2016

# Основы программирования

МГТУ им. Н.Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети
Лектор: д.т.н., проф.
Иванова Галина Сергеевна

# Дисциплина Основы программирования

**Содержание дисциплины:**

*Модуль 1*. Основы алгоритмизации и программирование с использованием скалярных типов данных.

*Модуль 2*. Структурные типы данных и модульное программирование.

*Модуль 3*. Организация данных на внешних носителях и в оперативной памяти.

*Язык программирования:* **Паскаль (Delphi Pascal)**
*Среда программирования:* **Turbo Delphi 2006 (Free version)**

**Объем дисциплины – 7 зачетных единиц – 256 часов:**

- лекции – 51 час - знакомство с теоретическим материалом;
- семинары – 34 часа - разработка алгоритмов решения задач;
- лабораторные работы – 34 часа – 8*4+2(зачет) часов - изучение приемов программирования;
- самостоятельная работа – 16*6 часов - закрепление материала.

# Расписание лабораторных работ:

ИУ6-11 – числ. пятница $10^{15}$-$13^{35}$

ИУ6-12 – знам. пятница $10^{15}$-$13^{35}$

ИУ6-13 – числ. понедельник $8^{30}$-$11^{50}$

ИУ6-14 – знам. понедельник $8^{30}$-$11^{50}$

ИУ6-15 – числ. понедельник $12^{00}$-$15^{25}$

Место проведения: кафедра КС и С, ауд. № 805 (ГК, 8 этаж)

**С собой иметь:** тетрадь, ручку, карандаш, линейку, флешку, материалы лекций или учебник.

**Посещение всех занятий обязательно!**

Отчетность по дисциплине:

- 3 рубежных контроля (РК):
  - РК 1. Итерационные циклы - 2 часа – 5-6 недели.
  - РК 2. Матрицы и подпрограммы - 2 часа – 10-11 недели.
  - РК 3. Файлы и дин. память - 2 часа – 15-16 недели.
- экзамен.

# Учебные материалы

**Учебники**:

1. Иванова Г.С. Программирование: Уч. для ВУЗов – М.: Кнорус, 2014.
2. Алексеев Ю.А., Ваулин А.С., Куров А.В. Практикум по программированию: Обработка числовых данных. Учебное пособие. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2008.

**Материалы** (задания, методички и слайды) – на сайте кафедры ИУ6.


**Turbo Delphi:**

http://code-man.narod.ru/delphi/setup/turbo/

# Консультации проф. Г.С. Ивановой

**Консультации проводятся:**

**а) лично - на кафедре ИУ6 (главное здание, 8 этаж, ауд. 807):**

- вторник числ. с 13-30 до 14-30;
- пятница с 12-00 до 13-00

**б) по электронной почте:**

**gsivanova@gmail.com**

# Анкета

1. **Фамилия, имя, отчество, адрес эл. почты**
2. **Оцените свой уровень владения компьютером:**
    1. **Знаком с клавиатурой, играл в игры…**
    2. **Могу скопировать файлы на флешку и обратно…**
    3. **Хорошо ориентируюсь в файловой системе, могу установить пути для программ…**
3. **Укажите, с какими операционными системами работали?**
4. **Изучали ли вы программирование в школе? Сколько лет?**
5. **Какие языки программирования изучали? В каких средах?**
6. **Оцените уровень ваших знаний:**
    1. **Имею представление о программировании…**
    2. **Могу посчитать площадь треугольника…**
    3. **Могу решать задачи на обработку матриц…**
    4. **Могу использовать динамические структуры данных…**
    5. **Могу использовать объектно-ориентированное программирование…**
7. **Есть ли дома доступ к компьютеру?**
8. **Есть ли дома доступ к Интернету?**

# Часть 1. Основы алгоритмизации и процедурное программирование

МГТУ им. Н.Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети
Лектор: д.т.н., проф.
Иванова Галина Сергеевна

# Введение

***Паскаль*** – универсальный язык программирования высокого уровня. Поддерживает структурный и объектный подходы. Первоначально предназначен для обучения студентов, затем, в совокупности со средой программирования Turbo Pascal, стал профессиональным.

***Автор языка***: Николаус Вирт, Цюрих, Швейцария.

***Год создания языка:*** 1971 г.

В основе языка хорошо продуманные, логически стройные концепции. Язык имеет простой, но хорошо защищенный синтаксис и сравнительно ясную семантику, что упрощает обучение азам программирования.

**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.

***Delphi Pascal*** – одна из реализаций языка программирования Паскаль, используемая в среде быстрой разработки программ Delphi.

# Среды программирования

***Среда программирования*** – собранная в единую программную систему совокупность программных средств, предназначенный для разработки программных продуктов. Обычно включает: редактор текстов, компилятор языка программирования, компоновщик, отладчик, библиотеки подпрограмм и/или классов и т.п.

Среда программирования ***Turbo Delphi*** – бесплатная для обучающихся (*free*) версия среды Delphi, которая является частью пакета разработки Windows-приложений **Borland Developer Studio 2006**. Лицензия дана сроком на 10 лет.

Среда программирования ***Lazarus*** – бесплатная профессиональная многоплатформная среда разработки программ, по основным функциональным возможностям совместимая с Turbo Delphi. Имеет схожий интерфейс, но последний включает много отдельных окон, в которых начинающим программистам тяжело разобраться.

# Этапы создания ПО

1. **Постановка задачи** – неформальное описание задачи.

2. **Анализ и уточнение требований** – формальная постановка задачи и выбор метода решения.

3. **Проектирование** – разработка структуры программного продукта, выбор структур данных, выбор метода решения, разработка алгоритмов обработки данных, определение особенностей взаимодействия с программной средой и т.п.

4. **Реализация** – составление программ, их тестирование и отладка.

5. **Модификация** – выпуск новых версий.

# Пример разработки программы

1. **Постановка задачи**: Разработать программу, которая определяет наибольший общий делитель (НОД) двух целых чисел.
2. **Анализ и уточнение требований:**

   1) *Функциональные требования*

   исходные данные: a, b – натуральные числа; 0 < a, b < **?** ;

   результат: x – натуральное число, такое, что

**$x = \max \{y_i \,/\, i = \overline{1,n}\}$, где $((a \bmod y_i) = 0) \,\&\, (b \bmod y_i) = 0)$**

   Методы решения:

   а) найти делители Y = { $y_i$ } и определить x = max {Y};

   б) метод Евклида

Пример 1:

| a | b |
|---|---|
| 24 | 18 |
| 6 | 18 |
| 6 | 12 |
| 6 = | 6 |

Пример 2:

| a | b |
|---|---|
| 3 | 4 |
| 3 | 1 |
| 2 | 1 |
| 1 = | 1 |

# Пример разработки программы (2)

2) *Эксплуатационные требования*:

а) операционная система – Windows XP и выше (консольный режим);

б) процессор – не ниже Pentium;

в) предусмотреть запрос на ввод данных с клавиатуры;

г) результаты вывести на экран.

3) *Технологические требования*:

а) язык программирования: Pascal;

б) среда программирования: Turbo Delphi 2006 (free);

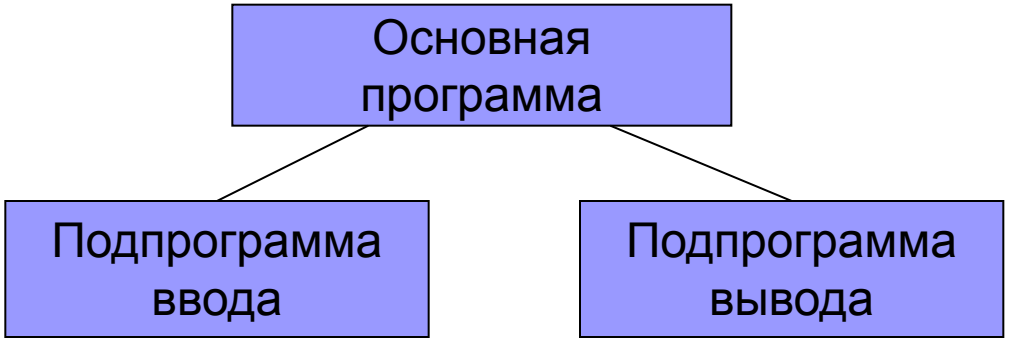
в) технология программирования: структурный подход.

# Пример разработки программы(3)

**3. Проектирование**

**Виды проектной документации:**

**1. Структурная схема ПО –** показывает взаимодействие по управлению основной программы и подпрограмм.



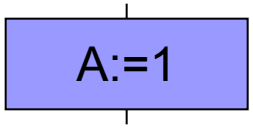

**2. Схемы алгоритмов программы и подпрограмм**

# Схемы алгоритмов

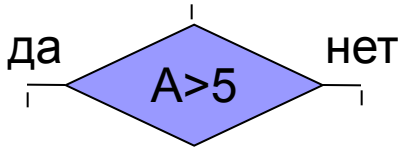
Обозначения по ГОСТ 19.701 – 90

1. Терминатор (начало/конец)

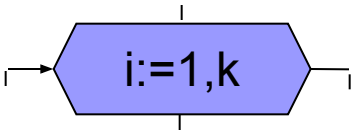

2. Процесс (вычисление)



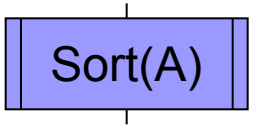

3. Анализ (проверка)



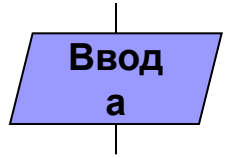

4. Модификатор (автоматическое изменение)



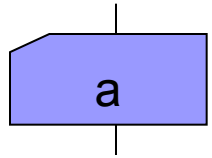

5. Предопределенный процесс (подпрограмма)



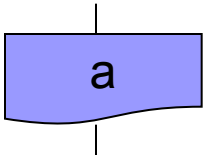

6. Ввод/вывод данных




7. Ввод с перфокарт

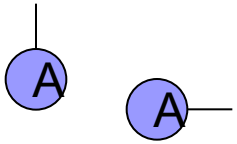



8. Вывод на принтер




9. Комментарий


10. Соединитель



# Правила выполнения схем алгоритмов

- Схемы алгоритмов должны быть выполнены аккуратно, желательно с применением карандаша и линейки или графических редакторов на компьютере.
- Стрелки на линиях, идущих **сверху вниз** и **слева направо**, т. е. в направлении письма, **не ставят**, чтобы не затенять схему.
- Если линия – ломанная, и направление ее хотя бы в одном сегменте не совпадает со стандартными, то стрелка ставится только **в конце линии**, перед блоком, в который она входит.
- Если схема не умещается на странице или линии многократно пересекаются, то линии разрывают. Один соединитель ставится в месте разрыва, второй – в месте продолжения линии. Оба соединителя помечаются одной и той же буквой или цифрой.
- Для простоты чтения схемы ее начало должно быть сверху, а конец – снизу. При этом количество изгибов, пересечений и обратных направлений соединительных линий должно быть **минимальным**.

# Пример неудачного изображение схемы

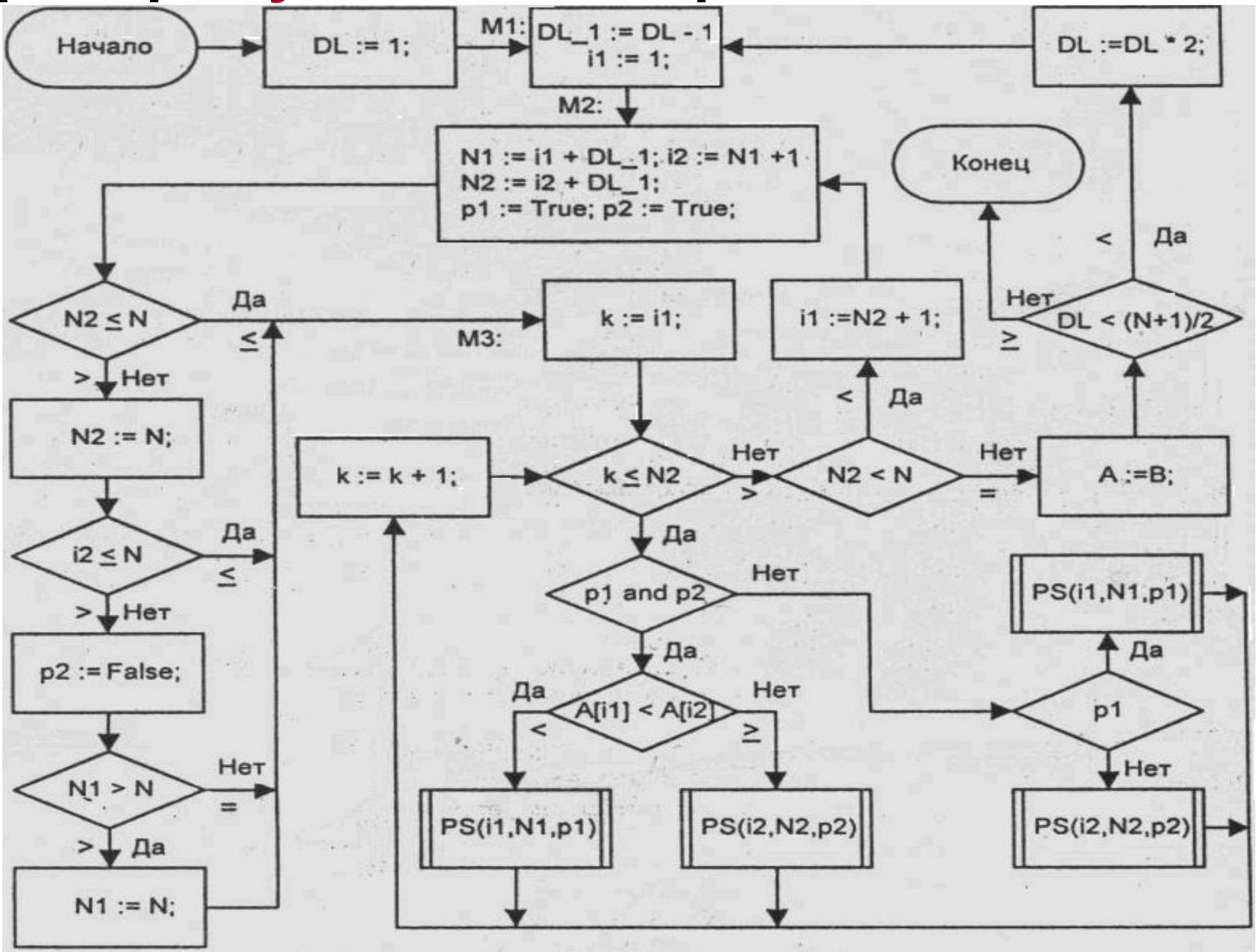
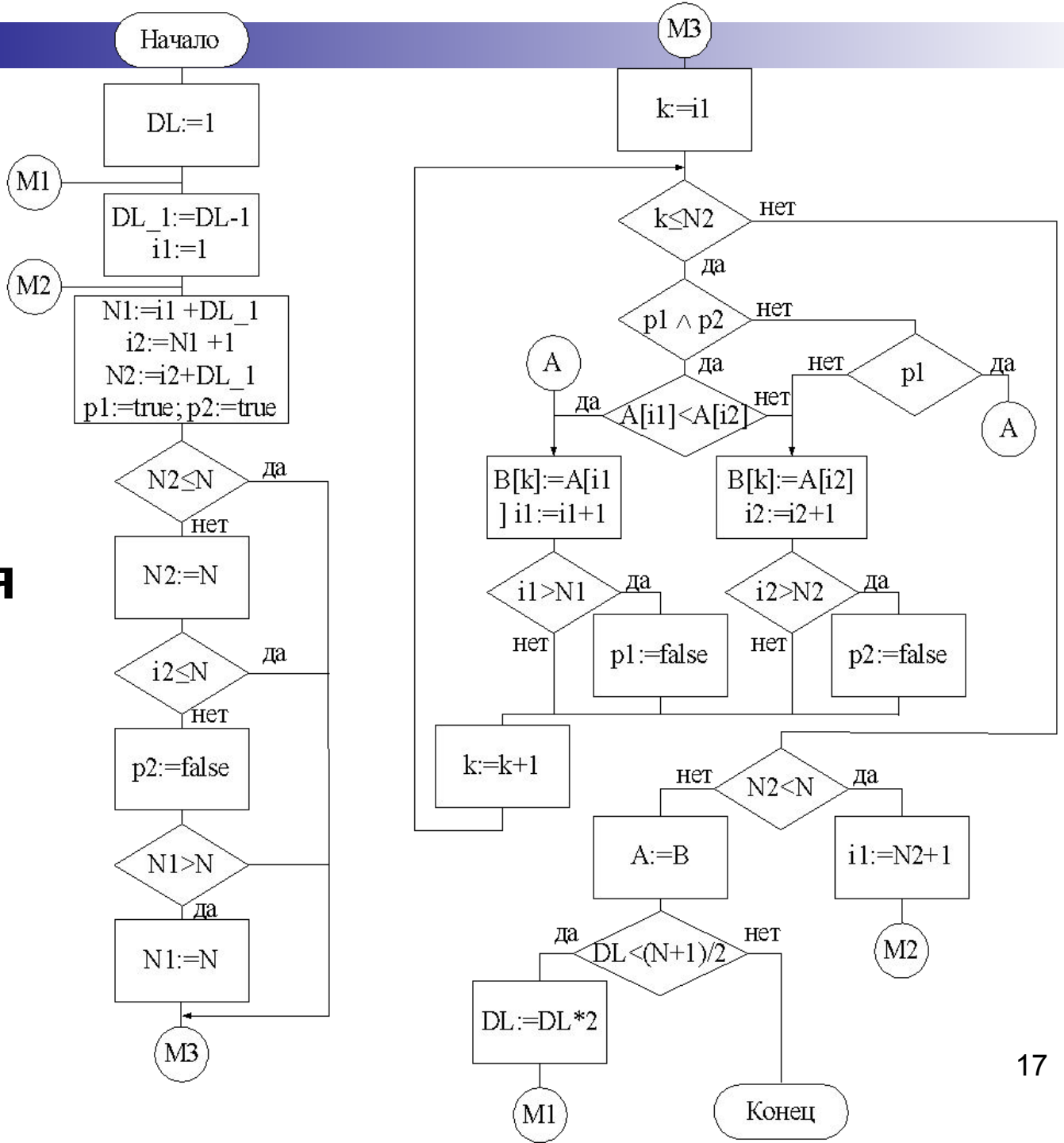
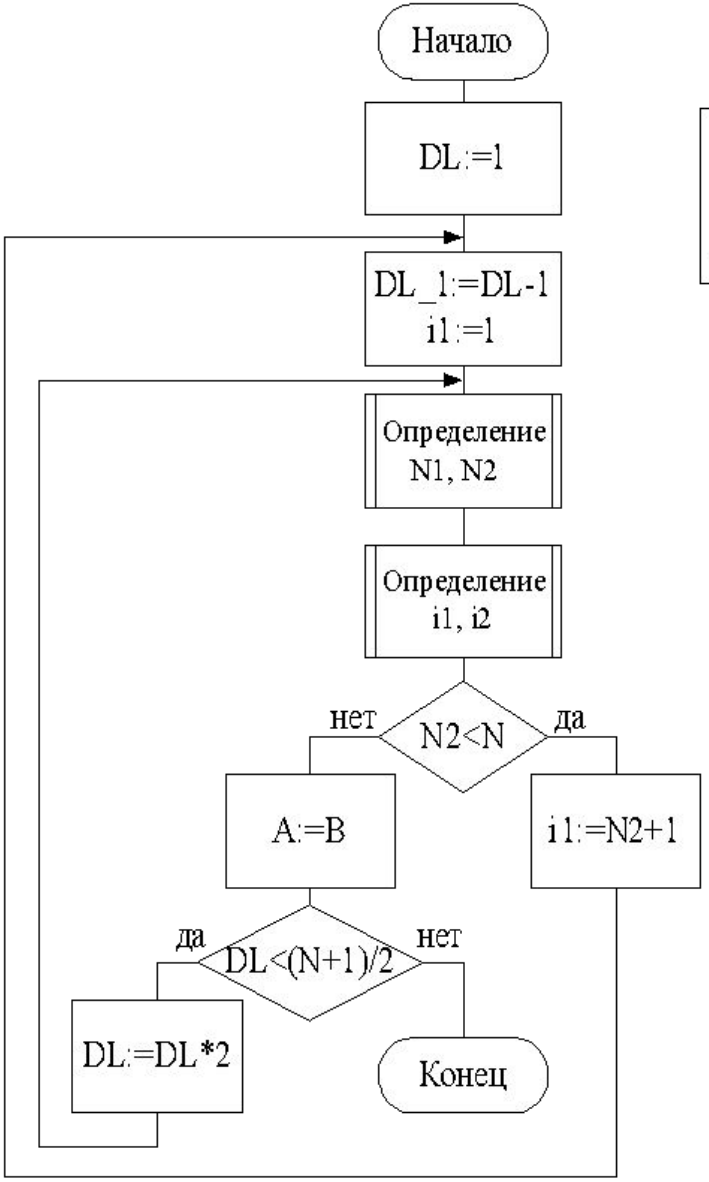


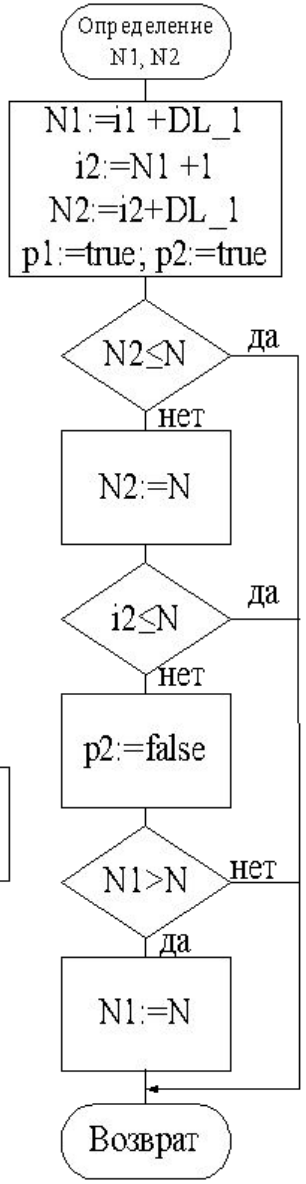

Рис. 14.18
Схема алгоритма сортировки последовательными слияниями

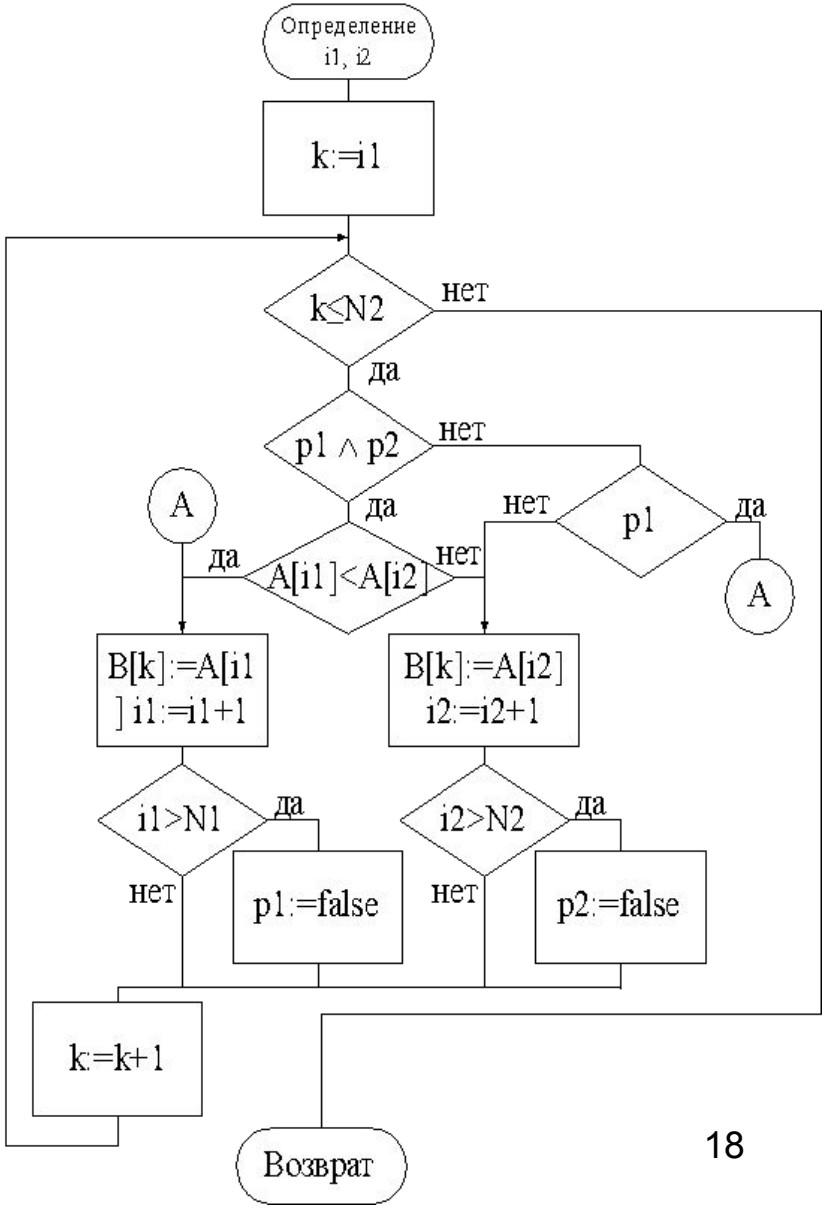
# 1-й вариант более читаемого изображения схемы алгоритма



# 2-й вариант: выделение подпрограмм

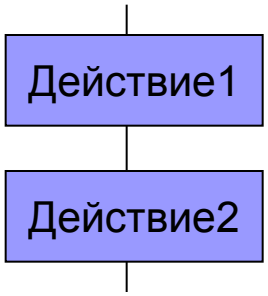


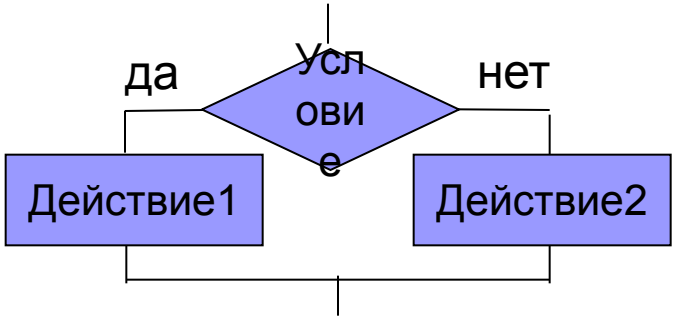



# Основные структурные конструкции алгоритма

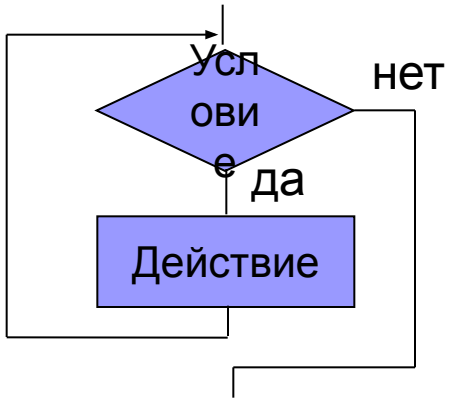
**1. Следование**




**2. Ветвление**




**3. Цикл-пока**



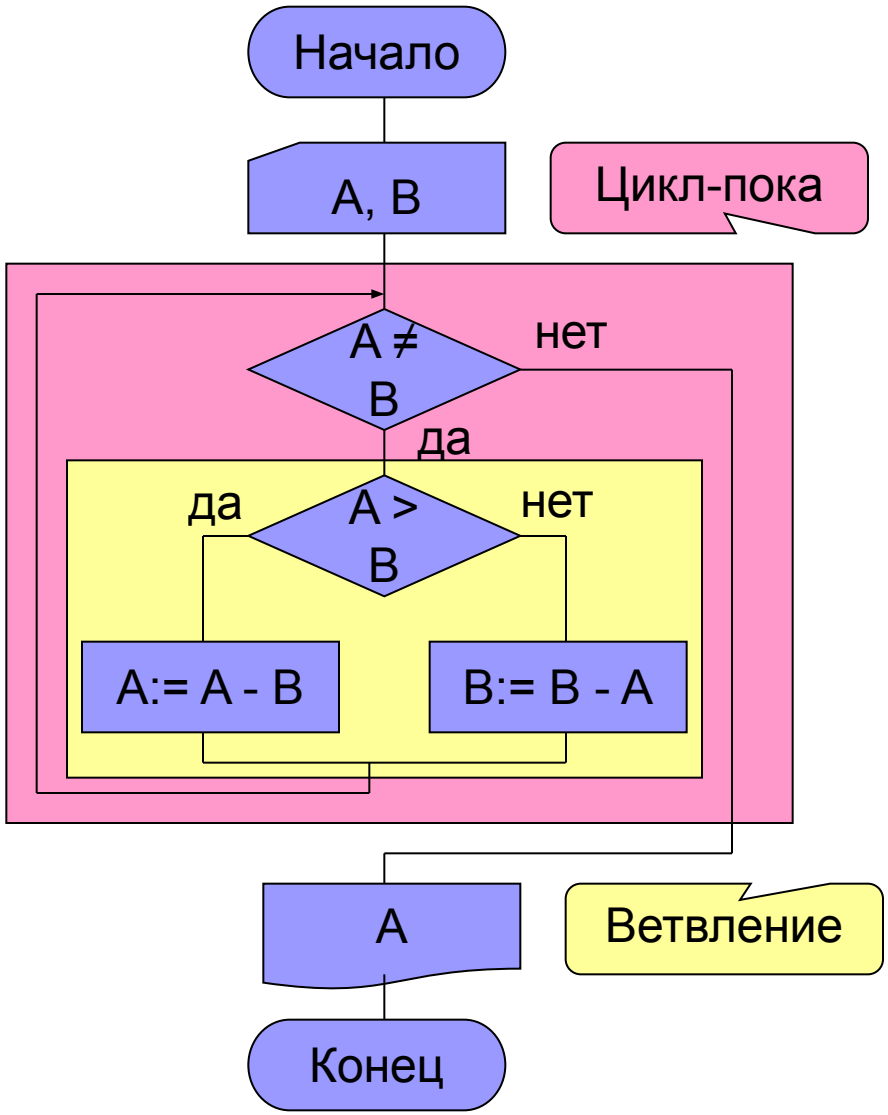

**Псевдокод:**

…
Действие 1
Действие 2
…

…
**Если** Условие
**то** Действие 1
**иначе** Действие 2
**Все-если**
…

…
**Цикл-пока** Условие
Действие
**Все-цикл**
…

# Схема и псевдокод алгоритма программы поиска НОД




**Алгоритм Евклида:**

**Ввести A,B**

**Цикл-пока A ≠ B**

**Если A > B**

**то A := A – B**

**иначе B := B – A**

**Все-если**

**Все-цикл**

**Вывести A**

**Конец**

4. **Реализация программы, ее тестирование и отладка.**

# Структура консольной программы

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

```
Program Ex1_01;//Определение наибольшего общего делителя
{$APPTYPE CONSOLE}
Uses SysUtils;
Var a,b:integer;
begin
    Write('Input two numbers:');
    Readln (a,b);
    while a<>b do
      if a>b then a:=a-b
             else b:=b-a;
    Writeln('Result:', a);
    Readln;
end.
```

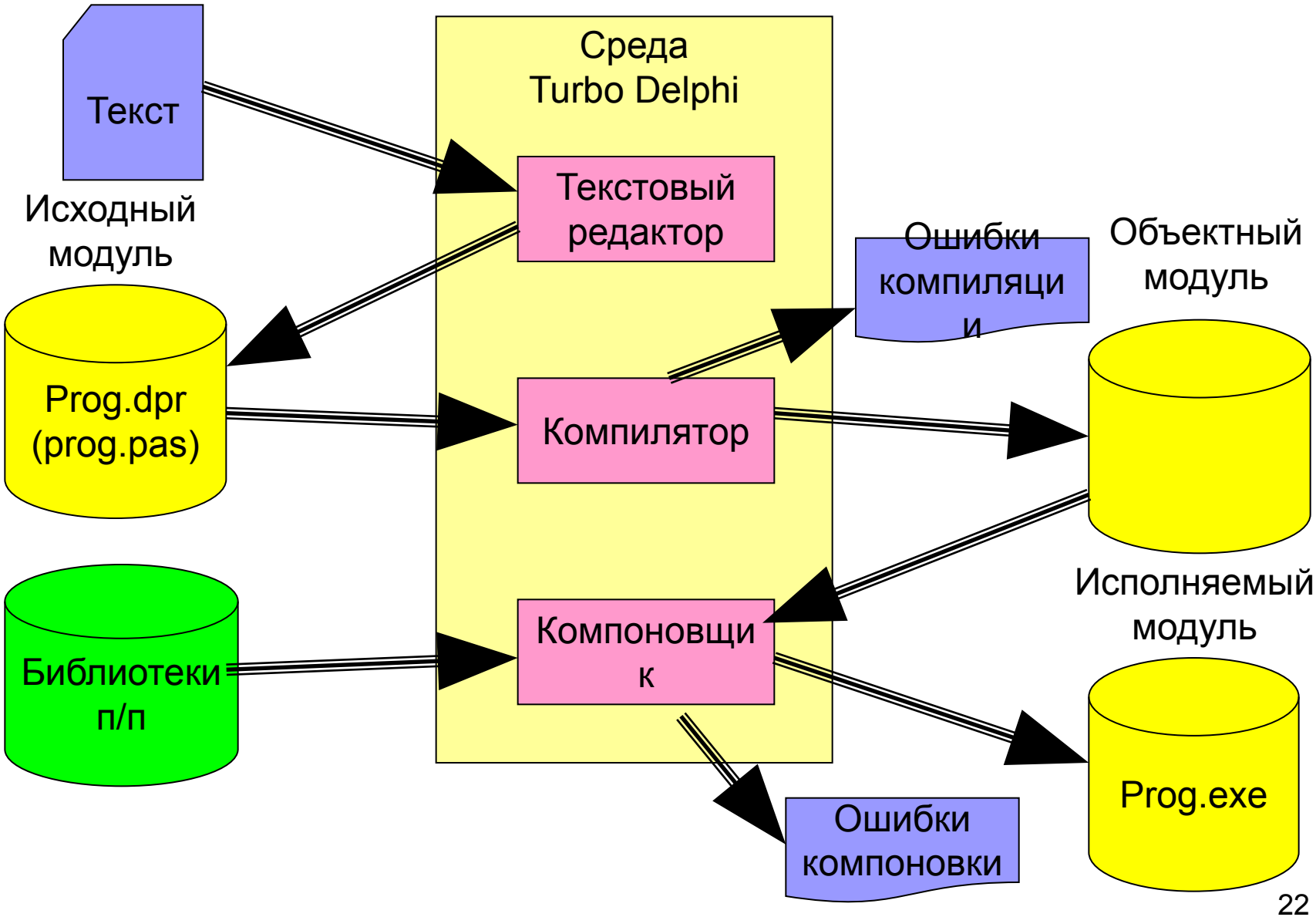
**Заголовок**

**Раздел описаний**

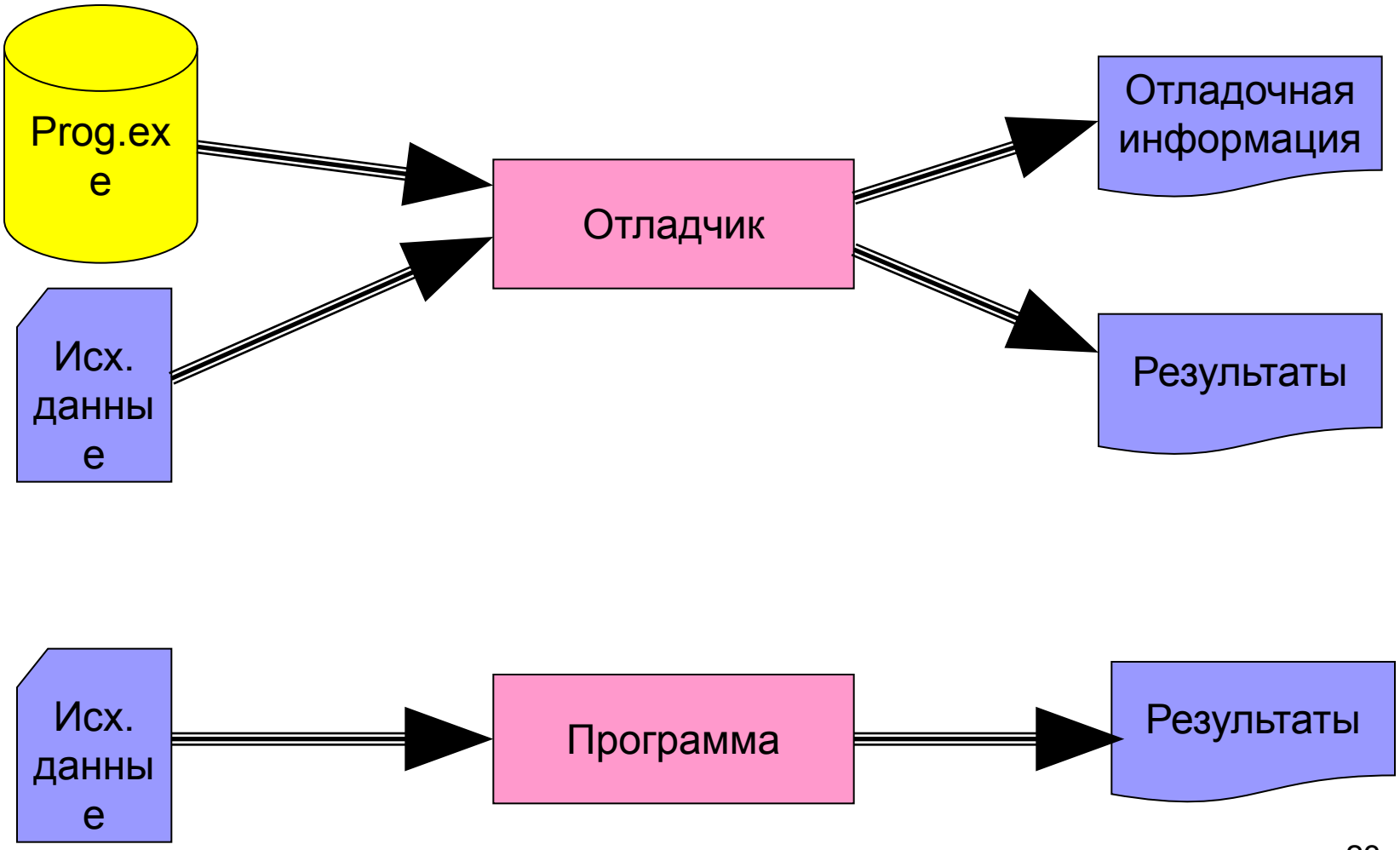
**Раздел операторов**

**Удерживает окно консоли в открытом состоянии, пока пользователь не нажал клавишу Enter**

# Схема процесса подготовки программы



# Схемы процессов отладки и выполнения программы



# Глава 1 Простейшие конструкции языка Delphi Pascal

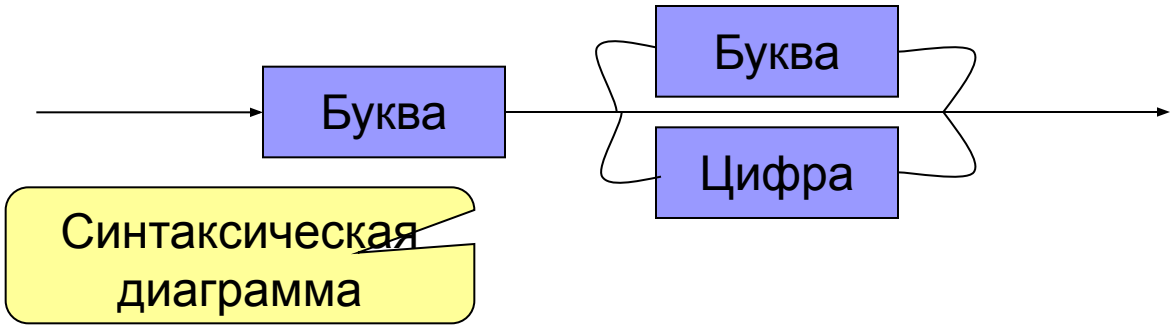
# 1.1 Синтаксис и семантика языка программирования

**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, a..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.

**Пример**: конструкция «Идентификатор»:




Правильные идентификаторы:

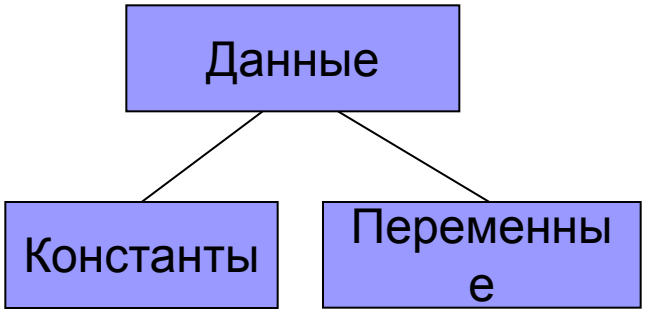
A, a21, n1dw, kkk

Неправильные идентификаторы:

12sdd, ?hjj, s21*5

# 1.2 Константы и переменные. Типы переменных

**Константы** – данные, не изменяемые в процессе выполнения программы.




**Литералы** – константы, указанные непосредственно в тексте программы.

**Примеры литералов**:

а) `-25, 2.5, 0.1e6` $\{= 0{,}1 \cdot 10^6\}$ – числовые литералы;

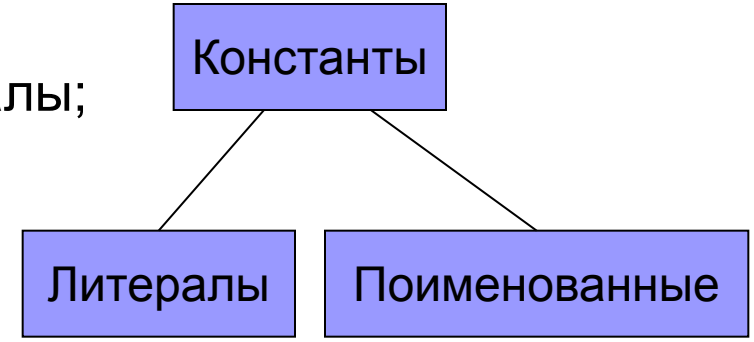
б) `$2a` – шестнадцатеричное число;

в) `true`, `false` – логические константы;

г) `'d'`, `#65 = 'A'` – символьные константы;

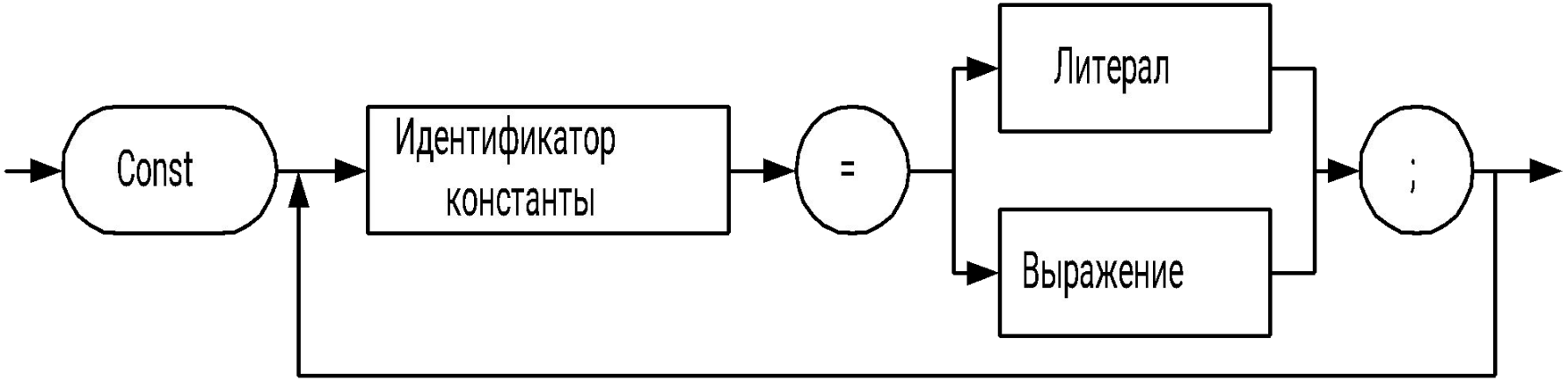
д) `'abcd'` – строковая константа;

е) `nil` – адресная константа.



# Поименованные константы

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

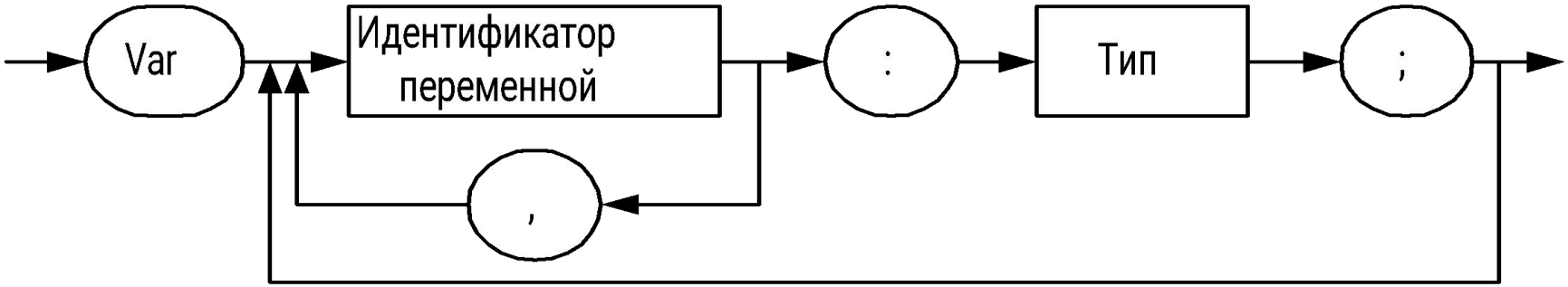



**Пример:**

```
Const min = 0; max = 100;
      center = (max - min) div 2;
```

# Переменные

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




**Пример**:

```
Var a,b:integer;
    c:real;
```

**При установленной опции Extended syntax {$X+}** (расширенный синтаксис) переменным при объявлении можно задавать начальные значения.

**Пример**:

```
Var a:integer=56; b:integer=85;
```

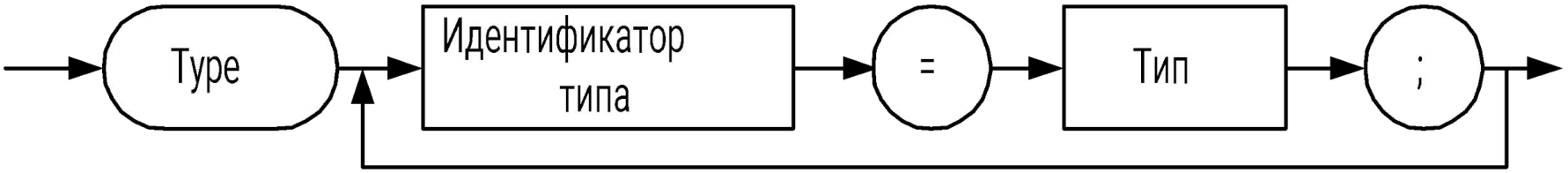
# Типы данных

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

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

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

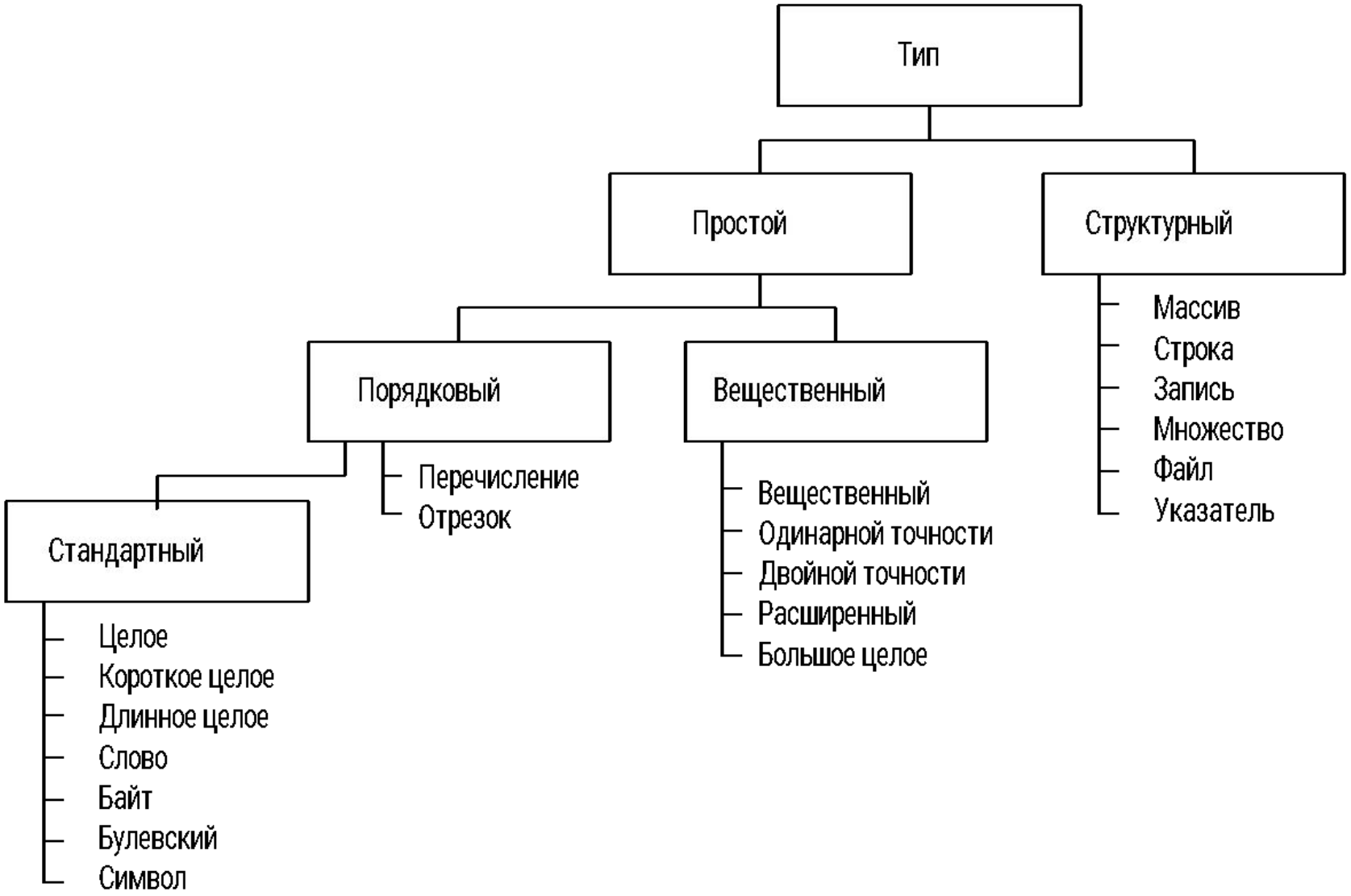
Для объявления новых типов данных используется конструкция:




**Пример**:

```
Type date = 1..31;  // объявление нового типа данных
Var d1:date;        // объявление переменной этого типа
```

# Классификация типов данных языка



# Основные стандартные типы данных Delphi Pascal

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;c: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)

# Порядковые типы

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

**Пример:**

```
Type Day = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
Var D:Day;
```

или

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

**Значения переменных**

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

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

**Пример:**

```
Type Date = 1..31; // значения – числа от 1 до 31
Var DataN: Date;
```

или

```
Var DataN: 1..31;
```

# Функции порядковых типов данных

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 типы

Вещественные числа представляются в компьютере с ограниченной точностью, определяемой разрядной сеткой.

Формат внутреннего представления:

$-0.5 \cdot 10^{23}$

Порядок

Мантисса

Знак мантиссы (1 бит) | Порядок | Мантисса

Стандартные вещественные типы:

| Тип | Значащих цифр | Диапазон 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 | $-2^{63}+1..2^{63}-1$ |

# 1.3 Выражения

1. **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)
```

8
1
2
1.6667

# Выражения (2)

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

< {меньше}, >{больше}, ={равно},

<>{не равно}, <={меньше или равно}, >={больше или равно}

**Пример:**

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

…

```
a > b
```

true

```
a = b
```

false

# Выражения (3)

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

**not {НЕ}**

| | |
|---|---|
| false | true |
| true | false |

**and {И}**

| | false | true |
|---|---|---|
| false | false | false |
| true | false | true |

**or {ИЛИ}**

| | false | true |
|---|---|---|
| false | false | true |
| true | true | true |

**xor {исключающее ИЛИ}**

| | false | true |
|---|---|---|
| false | false | true |
| true | true | false |

**Примеры:**

```
a:=true; b:=false;
a and b {false}
a or b {true}
```

# Выражения (4)

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

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

**Пример:**

```
var a:SmallInt = 5;
...
not a

{поразрядное
    НЕ}
```

00000000 00000101 ⇒ 11111111 11111010

$5_{10}$ $65530_{10}$

**a shl 2**

00000000 00000101 ⇒ 00000000 00010100

{сдвиг влево на два разряда}

$5_{10}$ $20_{10}$

# Математические функции

В выражениях можно использовать следующие математические функции:

```
Pi // число π
abs(<Целое или вещественное выражение>) // абс. значение
sqr(<Целое или вещественное выражение>) // x^2
sqrt(<Вещественное выражение>)          // √x
exp(<Вещественное выражение>)           // e^x
ln(<Вещественное выражение>)            // ln x
sin(<Вещественное выражение>)
cos(<Вещественное выражение>)
arctan(<Вещественное выражение>)        // arctg x
frac(<Вещественное выражение>)  // дробная часть числа
int(<Вещественное выражение>)   // целая часть числа
randomize                       // подготовка датчика случайных чисел
random (<В. выр.>) // генерация вещественного случайного числа
                              0 ≤ x < 1;
random (<Ц. выр. >) // генерация целого случайного числа
                              0 ≤ i < Целое;
```

# Правила вычисления выражений

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

| Операции | Приоритет |
|---|---|
| @, not | 1 |
| *, /, div, mod, and, shr, shl | 2 |
| +, -, or, xor | 3 |
| <, >, <=, >=, =, <> | 4 |

**Пример:**

1) $\frac{x(x+2)}{y(y-1)}$ ⇒ **x*(x+2) / y / (y-1)** или **x*(x+2) / ( y*(y-1))**

2) **(a < b) and (b>=1)**

# Правила вычисления выражений (2)

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.

**Пример:**

```
var a:single; k:integer;
…
a/k // число k преобразуются к типу single
```

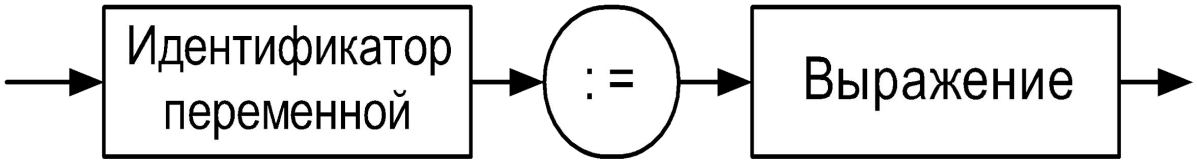
3. При сравнении вещественных чисел из-за их неточного представления проверку равенства и неравенства следует осуществлять ***с явным указанием допуска*****.**

**Пример:**

```
Var x,y:single;
x <> y  ⇒  abs(x-y) > 1e-10
x = y ⇒ abs(x-y) < 1e-10
```

# 1.4 Оператор присваивания

Используется для изменения значений переменных.




**Пример:**

```
Var v:integer; a,b:single;
… a:= v*b / 2.0;
```

single single

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.

# Неявное преобразования типов

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

**Пример:**

```
Var L:LongInt; E,x:extended; I:integer; R:Single;
Begin …
  R:= I * E / (x+L);
```

Single

Extended!

**Преобразование будет выполнено неявно (автоматически)**

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

**Исключение!** Для получения ошибки переполнения при работе с целыми числами необходимо установить опции компилятора **Overflow checking {$Q+}** и **Range checking{$R+}**.

# Явное преобразования типов

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.

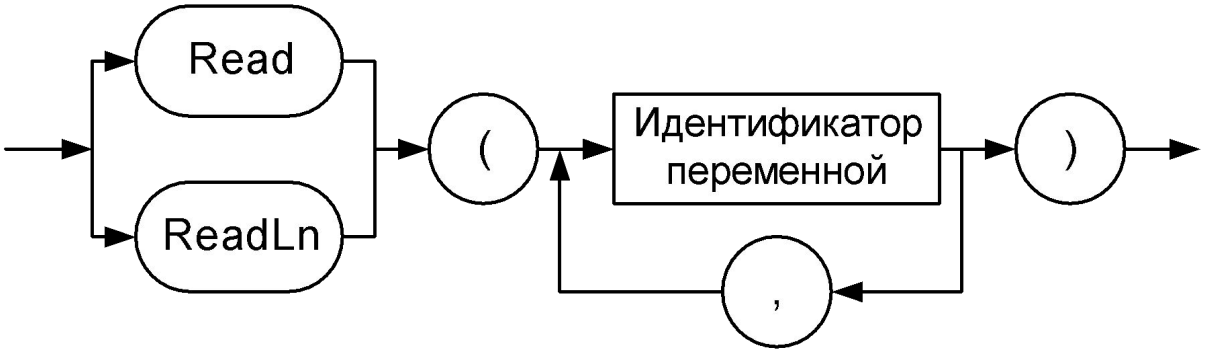
**Пример:** **`ord('A')`** = 65.

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

**Пример:** **`chr(65)`** = `'A'`.

# 1.5 Процедуры ввода-вывода

Ввод – операция по передаче данных от источника в память компьютера.




Вводимые числа разделяют пробелами или записывают на разных строках. По типу они должны соответствовать типам переменных.

**ReadLn** в отличие от **Read** после выполнения операции чтения переводит **курсор ввода** на следующую строку.

```
Read(a,b);
```

а) 30 40

б) 30
40

```
ReadLn(a,b);
```

а) 30 40

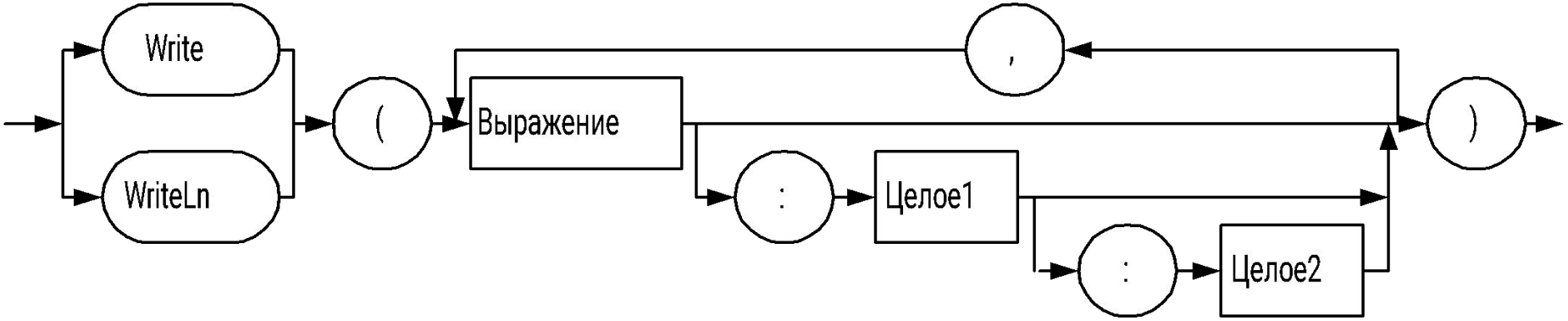
б) 30
40

```
ReadLn(a); ReadLn(b);
```

б) 30 **40**
40

# Процедуры ввода-вывода (2)

Вывод – операция по передаче данных из компьютера на внешнее устройство.




Целое1 – ширина поля вывода (число прижимается к правой границе);

Целое2 – количество выводимых цифр дробной части числа.

WriteLn – после вывода переводит курсор на следующую строку.

**Пример:** `Var a:integer=3; b:real=5.2;…`

```
writeln(a:3,b:6:2);
```

Результат: `_ _ 3 _ 5 . 2 _`

# Программа определения корней кв. уравнения

```
program Ex1_2;
{$APPTYPE CONSOLE}
Uses SysUtils;
Var A,B,C,D,E,X1,X2:Single;
Begin
    WriteLn('Input A,B,C');
    ReadLn (A,B,C);
    WriteLn('A=',A:3:1,' B=',B:3:1,' C=',C:3:1);
    D:=sqrt(sqr(B)-4*A*C);
    E:=2*A;
    X1:=(-B+D)/E;
    X2:=(-B-D)/E;
    WriteLn('X1=',X1:10:6,' X2=',X2:10:6);
    ReadLn;
End.
```