Лабораторная работа №1.

Арифметические операции.

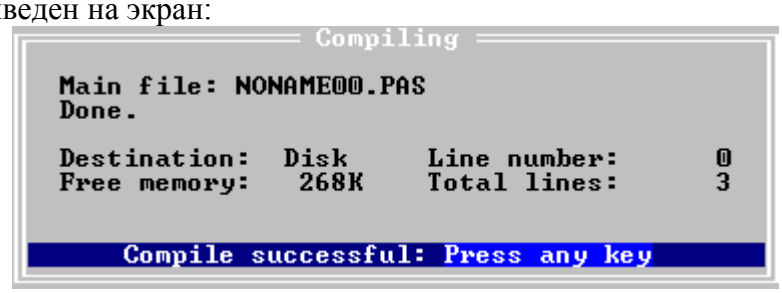
Время: 180 мин.
Что нужно освоить:
1) каким образом создать и запустить простейшую программу, написанную на языке Паскаль;
2) каким образом организовать диалог программы с пользователем;
3) описание переменных и констант, типы данных;
4) особенности работы с арифметическими и логическими операциями
5) стандартные арифметические функции и процедуры;
6) приоритет выполнения операций.

1. ПЕРВАЯ ПРОГРАММА.

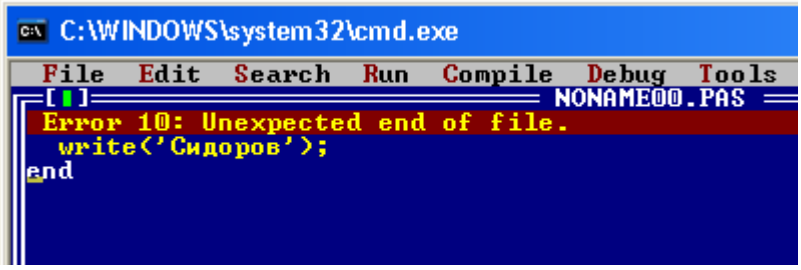
Текст программы на языке Паскаль располагается в операторных скобках: `begin end`. Заключительный символ программы это символ '.' – точка. Давайте напишем первую программу, она будет выводить на экран вашу фамилию. Наберите в редакторе Pascal следующий код:

```
begin
  write('Сидоров');
end.
```

После чего можно проверить корректность набранного кода – для чего нажмите клавишу F9. Программа будет откомпилирована, но не будет запущена на исполнение и результат будет выведен на экран:



Если код содержит ошибки, то будет выведено соответствующее сообщение, например, ошибкой считается отсутствие точки, завершающей программный код:



Чтобы запустить программу на исполнение следует нажать клавишу Ctrl+F9 – программа будет полностью откомпилирована, то есть переведена с языка высокого уровня на машинный язык, и запущена на исполнение. Но результата вы не увидите. Это происходит потому, что после выполнения всех операторов программы управление передается среде Pascal. Просмотреть результат можно нажав сочетание клавиш Alt+F5.

Чтобы обеспечить возможность просмотра результатов программы без дополнительного использования какого-либо сочетания клавиш следует их выводить в файл или как-то остановить на время процесс исполнения программы. Это можно сделать по разному, но для простоты предлагаю добавить к нашей программе одну строчку – readln:

```
begin
```

```
  write('Сидоров');
  readln;
end.
```

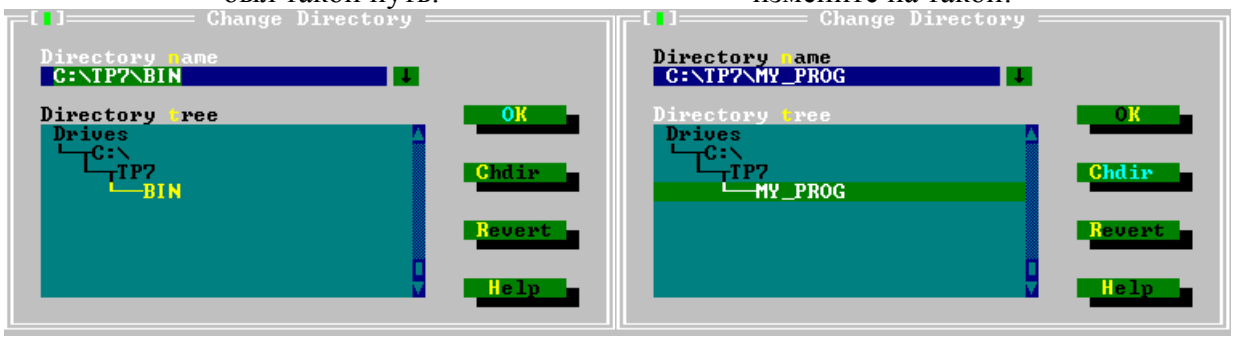
Процедура readln предназначена для считывания строки, введенной пользователем с клавиатуры. Признаком окончания ввода строки является «перевод каретки» – нажатие клавиши ввод. В нашем случае можно ничего не вводить, а просто нажать Enter.

Итак, запустите программу – фамилия будет выведена на экран. После фамилии вы обнаружите мигающий курсор – это и есть действие процедуры readln. Нажмите Enter для возвращения в среду программирования.

Пришло время сохранить программу. Нажмите F2 и вам будет предложено сохранить вашу программу в текущую папку – ту, из которой вы запускали Pascal. Пока не сохраняйте программу и нажмите Esc. Удобнее будет поступить так: создать специальную папку для хранения ваших программ и назначить путь к ней в среде Pascal. Создайте на диске папку (я назову её my_prog) и зайдите в меню Pascal: File / Change dir, там укажите путь к папке и нажмите OK.

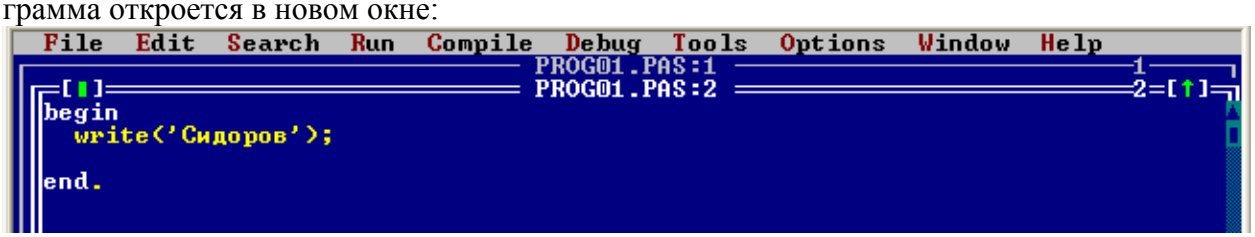
Пример:

был такой путь: измените на такой:



Теперь уже можно нажать F2 и сохранить программу в специально созданную папку. Все сохраненные в ходе выполнения лабораторной работы программы по окончанию заберите на свой носитель, а папку удалите, чтобы не захламлять жесткий диск. Обратите внимание, что в папке кроме файлов с текстами программ (*.pas) сохраняются и готовые к самостоятельному использованию без среды программирования, откомпилированные программы (*.exe).

Все сохраненные ранее программы, конечно, можно открывать и редактировать в среде программирования. Нажмите клавишу F3 и откроется диалог со списком программ в текущей папке. У вас пока только одна программа – её выделите и нажмите OK – программа откроется в новом окне:



Между окнами можно переключаться с использованием следующих сочетаний клавиш:

Alt+0 – вывести список окон;
Alt+цифра (номер окна) – перейти к окну с номером;
F6 – перейти к следующему окну;
Shift+F6 – перейти к предыдущему окну;

Закрыть окно можно сочетанием клавиш Alt+F3 или нажав мышкой на квадратик в левой верхней части окна. Распахнуть окно можно нажав на стрелочку в правой верхней части окна. Окно можно переместить мышкой, удерживая за заголовок окна.

В завершении части 1 рассмотрим еще один момент, связанный с оформлением вывода результатов. Вы заметили, что при выполнении программы кроме фамилии на экран также выводятся служебная информация и результаты предыдущего выполнения про-

граммы. Чтобы избавиться от загромождения экрана излишней информацией достаточно применить одну процедуру – процедуру очистки экрана **clrscr**:

```
begin
  clrscr;
  write('Сидоров');
  readln;
end.
```

Однако, если вы запустите программ у в таком виде, то получите сообщение об ошибке: «неизвестный идентификатор». Чтобы эта процедура стала доступна, следует подключить к нашей программе модуль, в котором она описана – **crt**:

```
uses crt;
begin
  clrscr;
  write('Сидоров');
  readln;
end.
```

Апробируйте работу программы в таком виде и сохраните её (prog01.pas).

Чтобы изменить цвет фона и текста вывода результатов программы можно также воспользоваться возможностями модуля crt. Процедура **textbackground** задает цвет фона, а **textcolor** – цвет шрифта. В качестве аргументов к этим процедурам можно использовать порядковый номер цвета от 0 (чёрный) до 15 (белый).

```
uses crt;
begin
  textbackground(1); {синий цвет фона}
  textcolor(14); {желтый цвет шрифта}
  clrscr;
  write('Сидоров');
  readln;
end.
```

Обратите внимание на форматирование кода программы – под форматированием имеется ввиду определенное оформление текста программы. В частности, в нашей программе используется структурный оператор begin end, являющийся, по сути, операторными скобками. Такой оператор предназначен для объединения нескольких операторов в группу. Группа операторов может являться телом цикла, ветвью в условном операторе. Для того, чтобы текст программы легче читался, имеет смысл организовать структурированную запись кода в виде отступов. Именно поэтому операторы в тексте программы в примере написаны не в столбик ровно один под другим, а с некоторыми отступами.

Следует заметить, что в одной строке можно записать более одного оператора – главное правило, между операторами должен стоять символ «;», по существу и сам являющийся оператором. Его, например, можно использовать для организации преднамеренного останова при отладке программы. А вот перед закрывающей операторной скобкой (перед оператором end) символ «;» можно и не ставить:

```
uses crt;
begin
  clrscr;
  write('Сидоров');
  ;
  readln
end.
```

2. ОРГАНИЗАЦИЯ ПРОСТЕЙШЕГО ДИАЛОГА С ПРОГРАММОЙ.

Итак, первая программа уже написана и работает, но она имеет существенный недостаток – она статична, то есть исполняемые действия неизменны и не могут быть адаптированы под текущие потребности пользователя. Как правило, программы могут динамически менять либо последовательность действий, либо значение переменных.

Закройте все открытые программы, откройте программу prog01.pas и сохраните её как prog02.pas – в ней будем производить модификации и исследования.

Все данные, которые используются в программе, могут быть либо константами, либо переменными. Константы, обычно используются для задания исходных данных или назначения величин, неоднократно используемых в алгоритме и неизменных до окончания всех действий. Константы можно задать в специальном разделе – разделе описания констант. Данный раздел начинается с идентификатора const:

```
program prog02;
uses crt;
const fam='Сидоров';
begin
  clrscr;
  write(fam);
  readln;
end.
```

Раздел названия программы, начинающийся зарезервированным словом program, является необязательным, и его я буду включать в текст только для удобства описания. Далее самостоятельно отслеживайте изменения названия программы, не забывая сохранять предыдущие версии.

Как вы видите, к константе можно обратиться посредством использования идентификатора. Апробируйте программу в таком виде – результат второй программы аналогичен результату первой.

В программе может быть несколько констант:

```
program prog02;
uses crt;
const fam='Сид'; iliya='оров';
begin
  clrscr;
  write(fam); write(iliya);
  readln;
end.
```

Введем в программу раздел описания переменных. Переменные в отличие от констант могут менять свое значение во время выполнения программы. У переменной есть имя (идентификатор) и значение переменной. Имя переменной остается неизменным во время выполнения программы, а значение можно изменять. Обращаться к переменной можно через её имя:

```
program prog03;
uses crt;
const fam='Сид'; iliya='оров';
var familiya: String;
begin
  clrscr;
  familiya:=fam+iliya;
  write(familiya);
  readln;
end.
```

Вы видите некоторое различие в разделах описания констант и переменных. Если тип констант автоматически опознается по форме записи, то тип данных переменной следует явно указывать. Цель назначения типов данных состоит в исключении ситуации

ошибочного выполнения операторов языка программирования. Например, операция сложения двух целых чисел «1» и «1» приведет к результату «2», а двух строк «1» и «1» – к результату «11». В нашем случае используется строковый тип данных, поэтому к переменной familiya можно будет применять только процедуры и функции допустимые для строковых величин. Кроме того следует понимать, что при назначении переменной определенного типа данных под неё в памяти компьютера выделяется определенное количество физической памяти (зависит от типа данных). Не следует злоупотреблять ресурсами компьютера. При наличии возможности использовать тип данных занимающий меньше места следует применить именно его. Оптимальное распределение ресурсов может быть значимой проблемой для написания программ управления базами данных, обработки многомерных массивов. В частности строковый тип данных имеет размер 255 байт, то есть всего четыре переменных, даже если они не используются, а только объявлены, уже занимают 1кб. Строковый тип, при необходимости можно сократить, к примеру, указав в разделе описаний переменных максимальный размер переменной:

```
var familiya: String[25];
```

Задача оптимизации ресурсов не является актуальной для простых программ и не оказывает заметного влияния на их работоспособность или быстродействие.

Программа №3 использует одну переменную, но при этом, все еще, не является адаптируемой под текущую ситуацию, под потребности пользователя. Следует организовать диалог с пользователем. Для этого можно использовать уже известную нам процедуру readln, но ранее мы использовали её без параметров, отчего, введенная пользователем строка, нигде не сохранялась и её нельзя было использовать в дальнейшем. Добавим процедуре чтения строки параметр – переменную ***name***:

```
program prog04;
uses crt;
var name: String[25];
begin
  clrscr;
  write('Введите имя : ');
  readln(name);
  write('Имя ',name,' содержит ',length(name),' символов.');
  readln
end.
```

Как вы уже догадались функция `length(name)` возвращает количество символов в строке. Обратите внимание на особенности использования процедуры write – в скобках через запятую можно указывать разнородную информацию (константы, переменные), которая будет выводиться последовательно в одной строке.

Усложним программу:

```
program prog05;
uses crt;
var f,i,o: String[25];
begin
  clrscr;
  write('Введите Фамилию : '); readln(f);
  write('Введите Имя : '); readln(i);
  write('Введите Отчество : '); readln(o);
  write('Пользователь - ',f,' ',i,' ',o);
  readln
end.
```

Такой линейный алгоритм может служить основой для организации простейшего интерфейса программы.

3. ОСНОВНЫЕ ТИПЫ ДАННЫХ.

Итак, программа оперирует данными посредством обращения к ним через переменные или константы, которые имеют определенный тип. Мы уже апробировали один из основных типов данных, используемых в Pascal – строковый (String). Рассмотрим и другие типы данных. Строка состоит из символов. Символ может храниться в переменной символьного типа – **char**. Пример 6:

```
program prog06;
uses crt;
var c1,c2: char;
    s: string;
begin
  clrscr;
  c1:='д'; c2:='a';
  s:=c1+c2;
  write(s);
  readln;
end.
```

Переменную типа char причисляют к переменным *порядкового типа*, так как каждый символ имеет строго определенный и заранее заданный порядковый номер. Номер этот характеризует место символа в таблице символов ASCII. Диапазон номеров лежит в пределах от 0 до 255, поэтому для хранения переменной типа char достаточно одного байта (1 байт = 8 бит => 2^8=256 – от 0 до 255 включительно). По номеру можно получить символ, а по символу его порядковый номер. Для этого сделаны специальные функции – **chr** (преобразует номер в символ) и **ord** (возвращает положение заданного символа в таблице ASCII).

Пример 7:

```
program prog07;
uses crt;
var c: char;
begin
  clrscr;
  write('Введите один символ: ');
  c:=readkey; writeln(c);
  writeln('Код символа: ',ord(c));
  readln;
end.
```

Стандартная функция **readkey** возвращает введенный с клавиатуры символ. Разница между процедурами write и writeln состоит в том, что write выводит текст без переноса курсора на следующую строку, а writeln – с переносом (ln – от line).

Функция chr преобразует число (порядковый номер) в символ, поэтому для демонстрации её возможностей в программу необходимо ввести переменную целочисленного типа (этот тип данных подробнее рассмотрим позже). Сейчас будем использовать переменную n типа byte:

```
program prog08;
uses crt;
var c: char;
    n: byte;
begin
  clrscr;
  write('Введите число в диапазоне от 32 до 255: ');
  readln(n);
  writeln(n,' - это номер символа: ',chr(n));
  readln;
end.
```

Апробируйте программу. Служебные символы располагаются в диапазоне до 31, 32 – символ «пробел», далее идут знаки препинания, цифры, буквы английского и русского алфавитов.

К переменным *порядкового типа* можно применять функции вызова предыдущего и следующего значений **pred** и **succ**:

```
program prog09;
uses crt;
var c: char;
begin
  clrscr;
  write('Введите один символ: ');
  c:=readkey; writeln(c);
  writeln('Предыдущий - ',pred(c));
  writeln('Следующий - ',succ(c));
  readln;
end.
```

Апробируйте эту программу – на запрос введите символ «1», в ответе получите «0» и «2». Попробуйте самостоятельно узнать их (и остальных цифр) порядковые номера в таблице символов.

Рассмотрим теперь порядок работы с числами и начнем с целых чисел. В Pascal предопределено пять целочисленных типов. Каждый тип обозначает подмножество целых чисел:

Тип	Диапазон	Формат	
Shortint	-128..127	Знаковый	8 бит
Integer	-32768..32767	Знаковый	16 бит
Longint	-2147483648..2147483647	Знаковый	32 бита
Byte	0..255	Беззнаковый	8 бит
Word	0..65535	Беззнаковый	16 бит

Все целочисленные типы являются *порядковыми*.

Назначая переменной определенный тип данных следует помнить, что при выполнении операций над переменной при превышении допустимого диапазона значений переменной произойдет останов программы.

Рассмотрим пример 10:

```
program prog10;
uses crt;
var n: byte;
begin
  clrscr;
  n:=100+200;
  writeln('Ответ: ',n);
  readln;
end.
```

Если вы попытаетесь запустить эту программу, то получите сообщение об ошибке – выход за пределы допустимого диапазона значений. Если вы исправите строку «`n:=100+200;`» на «`n:=100+100;`», то ошибки не будет. Но, если требуется проводить вычисления с числами бо́льшего диапазона, чем допускает byte, тогда следует изменить тип переменной. Например, так: «`var n: word;`». Апробируйте программу и убедитесь, что можно производить вычисления с числами бо́льшими, чем 255.

Однако и тут могут возникнуть ошибки. Изменим немного программу:

```
program prog11;
uses crt;
var n: word;
begin
  clrscr;
  n:=100-200;
  writeln('Ответ: ',n);
  readln;
end.
```

Запустите программу и опять получите сообщение об ошибке – выход за пределы диапазона. Дело в том, что тип word предопределяет только положительные целые числа. Для работы с отрицательными числами следует выбрать тип, задающий знаковый формат, например, «`var n: integer;`». Внесите самостоятельно изменения в программу.

Давайте ещё поработаем с этой программой – используем операцию деления:

```
program prog12;
uses crt;
var n: integer;
begin
  clrscr;
  n:=100/200;
  writeln('Ответ: ',n);
  readln;
end.
```

При запуске такой программы опять получите ошибку, так как результат от деления не является целым числом. В этой ситуации правильно использовать переменную вещественного типа. Вещественный тип имеет набор значений, который является подмножеством вещественных чисел, которые могут быть представлены в виде числа с плавающей точкой и с фиксированным количеством разрядов.

Pascal использует пять предопределенных вещественных типов. Каждый тип имеет свой диапазон и точность:

Тип	Диапазон		Точность	Байт
Real	2.9e-39..	1.7e38	11-12	6
Single	1.5e-45..	3.4e38	7-8	4
Double	5.0e-324..	1.7e308	15-16	8
Extended	3.4e-4932..	1.1e4932	19-20	10
Comp	-9.2e18..	9.2e18	19-20	8

Обратите внимание: тип Comp – 64-разрядное целое число.

Приведем программу в работоспособное состояние:

```
program prog12;
uses crt;
var n: real;
begin
  clrscr;
  n:=100/200;
  writeln('Ответ: ',n);
  readln;
end.
```

Правильный ответ уже получается, но читать его крайне неудобно: 5.00000000000000E-0001, что в переводе на математический язык означает 5×10^{-1} или просто 0,5. Для удобства чтения можно использовать **форматированный вывод**: «`writeln('Ответ: ',n:5:2);`». В данном случае число после первого символа «:» означает сколько всего символов выводить, а число после второго двоеточия – сколько после запятой. Ответ будет читаться намного удобнее: 0.50.

► Самостоятельно разработайте программы:

– labrab01.pas – просит у пользователя ввести два числа, а затем выводит на экран их сумму, разность, произведение и частное от деления;

– labrab02.pas – просит у пользователя ввести три числа, после чего программа рассчитывает их среднее арифметическое и выводит результат на экран. ◄

И ещё один тип данных – *логический*. В разделе описания переменных задается так: `var b: boolean;`

Переменная типа boolean может принимать значения true («истина») или false («ложь») в зависимости от выполнимости или не выполнимости логического выражения. Переменные такого типа удобно использовать как переключатели, в операторах ветвления, при построении логических выводов. Кроме того, к переменным типа Boolean, как к переменным *порядкового типа*, можно применять функции вызова предыдущего и следующего значений pred и succ:

```
program prog13;
uses crt;
var b: boolean;
```

```
begin
  clrscr;
  b:=2+3>4;
  writeln(b);
  writeln(pred(b));
  readln;
end.
```

В дальнейшем подробнее остановимся на использовании переменных логического типа.

4. ОСНОВНЫЕ АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПЕРАЦИИ.

Итак, мы уже приступили к использованию арифметических операций: «+», «–», «/». Кроме этого, естественно, есть и операция умножения: «*». Надо помнить, что результат операции деления всегда число типа real в не зависимости от типов входных операндов. Остальные операции не меняют тип используемых данных в процессе выполнения.

Рассмотрим остальные арифметические операции языка Pascal.

Операции целочисленного деления – **div** и остаток от деления – **mod** рассмотрим на примере 14:

```
program prog14;
uses crt;
var n,d,m: byte;
begin
  clrscr;
  n:=7;       writeln('Исходное число : ',n);
  d:=n div 2; writeln('Целочисленное деление: ',d);
  m:=n mod 2; writeln('Остаток от деления: ',m);
  readln;
end.
```

Как видите, результаты операций div и mod являются целочисленными.

► Разработайте программу labrab03.pas вычисляющую сумму цифр двухзначного числа. Программа должна предлагать пользователю ввести двухзначное число, разбивать его на отдельные цифры с помощью операций div и mod, и на основании этой информации определять сумму цифр. ◄

К логическим операциям можно отнести: **and** («И»), **shl** (побитовый сдвиг влево), **shr** (побитовый сдвиг вправо), **or** («ИЛИ»), **xor** («Исключающее ИЛИ»), **not** (отрицание).

Чтобы подробнее узнать о синтаксисе и порядке использования стандартных функций можно воспользоваться справкой Pascal – для этого установите курсор на интересующую вас функцию (наберите, например, в любом месте редактора кода shr) и нажмите сочетание клавиш Ctrl+F1 – в результате откроется контекстная помощь.

В качестве примера рассмотрим программу, осуществляющую операцию «И» с двумя переменными n и m:

```
program prog15;
uses crt;
var n,m,otv: byte;
begin
  clrscr;
  readln(n,m);
  otv:=n and m;
  writeln(otv);
  readln;
end.
```

Отметим, что *в процедуре readln может быть несколько входных аргументов*, тогда их следует перечислять через запятую: Во время выполнения программы при последова-

тельном вводе значений их можно разделять пробелами или просто после каждого введенного значения нажимать Enter.

Запустите программу и введите значения 6 и 3. Так как логические операции выполняются побитно, то ожидаемый результат 2. Он получается побитным выполнением операции «И» над двоичными числами 110_2 и 011_2. Получающийся результат 010_2 выводится в десятичной системе счисления – 2_{10}.

Апробируйте самостоятельно все оставшиеся логические операции. Разработанные программы сохраните.

5. ОСНОВНЫЕ АРИФМЕТИЧЕСКИЕ ФУНКЦИИ И ПРОЦЕДУРЫ.

К наиболее часто используемым функциям следует отнести: **abs** (абсолютное значение числа), **sqr** (возведение числа в квадрат), **sqrt** (квадратный корень числа), **exp** (экспонента), **ln** (натуральный логарифм), **sin** (синус), **cos** (косинус). Подробное описание функций с примерами смотрите в справке Pascal.

Вызов стандартной функции осуществляется путем указания в нужном месте программы имени функции и её аргумента, заключенного в круглые скобки. Например, в результате такой операции `y:=abs(-5);` в переменную `y` будет записано число `5`.

Есть также и безаргументная функция **`Pi`**, которая возвращает значение числа *Пи*. Пример использования: «`S:=2*Pi*R;`».

Аргумент тригонометрических функций sin и cos задается в радианах. Напомню, что для преобразования значения угла из градусной меры в радианную следует умножить величину угла на число `Pi/180`, а для обратного перевода – на `180/Pi`.

► Разработайте программу labrab04.pas вычисления площади треугольника, если известны величины двух сторон треугольника (a, b) и величина угла (γ) в градусах между ними. Площадь треугольника вычисляется по формуле: S=a*h/2, где высота треугольника h вычисляется по формуле: h=b*sin(γ). Программа должна предлагать пользователю ввести величину переменных a, b и γ и на основании этой информации производить расчет S. ◄

Часто программисты пользуются процедурами изменения значения переменной на фиксированный шаг: **dec** (от decrement – уменьшение) и **inc** (от increment – увеличение):

```
program prog16;
uses crt;
var x: byte;
begin
  clrscr;
  x:=5;
  dec(x,3);
  writeln('x=',x);
  inc(x,4);
  writeln('x=',x);
  readln;
end.
```

Результат:

```
x=2
x=6
```

Второй аргумент процедур dec и inc означает шаг изменения первого аргумента. Второй аргумент является необязательным параметром, при его отсутствии шаг изменения первого аргумента равен 1:

```
program prog17;
uses crt;
```

```
var x: byte;
begin
  clrscr;
  x:=5;
  dec(x);
  writeln('x=',x);
  inc(x);
  writeln('x=',x);
  readln;
end.
```

Результат:

```
x=4
x=5
```

Очень удобными функциями для различного рода вычислений являются функции обработки дробной части числа: **int** (возвращает целую часть числа) и **frac** (возвращает дробную часть числа). Например, write(int(5.9):3:0); дает результат 5. а write(frac(5.9):3:1); дает результат 0.9.

При решении вычислительных задач периодически возникает необходимость преобразования типов данных, в частности, приведения вещественного операнда к целочисленному. В этих целях используют функции **trunc** (от гл. truncate – обрезать) и **round** (от гл. rounded – округлять). Функция trunc возвращает ближайшее целое число меньшее по абсолютному значению аргумента, а функция round возвращает значение аргумента, округленное в математическом смысле до ближайшего целого числа. Сравните:

trunc(6.7) – возвращает 6; trunc(–1.6) – возвращает –1;

round(6.7) – возвращает 7; round(–1.6) – возвращает –2.

6. ПРИОРИТЕТ ОПЕРАЦИЙ.

Последовательность выполнения операций в составе сложного выражения происходит с учетом их приоритета:

1) высший приоритет имеют унарные операции отрицания: not и «унарный минус» (тот, что делает из положительного числа отрицательное) – они в выражении выполняется в первую очередь;

2) затем выполняются операции типа «умножение»: *, /, div, mod, and, shl или shr;

3) далее выполняются операции типа «сложение»: +, –, or, xor;

4) и, наконец, в последнюю очередь операции отношения: =, <>, <, >, <=, >=.

Если в выражении встречаются операции с равным приоритетом, то они выполняются в порядке очереди, то есть слева направо.

Можно **изменить порядок выполнения операций**, для этого необходимую часть выражения следует взять в **скобки**. Сначала вычисляется выражение в скобках как отдельный операнд. Если в выражении есть ряд вложенных скобок, то в первую очередь вычисляется часть выражения находящаяся в самых внутренних скобках. Например, результат вычисления выражения 3+4*5 равен 23, а при наличии скобок (3+4)*5 равен 35.

► Разработайте программу labrab05.pas вычисления сложного выражения: $z=\frac{x^2+y^2}{2-\sqrt{x+y}}$. Программа должна предлагать пользователю ввести величину переменных x и y и на основании этой информации производить расчет z. ◄