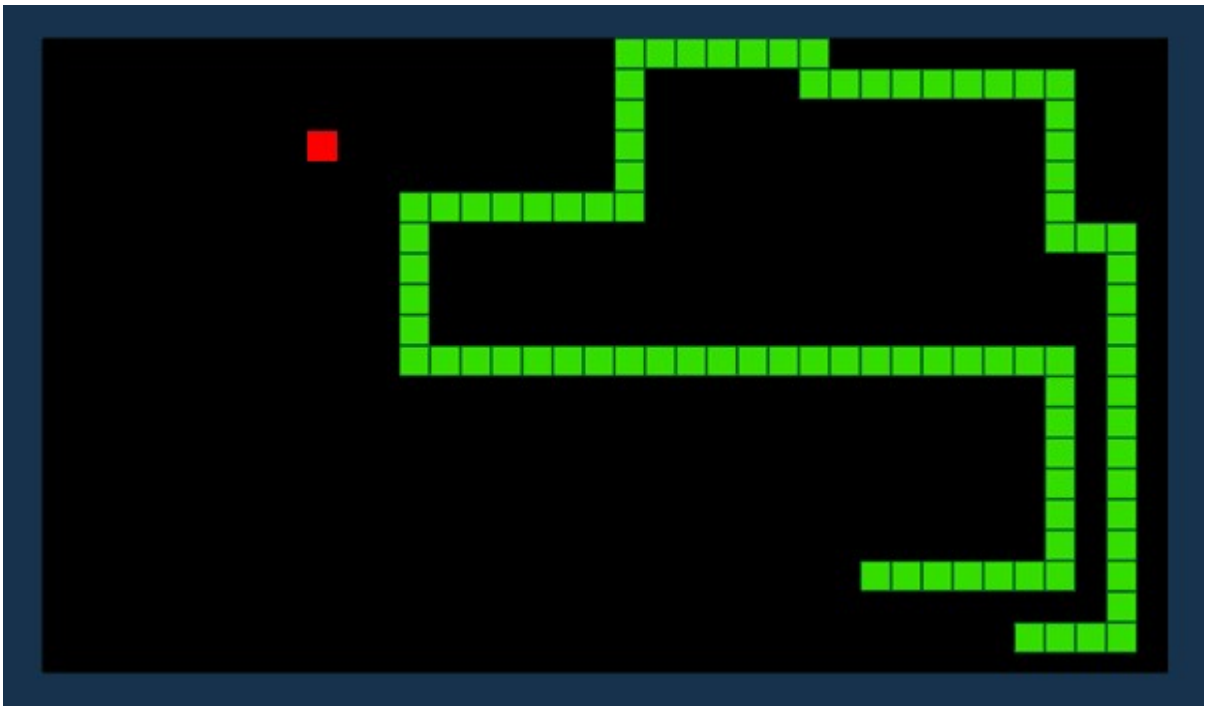
ТУБ №27:
Перечислимый тип в Delphi

Предположим, Вы пишете игру «Змейка». Знаете такую?



Ну, прямо скажем, это игра для уже не самых начинающих программистов, но это ничуть не помешает нам порассуждать о том, как там внутри могут быть реализованы отдельные части её логики.

Так вот, есть вопрос. Когда змейка должна продвинуться вперёд ещё на один шаг, нужно знать направление движения её «головы». Какой тип данных использовать для такой величины?

Что если завести 4 переменных логического типа: `IsUp`, `IsDown`, `IsLeft` и `IsRight`. В каждый момент времени будем присваивать True только одной из них, а в остальные будем записывать False. Должно сработать, правда?

Сработать-то должно, но теперь, когда игрок захочет поменять направление, придётся писать целых 4 присваивания на каждый случай:

```
if <Нажата_стрелка_вверх> then
begin
  IsUp := True;
  IsDown := False;
  IsLeft := False;
  IsRight := False;
end;
if <Нажата_стрелка_вниз> then
begin
  IsUp := False;
  IsDown := True;
  IsLeft := False;
  IsRight := False;
end;
if <Нажата_стрелка_влево> then
begin
  IsUp := False;
  IsDown := False;
  IsLeft := True;
  IsRight := False;
end;
if <Нажата_стрелка_вправо> then
begin
  IsUp := True;
  IsDown := False;
  IsLeft := False;
  IsRight := True;
end;
```

Пожалуй, самое неприятное здесь заключается в том, что повторяющиеся фрагменты кода с небольшими отличиями друг от друга — отличный способ допустить ошибку. Как это и произошло в приведённом примере. Заметили ошибку? А теперь представьте, что в итоговой программе это будет лишь малая её часть.

Попробуем подойти к проблеме с другой стороны. Давайте просто договоримся, что 0 — это вверх, 1 — это вправо, 2 — это вниз, а 3 — это влево, и будем обозначать направление обычным целым числом. Почему именно в таком порядке? На самом деле неважно, в каком, но почему бы и не в таком?

Хорошо, но какие возникнут проблемы? Наверное, самая очевидная заключается в том, что, если у нас есть переменная целочисленного типа, в ней по ошибке может оказаться любое другое число, кроме тех, которые мы договорились использовать. Например, Вы хотели изменить направление на «вправо», случайно зацепили клавишу и не заметили, как присваивание слегка вышло из-под контроля:

```
Direction := 91;
```

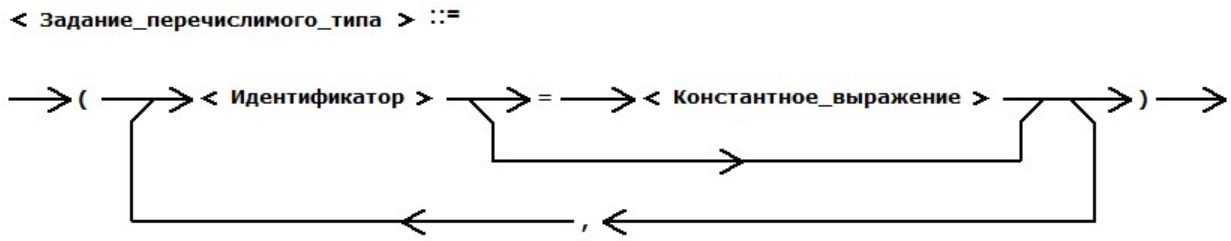
Конечно, можно решить эту проблему с помощью констант:

```
const
  Direction_Up    = 0;
  Direction_Right = 1;
  Direction_Down  = 2;
  Direction_Left  = 3;
```

и договориться присваивать переменной направления только эти константы. В целом это может сработать, но договорённости имеют свойство бывать нарушенными по незнанию, да и со значением такой переменной можно по ошибке начать делать странные вещи, например, складывать, умножать и делить, что, очевидно, смысла иметь не будет.

Для того, чтобы компилятор в подобных ситуациях мог помогать программисту отслеживать правильность программы и не позволять складывать мух с котлетами, придумали *перечислимые типы*.

При задании перечислимого типа используется следующий синтаксис:



Например, для направления движения змейки переменную можно объявить следующим образом:

```
var
  ...
  Direction: (dirLeft, dirRight, dirUp, dirDown);
  ...
```

Или объявить тип отдельно, а затем использовать его при объявлении переменной:

```
type
  TDirection = (dirLeft, dirRight, dirUp, dirDown);

var
  Direction: TDirection;
```

Перечислимый тип задаётся перечислением всех своих значений. Это означает, что в нашем примере переменные типа `TDirection` смогут принимать значения `dirLeft`, `dirRight`, `dirUp` и `dirDown`. Идентификаторы, перечисленные при задании перечислимого типа, становятся литералами этого типа.

В памяти значения перечислимых типов хранятся как целые числа, причём, если ничего не предпринимать, то литералам, использованным при объявлении, эти числа назначаются последовательно, начиная с 0, т.е. в нашем примере `dirLeft` будет представляться в памяти числом 0, `dirRight` — числом 1, `dirUp` — числом 2, а `dirDown` — числом 3.

Количество памяти, выделяемое под переменную перечислимого типа, определяется двумя факторами:

- количеством и значениями этого типа;
- настройками компилятора.

По умолчанию размер перечислимого типа совпадает с размером минимального целочисленного типа, необходимого для хранения чисел, которые поставлены в соответствие значениям этого перечислимого типа. Например, для нашего типа `TDirection` достаточно 1 байта. Тем не менее, имеется директива компилятора {$Z}, которая позволяет задавать минимальный размер для переменных перечислимого типа. {$Z1} устанавливает минимальный размер в 1 байт, {$Z2} — в 2 байта, при {$Z4} любая переменная перечислимого типа будет занимать не менее 4 байт. Для совместимости с ранними версиями Delphi допускаются также записи {$Z+} и {$Z–}, которые эквивалентны {$Z1} и {$Z4} соответственно. Эта более тонкая настройка бывает необходима при взаимодействии с библиотеками, написанными на других языках, при работе с файлами сложных форматов и т.п.

Ещё одна полезная возможность — явное задание значений, соответствующих литералам перечислимого типа. Например:

```
type
  TFieldSize = (
    fsTiny = 5,                  // 5
    fsSmall,                     // 6
    fsMedium = 10,               // 10
    fsLarge,                     // 11
    fsGiant = fsTiny + fsMedium // 15
  );
```

В основном применение этой возможности такое же, как и у директивы {$Z}. Каждое явное задание значения становится новой точкой отсчёта, и каждый последующий литерал, не имеющий явно заданного значения, получает значение на 1 больше. В приведённом примере значения, которыми будут представлены в памяти литералы перечислимого типа, указаны в комментариях.

Идентификаторы, используемые в качестве литералов перечислимых типов, не должны совпадать с идентификаторами переменных, констант, типов и значений других перечислимых типов. Причина достаточно проста. Предположим, мы пишем не обычную «Змейку», а такую, в которой змейке разрешается врезаться в одну из стен, ограничивающих игровое поле, и продолжать игру: например, при этом направление змейки автоматически меняется на противоположное — причём время от времени выбирается другая стена. Тогда мог бы получиться такой код:

```
type
  TDirection = (Left, Right, Up, Down);

var
  // When running into a wall in this direction, snake is left alive
  Left: TDirection;

  // Current snake direction
  Direction: TDirection;

...

Direction := Left;

...
```

У нас была бы переменная `Left`, задающая направление, в котором змейка оставляется живой (is left alive), причём значением переменной может быть любое из 4 направлений, и было бы значение (литерал) перечислимого типа `Left`, соответствующее направлению

влево. Чем являлся бы идентификатор `Left` в операторе присваивания? Чтобы избежать таких неоднозначностей, подобное дублирование идентификаторов запрещено.

Среди Delphi-программистов принято соглашение о том, что литералы перечислимых типов должны начинаться с префикса, указывающего на то, к какому из перечислимых типов этот литерал относится. Например:

```
type
  TWindowState = (wsNormal, wsMinimized, wsMaximized);
  TBorderStyle = (
    bsNone, bsSingle, bsSizeable, bsDialog, bsToolWindow, bsSizeToolWin
  );
  TPosition = (
    poDesigned, poDefault, poDefaultPosOnly, poDefaultSizeOnly,
    poScreenCenter, poDesktopCenter, poMainFormCenter, poOwnerFormCenter
  );
```

В первых двух случаях в качестве префикса использованы первые буквы слов, входящих в имя перечислимого типа, в третьем — первые две буквы единственного слова. В нашем примере с направлением в качестве префикса для тех же целей использовались три буквы «dir».

Над значениями перечислимого типа определены только операции сравнения. Большим считается то значение, которое в памяти представлено большим числом. Кроме того, доступны следующие функции:

Имя	Вид	Описание	Тип результата
Определённые для всех перенумерованных типов			
Ord(x)	Функция	Возвращает порядковый номер значения аргумента во множестве значений типа	Integer
Pred(x)	Функция	Возвращает значение, следующее во множестве значений типа перед значением аргумента	Тип x
Succ(x)	Функция	Возвращает значение, следующее во множестве значений типа после значения аргумента	Тип x
Определённые для всех типов			
SizeOf(x)	Функция	Возвращает размер типа в байтах	Integer

Как нетрудно догадаться по первым трём функциям, значения перечислимых типов обладают свойством перенумерованности. Порядковый номер значения, возвращаемый функцией Ord, совпадает с числовым представлением этого значения в памяти.

Дополнительные вопросы

1. Даны следующие объявления:

```
type
  TMonth = (
    monJan, monFeb, monMar, monApr, monMay, monJun,
    monJul, monAug, monSep, monOct, monNov, monDec
  );

var
  Month: TMonth;
```

В переменную `Month` записан один из месяцев. Как проверить, является ли этот месяц летним? Зимним?

2. Как скорректировать объявление типа `TMonth`, чтобы проверка на попадание в любое из времён года выполнялась одинаково просто?