ТУБ №25:
Простейшие операторы языка Delphi

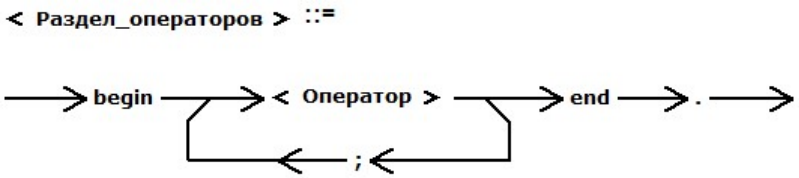
Одним из базовых элементов многих языков программирования, в том числе и Delphi, являются операторы.

> *Оператор* — законченное высказывание на языке программирования, задающее действия, которые должны быть выполнены программой.

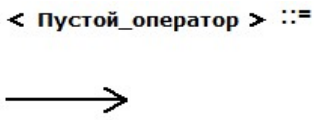
Начнём обсуждение с двух операторов, которые выполняют вспомогательную роль в языках Pascal и Delphi: составного и пустого.

Составной оператор применяется там, где по синтаксису языка допускается применение лишь одного оператора, а по логике работы программы их нужно несколько. Внутри себя составной оператор содержит один или несколько других операторов, разделённых точкой с запятой, сам при этом выступая в качестве одиночного оператора.

Нетрудно заметить сходство в синтаксисе составного оператора и раздела операторов:



Таким образом, можно рассматривать раздел операторов как составной оператор, за которым следует точка и который представляет собой всю программу целиком.



Пустой оператор имеет самый простой синтаксис в языке Delphi — не представлен ни одной лексемой и ни одним символом. Тем не менее, он играет важную роль, встречаясь в текстах программ довольно часто. Рассмотрим такую программу:

```
program CalcSum;

{$APPTYPE CONSOLE}

var
  X1, X2: Integer;
  Sum: Integer;

begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  WriteLn(Sum);
  ReadLn;
end.
```

По приведённой ранее синтаксической диаграмме между любыми двумя операторами внутри составного оператора (или внутри раздела операторов) ставится точка с запятой. Но какие два оператора разделяет последняя точка с запятой?

```
begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  WriteLn(Sum);
  ReadLn;
end.
```

Последний оператор — это и есть пустой оператор. Его можно и убрать:

```
begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  WriteLn(Sum);
  ReadLn
end.
```

но тогда после последнего «видимого» оператора не будет стоять разделителя — точки с запятой. На работу программы такое изменение никак не повлияет, однако кроме чисто эстетических соображений здесь возникают ещё и вопросы последующей работы с текстом программы.

В программировании хоть и необязательно по синтаксису большинства языков, но считается хорошим стилем записывать каждый оператор с отдельной строки. Каждый оператор, как мы помним, представляет собой законченную мысль, указание, что должна делать программа. Иногда возникает необходимость изменить порядок выполнения операторов: сначала предполагалось, что действия должны выполняться в одном порядке, но затем оказалось, что правильнее (лучше, оптимальнее, удобнее) будет другой порядок.

Предположим, нам нужно поменять местами два последних оператора. Это не имеет смысла в данной конкретной программе, но давайте просто смоделируем ситуацию. В первоначальном варианте (с пустым оператором) это делается очень легко. Выделяем одну из строк…

```
begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  WriteLn(Sum);
  ReadLn;
end.
```

… вырезаем её…

```
begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  WriteLn(Sum);
end.
```

… помещаем курсор в строку, перед которой необходимо вставить вырезанный оператор, и выполняем вставку:

```
begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  ReadLn;
  WriteLn(Sum);
end.
```

Готово. Полученная программа осталась синтаксически корректной. Но что будет, если в изначальной программе не было пустого оператора?

```
begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  ReadLn
  WriteLn(Sum);
end.
```

Как видим, для приведения программы в порядок теперь недостаточно просто передвинуть строку: приходится ещё перемещаться по строке и дописывать точку с запятой. А если в изначальной программе Вы не ставили точку с запятой осознанно, то хорошо бы ещё и в той строке, которая теперь стала последней, убрать этот знак:

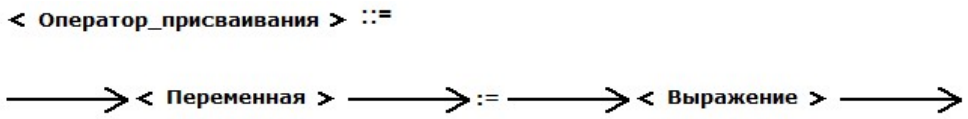
```
begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  ReadLn;
  WriteLn(Sum)
end.
```

Много лишних телодвижений, неправда ли? И всё это только лишь из-за отказа от пустого оператора. Так есть ли смысл?

Правило, по которому точка с запятой выступает в роли *разделителя* операторов, а не признака окончания оператора, обусловлено несколькими причинами. Во-первых, в некоторых случаях так намного проще реализовывать компилятор, а более простой компилятор — более быстрый компилятор. Во-вторых, в 1970-е гг. было очень сильно стремление к тому, чтобы приблизить язык общения с компьютером к естественным языкам, а в естественных языках, как Вы понимаете, точка с запятой ставится именно *между* частями большого высказывания, а не в конце. Наконец, в-третьих, такой подход используется во многих языках программирования. Введение же пустого оператора не стоит ничего, но позволяет программисту выбирать любой из стилей.

Оператор присваивания

Для того, чтобы записать некоторое значение в некоторую переменную, в Delphi предусмотрен оператор присваивания. В действительности мы его уже применяли, просто не уточняли, что эта запись является оператором. Синтаксис оператора присваивания в Delphi в точности соответствует обозначению присваивания в математике:



Слева от знака **:=** записывается нечто, что может рассматриваться в качестве переменной. Чаще всего это просто переменная, но бывают и чуть более сложные случаи. В правой части записывается произвольное выражение, которое должно быть того же или аналогичного (ещё говорят «совместимого по присваиванию») типа, что и переменная в левой части.

При выполнении оператора присваивания вычисляется значение выражение в правой части, а результат записывается в переменную, заданную в левой части.

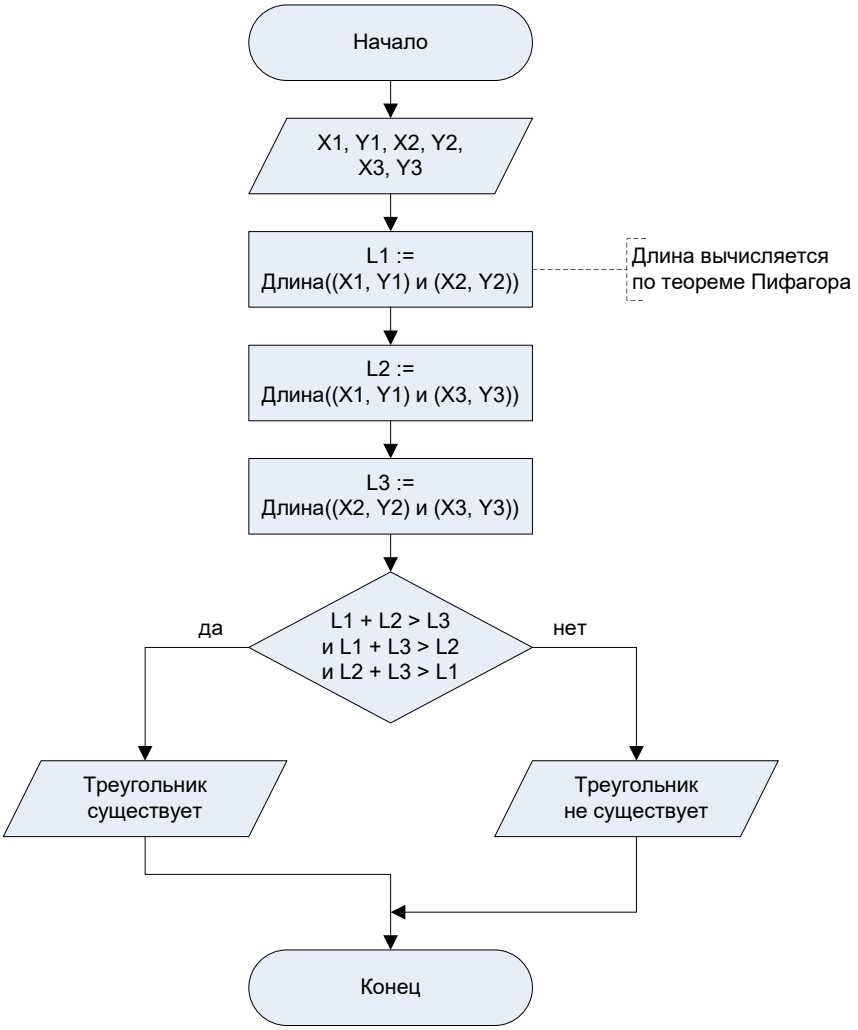
Проверяем условия

Не всё в этом мире линейно. Вот, скажем, пишете Вы программу для расчёта чего-нибудь из архитектуры или геодезии, а там где-то в глубинах алгоритмов требуется такую подзадачу решить:

> Даны координаты трёх точек: (X1; Y1), (X2; Y2), (X3; Y3). Определить, являются ли эти точки вершинами невырожденного треугольника.

Разберёмся, как подобраться к этой задаче. Во-первых, чтобы треугольник был невырожденным, нужно, чтобы это были три разные точки. Во-вторых, по той же причине они не должны лежать на одной прямой. Но если сейчас начать проверять попадание точек на одну прямую, то мы здесь надолго застрянем с угловыми коэффициентами и уравнениями прямых.

Вспомним, что треугольник со сторонами a, b и c существует в том случае, если сумма длин любых двух сторон больше длины третьей стороны. Получается, достаточно вычислить три длины сторон по координатам (в этом поможет теорема Пифагора) и проанализировать их. Попробуем…



Попробуем теперь написать программу, реализующую этот алгоритм.

```
program CheckTriangle;

{$APPTYPE CONSOLE}

var
  X1, Y1, X2, Y2, X3, Y3: Real;
  L1, L2, L3: Real;
  IsValid: Boolean;
```

Исходными данными у нас будут координаты точек и, поскольку точки не обязаны иметь целые координаты, использует вещественный тип данных. Из промежуточных данных у нас длины трёх отрезков — они тоже вещественные. Результат будет иметь логический тип: либо точки образуют треугольник, либо нет.

```
begin
  WriteLn('Program checks if 3 points with given coordinates');
  WriteLn('form a valid triangle.');
  WriteLn;

  Write('Enter X1: ');
  ReadLn(X1);
  Write('Enter Y1: ');
  ReadLn(Y1);

  Write('Enter X2: ');
  ReadLn(X1);
  Write('Enter Y2: ');
  ReadLn(Y1);

  Write('Enter X3: ');
  ReadLn(X1);
  Write('Enter Y3: ');
  ReadLn(Y1);
```

Уф-ф-ф, если бы за каждую строчку давали по пачке мороженого, ангина нам была бы обеспечена. Но ради довольного пользователя можно и потерпеть. Давайте-ка теперь рассчитаем длины трёх отрезков:

```
L1 := Sqrt(Sqr(X1 - X2) + Sqr(Y1 - Y2));
L2 := Sqrt(Sqr(X1 - X3) + Sqr(Y1 - Y3));
L3 := Sqrt(Sqr(X2 - X3) + Sqr(Y2 - Y3));
```

Выходим на финишную прямую. Сейчас нам нужно записать логическое выражение, которое будет давать ответ на вопрос о том, может ли существовать треугольник с такими длинами сторон.

```
IsValid := (L1 + L2 > L3) and (L1 + L3 > L2) and (L2 + L3 > L1);
```

Само по себе это условие должно быть Вам известно из математики, и оно действительно работает. Но у нас изначально задача состояла в проверке существования треугольника не по длинам сторон, а по координатам точек. На всякий случай нужно остановиться и убедиться, что наше выражение даст правильный ответ не только в случае существования треугольника, но и для всех вырожденных случаев.

Что будет, если какие-нибудь две точки имели одинаковые координаты? Тогда одна из переменных `L1`, `L2` или `L3` окажется равной 0, а две другие будут равны некоторому L. При подстановке таких величин в наше логическое выражение одна из скобок даст истину (т.к. сумма двух ненулевых длин очевидно больше 0), а две другие скобки превратятся в подвыражения вида

```
0 + L > L
```

Очевидно, это неравенство не выполняется ни при каких значениях L, а значит, и всё наше логическое выражение окажется ложным (False).

Если совпадут координаты всех трёх точек, мы получим подвыражения вида

```
0 + 0 > 0
```

во всех трёх скобках и ложное значение всего выражения.

Если все три точки различны, то остаётся единственный случай, когда треугольник не существует: если точки расположены на одной прямой. Предположим, дальше всего друг от друга расположены точки (X1; Y1) и (X3; Y3). Тогда расстояние между ними будет в точности равно сумме расстояний от каждой из этих точек до (X2; Y2), а это значит, что подвыражение

```
L1 + L3 > L2
```

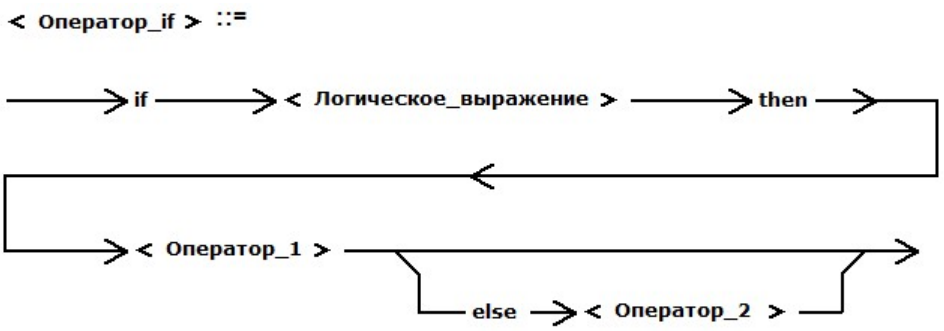
окажется ложным и сделает ложным всё наше выражение.

Дело остаётся за малым: вывести пользователю ответ. Разумеется, нас не устроит такой вариант:

```
WriteLn(IsValid);
```

Нет, этот вариант, конечно, тоже сработает, но пользователь явно будет не в восторге, поэтому придётся всё же как-то выводить два разных фрагмента текста в зависимости от условия. Способов сделать это в Delphi существует немало, но попробуем обойтись минимальным набором возможностей языка.

На помощь придёт *условный оператор* if. Его синтаксическая диаграмма выглядит так:



У оператора if имеется две так называемые ветви: then-ветвь и else-ветвь. Если логическое выражение, записанное после ключевого слова if, истинно, выполнится оператор, записанный в ветви then, если ложно — в ветви else. Ветвь else необязательна: если её нет, а логическое выражение оказалось ложным, просто не будет выполняться ничего и управление перейдёт к оператору, следующему за оператором if.

Применим этот оператор:

```
  if IsValid then
    WriteLn('Points form a valid triangle.')
  else
    WriteLn('Points do NOT form a valid triangle.');

  ReadLn;
end.
```

Программа готова. Можно убедиться, что она работает корректно.

Разумеется, можно было бы не отводить отдельную переменную для результата, а записать вот так:

```
if (L1 + L2 > L3) and (L1 + L3 > L2) and (L2 + L3 > L1) then
  WriteLn('Points form a valid triangle.')
else
  WriteLn('Points do NOT form a valid triangle.');
```

В конце концов разрешено указывать любое выражение логического типа, причём любой сложности. Но в таких вещах нужно знать меру. В нашем случае смысл этого громоздкого выражения достаточно очевиден, да и само значение выражения больше нигде не используется, но в большом проекте вместо вывода осмысленных сообщений будет продолжение вычислений, а условие будет проверяться неоднократно, так что в долгосрочной перспективе вариант с отдельной переменной предпочтительнее.

Более того, Вы же помните, что оператор — это высказывание на языке программирования? Сравните теперь такой код:

```
if IsValid then
  WriteLn('Points form a valid triangle.')
else
  WriteLn('Points do NOT form a valid triangle.');
```

с фразой на английском языке:

> If [triangle with given vertices] is valid then write “Points form a valid triangle.” else write “Points do NOT form a valid triangle”.
>
> *(Если [треугольник с заданными вершинами] невырожденный, то написать «Точки образуют невырожденный треугольник.», иначе написать «Точки не образуют невырожденного треугольника.»)*

При правильном подборе идентификаторов текст программы может читаться почти как книга на английском языке, а значит, смысл такой программы будет проще понять как другим людям, так и самому автору образца «5 лет спустя».

Треугольники наносят ответный удар

Давайте немного дополним условие нашей задачи:

> Даны координаты трёх точек: (X1; Y1), (X2; Y2), (X3; Y3). Определить, являются ли эти точки вершинами невырожденного треугольника. *Если точки образуют невырожденный треугольник, вывести его площадь.*

Может показаться, что вот тут-то мы точно закопаемся в математике: вычислять площадь треугольника по координатам его вершин — то ещё удовольствие. Но подождите! У нас уже вычислены длины его сторон, можно применить формулу Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где a, b и c — длины сторон треугольника, а p — его полупериметр, т.е.

$$p = \frac{a+b+c}{2}.$$

Ну что же, выглядит не так уж плохо, давайте дорабатывать программу. Во-первых, объявим парочку новых переменных:

```
var
  X1, Y1, X2, Y2, X3, Y3: Real;
  L1, L2, L3: Real;
  IsValid: Boolean;

  // Переменные для вычисления площади
  P, S: Real;
```

Теперь вот в этом фрагменте программы

```
if IsValid then
  WriteLn('Points form a valid triangle.')
else
  WriteLn('Points do NOT form a valid triangle.');
```

в ветвь then нужно добавить вычисление площади и вывод полученного значения. Но по синтаксису оператора if там может быть только один оператор. Как быть?

Можно, конечно, выкрутиться вот так:

```
if IsValid then
  WriteLn('Points form a valid triangle. S = ',
    Sqrt((L1 + L2 + L3) / 2
      * ((L1 + L2 + L3) / 2 - L1)
      * ((L1 + L2 + L3) / 2 - L2)
      * ((L1 + L2 + L3) / 2 - L3)):0:4)
else
  WriteLn('Points do NOT form a valid triangle.');
```

Если такой стиль написания программы кажется Вам отличным образцом для подражания, пожалуйста, отхлещите себя по лицу губкой для мытья посуды, а затем примите меры, чтобы оградить от себя человеческую цивилизацию. Программировать для атомных электростанций Вам при таких предпочтениях лучше не доверять.

Посмотрим на проблему ещё раз: в каком-то месте в программе можно записать только один оператор, а нужно несколько. Это как раз задача для составного оператора! Используем его, строго следуя синтаксическим диаграммам:

```
if IsValid then
begin
  WriteLn('Points form a valid triangle.');
  P := (L1 + L2 + L3) / 2;
  S := Sqrt(P * (P - L1) * (P - L2) * (P - L3));
  WriteLn('Its area is: ', S:0:4);
end
else
  WriteLn('Points do NOT form a valid triangle.');
```

Как видим, составной оператор действительно иногда здорово выручает.

Давайте напоследок рассмотрим ещё один, на тот раз абстрактный, пример:

```
if X > 2 then if X < 6 then WriteLn('A') else WriteLn('B');
```

Вопрос: как поведёт себя этот фрагмент программы, если X = –4? А если X = 8?

Ситуация, с которой мы здесь сталкиваемся, называется *проблемой dangling else* (или висящего else). Озвученный чуть ранее вопрос можно переформулировать следующим образом:

> К какому из операторов if относится else-ветвь?

Другими словами, как будет правильнее отформатировать текст программы?

```
if X > 2 then
  if X < 6 then
    WriteLn('A')
  else
    WriteLn('B');
```

```
if X > 2 then
  if X < 6 then
    WriteLn('A')
else
  WriteLn('B');
```

Правило в этом случае звучит следующим образом:

Else-ветвь всегда относится к ближайшему предшествующему оператору if, у которого ещё нет else-ветви.

Другими словами, чтобы понять, к какому оператору if относится else, необходимо начать двигаться от него назад по тексту программы. В приведённом выше примере при X = –4 программа не будет выводить ничего, а при X = 8 выведет букву «B».

Обратите внимание, что приведённые выше два способа записи этого фрагмента программы абсолютно одинаковы с точки зрения компилятора и отличаются только расстановкой пробельных символов. Тем не менее, неправильная расстановка отступов может ввести в заблуждение читающего программу.

На практике это означает, что в подобных ситуациях целесообразно явно указывать свои намерения с помощью составного оператора:

```
if X > 2 then
begin
  if X < 6 then WriteLn('A') else WriteLn('B');
end;
```

Уж в таком виде-то точно нельзя понять программу неправильно.

Дополнительные вопросы

1. Может ли в программе на языке Delphi после ключевого слова then сразу же следовать ключевое слово else? Как в этом случае воспринимается такая программа компилятором? Почему такая запись нежелательна и как обойтись без этого?

2. Напишите программу, которая для заданных пользователем длин сторон двух треугольников определяет, являются ли эти треугольники подобными.

3. Напишите программу, которая для заданного пользователем числа X вычисляет значение $\operatorname{sgn} X$.