Билеты по основам программирования

Билет №1

Синтаксис, семантика и алфавит языков программирования.

Язык программирования – это формальный язык, предназначенный для записи компьютерных программ.

Программа – последовательность инструкций, адресованных компьютеру, которая точно определяет, как следует решать задачу.

Синтаксис – правила, определяющие допустимые конструкции языка (слова, предложения), построенные из символов его алфавита.

«Защищенный» синтаксис предполагает, что предложения языка строятся по правилам, которые позволяют автоматически выявлять большой процент ошибок в программах.

Семантика – правила, определяющие смысл синтаксически корректных предложений.

Ясная или «интуитивно-понятная» семантика – семантика, позволяющая без большого труда определять смысл программы или «читать» ее.

Алфавит языка программирования Паскаль включает:

1) латинские буквы без различия строчных и прописных;

2) арабские цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

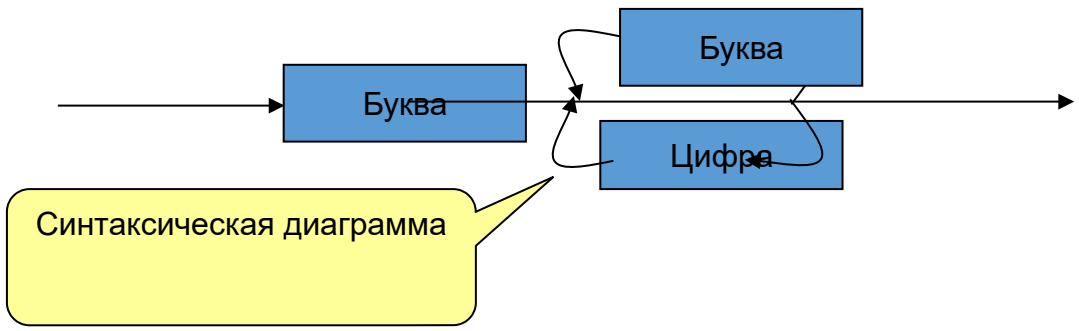
3) шестнадцатеричные цифры: 0..9, a..f или A..F;

4) специальные символы: + - * / = := ; и т. д.;

5) служебные слова: do, while, begin, end и т. д.

Идентификатор – это неделимая последовательность символов.

Пример: конструкция «Идентификатор» (имя):



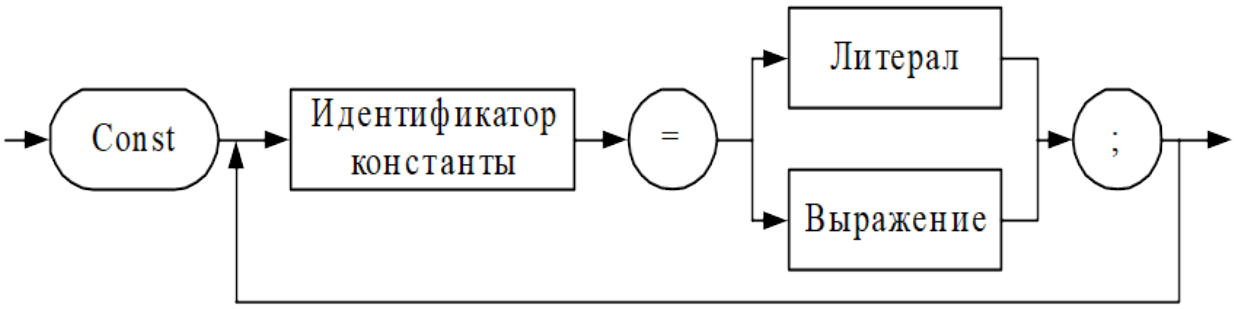
Билет №2

Представление данных: константы и переменные.

Константы – данные, не изменяемые в процессе выполнения программы.

Литералы – константы, указанные непосредственно в тексте программы.

Поименованные константы – константы, обращение к которым выполняется по имени. Объявляются в разделе описаний:

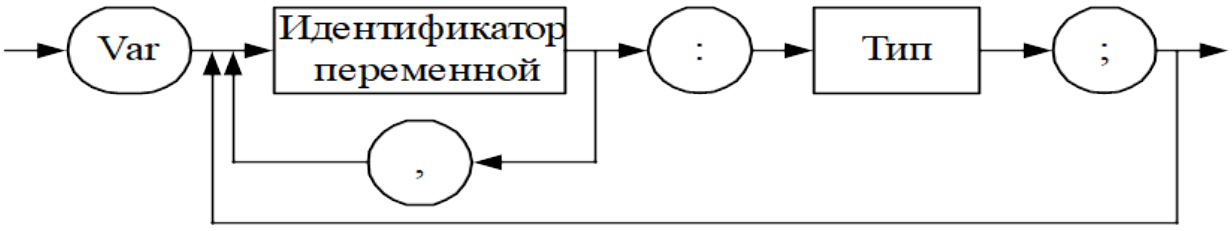


Пример:

Const min = 0; max = 100;

center = (max - min) div 2;

Переменные – поименованные данные, которые могут изменяться в процессе выполнения программы. Объявляются также в разделе описаний:



Пример:

Var a,b:integer;

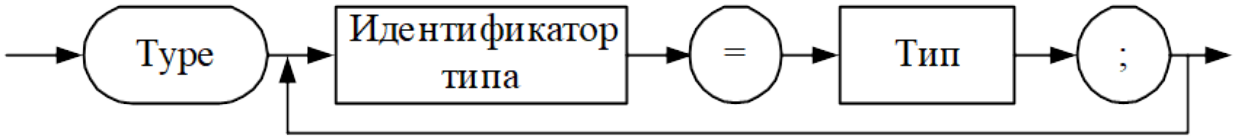
c:real;

Тип данных – описатель данных, который определяет:

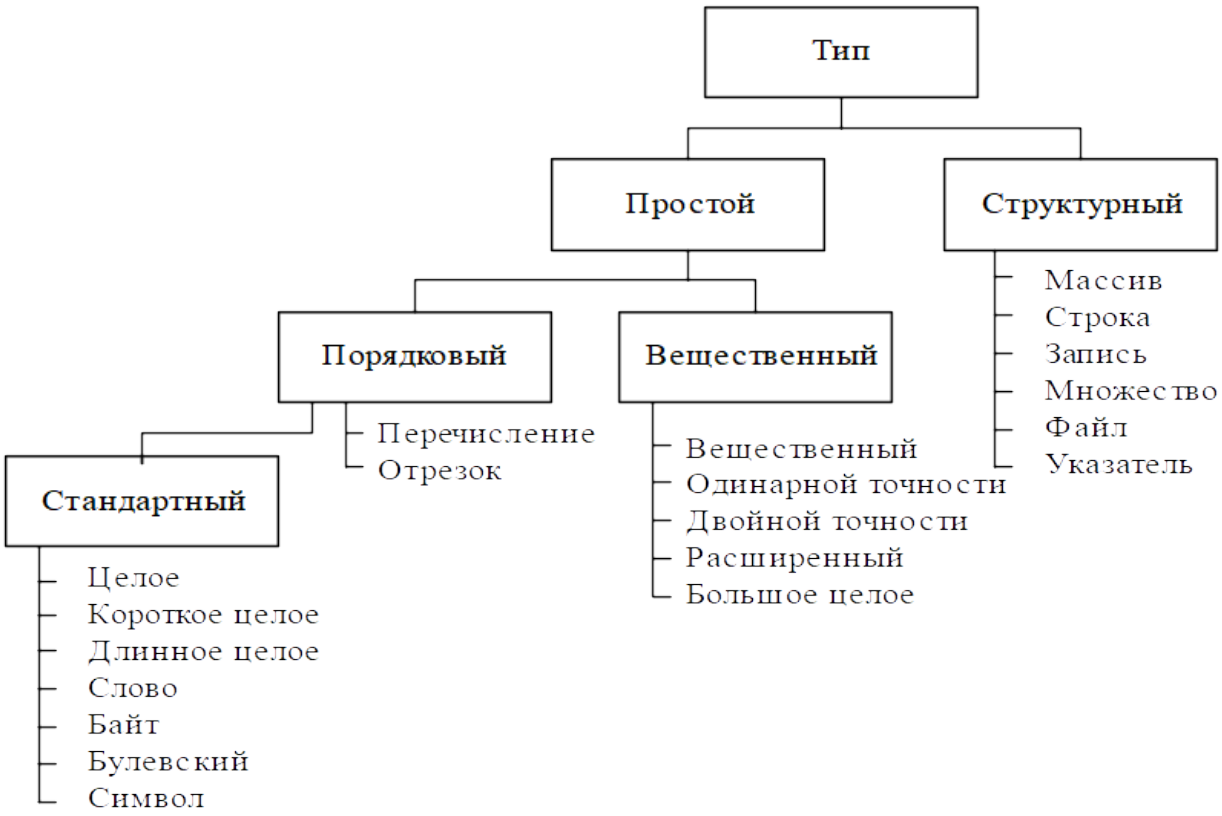
а) ***диапазон изменения значения*** переменной, задавая размер ее внутреннего представления;

б) ***множество операций***, которые могут выполняться над этой переменной.

Для объявления новых типов данных используется конструкция:



Классификация типов данных:



К стандартным **скалярным** типам относятся целочисленные, вещественные, символьные и булевские **типы данных**.

Операции и выражения:

Арифметические операции – применяют к вещественным и целым константам и переменным.

+, -, *,

/ {вещественное деление},

div {целочисленное деление},

mod {остаток от деления}

Пример:

var a: integer = 5; b: integer = 3;

a+b , a div b, a mod b, a / b, (a+b)/(a-b*a)

Операции отношения (больше, меньше, равно и т.д.) – применяют к числам, символам, строкам – в результате получают логическое значение.

Логические операции – применяют к логическим значениям – результат логическое значение.

Примеры:

a:=true; b:=false;

a and b {false}

a or b {true}

Два типа считаются **совместимыми**, если:

- оба они есть один и тот же тип;
- оба вещественные;
- оба целые;
- один тип есть тип-диапазон второго типа;
- оба являются типами-диапазонами одного и того же базового типа;
- оба являются множествами, составленными из элементов одного и того же базового типа;
- оба являются упакованными строками (определены с предшествующим словом PACKED) одинаковой максимальной длины;
- один тип есть тип-строка, а другой - тип-строка, упакованная строка или символ;
- один тип есть любой указатель, а другой - нетипизированный указатель;
- один тип есть указатель на объект, а другой - указатель на родственный ему объект;
- оба есть процедурные типы с одинаковыми типом результата (для типа-функции), количеством параметров и типом взаимно соответствующих параметров.

Преобразование типов:

Если типы результата и переменной не совпадают, но совместимы, то при выполнении присваивания выполняется ***неявное автоматическое преобразование*.**

Если результат не умещается в разрядную сетку переменной, то автоматически генерируется ошибка **«Переполнение разрядной сетки».**

Для несовместимых типов результата и переменной, в которую его необходимо занести, при выполнении присваивания необходимо **явное преобразование типов**, например, посредством специальных функций:

trunc(<Вещественное выражение>) – преобразует вещественное число в целое, отбрасывая дробную часть.

round(< Вещественное выражение>) – округляет вещественное число до целого по правилам арифметики.

Пример: trunc(4.5) = 4, round(4.5) = 5

ord(<Порядковое выр.>) – преобразует значение в его номер.

Пример: ord(’A’) = 65.

chr(<Ц. выр.>) – преобразует номер символа в символ.

Пример: chr(65) = ’A’.

Val (Str1, vaue, e) – преобразует строку str1 в число и записывает его значение в переменную value (если это невозможно, то e>0, иначе e=0);

Str(I, str1) –преобразует целое значение I в строку str1.

Билет №3

Основные операторы:

Оператор присваивания - основной оператор любого языка программирования. Общая форма записи оператора:

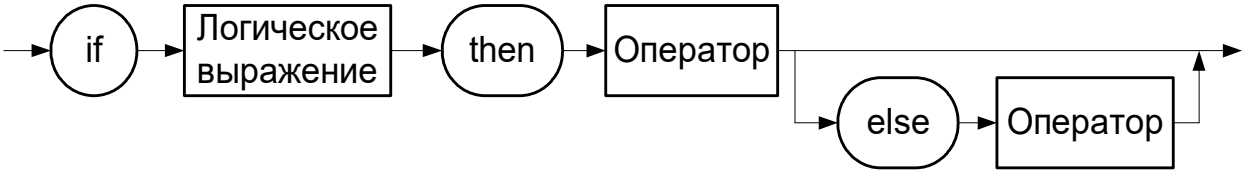
имя величины := выражение

Например, V:=A; или V:=A+1;

При помощи оператора присваивания переменной могут присваиваться константы и выражения, значения переменных её типа.

Как только в программе встречается переменная, для неё в памяти отводится место. Оператор присваивания помещает значение выражения в место, отведённое переменной.

Оператор условной передачи управления используется при обработке вариантов вычислений и реализует конструкцию ветвления.



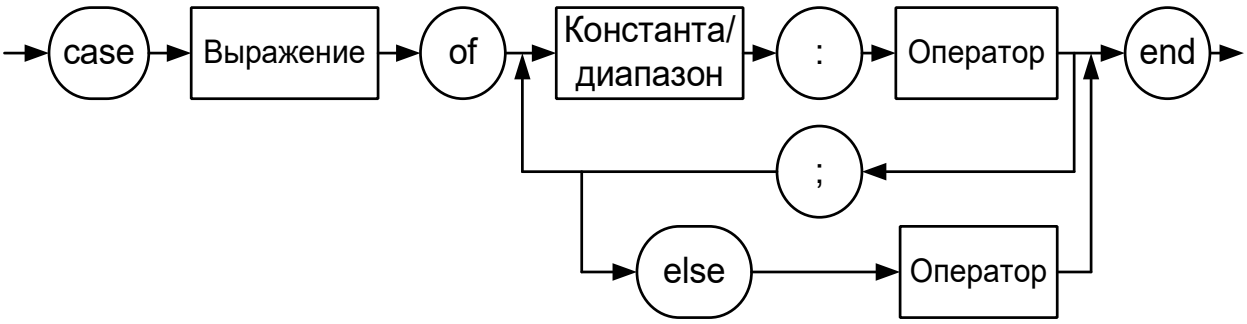
Пример:

if <Условие1> then

if <Условие2> then <Действие1>

else <Действие 2>

Оператор выбора позволяет программировать несколько вариантов решения.



Пример:

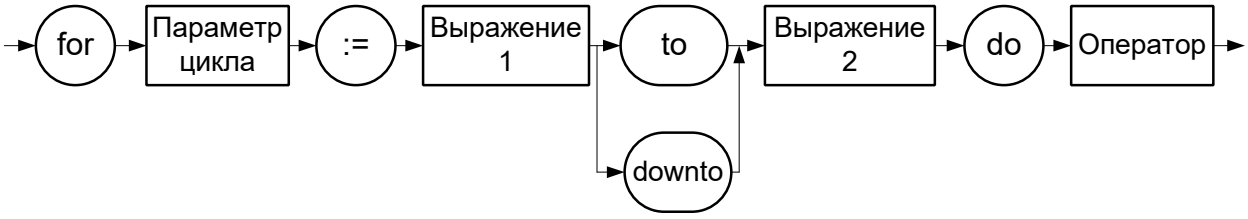
```
case 1+2*j of
        3:      z:=sin(x);
        -1..1,10: z:=cos(x);
        else    z:=0;
```

Операторы организации циклов:

Циклы бывают**:**

Счётными (цикл-for):

Счетный цикл – цикл, количество повторений которого известно или можно посчитать. Выход из такого цикла программируется по счетчику.

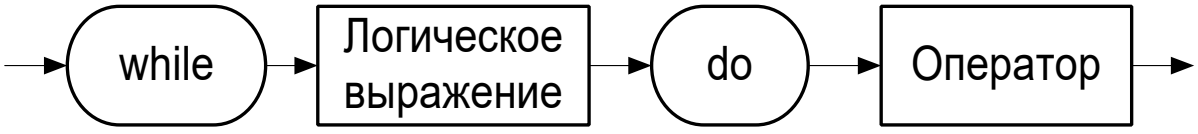


Пример:

```
for   i:=1 to 10 do
      begin
            x:=x+1;
            e:=e/10;
      end;
```

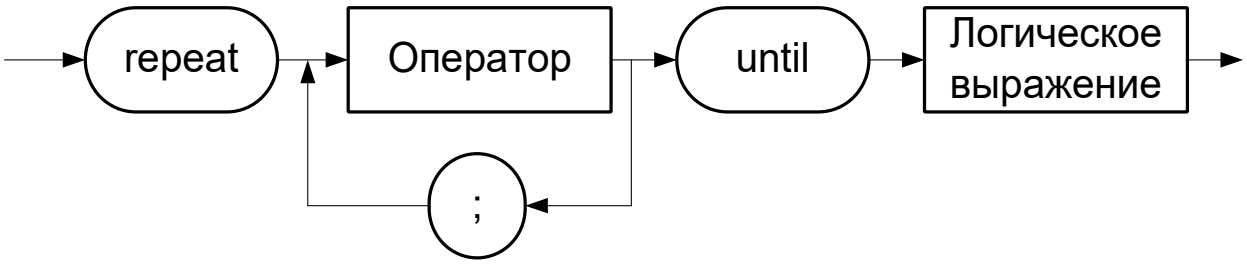
Итерационными (цикл-пока, цикл-до):

Итерационный цикл – цикл, количество повторений которого неизвестно или считается неизвестным при построении цикла. Выход из цикла программируется по выполнению или нарушению условия.



Пример:

```
while abs(e)>=1e-5 do
        begin
              x:=x+1;
              e:=e/10;
        end;
```



Пример:

```
repeat
        x:=x+1;
        e:=e/10;
 until  abs(e)<1e-5
```

Поисковыми:

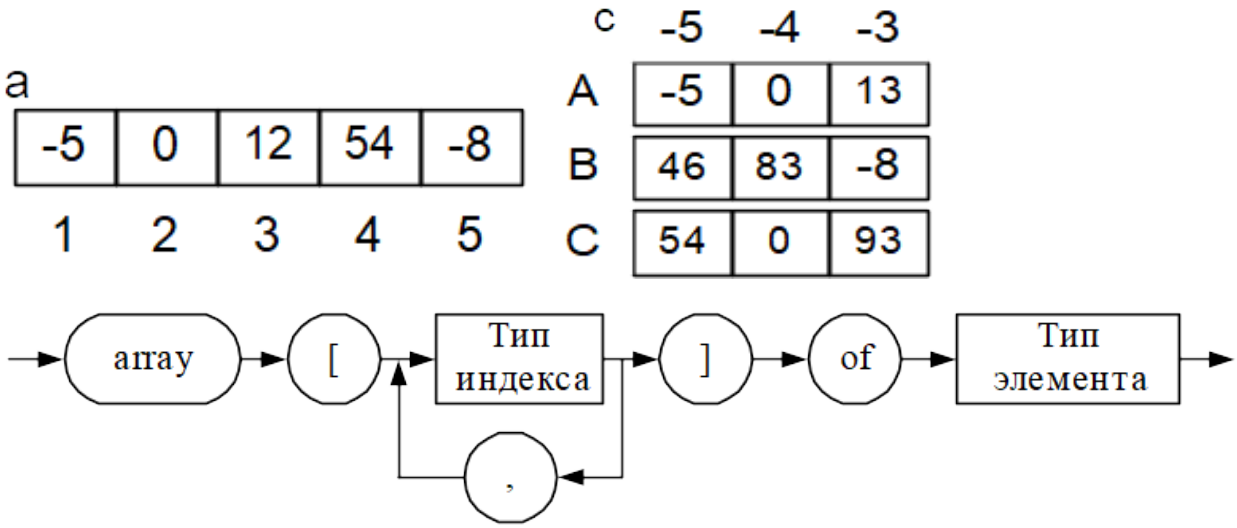
Поисковый цикл имеет два выхода – нашли и перебрали все и не нашли.

Билет №4

Структурные типы данных: массивы и строки.

Массивы:

Массив – это упорядоченная совокупность ***однотипных*** *данных*. Каждому элементу массива соответствует один или несколько ***индексов порядкового типа***, определяющих положение элемента в массиве.



Количество ***типов*** индексов задает ***размерность*** **массива**.

Тип индекса – порядковый – определяет доступ к элементу.

Тип элемента – любой кроме файла, в том числе массивы, строки и т.п.

(Массив в памяти не может занимать более 2 Гб.)

Объявление массива:

```
Var a:array[1..5] of integer;
        c:array['A'..'C',-5..-3] of byte;
        b:array[byte] of char;
Type mas=array[1..10] of integer;
Var a:mas;
```

Инициализация массива при объявлении

```
Var a:array[1..5]of real=(0,-3.6,7.8,3.789,5.0);
    b: array[boolean, 1..5] of real=
                        ((0,-3.6,7.8,3.789,5.0), (6.1,0,-4.56,8.9,3.0));
```

Операции над массивами:

1. Операция присваивания (только для массивов одного типа):

Пример:

```
Var a, b:array[boolean] of real;
    ...
    a:=b;
```

2. Доступ к элементу массива:

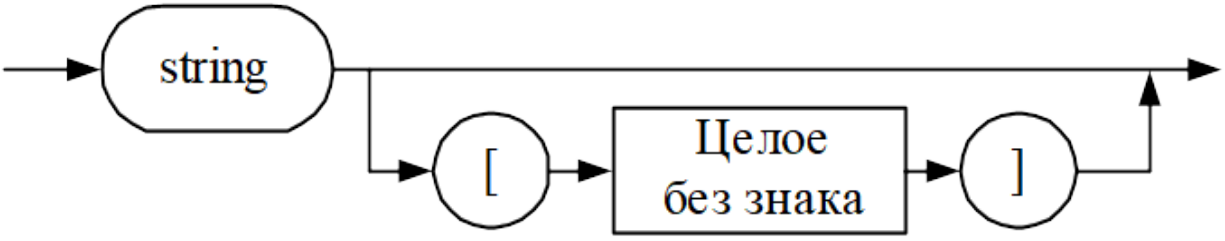
Пример:

```
Var a:array[char,boolean] of real;
    ...
    a['A',true]:=5.1; {прямой доступ}
    ...
    Ch:='B'; b:=false;
    a[Ch,b]:=3; {косвенный доступ: значения индексов
        находятся в переменных}
```

3. Ввод и вывод массива производятся поэлементно.

Строки:

Строка – последовательность символов**.**



«Целое» – максимальная длина строки.

Описание строк:

1) Var S1,S2:string[40]; S3:string;

2) С предварительным объявлением типов:

```
Type S40 = string[40];
  ST = string;
Var  S1,S2: S40;
  S3:ST;
```

3) С инициализацией

```
Var S:string[40]=’Строковая константа’;
S1:string = ’’;
```

Операции над строками:

1. Присваивание строк:

```
S1:=′ABCD′;
S1:=S2;
S1:=′A′;
S1:=′′;{пустая строка}
```

2. Обращение к элементу:

```
S1[5] - прямое
S1[i] - косвенное
```

3. Конкатенация (сцепление) строк:

```
St:=St + ’A’;
St:=’A’ + ’B’;
```

4. Операции отношения – выполняется попарным сравнением кодов символов, результат определяется по отношению кодов первых различных символов:

```
b:= S1 > S2;
’T’ < ’Ta’
```

5. Ввод-вывод строк:

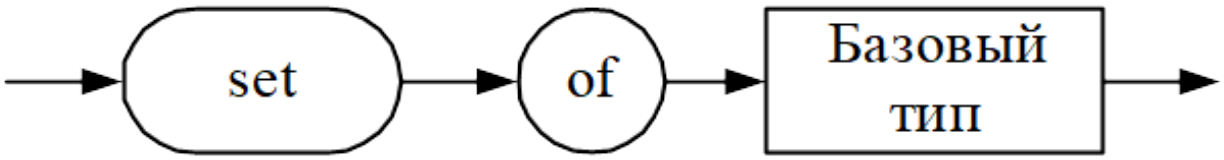
```
ReadLn(S1);
{Строка вводится до Enter
 или указанной длины}
WriteLn(S1);
```

Билет №5

Структурные типы данных: множества и записи.

Множества:

Множество – неупорядоченная совокупность неповторяющихся элементов



(Количество элементов не должно превышать 256)

Объявление множеств:

```
Type
  Digits = set of 1..100;
  Setchar = set of char;
  letter = set of 'a'..'z';
Var mychar: setchar;
   mydig: Digits;
   simst: letter;
```

или

```
Var number: set of 1..31;
          cif: set of  0..9;
   kods: set of #0..#255;
```

Инициализация множеств:

Конструкторы множеств – константы множественного типа:

[] – пустое множество;

[2,3,5,7,11] – множество чисел;

[’a’,’d’,’f’,’h’] – множество символов;

[1,k] – множество чисел, переменная k должна содержать число;

[2..100] – множество содержит целые числа из указанного интервала;

[k..2*k] – интервал можно задать выражениями;

[red,yellow,green]- множество перечисляемого типа

Инициализация множеств при объявлении:

Type setnum = set of byte;

Var S:setnum = [1..10];

Операции над множествами:

1. Присваивание:

A:=B;

A:=[];

2. Объединение, пересечение и дополнение:

- *A+B* (A∪B) – объединение множеств A и B – множество, состоящее из элементов, принадлежащих множествам A и B
- *A*B* (A∩B) – пересечение множеств A и B – множество, состоящее из элементов, принадлежащих одновременно и множеству A и множеству B.
- *A-B* (A \ B) – дополнение множества A до B – множество, состоящее из тех элементов множества A, которые не принадлежат множеству B.

Примеры:

[1,2]+[3,4] = [1,2,3,4];

[1..10]*[3,8,9,15,23,45] = [3,8,9];

[1..15]-[3,8,9,15,23,45] = [1,2,4..7,10..14];

[red,blue,green,black]*[blue,magenta,yellow] = [blue]

3. Операции отношения:

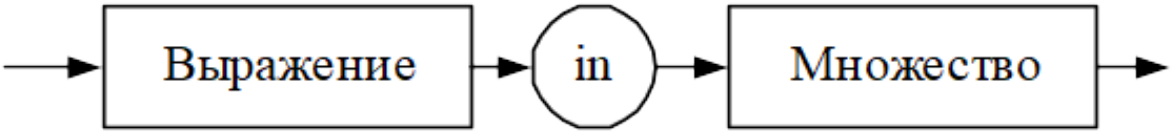
A = B – проверка совпадения множеств A и B (если совпадают – true)

A <> B – проверка не совпадения множеств A и B (не совпадают – true).

A <= B – проверка нестрогого вхождения A в B (если входит – true).

A > B – проверка строгого вхождения B в A (если входит – true).

4. Проверка вхождения элемента во множество:



Пример:

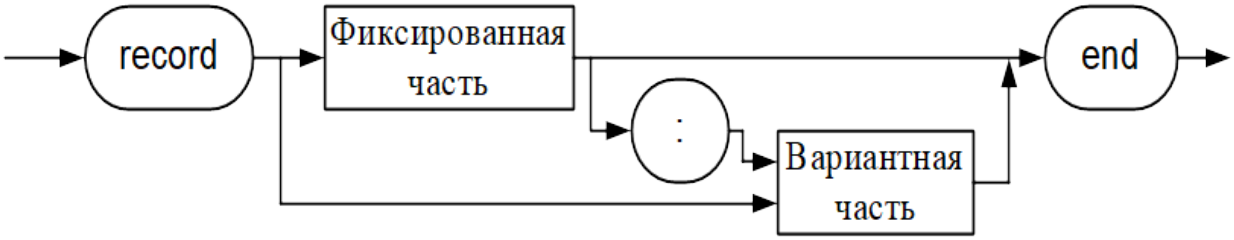
if a in [2..6] then …

Записи:

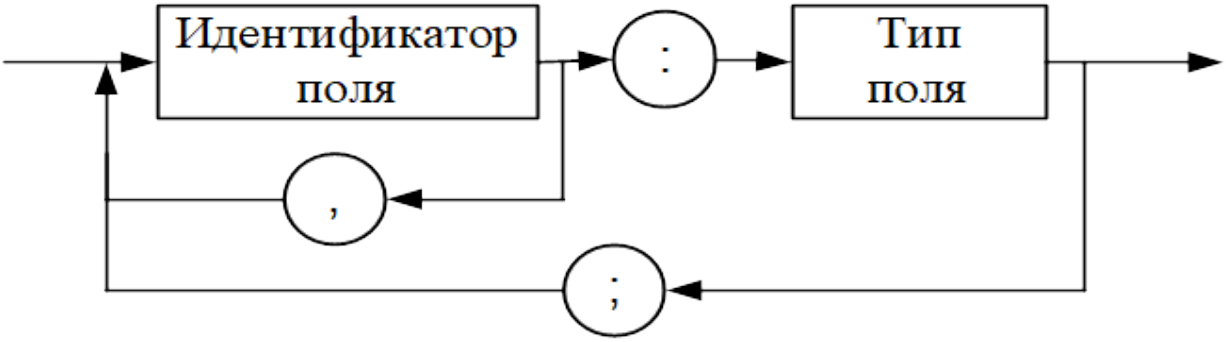
Запись – это структура данных, образованная фиксированным числом разнотипных компонентов, называемых полями записи.

Пример записи: Иванов Иван 20 лет студент 1 курса ⇒

Иванов | Иван | 20 | студент | 1



Фиксированная часть записи:



Объявление и инициализация записей:

Примеры:

a) Var Zap1: record

Day:1..31;

Month: 1..12;

Year: word;

```
        end;
```

б)
```
Type Data = record
              Day:1..31;
              Month: 1..12;
              Year: word;
            end;
  Var Zap1:Data;
```

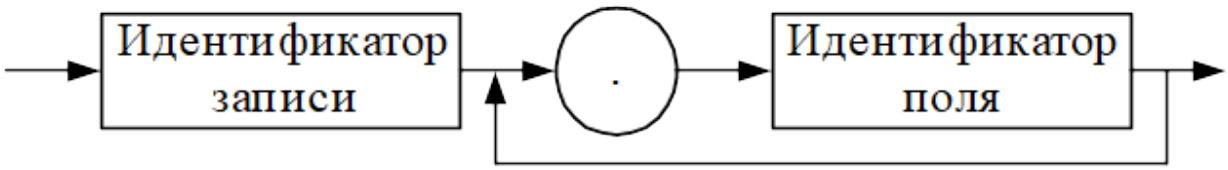
в)
```
Var BirthDay: Data = (Day:30; Month:6; Year:1973);
```

Операции над записями:

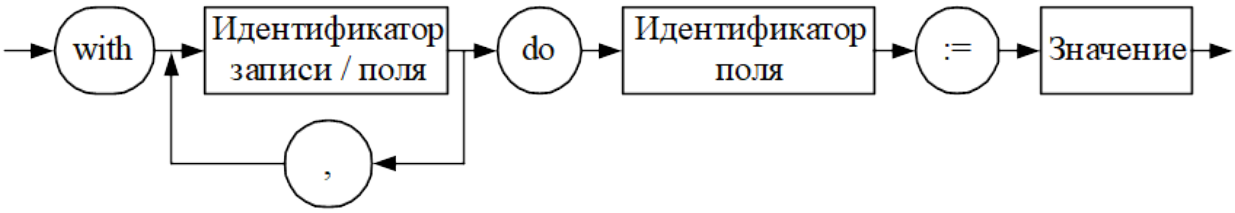
1. Присваивание записей одного типа:

```
    Var A,B: record Day:1..31; Month: 1..12; Year: word; end;
  …
    A:=B;
```

2. Доступ к полям записи:




```
    A.Day:=21; {точечная нотация}
```




```
    with A do Day := 21; {оператор доступа}
```

3. Ввод и вывод записей осуществляется по полям.

Билеты №6 и №7.

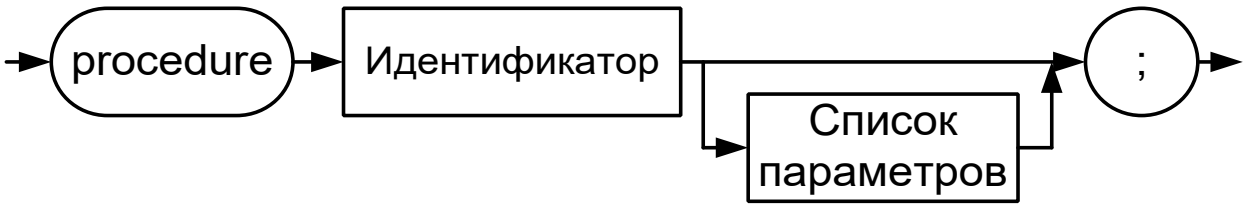
Процедуры и функции. Способы передачи данных в подпрограмму. Локальные и глобальные переменные, законы «видимости» идентификаторов. Параметры-строки и параметры-массивы.

Процедуры и функции – самостоятельные фрагменты программы, соответствующим образом оформленные и вызываемые по имени (программные блоки).

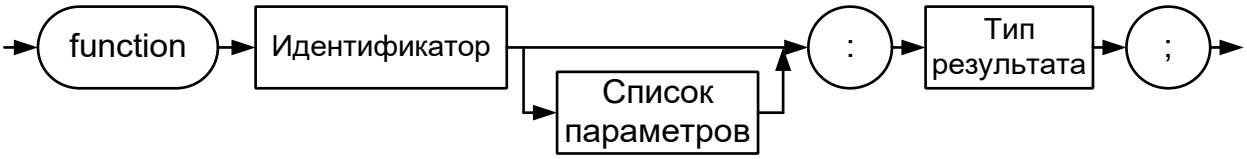
Программный блок:

Функция в Паскале- это подпрограмма, которая в отличие от **процедуры <u>всегда</u> возвращает какое-либо значение (!)**. Для этого в теле функции её имени присваивается вычисленное значение — результат, который она возвращает.

Процедура:



Функция:



Локальные и глобальные переменные:

Классы переменных	Время жизни	Доступность
Глобальные – объявленные в основной программе	От запуска до завершения программы	Из любого места программы, включая подпрограммы
Локальные – объявленные в подпрограмме	От вызова подпрограммы до возврата управления	Из подпрограммы и подпрограмм, вызываемых из нее

Подпрограмма может получать данные из основной программы:

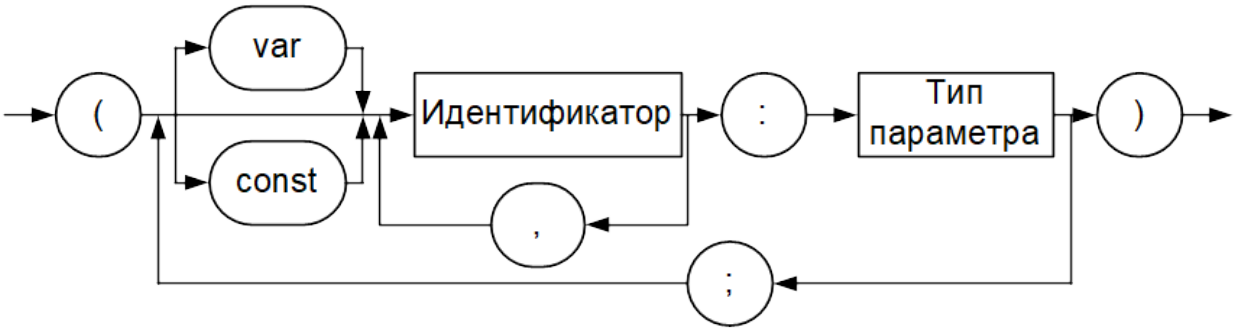
а) **неявно** – с использованием свойства доступности глобальных переменных;

б) **явно** – через параметры.

Не рекомендуется использовать неявную передачу данных, так как она приводит к большому количеству ошибок и жёстко связывает подпрограмму с данными**!!!**

Передача данных через параметры:

Список параметров описывается в заголовке.



Параметры, описанные в заголовке – ***формальные***.

При вызове подпрограммы необходимо определить ***фактические*** значения этих параметров – аргументы (константы и переменные).

Формальные и фактические параметры должны соответствовать по количеству, типу и порядку:

function proc(a:integer; b:single):byte; …

n:= proc(5,2.1);

Есть три способа передачи параметра в подпрограмму:

- **По значению:**

Никак не помечается. В таком случае подпрограмма **будет работать с копией** перереданного параметра и никакие его изменения, происходящие внутри подпрограммы, в основной программе отображаться не будут.

- **По ссылке:**

Помечается служебным словом **var.** При этом способе передачи параметра подпрограмма **по адресу получает доступ к самому параметру, а не к его копии**, как это б ыло в предыдущем случае, поэтому все изменения, происходящие внутри подпрограммы, в основной программе будут отображаться. (нельзя передавать литералы по ссылке!)

- **Как константа**

Помечается служебным словом **const.** В этом случае в подпрограмму тоже **передаётся адрес фактического параметра**, как и в предыдущем случае, **НО** при попытке изменить параметр, компилятор выдаёт сообщение об ошибке.

Структурные типы параметров должны быть предварительно объявлены!

Билеты №8 и №9

Процедуры и функции. Принципы разработки универсальных подпрограмм. «Открытые» массивы и строки. Нетипизированные параметры, параметры процедурного типа.

«Открытые» массивы:

«Открытый» массив – конструкция описания типа массива без указания типа индексов. Используется только при объявлении формальных параметров.

Примеры:

array of single;

array of integer;

Индексы открытого массива всегда начинаются с 0.

Размер можно:

- передать через дополнительный параметр;
- получить, используя функцию High(<Идентификатор массива>).

«Открытые» строки:

Для строк, передаваемых в подпрограмму как параметр-переменная, Паскаль осуществляет контроль длины строки. Чтобы избежать его необходимо использовать **«открытые» строки**.

Пример: Procedure Add(var s:openstring);

Нетипизированные параметры:

Нетипизированные параметры – параметры-переменные, тип которых при объявлении не указан.

Для приведения нетипизированного параметра к определенному типу можно использовать:

1) автоопределенное преобразование типов:

```
Procedure Proc(Var a); ...
 ...b:= Integer(a)+10; ...
```

2) наложенное описание переменной определенного типа:

```
Procedure Proc(Var a); ...
Var r:real absolute a;...
```

Параметры процедурного типа:

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;
```

Значениями переменных процедурных типов являются идентификаторы процедур и функций с соответствующими заголовками:

```
Var  f:func;
...
f:=fun1;...
```

Билет №10

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

Модули:

Модуль – это автономно компилируемая коллекция программных ресурсов, предназначенных для использования другими модулями и программами.

Ресурсы – переменные, константы, описания типов и подпрограммы.

Все ресурсы, определенные в модуле делят на:

1) ***внешние*** – предназначенные для использования другими программами и модулями.

2) ***внутренние*** – предназначенные для использования внутри модуля.

Структура модуля:

Unit <Имя модуля>;

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

Interface

<Интерфейсная секция>

Implementation

<Секция реализации>

[Initialization

<Секция инициализации>

[Finalization

<Секция завершения>]]

End.

Подключение модуля:

Подключение модуля к программе осуществляется по имени:

Uses <Имя модуля1>, <Имя модуля2>, ...;

Объявление модулей в файле проекта

Если:

- к проекту подключается модуль, который находится в каталоге, не совпадающем с каталогом проекта и не указанном в путях компилятора;
- в путях компилятора имеется несколько модулей с одинаковыми именами,

то необходимо указать местонахождение модуля:

Uses Strings in 'C:\Classes\Strings.pas';

Uses Strings in '..\Strings.pas'; {относительно текущего кат.}

Модули, объявленные в файле проекта с указанием in …, считаются частью проекта, т. е. доступны через средства работы с проектом среды.

Использование указания in … в файлах с расширением pas не допустимо.

Законы видимости:

Ресурсы модуля перекрываются ресурсами программы и ранее указанных модулей. Для доступа к перекрытым ресурсам модуля используют точечную нотацию:

<Имя модуля>.<Имя ресурса>

Пример:

```
Unit A;
  Interface
    Var X:real; …
End.
```

```
Program ex;
Uses A;
 Var X:integer;
  Begin
    X:=10;
     A.X:=0.45; …
  End.
```

Билет №11

Рекурсия. Особенности программирования. Достоинства и недостатки.

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

Различают два вида рекурсии подпрограмм:

- **Прямая или явная** - характеризуется существованием в теле подпрограммы оператора обращения к самой себе;
- **Косвенная или неявная** - образуется при наличии цепочки вызовов других подпрограмм, которые в коечном итоге приведут к вызову исходной (требует предопределения forward).

Каждое обращение к рекурсивной подпрограмме вызывает независимую активацию этой подпрограммы. Совокупность данных, необходимых для одной активации рекурсивной подпрограммы, называется **фреймом активации**, который включает:

- копии всех локальных переменных подпрограммы;
- копии параметров-значений;
- четырёхбайтовые адреса параметров-переменных и параметров-констант;
- копию строки результата (для функций типа string);
- служебную информацию - более 12 байт, точный размер этой области зависит от внутренней организации подпрограмм.

Структура рекурсивной программы: "Операторы после вызова" выполняются после возврата управления из рекурсивно вызванной подпрограммы.

При работе рекурсивных процедур среда незавершенного вызова сохраняется, т.е. запоминается адрес точки прерывания, сохраняются значения переменных, параметров и т.д. Сохранение среды для незавершенных вызовов производится по принципу стека. Таким образом, предложенный рекурсивный алгоритм вычисления факториала неэффективен как с точки зрения временной сложности, так и с точки зрения пространственной сложности.
Использовать рекурсию удобно, например, в тех случаях, когда отказ от нее заставил бы программиста самого организовывать стек для хранения промежуточных результатов. Рекурсия весьма эффективна при работе с графами и древовидными списками. Ее применение часто позволяет давать более прозрачные и сжатые описания алгоритмов, чем это было бы возможно без рекурсии.

Билет №12

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

Минимальная адресуемая единица памяти – *байт*.

Физический адрес $\mathbf{A_ф}$ – номер байта оперативной памяти.

Адресация по схеме «база+смещение»:

$$\mathbf{A_ф = A_б + A_{см}},$$

где $\mathbf{A_б}$ **– адрес базы** – адрес, относительно которого считают остальные адреса;

$\mathbf{A_{см}}$ **– смещение** – расстояние от базового адреса до физического.

Указатель – тип данных, используемый для хранения *смещений*.

В памяти занимает 4 байта, адресует сегмент размером $V = 2^{32} = 4$ Гб.

Базовый адрес = адрес сегмента.

Типизированные и нетипизированные указатели:

Различают указатели:

- **типизированные** – адресующие данные конкретного типа;
- **нетипизированные** – не связанные с данными определенного типа.

Объявление типизированного указателя:

Объявление нетипизированного указателя: **pointer**

Примеры:

а) **Type tpi=^integer;**

Var pi:tpi;

или

Var pi: ^integer;

б) **Var p:pointer;**

в) **Type pp = ^percon;**

percon = record

name: string:

next: pp;

end;

г) **Var r:^integer = nil;**

Операции над указателями:

- Присваивание; (Допускается присваивать указателю только значение того же или неопределенного типа.)
- Получение адреса (@);
- Доступ к данным по указателю (разыменовывание «^») (нетипизированные указатели разыменовывать **нельзя!);**
- Отношение (=, <>);

Управление выделением и освобождением памяти осуществляется посредством специальных процедур и функций:
1. Процедура **New(var P: ^<тип>)** – выделяет память для размещения переменной, размер определяется типом указателя.
2. Процедура **Dispose(var P: ^<тип>)** – освобождает выделенную память.

Билет №13

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

Список – способ организации данных, предполагающий использование указателей для определения следующего элемента.

Элемент списка состоит из двух частей: ***информационной*** и ***адресной*.**

Информационная часть содержит поля данных.

Адресная – включает от одного до n указателей, содержащих адреса следующих элементов. Количество связей, между соседними элементами списка определяет его связность: односвязные, двусвязные, n-связные.

В зависимости от количества полей в адресной части и порядка связывания элементов различают:

- **Линейные односвязные списки** - единственное адресное поле содержит адрес следующего элемента или nil;
- **Кольцевые односвязные списки** - единственное адресное поле содержит адрес следующего элемента, а последний элемент ссылается на первый;
- **Линейные двусвязные списки** - каждый элемент содержит адреса предыдущего и следующего элемента, соответственно первый и последний элементы в соответствующих адресных полях содержат nil;
- **Кольцевые двусвязные списки** - каждый элемент содержит адреса предыдущего и следующего элемента, причём в соответствующих адресных полях первый ссылается на последний, а последний на первый;
- **n-связные списки** - каждый элемент включает несколько адресных полей, в которых записаны адреса других элементов или nil.

Пример описания элементов списка:

Односвязный список:

```
Type pe = ^element;      {тип указателя}
   element = record
      name: string[16];  {информационное поле 1}
      telefon:string[7]; {информационное поле 2}
      p: pe;             {адресное поле}
   end;
```

Двусвязный список:

```
Type pe = ^element;        {тип указателя}
   element = record
             name: string[16];  {информационное поле 1}
             telefon:string[7]; {информационное поле 2}
             prev: pe;    {адресное поле «предыдущий»}
             next: pe;    {адресное поле «следующий»}
   end;
```

Односвязные списки аналогично одномерным массивам организуют последовательность элементов.

Использование списков позволяет:

- работать с произвольным количеством элементов, добавляя и удаляя их по мере необходимости;
- осуществлять вставку и удаление записей, не перемещая остальных элементов последовательности.

Но:

- не допускают прямого обращения к элементу по индексу;
- требуют больше памяти для размещения.

Объявление типа элемента и переменных:

Описание типа элемента:

```
Type tpel=^element;   {тип «указатель на элемент»}
   element=record
      num:integer;  {число}
      p:tpel;     {указатель на следующий элемент}
   end;
```

Описание переменной – указателя списка и несколько переменных-указателей в статической памяти:

```
Var first,       {адрес первого элемента}
  n,f,q:tpel; {вспомогательные указатели}
```

Исходное состояние – «список пуст»:

```
        first:=nil;
```

Добавление элементов к списку:

1 Добавление элемента к пустому списку:

```
new(first);
first ^.num:=5;
first ^.p:=nil;
```

2 Добавление элемента перед первым (по типу стека):

```
new(q);
q^.num:=4;
q^.p:=first;
first:=q;
```

3 Добавление элемента после первого (по типу очереди):

```
new(q);
q^.num:=4;
q^.p:=nil;
first^.p:=q;
```

Удаление элемента осуществляется с помощью процедуры **dispose.**

Алгоритм:

1. Указатель предыдущего элемента приравнять указателю удаляемого.
2. Очищаем память по адресу удаляемого элемента.

Пример:

```
 t := q;
p^.pe := q^.pe;
q := q^.pe;
i := i + 1;
dispose(t);
```

Билет №14

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

Файловая система:

Файл – поименованная последовательность элементов данных (компонентов файла), хранящихся, как правило, во внешней памяти.

Как исключение данные файла могут не храниться, а вводиться с внешних устройств (ВУ), например клавиатуры или выводиться на ВУ.

Полное имя файла включает:

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

Пример полного имени файла:

D:\Dir1\Dir2\File9.pas

Имя файла в Windows составляют из строчных и прописных букв латинского и русского алфавитов, арабских цифр и некоторых специальных символов, например, символов подчеркивания «_» или доллара «$»

Расширение определяет тип хранящихся данных, например:

COM, EXE – исполняемые файлы (программы);

PAS, BAS, CPP – исходные тексты программ на алгоритмических языках ПАСКАЛЬ, БЭЙСИК и С++;

BMP, JPG, PIC – графические файлы (рисунки, фотографии);

WAV,MP3,WMA – музыкальные файлы.

Так как во внешней памяти хранится большое количество файлов, то для удобства и ускорения работы применяется древовидная структура каталогов. Главным является корневой каталог, не имеющий имени и создаваемый в процессе форматирования диска, в котором могут фигурировать каталоги второго уровня (подкаталоги), каждый из которых может содержать подкаталоги следующего уровня.

Различают два типа файлов: **Дисковые и файлы "логические устройства"**. Дисковый файл представляет собой поименованную область внешней памяти на устройстве хранения информации. Физически операции ввода-вывода с

файлами выполняются с использованием специального буфера, что позволяет существенно повысить скорость выполнения операций.
В отличие от дисковых файлов с логическими устройствами операции ввода-вывода осуществляются только последовательно, так как при выполнении операций вывода-вывода данные передаются покомпонентно.
Доступ к компонентам файла осуществляется через указатель файла.
При выполнении операции чтения или записи указатель автоматически перемещается на следующий компонент.

Файловые переменные делятся на:

- **Типизированные**;
- **Текстовые** (используют для работы с текстами, представленными в виде строк и переменной длины);
- **Нетипизированные** (применяют для скоростного обмена между внешней и оперативной памятью физическими записями указанной длины без преобразования и обработки).

Описание файловых переменных:

1. **Типизированные файлы: file of <Тип компонента>,**
где <Тип компонента> – любой тип данных, кроме файлового.

2. **Текстовые файлы: text**

3. **Нетипизированные файлы: file**

Примеры:

```
1) Var  F1: file of real;
       F2: file;
       F3: Text; …
2)   Type FF = file of integer;
       FR = file;
       FC = text;
  Var  F1:FF;
       F2,F3:FC;
       F4:FC; …
```

Работа с файлом включает:

- Инициализацию файловой переменной (assign, assignfile);
- Открытие файла (reset, rewrite, append);
- Обработку компонентов файла (создание, модификация, поиск записей);
- Закрытие файла (close, closefile).

Стандартные процедуры и функции: rename, erase, eof, ioresult, truncate, chdir, getdir, fileage...

Текстовые файлы:

Текстовый файл – файл, компонентами которого являются символьные строки переменной длины, заканчивающиеся специальным маркером – маркером «Конец строки».

В текстовом файле компоненты заканчиваются маркером конца строки (#13, #10). Текстовые файлы используют для хранения и обработки символов, строк, символьных массивов. Логические и числовые данные при записи в текстовые файлы должны преобразовываться в символьные строки. Текстовый файл можно открыть для записи, чтения и добавления записей в конец. Файл, открытый для записи, не может использоваться для чтения и наоборот.

Программе, работающей в консольном режиме, без объявления, инициализации файловой переменной и открытия доступны два стандартных текстовых файла, связанных с логическими устройствами ввода и вывода:

INPUT – файловая переменная для обозначения файла данных, вводимых с клавиатуры;

OUTPUT – файловая переменная для обозначения файла данных, выводимых на экран.

Процедуры и функции обработки текстовых файлов:

1. Функция EOLn([Var f]): Boolean – возвращает TRUE, если во входном текстовом файле достигнут маркер конца строки; при отсутствии файловой переменной проверяется файл INPUT, связанный с клавиатурой.
   - При работе с клавиатурой функция EOLn возвращает TRUE, если *последним считанным был* символ #13.
   - При работе с диском функция EOLn возвращает TRUE, если *следующим считанным будет* символ #13.
2. Процедура Read([Var f:text;]v1,v2,...vn) – обеспечивает ввод символов, строк и чисел. При вводе чисел пробелы и символы табуляции игнорируются. Если файловая переменная не указана, то ввод осуществляется из файла INPUT.
3. Процедура **ReadLn([Var f;][v1,v2,...,vn]) –** осуществляет ввод символов, строк и чисел. После чтения последней переменной оставшаяся часть строки до маркера конца строки пропускается так, что следующее обращение к ReadLn или Read начинается с первого символа новой строки.
4. Процедура **Write([Var f;]v1,v2, ...,vn)** – осуществляет вывод одного или более выражений типа CHAR, STRING, BOOLEAN, а также целого или вещественного типов. При выводе числовых значений последние преобразуются в символьное представление. Если файловая переменная не указана, то вывод осуществляется в файл OUTPUT.

Любой параметр из списка вывода может иметь формат:

<Параметр> [: <Целое1> [: < Целое2>]].

5. Процедура **WriteLn([Var f;][v1,v2, ...,vn])** – осуществляет вывод в текстовый файл. Если файловая переменная не указана, то вывод осуществляется в файл OUTPUT.

Выводимая строка символов завершается маркером конца строки. Если список вывода не указан, то в файл передается только маркер конца строки.

6. Функция **SeekEOLn([Var f]):boolean** – пропускает пробелы и знаки табуляции до маркера конца строки или до первого значащего символа и возвращает **TRUE**, при обнаружении маркера. Если файловая переменная не указана, то функция проверяет файл **INPUT**.

7. Функция **SeekEOF([Var f]):boolean** – пропускает все пробелы, знаки табуляции и маркеры конца строки до маркера конца файла или до первого значащего символа и возвращает **TRUE** при обнаружении маркера. Если файловая переменная отсутствует, то функция проверяет файл **INPUT**.

Нетипизированные файлы:

Нетипизированные файлы - файлы, объявленные без указания типа компонентов.

Операции чтения и записи с такими файлами осуществляются блоками, что позволяет организовать высокоскоростной обмен данными между диском и памятью. Отсутствие типа делает эти файлы совместимыми с любыми другими.

Нетипизированные файлы, как и типизированные, допускают организацию прямого доступа.

Нетипизированный файл можно открыть для записи и для чтения:

ReSet(Var f;[recsize:word]);

ReWrite(Var f;[recsize:word]);

где recsize – размер записи файла в байтах. Длину записи задают кратной 512 байт, например: 1024, 2048. Если длина записи не указана, она принимается равной 128.

Процедуры и функции для работы с нетипизированными файлами:

1. Процедура BlockRead(Var f:file; Var buf;Count:word[;Var res:word])– осуществляет чтение блока записей из файла в буфер buf.

Параметр res будет содержать количество фактически обработанных записей. Если последняя запись – неполная, то значение параметра res ее не учтет.

2. Процедура BlockWrite(Var f:file;Var buf;Count:word[;Var res:word])– осуществляет запись блока из буфера buf в файл.

Билет №15

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

Типизированный файл – файл, все компоненты которого одного типа, заданного при объявлении файловой переменной.
Компоненты хранятся на диске во внутреннем (двоичном) формате.

Типизированный файл можно открыть для записи и чтения. Файл, открытый для записи, может использоваться для чтения. В файл, открытый для чтения, можно писать.
Поскольку размер компонентов одинаков, принципиально возможен не только последовательный, но и прямой доступ.

Процедуры и функции обработки типизированных файлов:

1. **Процедура Read(Var f; c1,c2,...,cn)** – осуществляет чтение компонентов типизированного файла. Список ввода содержит одну или несколько переменных того же типа, что и компоненты файла. Если файл исчерпан, обращение к процедуре вызывает ошибку ввода-вывода.

2. **Процедура Write(Var f; c1,c2,...,cn)** – осуществляет запись компонентов в типизированный файл. Список вывода содержит одно или более выражений того же типа, что и компоненты файла.

3. **Процедура Seek(Var f; numcomp:longint)** – осуществляет установку указателя файла на компонент с номером numcomp.

4. **Функция FileSize(Var f):longint** – возвращает количество компонентов файла. Может использоваться для установки на конец файла совместно с Seek():

```
Seek(f, FileSize(f));
```

5**. Функция FilePos(Var f):longint** – возвращает порядковый номер компонента, который будет обрабатываться следующим.

6**. Процедура Truncate(Var f)** – выполняет «усечение» файла.

Билет №16, Билет №17

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

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

Объектно-ориентированное программирование – технология создания сложного программного обеспечения, основанная на представлении программы в виде системы объектов, каждый из которых является экземпляром определенного типа (класса), а классы образуют иерархию с наследованием свойств.

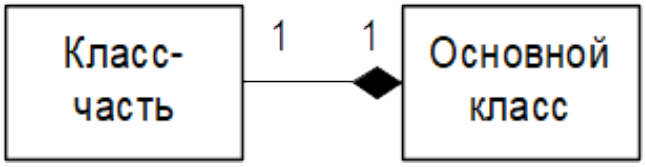
Класс - это структурный тип данных, который включает описание полей данных, а также процедур и функций, работающих с этими полями данных. Применительно к классам такие процедуры и функции получили название **методов**, а переменные - **полей**. Переменные типа класса обычно называют **объектами**.
Согласно общим правилам языка программирования объект-переменная должен быть: создан, инициализирован, уничтожен.

Методы построения классов:

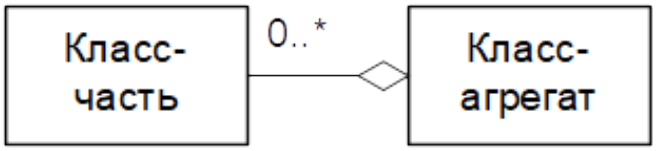
1) **Наследование –** механизм, позволяющий строить класс на базе более простого посредством добавления полей и определения новых методов. При этом исходный класс, на базе которого выполняется построение, называют родительским или базовым, а строящейся класс – потомком или производным классом.

   Если при наследовании какие-либо методы переопределяются, то такое наследование называется **полиморфным**.

2) **Композиция –** механизм, позволяющий включать несколько объектов других классов в конструируемый.



3) Наполнение – механизм, позволяющих включать указатели на объекты других классов в конструируемый.



Определение классов. Объявление объектов и инициализация полей.

С точки зрения синтаксиса ***класс*** – структурный тип данных, в котором помимо полей разрешается описывать ***прототипы*** (заголовки) процедур и функций, работающих с этими полями данных.

Объявление объекта класса:
Type <Имя класса> = object
<Описание полей класса>
<Прототипы методов>
end;

Доступ к полям и методам выполняется:

- с использованием точечной нотации;
- с использованием оператора with.

В консольном режиме Delphi Pascal можно **ограничить доступ к полям и методам** класса в пределах модуля. Для этого описание класса делится на специальные секции:

- **Public** - секция, содержащая описание общих или общедоступных полей и методов класса;
- **Private** - секция, содержащая описание внутренних или скрытых полей и методов класса.

Поля объекта должны инициализироваться. **Инициализация полей** осуществляется тремя способами:

- Прямым занесением данных в поле (Возможен только для общедоступных полей через точечную нотацию):
  A.length:=3.5;
  A.width:=5.1;

- С использованием типизированных констант (совпадает с синтаксисом описания типизированных констант типа "запись" через const);
  Var A:TRoom = (length:3.5; width:5.1):

- Посредством специального метода:

```
Procedure TRoom.Init;
 Begin  length:=l;  width:=w;  End;
                ...
Var A:TRoom;
Begin
   A.Init(3.5,5.1);
                ...
```

Неявный параметр Self:

Любой метод неявно получает параметр **Self** – ссылку (адрес) на поля объекта, и обращение к полям происходит через это имя.

```
Function TRoom.Square;
  Begin
     Result:= Self.length* Self.width;
  End;
```

При необходимости эту ссылку можно указывать явно:

@Self – адрес области полей данных объекта.

Билет №18

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

Объектно-ориентированное программирование – технология создания сложного программного обеспечения, основанная на представлении программы в виде системы **объектов**, каждый из которых является экземпляром определенного **типа (класса)**, а классы образуют иерархию с наследованием свойств.

Объектная декомпозиция - процесс представления предметной области задачи в виде совокупности функциональных элементов (**объектов**), обменивающихся в процессе выполнения программы входными воздействиями (**сообщениями**).
Объект отвечает за выполнение некоторых действий, зависящих от полученных сообщений и параметров самого объекта.

Процедурная декомпозиция - процесс разбиения программы на подпрограммы.
Структурной называют декомпозицию, если:

- Каждая подпрограмма имеет один вход и один выход;
- Подпрограммы нижних уровней не вызывают подпрограмм верхних уровней;
- Размер подпрограммы не превышает 40-50 операторов;
- В алгоритме использованы только структурные конструкции.

Диаграмма класса:

Имя класса
Поля
Методы

Обычно классы строят на базе уже существующих, используя механизмы, реализующие определённое отношение существующего и строящегося классов между собой:

- **Наследование:**
  При наследовании один класс строится на базе второго посредством добавления полей и определения новых методов. Исходный класс называют родительским или базовым, а строящийся - потомком, или производным, или подтипом. При наследовании поля и методы родительского класса повторно не определяют, специальный механизм наследования позволяет использовать эти компоненты класса, особо этого не оговаривая.

- **Композиция:**
  При композиции один класс является неотъемлемой частью другого. Физически композиция реализуется включением в классе фиксированного количества полей, являющихся объектами другого класса. Такие поля обычно называют объектными.

- **Наполнение (агрегация):**
- При наполнении (агрегации) точное количество одного класса, включаемых в другой класс, не ограничено и может меняться от 0 до заранее неизвестных значений. Физически наполнение реализуется с использованием указателей на объекты.

- **Полиморфное наследование:**
  При полиморфном наследовании осуществляются переопределение методов класса-родителя потомком. Метод потомка в этом случае имеет то же имя, что и метод родителя, но выполняет другие действия.

Полиморфизм в языках программирования и теории типов – это способность функции обрабатывать данные разных типов.

Сложный полиморфизм – это _____
_____
_____
_____
_____
_____
_____
_____

Три случая **обязательного** использования **сложного полиморфизма**:

- Если наследуемый метод для объекта производного класса вызывает метод, переопределенный в производном классе.
- Если объект производного класса через указатель базового класса обращается к методу, переопределенному производным классом.
- Если процедура вызывает переопределенный метод для объекта производного класса, переданного в процедуру через *параметр-переменную*, описанный как объект базового класса («процедура с полиморфным объектом»).

Билет №19
Динамические объекты и объекты с динамическими полями в консольном режиме Delphi.

Создание полиморфных объектов:

Функция New(<Тип указателя>) – возвращает адрес размещенного и, возможно, сконструированного объекта.

Динамические объекты создаются через функцию **New(<Тип указателя>)**. Если объект не был сконструирован с помощью new, то необходимо вызвать конструктор. Для корректного освобождения памяти следует предварительно вызвать деструктор.

Деструктор - метод класса, который используется для корректного уничтожения полиморфного объекта, содержащего невидимое поле. Деструктор можно переопределять.

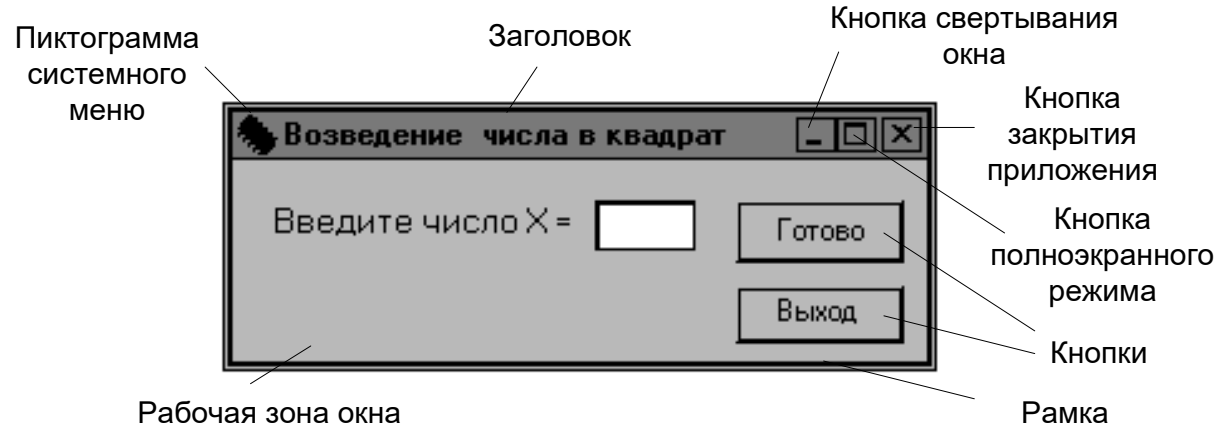
Чтобы уничтожить полиморфные объекты надо вызвать процедуру **Dispose(<Указатель>)** (перед вызовом процедуры необходим вызов деструктора, если он указан, и затем –выполняется освобождение памяти.)

Для реализации полиморфных объектов можно использовать динамические поля. **Динамическое поле** - это поле типа указателя, с помощью которого можно удобно реализовать композицию. В деструкторе необходимо освободить память, которая была выделена под динамические поля.

Билет №20
Технология событийного программирования. События Windows, сообщения и события Delphi. Основные события Delphi.

Событийным называется программирование, при котором программа представляет собой набор обработчиков некоторых событий. В качестве событий при этом могут интерпретироваться как нажатие какой-либо визуальной кнопки в окне программе, так и некоторые ситуации в самой программе. Таким образом основной цикл программы включает ожидание какого-либо события, вызов соответствующего обработчика, после чего цикл повторяется.
Такая программа не имеет алгоритма в традиционном смысле, так как связь между отдельными частями не задана жестко, а зависит от последовательности наступления тех или иных событий.

В операционной системе типа Windows взаимодействие между системой и приложениями реализуется посредством передачи сообщений. Каждый раз, когда в компьютере происходят какие-либо события, например, пользователь перемещает мышку, нажимает на ней клавишу, нажимает клавишу на клавиатуре, или происходит другое значимое событие для **приложения**, операционная система формирует специальное сообщение, которое называется **сообщением Windows**.
Приложение (в отличие от программы) – набор подпрограмм, вызываемых при наступлении некоторого события, которым считается любое изменение в системе, касающееся данного приложения.
Каждому приложению на экране соответствует окно:



Окно – самостоятельно существующий объект, параметры которого хранятся в специальной структуре данных, а поведение определяется обработчиками сообщений, составляющими ***оконную функцию***.

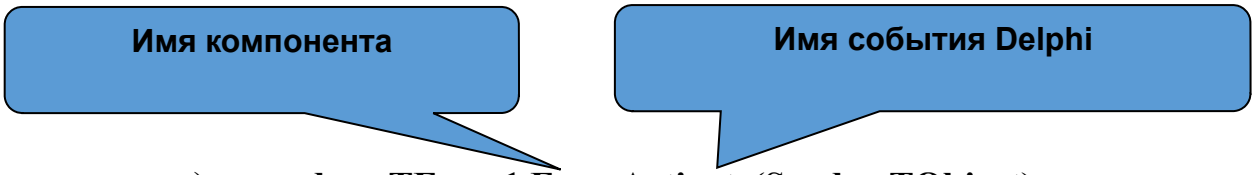
Сообщения Windows поступают в очередь сообщений приложения, откуда выбираются приложением, расшифровываются и обрабатываются.

Событие Delphi - условная ситуация, которая фиксируется Delphi, при получении информации о конкретном событии Windows с определёнными параметрами. При обнаружении каждой такой ситуации обработчик сообщения Windows осуществляет вызов соответствующего обработчика события Delphi.

Обработчики сообщений Windows предусмотрены у объектов класса **TForm** и классов управляющих компонентов, таких как кнопки, редакторы и т. п. Обработка выполняется следующим образом:

- В системе происходит событие и генерируется сообщение об этом событии - событие Windows;
- Сообщение Windows передаётся конкретному приложению, активному компоненту активного окна этого приложения;
- Стандартный метод обработки сообщения Windows компонента дешифрирует сообщение и генерирует заранее предусмотренные события Delphi;
- Если в приложении предусмотрены соответствующие обработчики событий Delphi , то они выполняются, если нет - то приложение возвращается в состояние ожидания.

Для каждого обработчика событий предусмотрен заголовок и определеный список передаваемых ему параметров:



а) **procedure TForm1.FormActivate(Sender:TObject);**

Параметр **Sender –** отправитель (инициатор события).

б) **procedure TForm1.Edit1KeyPress(Sender: TObject;**
var Key: Char);

Параметр **Key –** символ ANSI**.**

Класс формы TForm:

Основное окно приложения строится на базе класса **TForm**.

При входе в Delphi предоставляется заготовка приложения, которая «умеет» (содержит определенные обработчики сообщений Windows) выполнять стандартные действия.

Свойства:

Caption – заголовок окна (по умолчанию Form1);

FormStyle – вид формы(обычное, родительское, дочернее, неперекрываемое);

Width, Height, Top, Left – размеры и местоположение;

Color – цвет фона;

Font – характеристики шрифта;

Cursor – форма курсора мыши и т. д.

Методы:
Show – показать;
Hide – спрятать;
Close – закрыть;
ShowModal – показать в модальном режиме и т. д.

Основные события Delphi:

а) При изменении состояния формы:

OnCreate – в начальной стадии создания формы - используется при необходимости задания параметров (цвет или размер);

OnActivate – при получении формой фокуса ввода (окно становится активным и ему адресуется весь ввод с клавиатуры);

OnShow – когда форма (окно) становится видимой;

OnPaint – при необходимости нарисовать или перерисовать форму;

OnResize - при изменении размеров формы на экране;

OnDeactivate – при потере формой фокуса ввода (окно становится неактивным);

OnHide – при удалении формы с экрана (окно становится невидимым);

OnCloseQuery – при попытке закрыть форму - обычно используется для создания запроса-подтверждения необходимости закрытия окна;

OnClose – при закрытии формы;

OnDestroy – при уничтожении формы;

б) От клавиатуры и мыши:

OnKeyPressed – при нажатии клавиш, которым соответствует код ANSI;

OnKeyDown, OnKeyUp – при нажатии и отпускании любых клавишI;

OnClick, OnDblClick – при обычном и двойном нажатии клавиш мыши;

OnMouseMove – при перемещении мыши (многократно);

OnMouseDown, OnMouseUp – при нажатии и отпускании клавиш мыши;