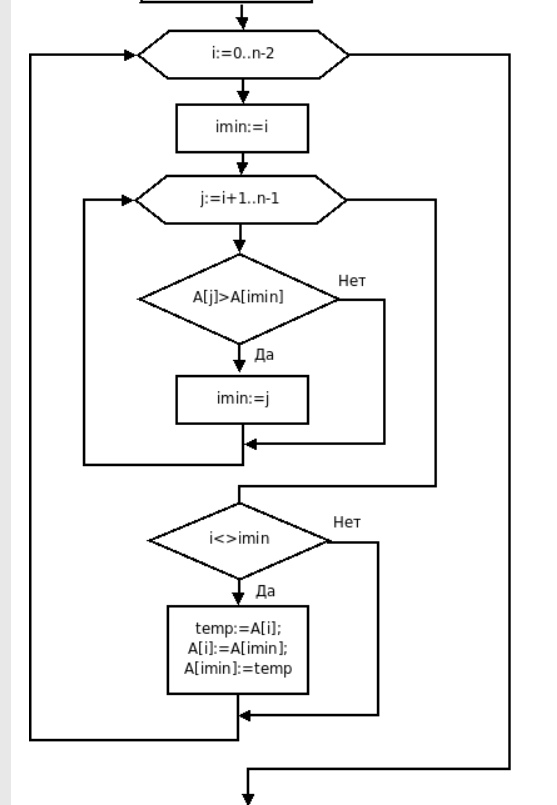
**Алгоритм** **сортировки** — это **алгоритм** для упорядочивания элементов в списке.

алгоритм сортировки пузырьком : Принцип действий прост: обходим массив от начала до конца, попутно меняя местами неотсортированные соседние элементы. В результате первого прохода на последнее место «всплывёт» максимальный элемент. Теперь снова обходим неотсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. И так далее 

Сортировка вставками (*Insertion Sort*) — это простой алгоритм сортировки. Суть его заключается в том что, на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем.

сортировка выбором -

1. берем первый элемент последовательности A[i], здесь i – номер элемента, для первого i равен 1;
2. находим минимальный (максимальный) элемент последовательности и запоминаем его номер в переменную key;
3. если номер первого элемента и номер найденного элемента не совпадают, т. е. если key≠1, тогда два этих элемента обмениваются значениями, иначе никаких манипуляций не происходит;
4. увеличиваем i на 1 и продолжаем сортировку оставшейся части массива, а именно с элемента с номером 2 по N, так как элемент A[1] уже занимает свою позицию;

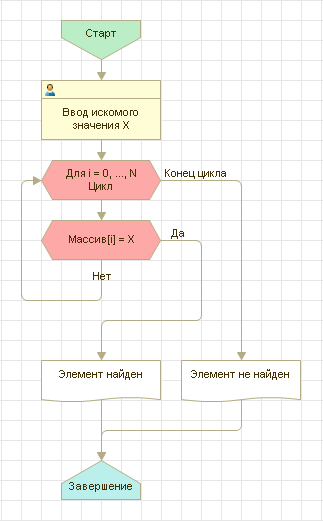


# **26. Алгоритмы поиска в одномерных массивах упорядоченных и неупорядоченных.**

## **Алгоритм линейного поиска**

**Линейный поиск** используется, если нет никакой дополнительной информации об организации элементов набора данных. Пусть задан некий массив неупорядоченных значений. Алгоритм заключается в последовательном сравнении элементов набора данных ***Массив[i]*** с искомым значением – ***X***. Сравнение производится до тех пор, пока не будет найден искомый элемент или не будут просмотрены все элементы набора данных. Если очередной элемент совпадает с ***X***, то задача решена и нужно вывести номер и значение найденного элемента. Соответственно алгоритм имеет два условия окончания поиска:

* искомый элемент найден, ***Массив[i] = Х*** ;
* весь набор данных просмотрен, совпадений не обнаружено.



Основой данного алгоритма служит цикл с условием выхода. В цикле каждый элемент ***Массив[i]*** сравнивается с ***X***. Если найден элемент, для которого выполняется условие ***Массив[i] = X***, то элемент с индексом ***i*** считается найденным.

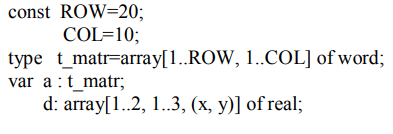
В том случае, если элементы последовательности ***Массив[i]*** имеют повторяющиеся значения, то согласно блок-схеме, приведенной на рисунке, будет найден элемент с минимально возможным индексом, для которого выполняется условие ***Массив[i] = X*** , т.е. это первый из таких элементов.

**Бинарный поиск** осуществляется на упорядоченном наборе данных, то есть значения элементов набора данных возрастают (убывают) с увеличением номера элемента.

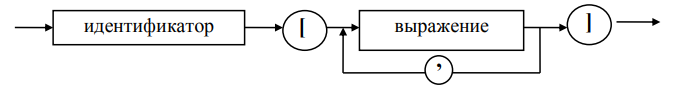


# **27. Многомерные массивы. Их описание и использование.**

Если количество типов индекса в описании массива равно n, то массив называют n-мерным. Формально размерность массива не ограничена, но фактически она зависит и от размера базового типа, и от ограничения на объем памяти структурированных типов.

Для представления матриц используются двумерные массивы. Тип первого индекса определяет число строк, а тип второго индекса − число столбцов матрицы. Например, 

Тип t\_matr − двумерный массив размером 20х10. Его можно использовать для работы с матрицами порядка не больше чем 20x10. Переменная d − трехмерный массив.

Обращение к элементам многомерных массивов: 

Идентификатор − имя массива. Число выражений (индексов) в квадратных скобках должно быть равно размерности массива. Например, a[3, 4], d [2, 1, y].

# **28. Строковый тип. Действия со строками.**

В ТР есть специальный тип для работы со строками переменной длины. Описание этого типа:

*Строкой* называется последовательность символов. Для работы со строками можно использовать упакованные символьные массивы с типом индекса 1..n

Использование упакованных символьных массивов дает некоторые преимущества по сравнению с обычными одномерными массивами:

1. Для совместимости по присваиванию не требуется тождественность типов, достаточно равенства размеров массивов.

2. Строковой переменной можно присвоить строковую константу такой же длины.

3. Определены операции сравнения строк одинаковой длины.

*Из двух строк равной длины та больше, у которой первый из неравных символов больше.*

Строковый тип ТР совместим по присваиванию:

1) с любым типом string;

2) c упакованным символьным массивом (равенство длин не требуется);

3) с символьным типом.

**Действия со строками**

**chr(m: byte): char;**

* функция, возвращающая символ по коду, который равен значению выражения m.

**ord(m: char): byte;**

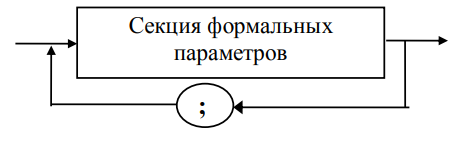
* функция, возвращающая код символа m.

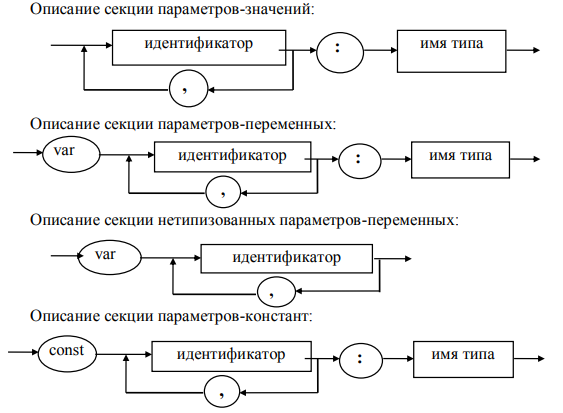
# **29. Подпрограммы, их назначение, виды подпрограмм**.

Один из принципов структурного программирования заключается в разбиении задачи на достаточно самостоятельные подзадачи. Для решения каждой подзадачи можно разработать свой алгоритм и записать его в виде самостоятельного программного блока, называемого **подпрограммой.** Подпрограммы имеют имена, и обращаться к ним можно по именам. Подпрограммы позволяют избегать многократной записи одной и той же последовательности действий. В языке Паскаль существует **два вида подпрограмм: процедуры и функции.** Программа, содержащая подпрограммы, проще для создания, легче для понимания, удобнее для использования. При разработке программ, содержащих подпрограммы, необходимо уметь описывать подпрограммы и обращаться к ним. Подпрограммы, как и другие программные объекты, должны быть описаны в разделе описаний. Структура описания подпрограммы такая же, как и структура программы, но в описании подпрограммы заголовок обязателен и описание подпрограммы заканчивается символом ‘;’, а не точкой.

# **30. Параметры подпрограмм.**

Обмен информацией между подпрограммой и вызывающей ее программой осуществляется через параметры и глобальные переменные. Но в подпрограммах не рекомендуется использовать глобальные переменные. В этом случае изменение программы может привести к неожиданным последствиям при выполнении подпрограммы, и наоборот.

Программы и подпрограммы будут максимально независимыми, если обмен информацией происходит в четко определенных границах, только через параметры. Передаваемые подпрограмме параметры описываются в списке формальных параметров заголовка подпрограммы, который имеет вид

В стандартном Паскале существует 2 вида формальных параметров: параметры-значения и параметры-переменные. В ТР, кроме этого, есть нетипизованные параметры-переменные и параметрыконстанты.

Следует обратить внимание на то, что при описании формальных параметров указывается имя типа. В последних версиях ТР, кроме имени типа, можно использовать конструкцию вида



Формальные параметры такого типа называются открытыми массивами В Паскале параметры любого вида доступны подпрограмме для чтения. Для записи доступны параметры-переменные (в том числе и нетипизованные). Как бы мы ни изменяли параметр-значение в теле подпрограммы, после выхода из подпрограммы он будет иметь первоначальное значение. Поэтому параметры-значения после использования переданной через них информации можно использовать как локальные переменные. Значения параметров-констант изменять нельзя. Значения, присвоенные параметрампеременным в подпрограмме, сохраняются после выхода из подпрограммы.

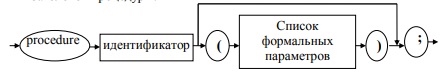
# **31. Глобальные и локальные программные объекты, область видимости и время жизни глобальных и локальных переменных.**

***Глобальными*** называют объекты, объявление которых дано вне функции. Они доступны (видимы) во всем файле, в котором они объявлены. В течение всего времени выполнения программы с глобальным объектом ассоциирована некоторая ячейка памяти.

***Локальными*** называют объекты, объявление которых дано внутри блока или функции. Эти объекты доступны только внутри того блока, в котором они объявлены. Объектам с локальным временем жизни выделяется новая ячейка памяти каждый раз при осуществлении описания внутри блока. Когда выполнение блока завершается, память, выделенная под локальный объект, освобождается, и объект теряет своё значение.

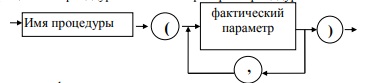
# **32. Процедуры, их описание и использование.**

Процедура − это подпрограмма, предназначенная для решения подзадачи, необязательно связанной с вычислением значений. Процедуры можно использовать для ввода или вывода данных структурированных типов, вычисления нескольких значений, преобразования данных и т.п. Заголовок процедуры:



Идентификатор после ключевого слова **procedure** − имя процедуры.

Обращение к процедуре является оператором процедуры и имеет вид:



Количество фактических параметров должно равняться количеству формальных параметров.

**Формальные параметры**-значения должны быть совместимы по присваиванию с соответствующими им фактическими параметрами. Последние представляют собой выражения, значения которых присваиваются формальным параметрам при вызове процедуры. Фактический параметр, соответствующий формальному параметру переменной, представляет собой переменную, тип которой тождественен типу формального параметра. Фактические параметры в списке разделяются запятыми.

Если параметр должен быть доступен подпрограмме для чтения, то он может быть описан и как параметр-переменная, и как параметр-значение, но использование параметров-значений простых типов имеет преимущества:

1) менее жесткое ограничение на типы − требуется только совместимость по присваиванию, а не тождественность;

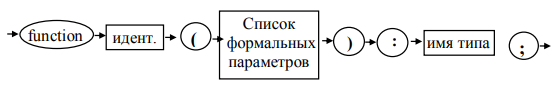
2) фактический параметр является выражением, необязательно переменной;

3) информация не может быть испорчена через этот параметр, то есть даже если этот параметр является переменной, то его значение при выходе из подпрограммы остается тем же, что и при входе; ( ) Имя процедуры фактический параметр

4) параметры-значения в теле подпрограммы можно использовать как локальные переменные. Во время обращения к подпрограммам с фактических параметровзначений снимаются копии, они помещаются в некоторую область памяти, называемую стеком, и подпрограмма работает с этими копиями. К параметрам-переменным и параметрам-константам подпрограмма обращается по адресам, в стек помещаются адреса параметров. Поэтому, если подпрограмме нужен только для чтения параметр структурированного типа, занимающий большой объем памяти, то рекомендуется его передавать как параметр-константу. Все изменения параметров-переменных в теле подпрограммы сохраняются при возвращении в вызывающую программу.

# **33. Функции, их описание и использование.**

Функция − это подпрограмма, в результате выполнения которой происходит вычисление значения скалярного или строкового типа.

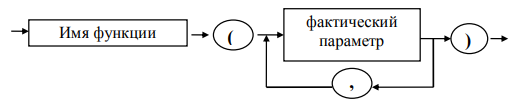
Заголовок функции имеет вид 

В разделе операторов описания функции должен быть хотя бы один оператор присваивания:



Значение последнего выражения, присвоенного имени функции, будет возвращено в точку вызова функции. Но использовать имя функции в качестве переменной, которой присвоено значение, недопустимо.

Синтаксически вызов функции не отличается от вызова процедуры:



Но вызов функции не является оператором. Обратиться к функции можно всюду, где допустимо использование значения типа, возвращаемого функцией. Обращение к функции может быть операндом в выражении, фактическим параметром-значением.

# **34. Побочный эффект функции.**

Основное назначение функции − возвращение значения в точку вызова, но функция так же, как и процедура, может изменять значения своих параметров-переменных. Такие функции называются функциями с побочным эффектом.

# **35. Рекурсивные и взаимно рекурсивные подпрограммы.**

В математике рекурсией называется способ описания функций или процессов через самих себя.

В некоторых языках программирования, в том числе и в Паскале, допустимо, чтобы подпрограмма вызывала себя. Такие подпрограммы называются рекурсивными. Рекурсивная подпрограмма обязательно удовлетворяет двум требованиям:

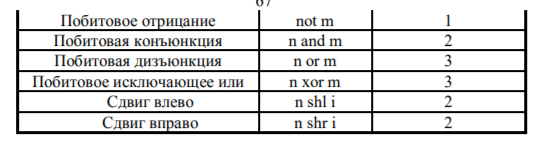
1) имеет нерекурсивный выход;

2) при каждом рекурсивном обращении задача упрощается, приближаясь к нерекурсивному решению.

При решении некоторых задач можно использовать как рекурсивный, так и итеративный алгоритм. Преимущество рекурсивных подпрограмм заключается в простоте написания, легкости понимания. Обычно это касается задач, связанных с процессами, рекурсивными по своей природе. Но рекурсивные алгоритмы, как правило, менее эффективны из-за затрат времени на организацию стека. При каждом обращении в стеке запоминаются значения всех локальных переменных, параметров и коды возврата. Глубина рекурсии (количество обращений к себе) ограничена объемом памяти стека. При глубокой вложенности рекурсии может произойти переполнение стека.

Две подпрограммы называются **взаимно рекурсивными**, если первая подпрограмма обращается ко второй, а вторая − к первой.

# **36. Побитовые операции.**

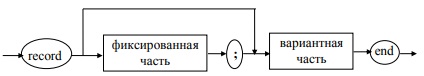


# **37. Комбинированный тип. Его характеристика, описание и использование.**

|  |
| --- |
| Комбинированный тип данных (запись) - структурированный тип, состоящий из компонентов, типы которых могут быть различны. Компоненты записей называются полями. Записи удобно использовать для хранения и обработки различных характеристик одного и того же объекта. Записи - основная структура данных, используемая в информационных системах.  Описание записи:  Идентификаторы - имена полей. Типы полей могут быть любыми.  Над записями, как едиными целыми, не определены никакие операции. Совместимость по присваиванию требует тождественности типов. Для описанных выше переменных допустимы присваивания:  Обращение к полю записи представляет собой составное имя:  Составное имя можно использовать везде, где допустим тип поля.. Имена полей записей могут совпадать с именами других переменных, при этом путаницы не возникает, так как обращение к ним иное..  **Оператор присоединения**  В Паскале есть возможность обращаться к полям записи без указания имени записи. Такую возможность дает оператор присоединения:  Перечисляемые в заголовке оператора присоединения идентификаторы являются именами записей. Оператор после заголовка - тело оператора присоединения. Если в теле оператора присоединения необходимо обратиться к переменной с именем, совпадающим с именем поля записи, указанной в заголовке, то используется составное имя: |

# **38. Записи с вариантами. Назначение, описание и использование.**

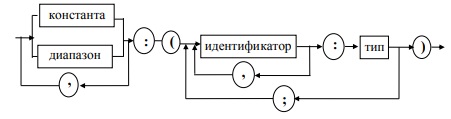
Часто необходимы записи с близкими, но не одинаковыми структурами. Для таких случаев используется тип-запись с вариантами:



Вариантная часть:



Альтернатива:

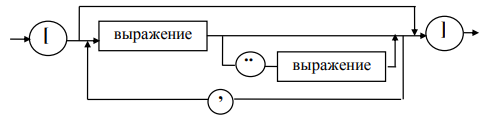


В круглых скобках − список полей альтернативы (это ветвь вариантной части). Для всех ветвей вариантной части выделяется одна и та же область памяти, где могут разместиться поля любой ветви. Идентификатор (дискриминант) определяет, какая из ветвей в данный момент активна. Но система не контролирует, к активным ли полям обращается программа.

# 39. Тип множество. Характеристика, описание и использование.

Описание типа множество:

Значениями типа множество являются все подмножества базового типа. Базовый тип играет роль универсального множества.

Число элементов базового типа множества ограничено. Ограничение определяется реализацией языка. В ТР число элементов базового типа не должно превышать 256. Базовый тип является перечисляемым типом или поддиапазоном перечисляемого типа. Кроме того, объем памяти, занимаемой значением базового типа, − один байт. Конструкция 

определяет значение типа множество. При этом значения всех перечисленных выражений принадлежат одному типу. Порядок перечисления выражений и диапазонов произвольный. Более того, в списке могут быть выражения с одинаковыми значениями.

# **40. Инициализация переменных при их описании.**

|  |
| --- |
| Тип переменных определяется пользователем в разделе описания переменных:В откомпилированной программе для всех переменных отведено место в памяти, и всем переменным присвоены нулевые значения.Для задания начальных значений переменным (инициализации перемен- ных) TURBO PASCAL позволяет присваивать начальные значения переменным одновременно с их описанием. Для этого используется конструкцияимя переменной: тип = значение;которая должна быть размещена в разделе описания констант |

# **41. Виды ошибок. Отладка и тестирование программ.**

Выделяют три типа ошибок: ошибки компиляции, ошибки времени выполнения и логические ошибки. Ошибки компиляции являются синтаксическими ошибками. Они выдаются при компиляции программы. Программа с синтаксическими ошибками не может быть выполнена. Ошибки времени выполнения связаны с невозможностью по какой-либо причине выполнить очередное действие, что приводит к аварийному останову программы. Такие ошибки возникают, например, при делении на 0, вычислении логарифма с отрицательным аргументом, переполнении стека при выполнении рекурсивной подпрограммы и т.п. Логические ошибки возникают при неправильном проектировании алгоритма или по невнимательности при записи или наборе программы. Программа с логическими ошибками может быть выполнена до конца, возможно даже, что при некоторых наборах исходных данных результаты будут правильными.

В процессе устранения логических ошибок можно выделить три этапа: 1. установление факта существования ошибки;

2. локализация ошибки;

3. устранение ошибки.

Существуют 2 метода обнаружения ошибок:

1. статическая (ручная) проверка, которая заключается в анализе программы без выполнения ее на ЭВМ;

2. тестирование − прогон на ЭВМ.

Оба метода требуют наборов тестовых данных, которые должны подбираться параллельно с разработкой алгоритма. В число наборов тестовых данных рекомендуется включить несколько типичных, среди них должны быть корректные и некорректные данные. Тестовые данные должны охватывать предельные случаи. Если в программе есть разветвления, то необходимы наборы тестовых данных для каждой ветви. Если в программе есть циклы, то данные нужно подобрать так, чтобы цикл выполнялся минимально и максимально возможное число раз. При отладке программ для практического использования нужно подобрать дополнительные тестовые данные: 1. Получить реальные данные у потенциального пользователя.

2. Породить случайным образом наборы тестовых данных.

**ОТЛАДКА ПРОГРАММ**

Для успешного выполнения программы необходимо выполнение двух требований:

1. Программа должна быть правильной. Правильность программы определяется степенью соответствия формальным требованиям, то есть спецификации программы. 2. Входные данные должны быть правильными. Процесс нахождения и устранения ошибок в программе называется отладкой. Процесс поиска ошибок во входных данных называется проверкой корректности входных данных.

Нужно руководствоваться правилом: чем раньше обнаружена ошибка, тем легче ее исправить. Поэтому при разработке алгоритма и записи программы необходимо следить за инициализацией всех переменных в программе, необходимо устанавливать правильное завершение циклов, избегать зацикливания. Какой бы простой ни была программа, рекомендуется выполнить трассировку с записью результатов на каждом шаге. Статистика утверждает, что 70% ошибок можно устранить на этапе ручного тестирования.

**Машинное тестирование**

Известный специалист в области программирования Дейкстра считает, что тестирование доказывает наличие ошибок, а не их отсутствие. Выделяют два вида тестирования: разрушающее и диагностическое:

1. Разрушающее тестирование производится над программой, которая считается правильной, с целью заставить ее дать сбой.

2. Диагностическое тестирование выполняется с целью локализации ошибки, если известно о ее существовании. В ТР существуют системные средства для отладки программ. Они позволяют выполнить трассировку с выводом значений интересующих переменных и выражений, устанавливать точки останова, точки прерывания (контрольные точки).

# **42. Стиль записи программ.**

При создании понятных, удобочитаемых программ большое значение имеет стиль записи программы. Стандарта для стиля записи нет, но можно дать некоторые рекомендации:

1. Основные части программы разделять пустыми строками.

2. Служебные слова, которыми начинается и заканчивается оператор, располагать одно под другим, начиная с одной позиции по вертикали.

3. Операторы одного уровня вложенности начинать с одной позиции по вертикали.

4. Вложенные операторы записывать с некоторым смещением вправо.

5. Включать в программу комментарии.

**ВАЖНОЕ ДОПОЛНЕНИЕ**

**Операнд** - аргумент операции.

**Оператор** - это команда или набор команд.

**Подпрограмма** - это программа, которую основная программа вызывает для совершения каких либо действий.

**Формальные** переменные определяются в заголовке

**Фактические**, которые после заголовка.

**Множество** - это конечная совокупность элементов, принадлежащих некоторому типу данных.

**Запись** - структурированный комбинированный тип данных состоящий из фиксированного числа компонентов разного типа.