# Синтаксис и семантика языков программирования. Алфавит языка Delphi Pascal. Описание синтаксиса языка: синтаксические диаграммы.

**Синтаксис языка** - совокупность правил, определяющих допустимые конструкции (слова, предложения) языка, его форму.

**Семантика языка** – совокупность правил, определяющих смысл синтаксических конструкций языка.

Описание синтаксиса языка включает определение алфавита и правил построения различных конструкций языка

**Синтаксические диаграммы** отображают правила построения конструкций более наглядно.

**Алфавит Borland Pascal** содержит: строчные и прописные буквы латинского алфавита, символ подчеркивания \_, цифры 0..9, специальные знаки, состоящие из одного или двух символов(<>-+:=), служебные слова(and, or, begin, interface…).

# Представление данных в Delphi Pascal: константы и переменные. Классификация скалярных типов данных, операции над ними, совместимость типов данных и операции преобразования типов.

**Константы** определяются один раз и не изменяются во время выполнения программы.

Используют следующие типы констант:

1. Целые и вещественные десятичные числа
2. Шестнадцатеричные числа($64)
3. Логические константы
4. Символьные константы(‘a’, #65)
5. Строки символов
6. Конструкторы множеств
7. Нулевой адрес nil.

Константы используются в двух формах:

* + - 1. **Литерал** – значение константы, записанное непосредственно в программе.
      2. **Поименованные константы** объявляются в инструкции раздела описаний const. Обращение к ним осуществляется по идентификатору.

**Переменные** – поименованные значения, которые могут изменяться в процессе выполнения программы. При объявлении переменной указывается ее идентификатор и тип.

Наложение используется **absolute**:

1. Наложение по абсолютному адресу: var a: word absolute $0000:$00FF; - первое число – адрес сегмента, второе – смещение.
2. Наложение на ранее определенную переменную: var c:byte; a:real absolute c;

Классификация:

1. Структурный:
   1. Массив
   2. Строка
   3. Запись
   4. Множество
   5. Файл
   6. Указатель
2. Простой-вещественный (со знаком):
3. Вещественный Real 6 байт
4. Одинарной точности Single 4 байт
5. Двойной точности Double 8 байт
6. Расширенный Extended 10 байт
7. Большое целое Comp 8 байт
8. Порядковый:
9. Перечисление var d:(..);
10. Отрезок type date=1..31; var d:date;
11. Стандартный:
12. Целое Integer - 2 байт (со знаком)
13. Короткое целое Shortint - 1 байт (со знаком)
14. Длинное целое Longint - 4 байт (со знаком)
15. Слово Word - 2 байт (без знака)
16. Байт Byte - 1 байт (без знака)
17. Булевский Boolean - 1 байт
18. Символ Char - 1 байт

Операции:

1. Арифметические операции
2. Операции отношения
3. Логические операции
4. Поразрядные операции
5. Строковая операция
6. Операции над множествами
7. Операция над указателями

Приоритет:

1. (@,not)
2. (\*,/,div,mod,and,shr,shl)
3. (+,-,or,xor)
4. (<,>,<>,=,<=,>=,in)

**Совместимые типы** – это совпадение двух типов

Когда типы не совпадают, то есть возможность преобразовать один тип в другой.

Выделают два вида:

1. Явный
2. Неявное

# Основные операторы Delphi Pascal: присваивание, условный оператор, оператор выбора и операторы циклов. Синтаксис операторов, их особенности и примеры использования.

С помощью **оператора присваивания** в программе изменяются значения переменных.

* Оператор условной передачи управления **if**
* Оператор выбора **switch-case**
* Операторы организации циклической обработки. В теории программирования выделяют несколько основных видов циклов: цикл-пока, цикл-до, счетный цикл

1. Цикл-пока. (**While**)
2. Цикл-до. (**repeat**)
3. Счетный цикл. (**For**) Есть **to**-следующее значение порядкового типа, **downto**-предыдущее.

# Структурные типы данных Delphi Pascal: массивы, строки. Описание, операции над структурами и их элементами.

**Массив -** упорядоченная совокупность однотипных данных. Тип элементов массива – любой допустимый кроме файлов. Суммарная длина массива не должна превышать 65537 байт. Для инициализации массивов используются типизированные константы.

Над массивами в целом определена только операция присваивания. Различают прямое и косвенное задание индексов.

**Строка –** тип, приспособленный для обработки символьной информации. Максимальная длина строки 255 байт. Var s1: string[40];

Нулевой элемент строки содержит её текущую длину.

Над переменными строкового типа можно осуществлять:

1. Операции доступа к символам
2. Операции присваивания
3. Сцепления (конкатенации) «привет»+ «мир»
4. Отношений. Операции отношения позволяют сравнивать строки на отношение равенства (=), неравенства (<>), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=). В результате сравнения двух строк получается логическое значение (true или false).

# Структурные типы данных Delphi Pascal: множества, записи. Описание, операции над структурами и их элементами.

**Множества –** неупорядоченная совокупность неповторяющихся объектов. Количество элементов от 0 до 255. **type letter=set of ‘a’..’z’;**

Операции над множествами:

1. a+b – объединение
2. a-b – дополнение
3. a\*b – пересечение
4. c in a – проверка вхождения элемента в множество
5. a>=b – b принадлежит a
6. a<=b – a принадлежит b

**Запись –** структура данных, состоящая из фиксированного числа разнотипных компонент, называемых полями записи.

Запись с фиксированными полями:

Var zap1: record

f:real;

a,b:integer;

end;

Запись с вариантами:

type figure=(s,t,c);

paramf=record

x,y:real;

case

fig:figure of

s:(side:real);

t:(side1,side2,side3:real);

c:(radius:real);

end;

var param:paramf;

Операции над записями:

1. Доступ к полям записи. В случае, если доступ к полям записи осуществляется многократно, целесообразно обращаться к полям записи с использованием оператора присоединения with.
2. Присваивание записей. Операция возможна при совпадении типов записей.

# Процедуры и функции Delphi Pascal. Способы передачи данных в подпрограмму. Локальные и глобальные переменные, законы «видимости» идентификаторов.

Процедуры и функции представляют собой относительно самостоятельные фрагменты программы, соответствующим образом оформленные и снабженные именем (программные блоки). Каждый блок включает заголовок, раздел описаний, раздел операторов.

Procedure <идентификатор> [<список параметров>];

Function <идентификатор> [<список параметров>]:<тип результата>;

Параметры, указываемые в заголовке подпрограммы, формальные.

При обращении к ней для каждого параметра должно быть указано фактическое значение – литерал, константа или переменная того же типа.

Из основной программы данные могут быть получены:

1. Неявно – переменные, глобальные константы
2. Явно – через параметры

Если имя в подпрограмме совпадает с именем глобального ресурса, то соответствующий глобальный ресурс в подпрограмме становится недоступным (перекрывается локальным ресурсом).

Параметры в подпрограмму могут передаваться 3 способами:

1. По значению
2. Как переменные (по ссылке, var)
3. Как неизменяемые переменные (по ссылке, const).

# Процедуры и функции Delphi Pascal. aa. Параметры-строки, параметры-массивы.

Параметры передаются в подпрограмму.

Количество элементов фактического параметра-массива можно определить с помощью функций high и low.

Если строка передается в подпрограмму как параметр-значение, то длина ее не контролируется.

Контроль длины осуществляется только для строк, передаваемых как параметр-переменная.

# Процедуры и функции Delphi Pascal. Принципы разработки универсальных подпрограмм. «Открытые» массивы и строки.

**Открытые массивы** – конструкция описания типа массива без указания типа индексов. (array of integer;). Индексы открытых массивов, всегда начинаются с нуля.

Количество элементов фактического параметра-массива можно определить с помощью функций high и low.

Если строка передается в подпрограмму как параметр-значение, то длина ее не контролируется. Контроль длины осуществляется только для строк, передаваемых как параметр-переменная. Поэтому при написании универсальных программ, работающих со строками произвольного размера, необходимо включать режим открытых строк {$P+} или объявлять параметры-строки как openstring.

# Процедуры и функции Delphi Pascal. Принципы разработки универсальных подпрограмм. Нетипизированные параметры, параметры процедурного типа.

В объявлении подпрограммы можно не указывать тип параметра-переменной:

procedure pr ( var x );

Такой параметр будет называться нетипизированным.

Нетипизированные параметры передаются в подпрограмму только по ссылке.

Для приведения нетипизированного параметра к определенному типу можно использовать:

1) автоопределенное преобразование типов:

Procedure Proc(Var a); ...

...b:= Integer(а)+10; ...

2) наложенное описание переменной определенного типа:

**Procedure Proc(Var a); ...**

**Var r:real absolute a;...**

Параметры процедурного типа используются для передачи в подпрограмму имен процедур и функций.

Для объявления процедурного типа используется заголовок подпрограммы, в котором отсутствует имя:

Type proc=procedure (a,b,c:real;Var d:real);

func=function(x:real):real;

Процедуры или функции, идентификаторы которых будут передаваться в качестве параметров процедурного типа, необходимо компилировать в режиме дальнего вызова ({$F+} или far) – в этом режиме используются 4-байтовые адреса вместо 2-байтовых.

# Модули Delphi Pascal. Структура модуля. Законы видимости идентификаторов. Доступ к «перекрытым» идентификаторам.

**Модуль** – автономно компилируемая коллекция программных ресурсов, предназначенная для использования другими модулями или программами.

Структура модуля:

Unit <имя модуля>; Cодержит объявление внешних ресурсов

Interface <интерфейсная секция>; содержит операторы, которые выполняются при подключении модуля

Implementation <секция реализации>; Cодержит описание подпрограмм, объявленных в интерфейсной секции, и описание внутренних ресурсов модуля.

[Begin <секция инициализации> End].

Все ресурсы модуля делятся на 2 группы:

1. **Внешние** – предназначенные для использования другими программными единицами
2. **Внутренние** – рабочие ресурсы данного модуля.

**Имя модуля должно совпадать с именем файла, в котором он содержится.**

При исполнении программы ее идентификаторы перекрывают идентификаторы используемых модулей. Если модуль A использует другой модуль B, то **при компиляции модуля A его идентификаторы перекрывают идентификаторы модуля B**. Для доступа к перекрытым идентификаторам используют обращения вида A.x, B.x.

# Рекурсия. Особенности программирования. Достоинства и недостатки.

**Рекурсия** – организация вычислений, при которой процедура или функция обращаются к самим себе.

Различают явную и косвенную рекурсии.

1. При явной – в теле подпрограммы существует вызов самой себя
2. При косвенной – вызов осуществляется в подпрограммах, вызываемых из рассматриваемой.

Косвенная рекурсия требует предопределения **forward**:

**Forward** - используется для того, чтобы можно было сначала вызвать процедуру, а потом использовать, например, для косвенной рекурсии. Сначала описываешь процедуру, затем Forward, а потом в любом месте ее описываешь (разумеется, в том же файле )

# Адресация динамической памяти: понятие адреса, операции получения адреса и разыменования. Процедуры получения памяти и освобождения ее.

Адрес = сегмент.

Переменные, созданием и уничтожением которых может явно управлять программист, называются динамическими.

Память принято делить на слова(2байта), двойные слова(4) и параграфы(16). При работе с памятью используется адресация по схеме "база + смещение".

Сегмент - это непрерывный участок памяти, имеющий длину 64к и начинающийся с адреса, кратного 16 (0,16,32,.....).

Существуют три вида сегментов:

* Сегмент кодов
* Сегмент данных
* Сегмент стека.

Сегменты могут следовать один за другим без промежутка, с некоторым интервалом или перекрывать друг друга.

В Паскале адрес байта памяти, состоящий из 2х слов типа word носит название указатель.

Над указателями определены операции:

1. Присваивания (допускается присваивать указателю только значение того же или неопределенного типа)
2. Операция разыменования (p1^:=2; p^:p1^+2;)
3. взятие указателя (p1:=@i; p1^:=5; - значит i:=5 для p1:^integer и i:integer)
4. отношения (p1=p2, p<>nil)

Заказать память требуемого объема из динамической области в Паскале можно несколькими способами:

1. Процедурой или функцией **new** (память, выделенная под динамические переменные после использования должна быть освобождена с помощью процедуры освобождения памяти **dispose**)
2. Динамическую память можно выделять фрагментом указанного размера. Для этой цели в Паскале используется процедура **GetMem (p:pointer,size:word).** Эта процедура запрашивает у системы память требуемого размера, указанного в параметре **size** (запрашиваемый объем не должен превышать 64к) и помещает адрес выделенного системой фрагмента в переменную типа указатель с именем p. После использования память обязательно нужно освободить процедурой **FreeMem (p:pointer,size:word)**, которая по адресу указателя p освобождает фрагмент размером **size**.

# Списковые структуры данных и основные приемы работы с ними: создание элемента, добавление элемента к списку, удаление элемента из списка. Область применения списковых структур данных.

Списковые структуры представляют собой совокупность элементов, количество которых заранее не известно. Оно определяется в процессе работы программы или программистом.

Элементом списка называется переменная типа запись, в качестве одного или нескольких полей которой определена переменная ссылочного типа - указатель на запись, в которой он описан. Остальные поля содержат информацию, подлежащую обработке и могут быть описаны как переменные любого допустимого типа, в том числе другого ссылочного.

TYPE ukzap = ^element; {ссылочный тип - указатель на запись}

element = record {новый тип - элемент, представляет собой запись о студенте}

name: string[16]; {фамилия}

b1,b2,b3: 2..5; {три отметки по экзаменам}

next: ukzap; {переменная ссылочного типа ukzap - указатель на след. элемент}

end;

Структуры различают:

* **Односвязные (однонаправленные) списки.** Каждый элемент такого списка содержит только одно поле ссылочного типа. Последний элемент такого списка должен содержать признак конца списка
* **Двусвязные (двунаправленные)списки.** Каждый элемент такого списка содержит 2 поля ссылочного типа. Указатель **pred**  содержит адрес предыдущего элемента, указатель **next** содержит адрес последующего элемента. Двунаправленный список характеризуется тем, что:

а) у первого элемента списка указатель **pred** устанавливается равным **nil**.

б ) у последнего элемента списка указатель **next** устанавливается равным **nil**.

* **N - связные списки.** Каждый элемент такого списка содержит n полей ссылочного типа.

Динамические линейные структуры:

1. Очередь – структура данных, реализующая: добавление – в конец, а удаление – из начала.
2. Стек – структура данных, реализующая: добавление и удаление с одной стороны.
3. Дек – структура данных, реализующая: добавление и удаление с двух сторон.

Выделение памяти:

**new(first);**

Добавление элемента к пустому списку:

**first ^.num:=5;**

**first ^.p:=nil;**

Добавление элемента перед первым (по типу стека):

**q^.num:=4;**

**q^.p:=first;**

**first:=q;**

Добавление элемента после первого (по типу очереди):

**q^.num:=4;**

**q^.p:=nil;**

**first^.p:=q;**

Удаление из памяти:

**dispose(first);**

# Основы файловой системы: файл, каталог, дисковод, полное имя файла, внутреннее представление информации в файле. Текстовый и нетипизированный файлы. Операции над файлами.

Файлом называют именованную последовательность элементов данных (компонент файла), имеющих одинаковую структуру и расположенных, как правило, на внешнем носителе. Имя(1..8 символов).Расширение(не более 3 символов).

**Каталог** – Путь к файлу.

**Дисковод** *–* представляет собой поименованную область внешней памяти на магнитном диске или логическим устройством.

Полное имя файла включает:

<Имя диска>:<Список имен каталогов>\<Имя файла>.<Расширение>

Доступ к компонентам файла осуществляется через указатель файла.

При выполнении операции чтения или записи указатель автоматически перемещается на следующий компонент.

После вывода последнего компонента файла система пишет специальную запись – маркер «Конец файла» (байт #26).

При обнаружении во время операции чтения маркера конца файла – операция завершается. Попытка читать маркер вызывает прерывание по ошибке чтения.

Виды файлов:

1. Текстовые (TYPE <имя> = TEXT;)
2. Нетипизированные (TYPE <имя> = FILE; применяется для скоростного обмена между внешней и оперативной памятью физическими записями указанной длины без преобразования, по умолчанию длина записи 128 байт.)

Процедуры и функции для работы с файлами:

* **assign(f,<имя файла>);** - инициализация файловой переменной
* **reset(f)** – открытие для чтения
* **rewrite(f)** – открытие для записи
* **close(f)** – закрытие
* **rename(f,<новое имя>)** – переименование (файл не должен быть открыт)
* **erase(f)** – удаление файла (файл не должен быть открыт)
* **eof(f):boolean** – проверка достижения конца файла
* **{$I-} ioresult(f):word** – получение информации об операции ввода-вывода (0, если все норм. Код ошибки в противном случае.)
* **read, write** – процедуры чтения/записи
* **seek(f,<номер компонента>)** – устанавливает указатель на требуемый компонент **filesize(f):longint** – количество компонент файла
* **filepos(f):longint** – возвращает номер компонента, который будет обрабатываться следующей операцией ввода-вывода (нумерация с нуля)
* **eoln(t):boolean** – проверка достижения конца строки
* **seekeoln(t):boolean** – то же, только пропускаются пробелы и знаки табуляции
* **seekeof(t):boolean** – проверка достижения конца текстового файла (пропускаются пробелы и знаки табуляции)
* **append(t)** - инициация записи в ранее существовавший текстовый файл.

# Типизированные файлы: внутреннее представление информации в файле, особенности обработки. Файловая переменная. Операции над файлом.

Типизированный файл – это файл, все компоненты которого одного типа, заданного при объявлении файловой переменной. TYPE <имя> = FILE of <тип компоненты>; где тип компоненты - любой тип переменной, принятой в Паскале, кроме FILE. Компоненты файла хранятся на диске во внутреннем(двоичном) формате и нумеруются с 0.

Поскольку длина каждой компоненты файла строго постоянна, помимо последовательного, возможно осуществление прямого доступа к компонентам файла.

* **read(f,c1,c2,…,cn)–** процедуры чтения. Компоненты должны быть того же типа, если файл исчерпан, то обращение к процедуре выдаст ошибку ввода-вывода;
* **write(f,c1,c2,…,cn);** – процедуры записи
* **seek(f,numcomp);** – устанавливает указатель на требуемый компонент
* **filesize(f):longint;** – количество компонент файла
* **filepos(f):longint** – порядковый номер компонента, который будет обрабатываться при следующей операции ввода-вывода

# Классы консольного режима Delphi: описание классов, поля и методы, объявление объектов класса, доступ к полям и методам объекта, ограничение доступа.

Процедуры и функции, заголовки которых описаны в классе, называются методы.

Type <имя класса> = object

<описание полей класса>

<прототипы методов>

end;

Тела методов класса описываются после объявления класса.

Procedure/Function<имя класса>.<имя метода>;

<локальные ресурсы процедуры/функции>

begin

<тело процедуры/функции>

end;

Описав класс, мы можем объявить любое количество объектов этого класса.

Обращение к полям и методам объекта выполняется так же, как к полям записи: с использованием точечной нотации или с использованием оператора **with**.

Для этого описание класса делится на специальные секции:

* **Public** – секция, содержащая описания **общедоступных полей и методов** класса
* **Private** – секция, содержащая описания **внутренних или скрытых полей и методов** класса.

# Классы консольного режима Delphi: Способы инициализация полей. Неявный параметр Self.

Поля объектов инициализируются:

– Прямым занесением в поле A.length:=3.5; и с использованием типизированных констант Const a: TRoom=(length:3.5);

– Посредством специального метода – инициализирующей процедуры, которую рекомендуется называть procedure Init(l:real);

procedure TRoom.Init;

begin

length:=l;

end;

Каждый объект данного класса при объявлении получает свой набор полей класса. Эти поля собраны в запись, внешнее имя которой совпадает с именем объекта. Любой метод класса, вызванный для конкретного объекта, неявно получает специальный параметр Self. Значением которого является адрес записи, объединяющий все поля этого объекта. Этот параметр иногда называют обобщенным внутренним именем объекта.

# Процедурная и объектная декомпозиция. Диаграммы классов. Отношения между классами

Объектной декомпозицией называют процесс представления предметной области задачи в виде совокупности функциональных элементов (объектов), обменивающихся в процессе выполнения программы входными воздействиями (сообщениями).

**Наследование** ([потомок]-|>[родитель]) – отношение между классами, при котором один класс строится на базе второго посредством добавления новых полей и определения новых методов.

**Композиция** ([часть]-1..5-♦[основной класс]) – способ конструирования классов, при котором в строящийся класс включают объекты других классов. Физически в класс включается фиксированное число полей, являющихся объектами другого класса.

**Наполнение/агрегация** ([часть]-0..\*-◊[агрегат]) – точное количество объектов одного класса, включаемых в другой класс, не ограничено и может меняться от 0 до достаточно больших значений. Физически реализуется с использованием указателей на объекты (массив указателей, списки).

**Полиморфное наследование** – наследование, при котором осуществляют переопределение методов класса-родителя потомком. (Метод потомка имеет то же имя, что и метод родителя, но выполняет другие действия.) Простой полиморфизм используют, если при вызове переопределенного метода тип объекта, для которого вызывается метод, точно известен. Нужный метод определяется на этапе компиляции (раннее связывание). Сложный полиморфизм используют, если при вызове переопределенного метода необходимо уточнить, метод родителя или метод потомка должен быть подключен. Нужный метод определяется на этапе выполнения программы (позднее связывание). Например, класс1 наследуется от класса2 с методами print и count, где метод print для класса1 остается прежним, а метод count переопределяется, при этом метод print в процессе выполнения вызывает метод count потомка, для этого

1. Переопределяемые методы должны быть описаны служебным словом **virtual**
2. К методам класса с виртуальными полиморфными методами надо добавить конструктор(**constructor**),
3. Необходимо вызвать конструктор прежде, чем произойдет первое обращение к виртуальным полиморфным методам. В переопределенном конструкторе можно использовать конструктор родителя с помощью служебного слова **inherited**.

# Динамические объекты и объекты с динамическими полями в консольном режиме Delphi.

Для динамического объекта необходимо описать указатель на объект, выделить память

(new(<тип указателя>[,<вызов конструктора>]))

А после использования – освободить память

(dispose(<указатель>[,<вызов деструктора>])).

При этом указателю базового класса допускается присваивать адрес объекта производного класса. Для корректного освобождения памяти для полиморфных объектов необходимо использовать деструктор, т.к. такие объекты содержат дополнительное поле с адресом ТВМ.

При работе с динамическими объектами или с динамическими полями в статистических объектах целесообразно использовать обычные средства контроля выделения памяти: функции MaxAvail, MemAvail, HeapError, HeapFunc.

# Технология событийного программирования. События Windows, сообщения и события Delphi. Основные события Delphi.

Принцип событийного управления Windows предполагает, что **приложение должно состоять из нескольких подпрограмм, вызываемых при наступлении некоторого события**, которым считается любое изменение в системе, касающееся данного приложения. О наступлении того или иного события приложение информируется посылкой соответствующих сообщений. В процессе работы приложение может генерировать сообщения windows другим приложениям или самому себе, инициируя все новые и новые события. Последовательность событий, как правило, неизвестна, поэтому части приложения, называемые обработчиками сообщений, взаимодействуют только через общие данные, как при ООП. Любое приложение состоит как минимум из двух частей: основной программы и оконной функции (поведение окна определяется обработчиками сообщений, составляющими оконную функцию). Приложение, создав окно и запустив цикл обработки сообщений, возвращает управление операционной системе и в дальнейшем получает управление только по приходу сообщений, предназначенных ему.