Слинкин Д.А.

От Turbo Pascal к Free Pascal.
Часть 2.

Переменные и константы, простые типы данных.

Во второй части книги раскрываются изменения и нововведения Free Pascal в определение переменных и констант, а также модификации и новшества в порядковых и вещественных типах, с множеством поясняющих примеров.

Материал предназначен для учителей информатики, может быть полезен для преподавателей программирования, а также школьников и студентов младших курсов, обучающихся на IT-специальностях.

Оглавление

Введение....3
Переменные и константы....3
Инициализация переменных....3
Восьмеричные и двоичные целочисленные константы, символьные константы....4
Ресурсные строковые константы....5
Информационные директивы....7
Модификаторы переменных....8
Потоковые переменные....10
Свойства-переменные....12
Порядковые типы данных....15
Целочисленные и неотрицательные типы данных....15
Логические типы данных....20
Перечислимые типы данных....22
Тип поддиапазона....24
Символьные типы....24
Вещественные типы данных....24
Заключение....26

Введение

Перед Вами - вторая часть учебно-методического пособия "От Turbo Pascal к Free Pascal".

В отличие от первой части, раскрывавшей технологические особенности работы с Free Pascal и средами программирования для него, вторая часть полностью посвящена нововведениям в языке программирования. Я не буду останавливаться на языковых концепциях, унаследованных из Turbo Pascal, они достаточно хорошо описаны в соответствующей литературе (например, в авторском пособии "Основы программирования на Турбо-паскале", полнотекстовая версия которого свободно доступна по адресу http://shgpi.edu.ru/biblioteka/katfree/2.pdf). В то-же время список новшеств в языке Free Pascal достаточно велик, некоторые из них не имеют даже примерных аналогов в Turbo Pascal, поэтому каждая описываемая языковая возможность будет сопровождаться подробными пояснениями. Более подробно о документации, печатных и электронных изданиях, посвященных Free Pascal, было рассказано в первой части пособия, в параграфе "Информационная поддержка".

Максимальное количество нововведений Free Pascal обеспечивает в расширенном режиме работы **objfpc**, который и будет применяться по умолчанию для всех примеров программ. Версия используемого компилятора Free Pascal - 2.6.2. В качестве базовой операционной системы используется Linux, дополнительной - Windows (в ситуациях, когда программа функционирует по разному в различных ОС). Для визуализации текста программы применяется Geany. Результат запуска программы чаще всего демонстрируется в терминале, генерируемом Geany, иногда - в отдельных терминалах Windows и Linux.

Переменные и константы

Раздел объявления переменных и констант Free Pascal претерпел значительные изменения, сохранив при этом практически полную обратную совместимость с Turbo Pascal. В данном параграфе я постараюсь описать наиболее важные языковые отличия Free Pascal в области объявления и манипуляций с переменными и константами. Каждое отличие сопровождается авторскими примерами с комментариями, демонстрирующими возможности его использования.

Инициализация переменных

Free Pascal позволяет проводить инициализацию переменных первоначальными значениями аналогично типизированным константам в Turbo Pascal.

Пример:

Программа **constvar.pp**, определяет и использует одну типизированную константу и одну инициализированную переменную.

На скриншоте справа изображена программа, приветствующая пользователя на двух языках. Типизированная константа **helloe** (строка 3) содержит английский вариант приветствия, инициализированная переменная **hellor** (строка 4) - русский эквивалент. Получить исполняемый файл для такой программы невозможно, т.к. используемый режим совместимости с Turbo Pascal (строка 1) заставляет компилятор генерировать следующую ошибку:

constvar.pp

```
1  {$mode tp}
2  { $mode objfpc}
3   const helloe:string='Hello!';
4   var hellor:string='Привет!';
5  begin
6   write('По английски: ',helloe);
7   writeln(', а по русски: ',hellor);
8  end.
```

```
Компиляция constvar.pp
constvar.pp(4,19) Фатально: Синтаксическая ошибка, ожидается ";", но обнаружено "="
Фатально: Компиляция прервана
```

Однако при выборе режима **objfpc** инверсией комментария в строках 1-2 (см. скриншот справа), компиляция программы завершится успешно и при ее запуске на экране появится ожидаемое приветствие:

```
По английски: Hello!, а по русски: Привет!
```

constvar.pp

```
1  { $mode tp}
2  {$mode objfpc}
3   const helloe:string='Hello!';
4   var hellor:string='Привет!';
5  begin
6   write('По английски: ',helloe);
7   writeln(', а по русски: ',hellor);
8  end.
```

Восьмеричные и двоичные целочисленные константы, символьные константы

В дополнение к десятичным и шестнадцатиричным целочисленным константам, унаследованным от Turbo Pascal, во Free Pascal были введены восьмеричные и двоичные целочисленные константы. Восьмеричная константа — это набор восьмеричных цифр, предваряемый знаком &. Двоичная константа — это набор двоичных цифр, предваряемый знаком %.

Примеры:

&1027 - восьмеричный эквивалент десятичного числа 535
%1001 - двоичный эквивалент десятичного числа 9
-&33 - восьмеричный эквивалент десятичного числа -27
&981 - неверно, цифры 8 и 9 не являются восьмеричными цифрами.
%1231 - неверно, цифры 2 и 3 не являются двоичными цифрами.
&10.6 - неверно, вещественные восьмеричные константы не поддерживаются.

В примере справа демонстрируются арифметические операции над десятичными, восьмеричными и двоичными константами. Результат запуска программы:

18
10
12

octalbin.pp

```
1  Begin
2    writeln(&10+10);
3    writeln(&10+%10);
4    writeln(%10+10);
5  end.
```

Для символьных констант, предваряемых знаком #, разрешено использовать значения, превышающие 255. Такие символы имеют тип WideChar,

обеспечивают поддержку Unicode и хранятся в двух байтах. Более подробно типе WideChar будет рассказано в следующих параграфах.

Примеры:

#$03B1 - малая греческая буква "альфа": α

#$20B9 - знак индийской рупии: ₹

Ресурсные строковые константы

Во Free Pascal введено понятие ресурсной строковой константы (ресурсной константы или ресурсной строки). Это нововведение впервые появилось в Delphi для обработки ресурсов Windows (данных, встроенных в исполняемые файлы и доступных для модификаций через API-функции Windows) и было унаследовано во Free Pascal, однако внутренняя структура, способы хранения и обработки ресурсных констант во Free Pascal коренным образом отличаются от аналогичных в Delphi. Дело в том, что Free Pascal, в отличие от Delphi, является кроссплатформенной системой и должен обеспечивать единообразие даже для тех платформ, которые не поддерживают понятие "ресурс".

Ресурсная строка определяется как обычная строковая константа, но вместо ключевого слова const используется ключевое слово **resourcestring**. Ресурсные константы обладают всеми признаками строковых констант, им нельзя присвоить значение, однако особый способ хранения в оперативной памяти позволяет во время исполнения программы выполнить их массовую модификацию. При компиляции модуля, содержащего ресурсные константы, их имена, значения и хэш-суммы имен автоматически сохраняются в файле с именем модуля и расширением .rst. Данный файл может в дальнейшем использоваться как основа для генерации трансляций ресурсных строк на различные языки. Такой подход используется, например, в широко распространенной библиотеке интернационализации gettext (http://www.gnu.org/software/gettext/) и Free Pascal обеспечивает ее поддержку.

Рассмотрим пример массовой замены содержимого ресурсных констант с использованием модифицированного .rst-файла:

Модуль rsunit.pp, содержащий две ресурсные строки	Файл rsunit.rst, созданный при компиляции rsunit.pp.

rsunit.pp | rsunit.rst

```
1  {$mode objfpc}
2  {$H+}
3  unit rsunit;
4  interface
5  resourcestring
6    rsError1 = 'First error';
7    rsError2 = 'Second error';
8  implementation
9  end.
```

rsunit.pp | rsunit.rst

```
1
2  # hash value = 102863554
3  rsunit.rserror1='First error'
4
5
6  # hash value = 66535202
7  rsunit.rserror2='Second error'
8
9
```

Воспользуемся автоматически созданным файлом rsunit.rst, выполнив перевод находящихся в нем ресурсных строк и сохранив результат в rsunit.rst.ru:

rsunit.pp | rsunit.rst | rsunit.rst.ru

```
1
2   # hash value = 102863554
3   rsunit.rserror1='Первая ошибка'
4
5
6   # hash value = 66535202
7   rsunit.rserror2='Вторая ошибка'
8
9
```

Разработаем программу демонстрации загрузки ресурсных строк из файла и применения их в программе.

rsunit.pp | rsunit.rst | rsunit.rst.ru | rstranslate.pp

```
1   {$mode objfpc}
2   uses rsunit,classes;
3
4   function changeRS(Name,Value: AnsiString; Hash: Longint; arg:pointer): AnsiString;
5   begin
6     with TStringList(arg) do result:=valueFromIndex[indexOfName(name)];
7     result:=copy(result,2,length(result)-2);
8   end;
9
10  var S:TStringList;
11
12  begin
13    S:=TStringList.Create();
14    S.loadFromFile('rsunit.rst.ru');
15
16    writeln('Ресурсная строка rsError1 : ', rsError1);
17    writeln('Ресурсная строка rsError2 : ', rsError2);
18    writeln();
19
20    SetUnitResourceStrings('rsunit',@changeRS,S);
21
22    writeln('Ресурсная строка rsError1, перевод на русский язык : ', rsError1);
23    writeln('Ресурсная строка rsError1, перевод на русский язык : ', rsError2);
24    writeln();
25
26    ResetResourceTables;
27
28    writeln('Ресурсная строка rsError1, restored : ', rsError1);
29    writeln('Ресурсная строка rsError2, restored : ', rsError2);
30    writeln();
31
32    S.Free();
33  end.
```

Для загрузки и обработки файла ресурсных строк в программе используется экземпляр класса TStringList (класс для манипуляции списком строк) из модуля Classes, который объявляется в 10-й строке, создается в 13-й, заполняется переведенным на русский язык ресурсным файлом в 14-й, применяется для доступа к содержимому ресурсной константы по ее имени в 6-й и уничтожается по окончании работы программы в 32-й.

Первоначальные значения ресурсных констант выводятся в 16-17-й строках программы. В 20-й строке вызывается процедура SetUnitResourceStrings для итерации по ресурсным константам модуля rsunit. В качестве второго параметра в данную процедуру передается адрес функции changeRS, которая будет вызвана для каждой ресурсной константы модуля rsunit и заменит ее содержимое переведенным значением. Третьим параметром передается ранее подготовленный список строк ресурсного файла. Функция changeRS возвращает новое значение ресурсной константы, удаляя из нее лидирующий и завершающий апострофы (строка 7).

Строки 22-23 демонстрируют изменения, произошедшие с ресурсными константами, строка 26 восстанавливает исходное содержимое ресурсных констант с помощью процедуры ResetResourceTables, что можно наблюдать по результатам выполнения строк 28-29.

Выполнение программы приводит к следующему выводу на экран:

```
Ресурсная строка rsError1 : First error
Ресурсная строка rsError2 : Second error

Ресурсная строка rsError1, перевод на русский язык : Первая ошибка
Ресурсная строка rsError1, перевод на русский язык : Вторая ошибка

Ресурсная строка rsError1, restored : First error
Ресурсная строка rsError2, restored : Second error
```

Все средства для обработки ресурсных строк сосредоточены в системном модуле objpas, который автоматически подключается к программе в режимах совместимости delphi и objpas. Более подробно о ресурсных строках можно узнать по адресу http://freepascal.org/docs-html/ref/refse11.html.

Информационные директивы

Во Free Pascal введены информационные директивы **deprecated**, **experimental**, **platform** и **unplemented**, которые могут завершать объявление любой переменной, процедуры или функции. Каждая директива побуждает компилятор выдавать предупреждение при попытке использовать помеченный идентификатор в программе. Предупреждения не считаются ошибками и не препятствуют компиляции, они должны информировать программиста о возможных проблемах при использовании соответствующих идентификаторов.

`deprecated [строковая_константа]`

Идентификатор, для которого определена данная директива, должен считаться устаревшим. За директивой может находиться строковое значение, которое обычно описывает причины устаревания идентификатора и рекомендации по его замене.

`experimental`

Директива помечает переменные, процедуры и функции как экспериментальные и требует осторожного обращения с ними, так как результат их использования может быть непредсказуем. Предполагается, что в последующих версиях они могут быть как исключены из программы, так и оставлены экспериментальными или стабилизированы.

`platform`

Такой директивой помечается платформо-зависимый идентификатор. Предполагается, что данный идентификатор может отсутствовать на некоторых платформах

`unimplemented`

Используется для функций и процедур, которые в данный момент не реализованы и фактически являются «заглушками».

Пример:

hintd.pp

```
1  {$mode objfpc}
2  var data_length:qword=0;
3      datalen:integer absolute data_length
4          deprecated 'устарело, используйте data_length';
5
6  var hashCodeSelf:string experimental;
7
8  var androidVersion:string='4.3.1' platform;
9
10 function makeHashCodeSelf():string; unimplemented;
11  begin result:='' end;
12
13 begin
14   writeln(datalen);
15   writeln(androidVersion);
16   hashCodeSelf:=makeHashCodeSelf();
17 end.
```

При компиляции примера получим несколько предупреждений, информирующих о попытках использовать идентификаторы, помеченные информационными директивами:

```
Компиляция hintd.pp
hintd.pp(14,17) Внимание: Symbol "datalen" is deprecated: "устарело, используйте data_length"
hintd.pp(15,24) Внимание: Символ "androidVersion" не портабелен
hintd.pp(16,14) Внимание: Символ "hashCodeSelf" является экспериментальным
hintd.pp(16,16) Внимание: Символ "makeHashCodeSelf" не реализован
Компоновка hintd
Сборка прошла успешно.
17 строк скомпилировано, 0.0 сек.
4 предупреждений
```

Получить более подробную информацию по данному вопросу можно по адресу http://freepascal.org/docs-html/ref/refse5.html

Модификаторы переменных

При объявлении переменных во Free Pascal можно использовать модификаторы **cvar**, **external**, **export** и **public**.

cvar

Многие языки программирования манипулируют регистрозависимыми идентификаторами, в то время как Free Pascal — регистронезависимый язык. Это часто препятствует подключению к программе на Free Pascal модулей на других языках и наоборот. Решить проблему помогает модификатор cvar. По умолчанию при компиляции все используемые в программе идентификаторы переводятся в верхний регистр и в таком виде хранятся в объектных файлах, используемых затем при компоновке программы, чем и достигается независимость идентификаторов от регистра символов. Модификатор cvar отменяет такое поведение компилятора, помеченные cvar переменные хранятся в объектных файлах в том виде, в котором были объявлены.

```
external [имя_библиотеки] [name имя_переменной]
```

Модификатор позволяет указать, что переменная хранится не в текущей программе (модуле), а во внешнем объектном файле (модуле) или библиотеке. Объектный файл должен быть подключен к программе с помощью директивы компилятора **{$L имя_объектного_файла}**. Если используется внешняя библиотека (статическая или динамическая), ее имя следует указать после модификатора, в результате чего компоновщик попытается собрать исполняемый файл совместно с указанной библиотекой. Разрешено использовать псевдонимы переменной, для этого реальное имя внешней переменной указывается после ключевого слова **name**. Если объектный файл или библиотека созданы на регистрозависимом языке, **external** может комбинироваться с **cvar**.

Рассмотрим пример совместного использования модификаторов **cvar** и **external**. Для этого создадим и скомпилируем в объектный модуль программу на язык C, определяющую единственную переменную. Затем создадим программу на Free Pascal, которая будет подключать указанный модуль и получать доступ к объявленной в нем переменной как напрямую, так и с помощью псевдонима.

var.c (программа на C)	var.c `1 long int currentID = 100;`
Компиляция программы в объектный модуль **var.o**	`cc -c var.c -o var.o`

getID.pp
В программе подключается объектный модуль **var.o**, полученный в результате компиляции **var.c** (строка 1). Затем определяются переменные **currentID** и **id** (строки 2 и 3), фактически ссылающиеся на одну и ту-же переменную, находящуюся в подключенном объектном модуле. В теле программы демонстрируется значение исходной переменной, производится ее модификация и вывод.
	getID.pp
	`1 {$L var.o}`
	`2 var currentID:integer; cvar; external;`
	`3 id:integer; external name 'currentID';`
	`4 begin`
	`5 writeln(id);`
	`6 id:=10;`
	`7 writeln(currentID);`
	`8 end.`

Результат исполнения программы **getID**	100 10

```
export [name имя_переменной]
public [name имя_переменной]
```

Модификаторы **export** и **public** являются синонимами и объявляют переменную экспортируемой, с возможностью указать внешнее имя переменной после ключевого слова **name**. Это означает, что к переменной можно обращаться из программ на других языках программирования. Рекомендуется комбинировать модификаторы **export** и **public** с модификатором

cvar.

В следующем примере определяем экпортируемую переменную в varpas.pp (модуль Free Pascal). После компиляции модуля получаем два файла: varpas.ppu и varpas.o. Первый представляет собой скомпилированное представление модуля и предназначен для подключения только к программам на Free Pascal, второй — является универсальным объектным модулем, предназначенным для подключения к программам на любых языках программирования. Затем создадим usevarpas.c (программу на C), подключающую модуль varpas.o и выводящую значение экпортированной переменной на экран.

<table>
<tr><td>Модуль varpas.pp на языке программирования Free Pascal. Модуль следует скомпилировать, например так:
fpc varpas.pp</td><td>varpas.pp
<pre>1 unit varpas;
2 interface
3 var currentID:integer=200; cvar; public;
4 implementation
5 end.</pre></td></tr>
<tr><td>Программа usevarpas.c на языке программирования Си</td><td>usevarpas.c
<pre>1 #include <stdio.h>
2 void main()
3 {
4 extern long int currentID;
5 printf("%ld",currentID);
6 }</pre></td></tr>
<tr><td colspan="2">Программу usevarpas.c следует скомпилировать с подключением объектного модуля varpas.o, например так:
cc usevarpas.c varpas.o -o usevarpas</td></tr>
<tr><td colspan="2">Результат запуска программы usevarpas</td></tr>
<tr><td colspan="2">200</td></tr>
</table>

Дополнительное описание модификаторов переменных можно получить по адресам http://freepascal.org/docs-html/ref/refse20.html и http://freepascal.org/docs-html/prog/progsu146.html

Потоковые переменные

Потоковая переменная — это глобальная переменная, которая существует отдельно для каждого программного потока исполнения. Потоковые переменные имеет смысл применять только в многозадачных программах. Для объявления потоковой переменной следует вместо **var** использовать ключевое слово **threadvar**. Когда стартует очередной поток управления (в терминологии - **thread** или **нить**), для него создаются копии всех потоковых переменных с нулевыми значениями. По окончании работы вторичного потока все ассоциированные с ним потоковые переменные исчезают.

В примере ниже создаем и запускаем 5 параллельных нитей, каждая из которых итерирует 3 раза по глобальной переменной **tv**, выводит на каждой итерации ее значение и увеличивает на единицу. Основная программа устанавливает первоначальное значение переменной **tv** в единицу, после чего запускает нити, дожидается их завершения и выводит окончательное значение **tv**. В начале каждой итерации нити блокируются на некоторое количество

микросекунд (от 1 до 10). Это позволяет увидеть практически реальную картину исполнения программы на высокозагруженном микропроцессоре.

Программа tvar.pp

tvar.pp

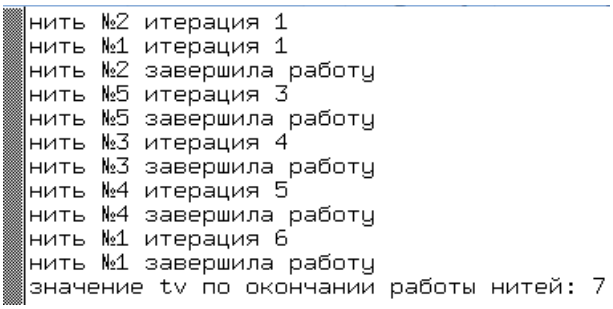
```
1  {$mode objfpc}
2  uses {$ifdef unix}cthreads,{$endif}sysutils;
3
4  const
5    threadcount = 5;  // кол-во нитей
6    incount=3;        // кол-во итераций в каждой нити
7
8  threadvar tv : longint;
9  //var tv : longint;
10
11 function f(p : pointer) : ptrint;
12 begin
13   while (tv<incount) do begin
14     sleep(random(10)+1);
15     Writeln('нить №',ptrint(p),' итерация ',tv);
16     inc(tv);
17   end;
18   result:=0;
19   writeln('нить №',ptrint(p),' завершила работу');
20 end;
21
22 var
23    i : ptrint;
24    threads:array[1..threadcount] of ptrint;
25 begin
26    tv:=1;
27    for i:=1 to threadcount do threads[i]:=BeginThread(@f,pointer(i));
28    for i:=1 to threadcount do waitForThreadTerminate(threads[i],0);
29    Writeln('значение tv по окончании работы нитей: ',tv);
30 end.
```

Результат выполнения программы (при каждом запуске может немного изменяться)

```
нить №3 итерация 0
нить №2 итерация 0
нить №5 итерация 0
нить №4 итерация 0
нить №1 итерация 0
нить №2 итерация 1
нить №4 итерация 1
нить №3 итерация 1
нить №1 итерация 1
нить №5 итерация 1
нить №4 итерация 2
нить №4 завершила работу
нить №2 итерация 2
нить №1 итерация 2
нить №2 завершила работу
нить №1 завершила работу
нить №3 итерация 2
нить №3 завершила работу
нить №5 итерация 2
нить №5 завершила работу
значение tv по окончании работы нитей: 1
```

Видим, что нити стартуют и завершаются в произвольном порядке, однако, как это и предусмотрено, каждая из них при первом входе получает значение потоковой переменной **tv** равное нулю и итерирует по нему ровно три раза. Значение **tv** в основной нити изменению не подверглось, что видно из последней строки вывода.

Результат выполнения программы после замены потоковой переменной **tv** на обычную глобальную переменную с помощью инверсии комментария в строках 8-9 (при каждом запуске может немного изменяться)



Видим, что количество итераций совершенно непредсказуемо, т. к. нити конкурируют за доступ к обычной глобальной переменной **tv**. Заметно также, что пока одна нить блокируется, другие успевают увеличить значение tv, что дает при выводе номера итерации некорректное значение. По окончании работы программы можно заметить, что значение **tv** значительно увеличилось, что еще раз доказывает, что нити работали с одной и той-же глобальной переменной.

Более подробное описание потоковых переменных доступно по адресам http://freepascal.org/docs-html/ref/refse23.html и http://freepascal.org/docs-html/prog/progse43.html

Свойства-переменные.

Свойства-переменные — концепция, импортированная Free Pascal из понятия «свойство» для класса. Суть свойства-переменной заключается в определении идентификатора, доступ к которому организован как к обычной переменной, однако при попытке записи значения в такую переменной вызывается процедура, а при попытке чтения — функция. Разрешается создавать свойства-переменные только для чтения или только для записи, а также в виде массивов (свойства-массивы), индексами которых могут быть не только порядковые, но и строковые значения.

Общий вид определения свойства-переменной:

```
property имя_переменной:тип
         [read функция_чтения] [write процедура_записи]
```

Функция чтения должна быть глобальной функцией без параметров, возвращающей значение свойства. Процедура записи быть глобальной процедурой, с единственным параметром, соответствующим типу свойства. Пример работы со свойством-переменной приведен в первой части пособия, в параграфе "Режимы совместимости"

Общий вид определения свойства-массива:

```
property имя_переменной "[" индекс1:тип1,индекс2:тип2, … "]"
         :тип[read функция_чтения] [write процедура_записи]
```

Функция чтения свойства-массива должна быть глобальной функцией, возвращающей значение свойства, с параметрами, соответствующими типу и порядку расположения индексов. Процедура записи быть глобальной

процедурой, с параметрами, соответствующими типу и порядку расположения индексов, с последним параметром, соответствующим типу свойства.

Дополнительную информацию о свойствах-переменных можно получить по адресу http://freepascal.org/docs-html/ref/refse24.html.

Рассмотрим следующий пример. В программировании часто возникает необходимость сохранять значения переменных между запусками программы, чтобы обеспечить продолжение работы при каждом новом запуске с того места, где она закончилась в предыдущий раз. Создадим прототип решения данной задачи с помощью модуля **filedata.pp**, в котором будут использованы двумерные байтовые свойства-массивы. Первым индексом свойств-массивов будет имя файла, вторым — позиция в файле, где хранится значение свойства. В целях удобства работы создадим три свойства-массива — для чтения-записи, только для чтения и только для записи. Это позволит в будущем на уровне компилятора отслеживать возможные ошибки программиста, когда он случайно попытается присвоить значение свойству только для чтения, или получить значение свойства только для записи.

Модуль filedata.pp

filedata.pp

```
1  {$mode objfpc}
2  unit filedata;
3  interface
4
5  Function fGetByte(filename:string; index:int64): byte;
6  Procedure fSetByte(filename:string; index:int64; Value:byte);
7
8  Property fData[name:string;index:int64]:byte Read fGetByte Write fSetByte;
9  Property fDataRead[name:string;index:int64]:byte Read fGetByte;
10 Property fDataWrite[name:string;index:int64]:byte Write fSetByte;
11
12 implementation
13
14 Function fGetByte(filename:string; index:int64): byte;
15  var f:file of byte;
16 begin
17  assign(f,filename);
18  try reset(f); except exit(0); end;
19  if filesize(f)<=index then begin close(f); exit(0); end;
20  seek(f,index); read(f,result);
21  close(f);
22 end;
23
24 Procedure fSetByte(filename:string; index:int64; Value:byte);
25  var f:file of byte;
26 begin
27  assign(f,filename);
28  try reset(f); except try rewrite(f); except exit; end; end;
29  seek(f,index); write(f,value);
30  close(f);
31 end;
32
33 end.
```

В модуле определены (строки 8-10) байтовые свойства-массивы fData (для чтения и записи), fDataRead (только для чтения) и fDataWrite (только для записи). Доступ к содержимому всех трех свойств организован одними и теми-же методами, в результате чего запись-чтение значений всех трех свойств приводит к идентичным результатам.

Манипуляции по записи значения свойства в файл реализованы в процедуре fSetByte (строки 24-31). Процедура открывает (при отсутствии - создает) файл байтовых значений, перемещает файловую позицию в соответствии со значением переданного индекса и осуществляет запись параметра Value в установленную позицию.

Манипуляции по чтению значения свойства из файла реализованы в функции fGetByte (строки 14-22). Функция открывает файл байтовых значений, в случае неудачи — возвращает значение 0. Если переданный индекс выходит за границы файла, функция также возвращает 0. Иначе производится установка файловой позиции в соответствии со значением переданного индекса, затем - чтение байтового значения из файла и его возврат в качестве результата функции.

Рассмотрим примеры использования модуля filedata.pp:

Программа установки значения свойств fd_write.pp

filedata.pp | fd_write.pp

```
1  {$mode objfpc}
2  uses filedata;
3  begin
4   fdataWrite['first',0]:=10;
5   fdataWrite['first',999]:=20;
6   fdata['second',0]:=100;
7   fdata['second',999]:=200;
8  end.
```

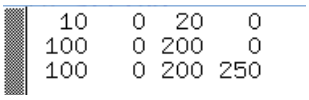
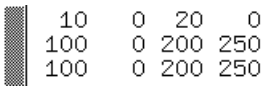
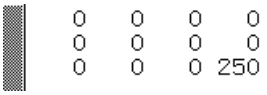
Результатом запуска данной программы будет появление в файловой системе двух бинарных файлов (first и second), размером в 1000 байт каждый. В обоих файлах будет только два значения, отличных от нуля, на позициях 0 и 999.

Программа чтения и установки значения свойств fd_readwrite.pp

filedata.pp | fd_write.pp | fd_readwrite.pp

```
1  {$mode objfpc}
2  uses filedata;
3   var fn:string;
4  begin
5   fn:='first';
6   writeln(fdataRead[fn,0]:4,fdataRead[fn,100]:4,
7           fdataRead[fn,999]:4,fdataRead[fn,9999]:4);
8   fn:='second';
9   writeln(fdata[fn,0]:4,fdata[fn,100]:4,fdata[fn,999]:4,fdata[fn,9999]:4);
10  fdata[fn,9999]:=250;
11  writeln(fdata[fn,0]:4,fdata[fn,100]:4,fdata[fn,999]:4,fdata[fn,9999]:4);
12 end.
```

Программа fd_readwrite.pp выводит содержимое некоторых элементов байтового свойства-массива. Вывод осуществляется для файлов first и second, позиций 0, 100, 999 и 9999. Затем в файле second устанавливается значение 250 в позицию 9999 и вывод для second повторяется. В результате работы программы размер файла second будет установлен в 10000 байт. Результат вывода на экран может в значительной степени варьироваться в зависимости от исходного содержимого файлов first и second

	Вывод fd_readwrite в случае предварительного запуска fd_write.
	Вывод повторного и всех последующих запусков fd_readwrite в случае предварительного запуска fd_write.
	Вывод fd_readwrite без предварительного запуска fd_write и в отсутствии файлов first и second. Побочным эффектом запуска будет появление в файловой системе файла second размером в 10000 байт.

При модификации программ fd_write.pp и fd_readwrite, либо при самостоятельном использовании модуля filedata.pp, следует соблюдать крайнюю осторожность в ОС Windows. Например, действие вида fdata['my.dat',100000000000]:=1 приведет к созданию в файловой системе файла my.dat размером в 100 гигабайт. И если для большинства файловых систем ОС Linux, данное действие будет практически мгновенным и его результатом будет создание так называемого "разреженного"(sparse) файла, в котором последовательности нулей практически не занимают дисковое пространство, то для ОС Windows указанное действие будет выполняться значительный промежуток времени, может закончиться исчерпанием свободного места и аварийной ситуацией.

Порядковые типы данных.

Free Pascal привнес в язык множество новых порядковых типов, а также модифицировал некоторые из унаследованных от Turbo Pascal.

Целочисленные и неотрицательные типы данных.

Тип данных	Новый/измененный	Диапазон значений	Размер в байтах
Smallint	новый	-32768 .. 32767	2
Integer	измененный	smallint либо longint	2 или 4
Longword	новый	0 .. 4294967295	4
Cardinal	новый	longword	4
Dword	новый	longword	4
Int64	новый	-9223372036854775808 .. 9223372036854775807	8
QWord	новый	0 .. 18446744073709551615	8

Таким образом, на сегодняшний день Free Pascal содержит следующий набор целочисленных и неотрицательных типов: однобайтовые byte, shortint; двубайтовые word, smallint, integer; четырехбайтовые dword, cardinal, longword, longint, integer; восьмибайтовые qword, int64.

Больший интерес представляет тип **integer**, размер которого теперь меняется в зависимости от используемого режима совместимости (см. параграф "Режимы совместимости" первой части пособия), что легко проверяется программой на скриншоте справа. Меняя текущий режим совместимости в строках 1-4, компилируя и запуская программу, получаем различный размер типа integer, как показано в следующей таблице:

int_size.pp

```
1  { $mode tp}
2  { $mode fpc}
3  { $mode delphi}
4  {$mode objfpc}
5  begin
6    writeln(sizeof(integer));
7  end.
```

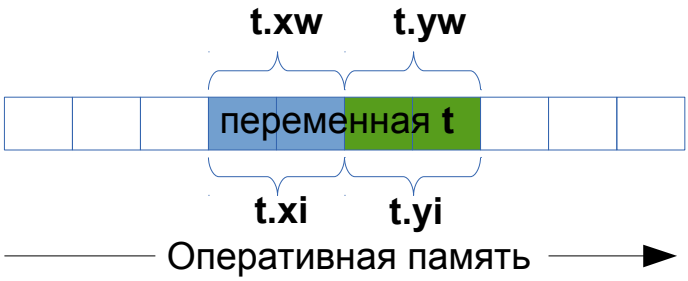
	tp	fpc	delphi	objfpc
размер типа integer	2 байта	2 байта	4 байта	4 байта

Если разработчик не учитывает данный факт, то корректно функционирующие в Turbo Pascal программы могут давать неожиданные результаты после перекомпиляции во Free Pascal с использованием режимов objfpc или delphi.

Рассмотрим конкретный пример. В программе справа разработчик задает запись с вариантной частью для хранения координат точки, применяя комбинацию типов integer и word. В результате он рассчитывает использовать как целочисленный диапазон координат от -32768 до 32767, так и, при необходимости, неотрицательный диапазон от 0 до 65525. Для экономии оперативной памяти и удобства перехода между целочисленными и неотрицательными значениями, разработчик совмещает в оперативной памяти соответствующие поля записи (**t.xw** разделяет память с **t.xi**, **t.yw** - с **t.yi**):

int_point.pp

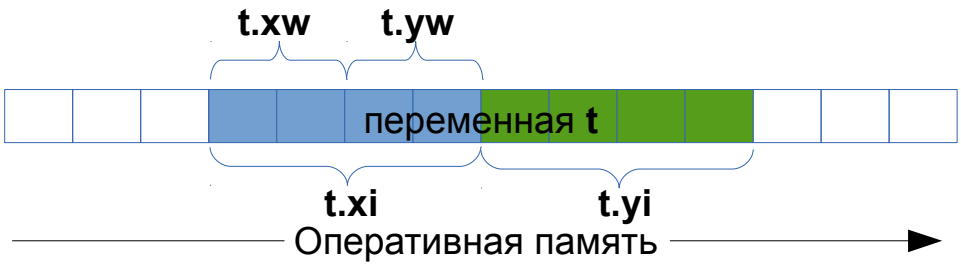
```
1  {$mode tp}
2  { $mode objfpc}
3  type Point=record
4         case boolean of
5           false: (xw,yw:word);
6           true: (xi,yi:integer);
7       end;
8  var T:point;
9  begin
10   t.xw:=10;
11   t.yw:=20;
12   writeln(t.xi,' ',t.yi);
13 end.
```



Данный пример работает корректно в Turbo Pascal и во Free Pascal в режиме совместимости с Turbo Pascal, однако возвращает ошибочный результат при использовании режима objfpc, что легко проверить инверсией комментариев в строках 1-2 программы:

Результат работы программы int_point.pp в режиме **tp**	Результат работы программы int_point.pp в режиме **objfpc**
10 20	1310730 0

Дело в том, что в режиме objfpc двухбайтовое поле **t.xw** разделяет память с младшим словом четырехбайтного поля **t.xi**, а поле **t.yw** - со старшим словом поля **t.xi**. В результате изменения значений полей **t.xw** и **t.yw**, изменяется только значение поля **t.xi**, причем совершенно не так, как задумал программист:



Значение **1310730** для **t.xi** будет получено на платформах, где используется порядок расположения байт в оперативной памяти от младшего к старшему (little-endian), например - на intel x86 и amd64/x86_64. На платформах, где применяется обратный порядок (big-endian), например на платформе SPARC, вместо **1310730** будет получено значение **655380**. Однако данный факт не исправит ошибку в работе программы.

Впервые размер типа **integer** был изменен в Delphi, а Free Pascal просто унаследовал данное поведение. В связи с динамичностью размера типа integer, стал востребованным целочисленный тип **smallint** с фиксированным размером в 2 байта. Данным типом рекомендуется заменить тип integer в коде всех программ, созданных для Turbo Pascal.

Четырехбайтный неотрицательный тип **cardinal** был введен в целях совместимости с Delphi и полностью соответствует типу **longword**.

Завершают таблицу два типа данных максимального среди порядковых типов объема - знаковый **int64** и неотрицательный **qword** . С учетом распространения 64-битных компьютерных систем ввод в язык восьмибайтных типов является очень важным нововведением, особенно в задачах обработки больших файлов и в адресной арифметике.

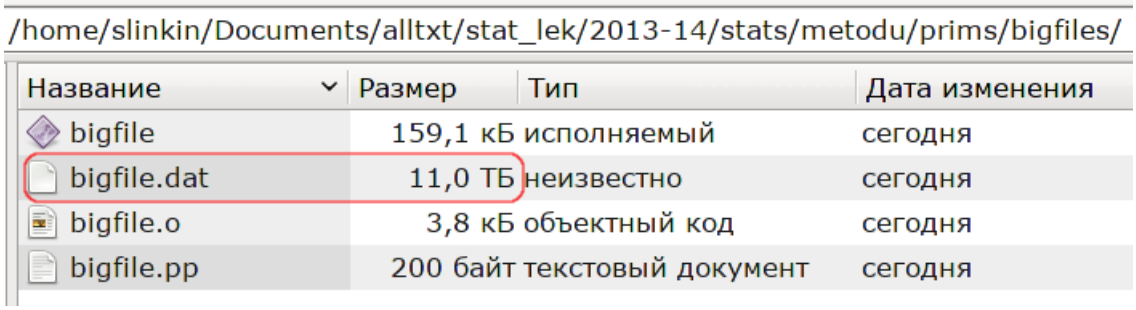
Например, функция **FileSize** во Free Pascal возвращает значение типа **int64**, что позволяет определять размер файла до 9 эксабайт (для сравнения - жесткий диск распространенного в настоящее размера 1 терабайт - в 9 миллионов раз меньше, чем размер одного такого файла), в то время как в Turbo Pascal максимально определяемый размер файла был 2 гигабайта благодаря используемому в функции FileSize типу longint. Тип int64 используется и во многих других функциях обработки файлов, позволяя выполнять с ними действия, недоступные в Turbo Pascal.

Программа справа демонстрирует возможности Free Pascal в обработке больших файлов. При запуске она создает файл **bigfile.dat** размером в 10 терабайт и возвращает его размер в байтах. Не пытайтесь запустить данную программу в файловых системах NTFS или exFAT ОС Windows, это приведет к попытке действительного создания файла указанного размера, что, скорее всего, окажется невозможным по причинам недостаточного объема жесткого диска. Если в ОС Windows применяется файловая система fat32, то создание файла окажется принципиально невозможным по причинам ограниченности максимального размера файла в 4 гигабайта. На моем компьютере, с операционной системой Linux и файловой системой ext4, создание данного файла завершается успешно, благодаря поддержке файловой системой

bigfile.pp

```
1  {$mode objfpc}
2   var f:file of byte;
3  begin
4   assign(f,'bigfile.dat');
5   rewrite(f);
6   seek(f,1024*1024*1024*1024*10-1);
7  //      1Kb->1Mb->1Gb->1Tb->10Tb
8   write(f,1);
9   writeln(fileSize(f));
10  close(f);
11 end.
```

"разреженных" (sparse) файлов и максимального размера одного файла в 16 терабайт. Результатом работы программы будет практически мгновенное создание в файловой системе "разреженного" файла и вывод его размера в байтах: **10995116277760**.



Не рекомендуется выполнять действий, предусматривающих обработку содержимого файла **bigfile.dat**, а именно: копирование, перенос, загрузку в различные программы и т. д. Это может закончиться аварийным остановом или зависанием соответствующей программы, если ее разработчики не предусмотрели возможность проведения операций с файлами такого размера. Наилучшим действием, если, конечно не предусматривается исследование возможностей "разреженных" файлов, будет его удаление.

Восьмибайтовые типы **int64** и **qword** по разному обрабатываются в зависимости от разрядности операционной системы. В 64-х разрядных системах операции над этими типами реализуются встроенными машинными командами. В 32-х разрядных системах такие операции приходится эмулировать, что значительно замедляет работу программы.

В следующем примере сравним время, затраченное на многократные вычисления с использованием идентичных арифметических операций (+,-,*,/) над значениями типа longint и int64 в 32-х разрядной и 64-х разрядной операционных системах.

32vs64.pp

```
1  {$mode objfpc}
2  uses sysutils,dateutils;
3   const MAX=200000000;
4  {$IFDEF CPU64} BITS=64; {$ENDIF}
5  {$IFDEF CPU32} BITS=32; {$ENDIF}
6  {$IFDEF CPU16} BITS=16; {$ENDIF}
7   var
8       i32,MulDiv32:longint;
9       i64,MulDiv64:int64;
10      dt:tdatetime;
11 begin
12  dt:=now();
13  i32:=1; MulDiv32:=10;
14  while i32<MAX do i32:=(((i32+2)*MulDiv32) div MulDiv32)-1;
15  writeln(millisecondsbetween(now(),dt),' мс: 32-битная арифметика для ',BITS,'-разрядной ОС');
16
17  dt:=now();
18  i64:=1; MulDiv64:=10;
19  while i64<MAX do i64:=(((i64+2)*MulDiv64) div MulDiv64)-1;
20  writeln(millisecondsbetween(now(),dt),' мс: 64-битная арифметика для ',BITS,'-разрядной ОС');
21 end.
```

В 14 и 19 строках программы выполняются вычисления с использованием соответственно 32-битных и 64-битных переменных. Каждое вычисление, не смотря на свою кажущуюся громоздкость, приводит к увеличению на единицу значения переменной i32 или i64, что позволяет достаточно просто управлять количеством итераций цикла, которое, в свою очередь подобрано таким, чтобы с одной стороны, не выйти за пределы 32-битного значения во время вычислений, а с другой - чтобы общее время выполнения программы на целевой системе было порядка 5 секунд, что вполне достаточно для примерной оценки производительности.

Компиляция программы была проведена в 4-х операционных системах. Во всех случаях использовался один и тот-же компьютер - ноутбук Dell Latitude E5520 с микропроцессором Intel(R) Core(TM) i5-2520M, где в двойной загрузке функционируют 64-разрядная ОС Windows 7 и 64-разрядная ОС Linux. В ОС Linux используется система виртуализации VirtualBox, где в свою очередь функционируют 32-разрядная ОС Windows XP и 32-разрядная ОС Linux. Так как применяется технология аппаратной виртуализации, обе 32-разрядные системы работают практически без потерь производительности.

На каждой ОС приложения были запущены не менее 10 раз, выявленный разброс значений не превысил 5%. Некоторые из полученных результатов приведены ниже:

Windows XP, 32bit

```
C:\>32vs64.exe
2313 мс: 32-битная арифметика для 32-разрядной ОС
6379 мс: 64-битная арифметика для 32-разрядной ОС
```

Linux, 32bit

```
[slinkin@p7server32 ~]$ ./32vs64
2420 мс: 32-битная арифметика для 32-разрядной ОС
6143 мс: 64-битная арифметика для 32-разрядной ОС
```

Windows 7, 64bit

```
C:\>32vs64.exe
3462 мс: 32-битная арифметика для 64-разрядной ОС
3510 мс: 64-битная арифметика для 64-разрядной ОС
```

Linux, 64bit

```
[slinkin@sdanout 32vs64]$ ./32vs64
3413 мс: 32-битная арифметика для 64-разрядной ОС
3418 мс: 64-битная арифметика для 64-разрядной ОС
```

Эксперимент в целом подтвердил низкую эффективность обработки в программах на Free Pascal 64-битных данных в 32-битных операционных системах (более чем в 2.5 раза), показал эквивалентную эффективность арифметических вычислений в Windows и Linux, а также выявил с первого взгляда парадоксальный факт более высокой эффективности работы 32-битных приложений по сравнению с 64-битными в случае обработки 32-битных данных (примерно в 1.5 раза). Последний вывод плохо согласуется с рекламными заявлениями производителей микропроцессоров, однако является неоспоримым и легко доказуемым фактом.

Кроме встроенных в язык типов, в автоматически подключаемом модуле system определяются множество дополнительных целочисленных и неотрицательных типов, выражаемых через стандартные типы данных (см. http://freepascal.org/docs-html/rtl/system/index-3.html). Это сделано для лучшей читаемости программ и возможности при необходимости изменять размер и/или строение типа на различных платформах.

В модуле system среди прочих определены типы uint8 (byte), uint16 (word), uint32 (dword), uint64 (qword), название которых позволяет догадаться об их размере и отношении к порядковым неотрицательным типам: например uint64 расшифровывается как **u**nsigned **i**nteger **64** bit, т. е. беззнаковый целый 64-разрядный тип. Еще один характерный пример имеет отношение к типу integer. Строго говоря, Free Pascal не относит тип integer к встроенным в язык типам данных, он определяется в модуле system как smallint (http://freepascal.org/docs-html/rtl/system/integer.html). Однако при использовании режимов

совместимости objfpc или delphi автоматически подключается модуль objpas, в котором тип integer переопределяется как longint (http://freepascal.org/docs-html/rtl/objpas/integer.html).

Логические типы данных

Во Free Pascal введено несколько новых логических типов в дополнение к типу Boolean. Некоторым модификациям подвергся и тип Boolean:

Тип	Новый/ измененный	Размер в байтах	ord(false)	ord(true)	ord(not false)
Boolean	измененный	1байт или 1бит	0	не ноль	1
ByteBool	новый	1	0	не ноль	-1
WordBool	новый	2	0	не ноль	-1
LongBool	новый	4	0	не ноль	-1

Уникальным новшеством во Free Pascal является возможность переменным перечислимого типа и типа поддиапазона занимать в оперативной памяти, при выполнении определенных условий, **один или несколько бит**, а переменным типа Boolean - **один бит**, что позволяет значительно экономить оперативную память при работе с большим количеством переменных, упрощает обработку низкоуровневых данных (пакетов сетевых протоколов, форматов изображений и т. п.), дает возможность создавать плотные битовые структуры данных, например - аналог типа множество (**set of ...**) без ограничений на 256 значений, обеспечивает прямое управление отдельными битами любой области оперативной памяти без применения сложных выражений с логическими операциями. Boolean занимает 1 бит при выполнении двух условий:

1) переменные типа Boolean должны храниться в упакованном массиве или записи;

2) должна быть установлена директива компилятора {$BITPACKING ON}.

Рассмотрим пример программы, демонстрирующий доступ к различным битам значения типа Word с использованием битового логического типа Boolean. Предполагаем, что программа запускается на платформе, использующей порядок расположения байт в оперативной памяти от младшего к старшему (little-endian), например - amd64/x86_64.

bitrec.pp

```
1  {$BITPACKING ON}
2  Type
3      TBitRec = packed record
4          case byte of
5              0: (B0,B1,B2,B3,B4,B5,B6,B7,B8,B9 : Boolean);
6              1: (W:word);
7      end;
8  var
9      Data:TBitRec;
10     A:packed array [0..15] of boolean absolute data;
11 begin
12     fillbyte(data,sizeof(data),0);
13     writeln('Размер записи Data: ', sizeof(data),', адрес ',ptruint(@data));
14     writeln('Размер поля Data.W: ', sizeof(data.W),', адрес ',ptruint(@data.W));
15     writeln('Размер массива A: ', sizeof(A),', адрес ',ptruint(@A));
16     writeln('Первоначальное содержимое поля data.W: ',data.W);
17     data.b0:=true; writeln('Бит 0 установлен, результат: ',data.W);
18     data.b9:=true; writeln('Бит 9 установлен, результат: ',data.W);
19     a[1]:=true; writeln('Бит 1 установлен, результат: ',data.W);
20     a[15]:=true; writeln('Бит 15 установлен, результат: ',data.W);
21 end.
```

```
Размер записи Data: 2, адрес 6448320
Размер поля Data.W: 2, адрес 6448320
Размер массива A: 2, адрес 6448320
Первоначальное содержимое поля data.W: 0
Бит 0 установлен, результат: 1
Бит 9 установлен, результат: 513
Бит 1 установлен, результат: 515
Бит 15 установлен, результат: 33283
```

Скриншот справа демонстрирует результат запуск программы **bitrec.pp**.

В программе определены две переменные: **A** - упакованный массив логических битовых значений (строка 10) и **Data** - упакованная запись с вариантной частью (строка 9). В записи в виде одного из вариантов описаны битовые логические поля **B0..B9** (строка 5), разделяющие оперативную память с полем **W** типа word (строка 6):

Первые три строки вывода программы демонстрируют тот факт, что обе переменные и поле записи разделяют одну и ту-же область оперативной памяти (выводимый адрес может измениться при повторном запуске программы) и имеют одинаковый размер.

В строках 17, 18, 19 и 20 производится установка битов 0, 9, 1 и 15 в разделяемой переменными области памяти с выводом содержимого памяти на экран при каждой установке:

Перечислимые типы данных

Free Pascal позволяет указывать целочисленное константное выражение для любого значения перечислимого типа в его определении. Такая возможность доступна только в режимах совместимости objfpc и fpc.

В примере справа в строке 3 определен перечислимый тип **TEnum**, содержащий семь значений от **e0** до **e6**. Основная программа (строки 4-9) выводит на экран числовое представление каждого значения данного типа.

e1.pp

```
1  {$mode objfpc}
2  type
3   TEnum=(e0,e1,e2:=10,e3,e4,e5:=20,e6);
4  begin
5  writeln(e0,'=',ord(e0),', ',e1,'=',ord(e1),', ',
6          e2,'=',ord(e2),', ',e3,'=',ord(e3),', ',
7          e4,'=',ord(e4),', ',e5,'=',ord(e5),', ',
8          e6,'=',ord(e6));
9  end.
```

Результат работы программы:

```
e0=0, e1=1, e2=10, e3=11, e4=12, e5=20, e6=21
```

Отсюда видно, что присвоение значению числовой константы порождает новую возрастающую последовательность внутри перечислимого типа.

Рекомендуется использовать такие числовые константы, чтобы вся последовательность значений в типе была возрастающей. В противном случае компилятор сгенерирует предупреждение.

Для примера заменим выражение **e5:=20** в строке 3 на **e5:=-1**, сохраним под именем e2.pp, скомпилируем и запустим программу. Результаты компиляции, в том числе и предупреждение, показаны справа, однако исполняемый файл программы будет успешно создан и при его запуске будет получен следующий результат:

```
Компилятор Free Pascal версии 2.6.2 [2013/03/11] для x86_64
Copyright (c) 1993-2012 by Florian Klaempfl
Целевая ОС: Linux for x86-64
Компиляция e2.pp
e2.pp(3,34) Заметка: Значения в перечисляемых типах должны быть возрастающими
Компоновка e2
/usr/bin/ld: warning: link.res contains output sections; did you forget -T?
Сборка прошла успешно.
9 строк скомпилировано, 0.1 сек.
1 заметок
```

```
e0=0, e6=1, e2=10, e3=11, e4=12, e5=-1, e0=0
```

Видим, что с **e5** началась новая возрастающая последовательность, а также, что значения

e6 и **e0** равны нулю. Компилятор оказался не в состоянии отличить значения **e6** от **e0**, в результате чего на экране вместо строки **e6** появилась строка **e0**.

Минимальный размер переменной перечислимого типа может быть равен 1, 2 или 4 байта и зависит от двух факторов: во первых, от директивы компилятора $Z или $PACKENUM (http://freepascal.org/docs-html/prog/progsu59.html), и во вторых - от диапазона числовых значений в перечислимом типе. Второй фактор присутствовал и в Turbo Pascal, хотя проверить возможность существования четырехбайтовых перечислимых типов было достаточно проблематично, т. к. для этого следовало в одном типе использовать более 65536 значений. Во Free Pascal, благодаря возможности самостоятельно определять значение констант перечислимого типа, такая проверка сложности не представляет.

В примере справа определяется 3 перечислимых типа различных размеров. С помощью директивы **{$Z1}** устанавливается минимальный размер перечислимого типа в 1 байт. Тип **TEnum1** содержит два значения, равных в числовом выражении 0 и 1, входящих в однобайтовый диапазон. Второе значение типа **TEnum2** равно 1000, что требует для хранения переменной такого типа минимум два байта. Второе значение типа **TEnum4** равно 100000, что обязует компилятор выделить для хранения переменной такого типа четыре байта. Результат запуска программы дает ожидаемые значения: 1 2 4 . Если заменить директиву **{$Z1}** на **{$Z2}**, установив тем самым минимальный размер перечислимого типа в 2 байта, то на выходе программы получим значения: 2 2 4 . Замена **{$Z1}** на **{$Z4}** даст следующий результат: 4 4 4 . Директива **{$Z4}** установлена по умолчанию для режимов совместимости objfpc и fpc, в то время как в режимах tp и delphi по умолчанию используется **{$Z1}**.

e3.pp

```
1  {$mode objfpc}
2  {$Z1}
3  type
4   TEnum1=(e1_0,e1_1);
5   TEnum2=(e2_0,e2_1:=1000);
6   TEnum4=(e4_0,e4_1:=100000);
7  begin
8   writeln(sizeof(TEnum1),
9        ' ',sizeof(TEnum2),
10       ' ',sizeof(TEnum4));
11 end.
```

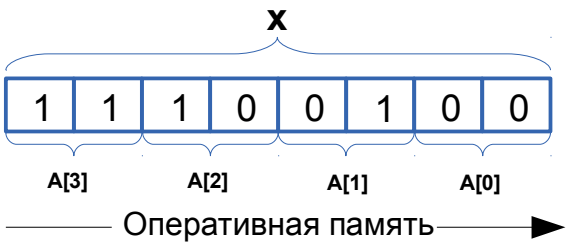
Перечислимый тип, как и логический, может занимать в оперативной памяти всего несколько бит, подчиняясь директиве $BITPACKING. Количество бит зависит от диапазона значений перечислимого типа и, в отличии от логического типа, может быть больше одного.

В примере справа определяется тип **bitenum** (строка 2), каждая переменная которого, при использовании в упакованном массиве **a** (строка 3), будет занимать два бита, т. к. числовые значения типа находятся в диапазоне от 0 до 3. Массив **a** содержит 4 двубитовых значения и занимает таким образом, 8 бит или 1 байт. Для проверки возможностей управления двубитовыми наборами определяется байтовая переменная **x**, которая разделяет память с массивом **a**. В строках с 6 по 9 двубитовыми блоками заполняется массив **a**, в 10 строке выводится содержимое сформированного байта. В результате работы программы на экране появится число 228 .

bitenum.pp

```
1  {$BITPACKING ON}
2  type bitenum=(null,one,two,three);
3  var a: packed array[0..3]of bitenum;
4      x: byte absolute a;
5  begin
6   a[0]:=null;
7   a[1]:=one;
8   a[2]:=two;
9   a[3]:=three;
10  writeln(x);
11 end.
```

Значения **bitenum** в двоичном преставлении равны: **null=%00**, **one=%01**, **two=%10** и **three= %11.** Схема справа иллюстрирует расположение и содержимое двубитовых элементов массива A в оперативной памяти по окончании работы программы. Значение байта рассчитываем следующим образом: 128+64+32+0+0+4+0+0= **228**.



Тип поддиапазона

Единственное отличие, внесенное Free Pascal в тип поддиапазона, это возможность переменной данного типа занимать несколько бит оперативной памяти аналогично тому, как это было реализовано для логического и перечислимого типов. При определении перечислимого типа его объем может вычисляться в битах на основе анализа нижней и верхней границы диапазона.

В примере справа определен тип поддиапазона **bit3** (строка 2), переменные которого могут хранить значения от 0 до 7. Для такого диапазона значений достаточно трех битов оперативной памяти. Поэтому упакованный массив **a** на десять элементов (строка 3) будет размером в 30 битов. Результат запуска программы: 4. Так как оперативная память выделяется программе не в битах, а в байтах, то размер массива будет определен в **4 байта** или 32 бита, из которых 2 последних бита не будут задействованы.

bitsubrange.pp

```
1  {$BITPACKING ON}
2  type bit3=0..7;
3  var a: packed array[1..10]of bit3;
4  begin
5   writeln(sizeof(a));
6  end.
```

Символьные типы

Free Pascal вводит новый символьный тип WideChar, так называемый **широкий символьный тип**. Это **двубайтовый** тип, позволяющий хранить один из символов стандарта Unicode (http://www.unicode.org/, http://ru.wikipedia.org/wiki/Unicode). Подробно об этом типе, а также об обработке строк Unicode будет рассказано в следующей части пособия.

В модуле System определено большое количество символьных типов, однако ни один из них не нуждается в дополнительных комментариях, так как все они определяются в конечном итоге через Char (AnsiChar, TAnsiChar), WideChar (WChar, UCS2Char, UnicodeChar) или через тип поддиапазона (UCS4Char).

Вещественные типы данных

Серьезные изменения были внесены в тип **Real**, добавлен новый тип **Currency**.

В Turbo Pascal тип Real был размером в 6 байтов, имел уникальный формат, а для манипуляций с ним использовалась специализированная математическая библиотека, аппаратные средства напрямую не применялись. Дело в том, что Turbo Pascal был разработан для платформы Intel/x86, в которой достаточно долгое время математический сопроцессор был опциональной возможностью.

Поэтому тип Real изначально был предназначен для обработки без использования математического сопроцессора, в отличие от вещественных типов Single, Double, Extended, Comp, соответствующих стандарту IEEE 754 (см. например http://en.wikipedia.org/wiki/IEEE_floating_point, http://ru.wikipedia.org/wiki/IEEE_754-2008) и ориентированных на аппаратную обработку.

Во Free Pascal тип Real соответствует восьмибайтовому типу Double. В ранних версиях Free Pascal и на некоторых платформах без поддержки аппаратной вещественной арифметики, тип Real соответствует четырехбайтовому типу Single. Однако на системах, которые рассматриваются в данном пособии, тип Real всегда эквивалентен типу Double.

В примере справа проводится исследование типа Real, вычисляются некоторые его характеристики: объем оперативной памяти, занимаемый переменной типа Real (строка 5), минимальная степень десятки, доступная для данного типа (строки 6-11), максимальная степень десятки, доступная для данного типа (строки 12-17), точность или количество значимых знаков после десятичной точки (строки 18-24). При запуске программы получаем:

```
Размер в байтах: 8
Минимальная степень: -324
Максимальная степень: 308
Точность: 15-16 знаков
```

Отсюда видим, что характеристики типа Real соответствуют типу Double (http://www.freepascal.org/docs-html/ref/refsu6.html). Если тип переменной **x** изменить на Double (строка 2), результат выполнения программы не изменится.

realdouble.pp

```
1  {$mode objfpc}
2  var x:real; delta:extended;
3      count:integer;
4  begin
5   writeln('Размер в байтах: ',sizeof(x));
6   x:=1.0; count:=0;
7   while (x<>0) do begin
8    x/=10;
9    inc(count);
10  end;
11  writeln('Минимальная степень: ',-count);
12  x:=1.0; count:=0;
13  while (true) do begin
14   try x*=10; except break; end;
15   inc(count);
16  end;
17  writeln('Максимальная степень: ',count);
18  delta:=1.0; count:=0;
19  repeat
20   x:=1.0; delta/=10; x+=delta;
21   inc(count);
22  until x=1.0;
23  writeln('Точность: ',count-1,
24          '-',count,' знаков');
25 end.
```

Во Free Pascal введен вещественный тип данных **Currency**, имеющий только 4 знака после десятичной точки. Ниже представлены его характеристики:

Диапазон значений	Точность	Байт на переменную
-922337203685477.5808 .. 922337203685477.5807	19-20	8

Currency предназначен, в основном, для финансовых вычислений, для которых важна высокая точность и где целая часть представляет собой основную денежную единицу, а дробная - разменную монету (рубли-копейки, доллары-центы, марки-пфеннинги, фунты-пенсы и т.п.). Такой-же точностью обладают еще два вещественных типа: Extended и Comp, однако Extended имеет размер в 10 байт против 8 байт Currency, а Comp, хотя и обрабатывается математическим

сопроцессором, является формально целочисленным и не имеет дробной части, что не позволяет вести расчеты в основных денежных единицах.

Заключение

Завершая вторую часть пособия, хочется отметить, что Free Pascal вбирает в себя изменения и нововведения множества программных концепций, обеспечивая при этом как языковую совместимость с современными коммерческими компиляторами языка Pascal (например - Embarcadero Delphi), так и уникальные возможности, недоступные в подобных системах (например - управление битовыми структурами данных).

Следующая, третья часть пособия, будет полностью посвящена символьным и строковым типам данных, а также возможностям, предоставляемым Free Pascal в обработке текста Unicode.