Билет 1

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

**Синтаксис** – правила, определяющие допустимые конструкции языка. «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.

**Пример**: if, then, else – синтаксически корректные единицы, но выражение if else then; не является семантически корректным предложением ЯП pascal.

**Алфавит** языка програмирования 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, а..f или A..F;

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

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

**Синтаксис** – правила, определяющие допустимые конструкции языка, построенные из символов его алфавита.

Буква

Буква

Цифра

Синтаксическая диаграмма

Билет 2

# Представление данных в языке Pascal: константы и переменные. Классификация скалярных типов данных, их внутреннее представление, операции над ними. Примеры.

**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:

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

**Тип** – описатель данных, который определяет:

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

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



1. ***Целые типы:***

**Integer, LongInt** (4 байта со знаком):  **-2147483648..2147483647;**

**SmallInt** (2 байта со знаком): **-32768..32767**

**ShortInt** (1 байт со знаком): **-128..127;**

**Word** (2 байта без знака): **0..65535;**

**Byte** (1 байт без знака): **0..255.**

**Пример: Var a,b:word;с:shortint;**

2. ***Символьные типы:***

**Char,** **AnsiChar** (1 байт без знака) – код символа по таблице ANSI;

**WideChar** (2 байта без знака) – код символа по таблице Unicode

3. ***Булевский тип:***

**Boolean** (1 байт без знака: 0 – false, 1 - true)

4. ***Перечисление*** – значения переменных этого типа описываются явно (перечисляются).

**Пример:**

**Туpe Day = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);**

**Var D:Day;**

или

**Var D:(Mon, Tue, Wed, Thu, Fri, Sat, Sun);**

D:=Fri; **// присваивание переменной D значения Fri**

5. ***Отрезок*** – значения переменных этого типа входят в определенный диапазон значений стандартного типа.

**Пример:**

**Туpe Date = 1..31; // значения – числа от 1 до 31**

**Var DataN: Date;**

или

**Var DataN: 1..31;**

1. **Ord** (<Выражение порядкового типа>) – возвращает номер значения по порядку (не применима к 64 битным аргументам).

**Пример:** **Ord(’A’)= 65 // номер символа в таблице ANSI**

2. **Pred** (<Выражение порядкового типа>) – возвращает предыдущее значение.

**Dec**(<Целое>) – возвращает значение, уменьшенное на 1.

**Пример:** **N:=5; k:= Pred(N){k=4}; m:= Dec(N){m=4};**

3. **Succ** (<Выражение порядкового типа>) – возвращает следующее значение.

**Inc**(<Целое>) – возвращает целое, увеличенное на 1.

**Пример:** **N:=5; k:= Succ(N){k=6}; l:= Inc(N){l=6};**

4. **High** (<Идентификатор>) – возвращает самое большое значение типа, также работает со строками и массивами (см. далее).

5. **Low**(<Идентификатор>) – возвращает самое маленькое значение типа, также работает со строками и массивами (см. далее).

Вещественные

Тип Значащих Диапазон

цифр порядка

Real (8 байт) (в старших версиях) 15-16 -324..308

Single (4 байта) 7-8 -45..38

Double (8 байт) 15-16 -324..308

Extended (10 байт) 19-20 -4951..4932

Comp (8 байт) 19-20 -263+1..263-1

1. **Арифметические** **операции** – применяют к вещественным и целым константам и переменным:

+, -, \*,

/ {вещественное деление},

div {целочисленное деление},

mod {остаток от деления}

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

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

not, and, or, xor

4. **Поразрядные операции** – выполняются поразрядно, применяют к целым, результат – целое число:

not, and, or, xor, shr {сдвиг вправо}, shl {сдвиг влево}

1. Порядок выполнения операций определяется ***приоритетами и скобками***

Операции Приоритет

@, not 1

\*, /, div, mod, and, shr, shl 2

+, -, or, xor 3

<, >, <=, >=, =, <> 4

Билет 3

# Совместимость типов данных и операции преобразования типов. Примеры.

2. При выполнении арифметических операций над числами различных типов автоматически осуществляется ***неявное преобразование:***

* целого и вещественного типов – к вещественному,
* с разными интервалами представлений – к типу с большим интервалом.
* 3.При сравнении вещественных чисел из-за их неточного представления проверку равенства и неравенства следует осуществлять ***с явным указанием допуска*.**
* **Пример:**
* **Var x,y:single;**
* **x <> y** ⇒ **abs(x-y) > 1e-10**
* **x = y** ⇒ **abs(x-y) < 1e-10**
* 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.
* **ord**(<Порядковое выр.>) – преобразует значение в его номер.
* **Пример:** **ord(’A’)** = 65.
* **chr**(<Ц. выр.>) – преобразует номер символа в символ.
* **Пример:** **chr(65)** = ’A’.

**Присваивание**

* Корректное выполнение оператора предполагает, что результат вычисления и переменная правой части ***одного типа*** или ***совместимы по типу*.**
* По правилам ***совместимы*:**
* а) все целые типы между собой;
* б) все вещественные типы между собой;
* в) отрезок базового типа и базовый тип;
* г) два отрезка одного и того же базового типа;
* д) символ и строка.

Билет 4

# Присваивание, условный оператор, оператор выбора. Синтаксис операторов, их особенности и примеры использования.

**Присваивание**

Используется для изменения значений переменных.

* Корректное выполнение оператора предполагает, что результат вычисления и переменная правой части ***одного типа*** или ***совместимы по типу*.**
* По правилам ***совместимы*:**
* а) все целые типы между собой;
* б) все вещественные типы между собой;
* в) отрезок базового типа и базовый тип;
* г) два отрезка одного и того же базового типа;
* д) символ и строка.



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

Оператор – простой или составной оператор языка.



Составной оператор – блок операторов в операторных скобках

**begin** …**end**.

**Оператор выбора**

Оператор позволяет программировать несколько вариантов решения.



**Пример:**

**case 1+2\*j of**

**3: z:=sin(x);**

**-1..1,10: z:=cos(x);**

**else z:=0;**

**end;**

Билет 5

# Операторы циклов языка Pascal. Синтаксис операторов, их особенности и примеры использования.

**Циклы**

**Счетные**

**Цикл-для**

**Цикл-пока**

**Цикл-до**

**Поисковые**

**Итерационные**

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

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

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



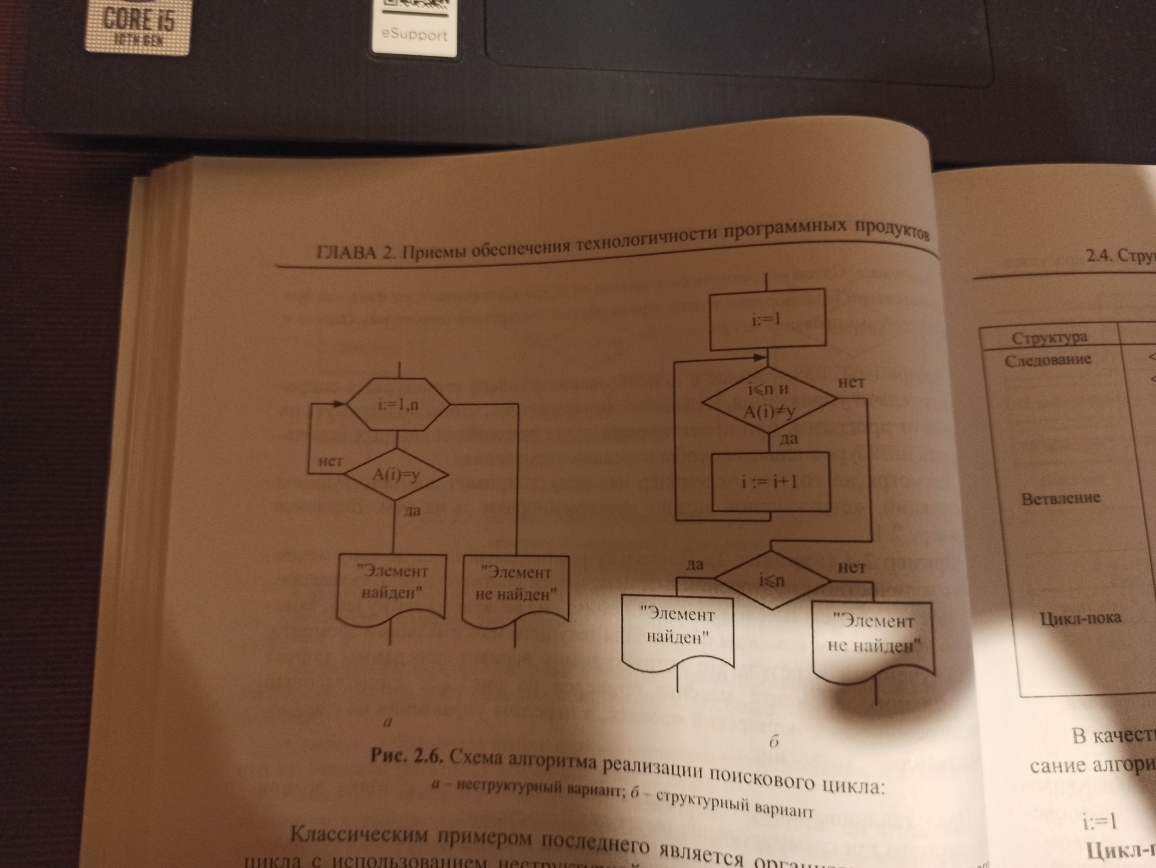




Билет 6

# Поисковый цикл. Неструктурная и структурная реализации поискового цикла.

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



Билет 7

# Массивы языка Pascal. Описание, внутреннее представление, операции над массивами и их элементами. Примеры.

***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.

Операции:

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

2. Доступ к элементу массива, прямой или косвенный (индекс – литерал или переменная соотв)

3. Ввод/вывод массивов осуществляется поэлементно

Билет 8.

# Строки языка Pascal. Описание, внутреннее представление, операции над строками и их элементами. Примеры.

Строка – последовательность символов.



Целое – максимальная длина строки.

Внутренний формат:

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);**

. Функция **Length(st):word** – возвращает длину строки st:

**n:=Length(st1);**

2. Процедура **Delete(st, index, count)** – удаляет count символов строки st, начиная с символа с номером index:

**S1: = ′dddddsssssfffff′;**

**Delete(S1,6,5);**

3. Процедура **Insert(St2,St1,index)** – вставляет подстроку символов St2 в строку St1, начиная с символа с номером index:

**S1 = ′dddddddddd′;**

**S2 = ′аааааа′ ;**

**Insert(S2,S1,6);**

**Insert(′Pas’,S1,6);** 4. Процедура **Str(x**[**:w**[**:d**]]**,St)** – преобразует результат выражения x, в строку st, содержащую запись этого числа в виде последовательности символов (как при выводе).

**x:=-5.67;**

**Str(x:7:3,s1);**

5. Процедура **Val(St,x,Code)** – преобразует строку St с записью числа в виде последовательности символов во внутреннее представление целого или вещественного числа и помещает его в переменную x. В целочисленной переменной Code процедура возвращает код ошибки:

**Var S:string; Code:integer; a:real; ...**

**...repeat**

**Write(′Input a:′);**

**ReadLn(S);**

**Val(S,a,Code);**

**if Code<>0 then WriteLn(′Input error′);**

**until Code=0; ...**  **6. Функция Copy(St,index,count):string – возвращает фрагмент строки St, длиной count символов, начиная с символа с номером index:**

**S1 = ′qqqEEEEEEuuuuu′;**

**S:= Copy(S1,4,6);**

**7. Функция Pos(St2,St1):integer – возвращает номер позиции первого вхождения подстроки St2 в строку St1. Если вхождение не найдено, то функция возвращает 0:**

**S1 = ′qqqEEррEEuuuuu′;**

**i:= Pos(′EE′,S1);**

**8. Функция UpCase(ch):char – возвращает символ, соответствующий символу верхнего регистра для ch, если таковой имеется, либо сам символ ch, если для него не определен символ верхнего регистра.**



Билет 9.

# Множества языка Pascal. Описание, внутреннее представление, операции над множествами и их элементами. Примеры.

***Множество*** – неупорядоченная совокупность неповторяющихся элементов.

Тип элементов – порядковый, кроме Word, Integer, SmallInt, LongInt. Количество элементов не должно превышать 256.

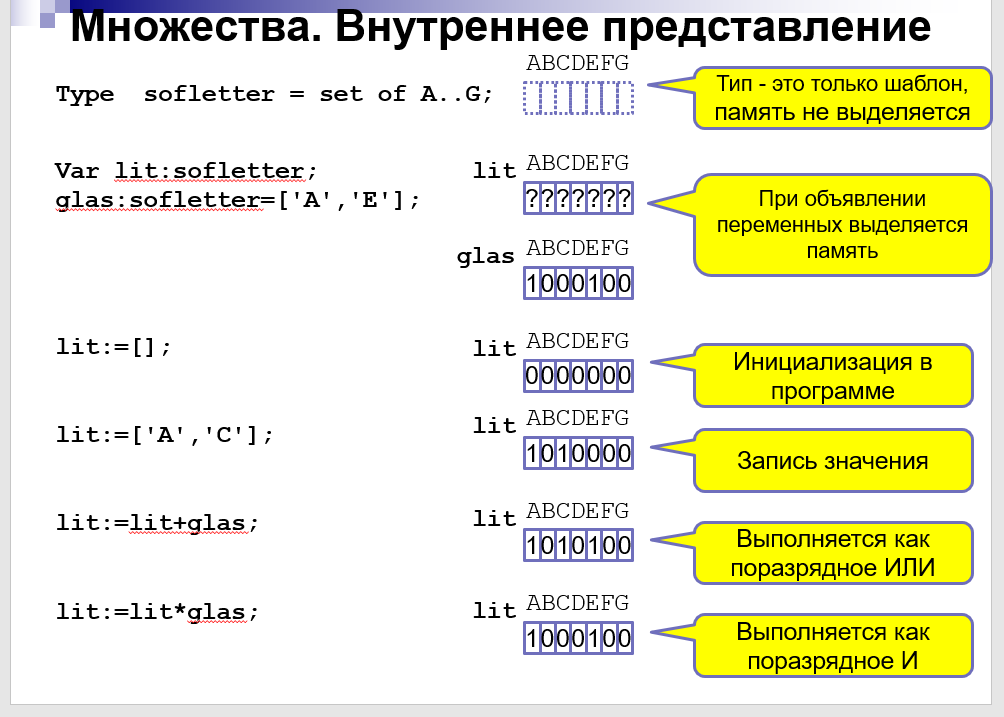
1. Присваивание:

**A:=B;**

**A:=[];**

2. Объединение, пересечение и дополнение:

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



Билет 10

# Записи языка Pascal. Описание, внутреннее представление, операции над записями и их элементами. Примеры.

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

1. Присваивание записей одного типа:

2. Доступ к полям записи: точкой или with … do …

3. Ввод и вывод записей осуществляется по полям.

Билет 11

# Процедуры и функции. Определение, описание, особенности. Примеры.

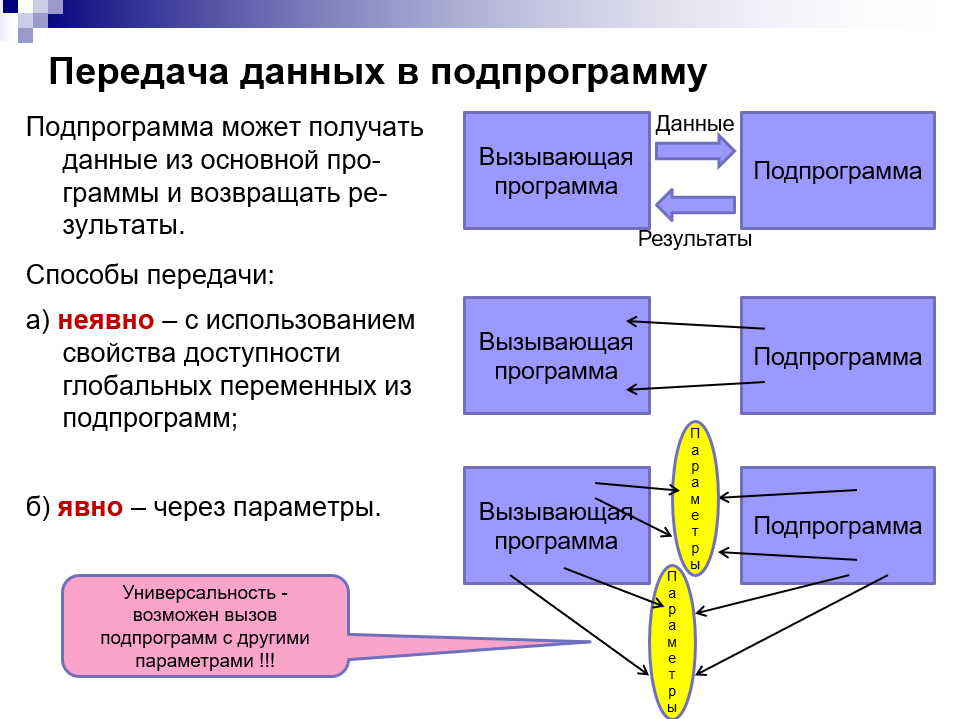
**Процедуры и функции** – самостоятельные фрагменты программы, соответствующим образом оформленные и вызываемые по имени (программные блоки).





Билет 12.

# Способы передачи данных в подпрограмму на языке Pascal. Примеры.



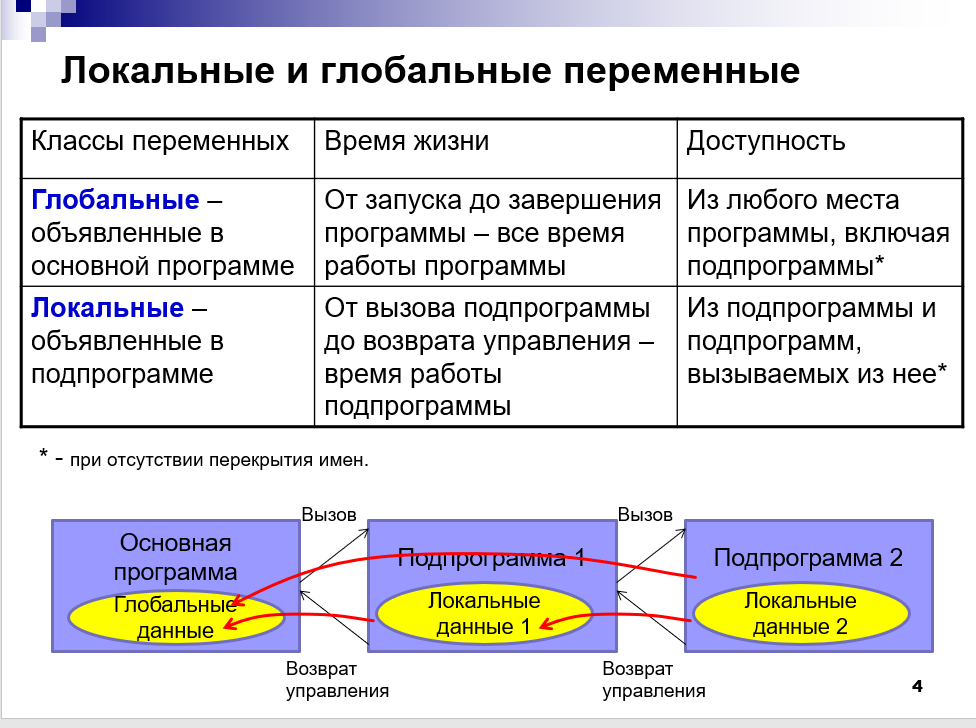
Недостатки неявной передачи данных:

1) жестко связывает подпрограмму и данные;

2) приводит к большому количеству ошибок.

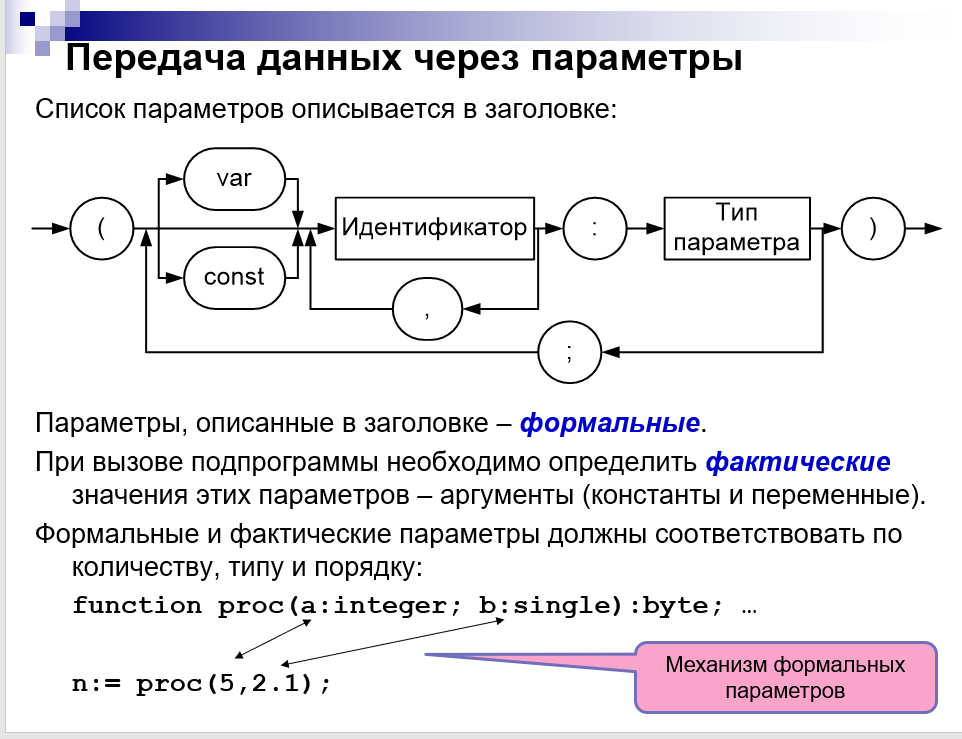
Билет 13.

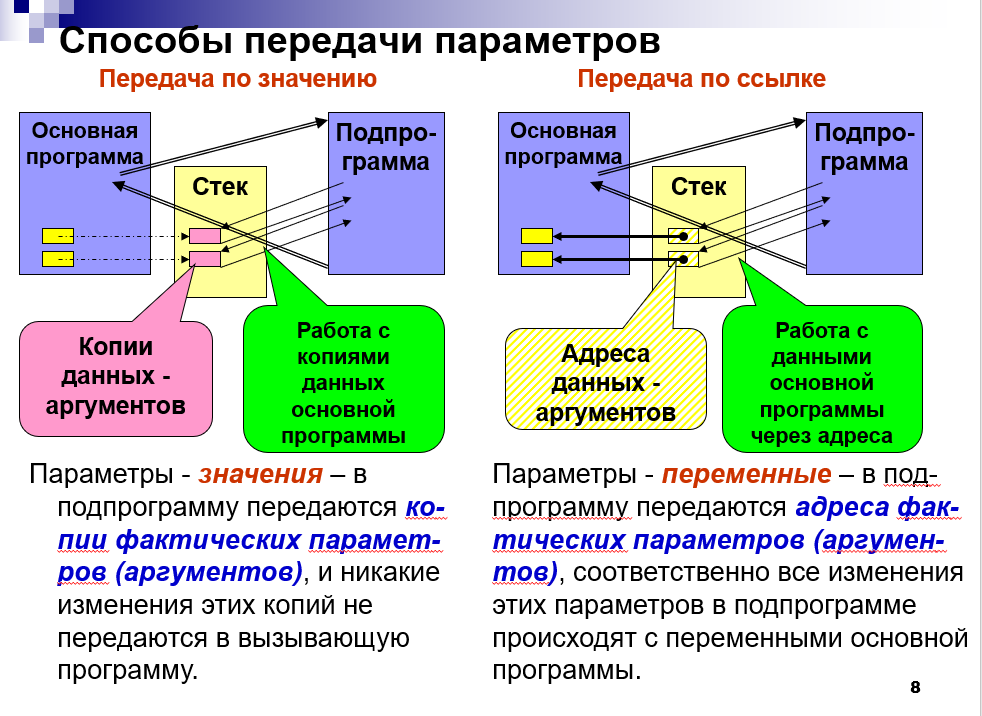
# Локальные и глобальные переменные, законы «видимости» идентификаторов. Примеры.



Билет 14.

# Формальные и фактические параметры подпрограмм языка Pascal. Примеры.





* ***Параметры-значения*** при описании подпрограммы не помечаются, например:

**function Beta(x:single; n:byte):integer;** .

* ***Параметры-переменные*** при описании подпрограммы помечаются служебным словом **var**, например:

**function Alpha(x:single; Var n:byte):integer;** .

*Ограничение: в качестве фактических значений параметров-переменных* ***нельзя использовать******литералы***:

**Alpha(2.5,5);** // ошибка!

правильно: **n:=5; Alpha(2.5,n);**

* ***Параметры-константы*** – в подпрограмму, так же как и в случае параметров-переменных, передаются ***адреса фактических параметров***, но при попытке изменить значение параметра компилятор выдает сообщение об ошибке; такие параметры при описании подпрограммы помечаются служебным словом **const**, например:

**function Alpha(const x:single; n:byte);** .

***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(а)+10; ...**

2) наложенное описание переменной определенного типа:

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

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

Билет 15.

# Параметры-строки, параметры-массивы. Примеры.

Структурные типы параметров (массивы, строки, множества, записи, указатели, файлы) должны быть предварительно объявлены.

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

P.S. Возможно использование конструкции открытого массива для объявления динамических массивов, память под которые выделяется во время выполнения программы во время выполнения процедуры SetLength.

**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>).