ТУБ №30:
Типизированные константы-массивы в Delphi

Задумывались ли Вы когда-нибудь, сколько секунд прошло с момента Вашего рождения? Миллион? Десять миллионов? Сто? Миллиард? Несколько простых действий на калькуляторе подсказывают, что миллиард секунд человеческие часики успевают натикать после исполнения 31.5 года. Но что, если попытаться подсчитать точнее и написать для этого небольшую программу?

Критически настроенный читатель скажет, что писать программу ради задачи, которая решается на калькуляторе, — занятие сомнительное. Но в действительности перевод дат в секунды и обратно происходит в современных компьютерах довольно часто. Например, один из самых популярных способов хранения даты и времени — так называемое UNIX-время [1] — представляет собой не что иное, как количество секунд, прошедших с полуночи 1 января 1970 г., а значит, когда пользователь вводит дату, выбирая по отдельности день, месяц и год, для её хранения и последующей обработки как раз и приходится считать секунды. Правда, уже немного с другого момента в прошлом.

Если пренебречь странностями в учёте времени, такими, например, как високосные секунды (leap seconds) [2] или переходы между зимним и летним временем, то алгоритм перевода дат в секунды становится достаточно очевидным. Но в какой-то момент мы столкнёмся с проблемой: в разных месяцах разное количество дней.

Предположим, у нас есть такие объявления:

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

var
  Month: TMonth;
  Year: Integer;
  DaysInMonth: Integer;
```

Как преобразовать `Month` в `DaysInMonth`?

Есть, конечно, очевидный способ:

```
if Month = monJan then
  DaysInMonth := 31
else
  if Month = monFeb then
  begin
    if (Year mod 400 = 0) then
      DaysInMonth := 29
    else
    begin
      if (Year mod 4 = 0) and (Year mod 100 <> 0) then
        DaysInMonth := 29
      else
        DaysInMonth := 28;
    end
  end
  else
    if Month = monMar then
      DaysInMonth := 31
    else
      if Month = monApr then
        ...
```

но этот фрагмент кода заставит рыдать даже самых стойких. Кто-то предложит записать его с чуть более хитрыми отступами:

```
if Month = monJan then
  DaysInMonth := 31
else if Month = monFeb then
begin
  if (Year mod 400 = 0) or ((Year mod 4 = 0) and (Year mod 100 <> 0)) then
    DaysInMonth := 29
  else
    DaysInMonth := 28;
end
else if Month = monMar then
  DaysInMonth := 31
else if Month = monApr then
  DaysInMonth := 30
else if Month = monMay then
  ...
```

Тем не менее, и такой способ записи помогает не больше, чем прикладывание к монитору листа подорожника. Главный признак того, что эти варианты кода что-то делают не так, — большое количество однотипных конструкций. Как быть? Решение, как ни странно, заключается в использовании массивов.

Типизированные константы-массивы

Давайте предположим, что у нас есть массив

```
var
  NumDays: array [TMonth] of Integer;
```

в котором индексы имеют тип `TMonth`, т.е. задают месяцы, а значения будут равны количеству дней в соответствующих месяцах. Тогда преобразования величины типа TMonth в соответствующее количество дней можно было бы записать очень кратко:

```
DaysInMonth := NumDays[Month];
```

Можно было бы записать даже что-то наподобие:

```
SummerDays := NumDays[monJun] + NumDays[monJul] + NumDays[monAug];
```

Представьте себе, как аналогичные вычисления выглядели бы, если бы пришлось обходиться двенадцатью с лишним `if`'ами! Мало того, что такой код был бы просто ужасно громоздким, каждый оператор `if` — это дополнительная проверка, которая тоже занимает какое-то время. Для получения количества дней в январе достаточно было бы одно такой проверки, а вот для декабря их потребовалось бы уже примерно 11 или 12, смотря как реализовать.

Использование массива для хранения ответов на интересующий нас вопрос позволяет решить обе проблемы: и код стал компактнее и нагляднее, и количество действий стало одинаковым и небольшим для всех месяцев — всего-то одно обращение к элементу массива во всех случаях.

Но как заполнить такой массив значениями? Очевидным вариантом выглядит такой:

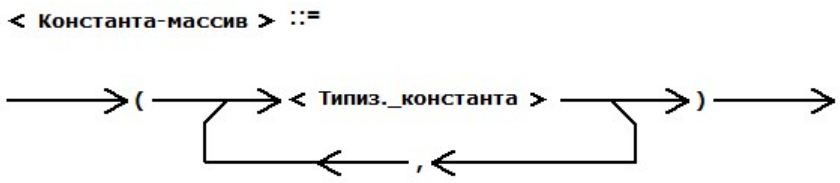
```
NumDays[monJan] := 31;
NumDays[monFeb] := 28;
NumDays[monMar] := 31;
...
NumDays[monDec] := 31;
```

но что-то подсказывает, что это не самая удачная идея. В самом деле, хорошо, что месяцев всего лишь 12, а не 365! А если когда-нибудь придётся заполнять, например, такой массив?

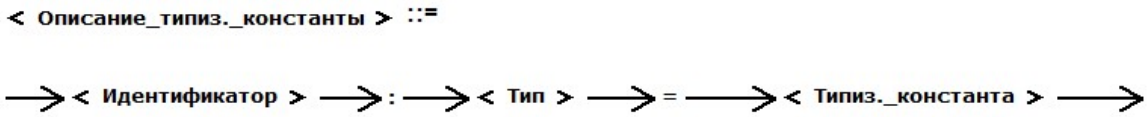
```
var
  // Элемент равен True, если день праздничный
  IsHoliday: array [1..366] of Boolean;
```

Простыне, которая бы получилась, позавидовал бы сам дедушка Лёва Толстой! А значит, надо искать другие способы.

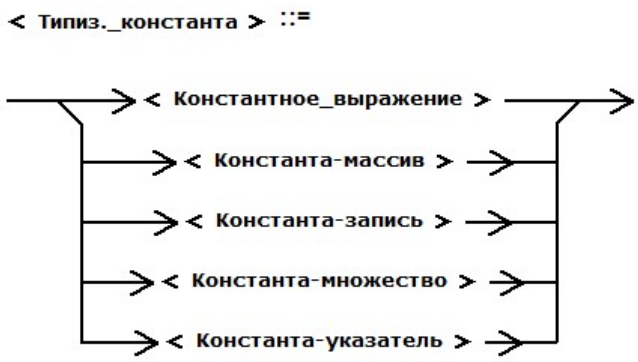
Обратите внимание, что данные, которые мы пытаемся прописать, — это в каком-то смысле константы. Если бы был способ задать массив как константу, да ещё и записывалось это компактнее, это бы решило нашу проблему. Такой способ есть, и он называется типизированная константа-массив.



Вспомним, что описание типизированной константы имеет следующий синтаксис:



а сама типизированная константа записывается так:



Собрав эту информацию воедино, решим поставленную нами задачу:

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

const
  NumDays: array [TMonth] of Integer =
    (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);

var
  Month: TMonth;
  Year: Integer;
  DaysInMonth: Integer;

...

DaysInMonth := NumDays[Month];
```

Куда компактнее и даже читается намного проще, чем все наши предыдущие попытки, не правда ли?

Хорошо, но у нас осталась ещё одна проблема — февраль. В примере для него записаны фиксированные 28 дней, но есть ощущение, что это не совсем так. Как быть?

Есть несколько способов и каждый из них со своими преимуществами и недостатками. Например, можно просто добавить поправку прямо в код:

```
DaysInMonth := NumDays[Month];
IsLeapYear := (Year mod 400 = 0) or
              ((Year mod 4 = 0) and (Year mod 100 <> 0));

if (Month = monFeb) and IsLeapYear then
  Inc(DaysInMonth);
```

На практике определение високосности года уже может быть выполнено в ходе предшествующих вычислений, поэтому код прирастёт только на один оператор `if`. Для проверки на високосность тоже есть способы сделать запись более краткой и понятной, но это отдельная тема.

Если Вы живёте в эпоху грандиозных перемен или же просто опасаетесь, что однажды логика станет сложнее и високосный год будет иметь совершенно другое разбиение на месяцы, чем обычный, можно использовать многомерную константу-массив:

```
const
  NumDays: array [Boolean, TMonth] of Integer = (
    (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
    (31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
  );
```

Как нетрудно догадаться, первый индекс определяет, интересует нас високосный год (True) или невисокосный (False). Для нынешнего положения дел это может показаться избыточным (ну уж прибавили бы единичку для февраля, корона бы с головы не упала!), но как знать, а вдруг однажды мы перейдём на альфацентаврианский календарь и придётся сделать так:

```
const
  NumDays: array [Boolean, TMonth] of Integer = (
    (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
    (31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30)
  );
```

или так:

```
const
  NumDays: array [Boolean, TMonth] of Integer = (
    ( 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
    (355,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1)
  );
```

Такой подход очевидно гибче, но кому-то подобные перестраховки могут показаться излишними, и в этом будет доля истины. С другой стороны, все ведь знают, что…

> Программист ставит себе на тумбочку перед сном два стакана. Один с водой — на случай, если захочет ночью пить. А второй пустой — на случай, если не захочет.

так что имеем право.

Следует иметь в виду, что во многих реализациях языка Pascal и в ранних версиях Delphi типизированная константа-массив в действительности ведёт себя как переменная, т.е. её значение можно произвольно изменять. Единственное отличие заключается в том, что такой «переменной» задано начальное значение. Это было сделано для удобства, поскольку технически такой запрет всё равно можно обойти.

Впоследствии это поведение было сделано настраиваемым: директива компилятора {$J+} или эквивалентная ей {$WRITEABLECONST ON} включает возможность изменения типизированных констант, {$J–} или {$WRITEABLECONST OFF} — отключает, делая такие константы самыми настоящими неизменяемыми константами. Изменяемые константы — возможность, необходимая для совместимости со старым кодом, в новых проектах рекомендуется отключать эту настройку, т.к. это позволяет компилятору генерировать более эффективный машинный код для некоторых платформ.

В Delphi 10.3 добавили синтаксис, позволяющий аналогичным образом инициализировать переменные, однако следует иметь в виду, что эта возможность позаимствована из C-подобных языков и при необдуманном использовании может значительно замедлять работу программы.

[1] https://ru.wikipedia.org/wiki/Unix-время
[2] https://ru.wikipedia.org/wiki/Дополнительная_секунда

Дополнительные вопросы

1. Предложите типы данных и способ записи расстановки шашек в игре «Нарды». Запишите объявления типов и типизированную константу-массив для начальной расстановки шашек.

2. Предложите типы данных и способ записи расстановки фигур в игре «Шахматы». Запишите объявления типов и типизированную константу-массив для начальной расстановки фигур.