НЕКОТОРЫЕ НОВЫЕ ВОЗМОЖНОСТИ ЯЗЫКА ФРИ ПАСКАЛЬ (для режима **-Mtp**)

1 Порядок разделов в описаниях

Если в стандарте языка Паскаль зафиксирован жесткий порядок следования разделов описаний в начале блоков (метки, константы, типы, переменные, процедуры и функции), то в языке Фри Паскаль эти разделы могут быть расположены в любом порядке, причем каждый раздел может появляться многократно. Однако при этом сохраняется общее правило Паскаля: на любое имя можно ссылаться только после того, как оно описано.

2 Дополнительные целые типы

Стандартный тип *integer* определяет целые числа размером в 2 байта (от -2^{15} до $2^{15}-1$). Помимо этого во Фри Паскале имеются и другие стандартные целые типы со следующими именами:

shortint - целые размером в 1 байт (от -128 до +127)
smallint - целые числа размером в 2 байта (от -2^{15} до $2^{15}-1$) – эквивалентен типу *integer*
longint - целые размером в 4 байта (от -2^{31} до $2^{31}-1$)
int64 - целые размером в 8 байтов (от -2^{63} до $2^{63}-1$)
byte - неотрицательные целые в 1 байт (от 0 до 255)
word - неотрицательные целые в 2 байта (от 0 до $2^{16}-1$)
longword - неотрицательные целые в 4 байта (от 0 до $2^{64}-1$)

Отметим также, что если некоторый символ (типа *char*) надо задать его явным десятичным кодом, например кодом 83, тогда вместо *char(83)* можно записать *#83*. Пример: `c:=#83`.

3 Типизированные константы

Так не очень удачно называются обычные переменные, но имеющие начальные значения, т.е. значения, которые они получат при входе в программу. Затем эти значения можно и изменить. Описываются эти переменные таким образом:

```
const <имя переменной>:<ее тип>=<константное выражение>;
```

где константное выражение (выражение, операндами которого могут быть только константы и обращения к стандартным функциям с параметрами-константами) как раз и задает начальное значение переменной. Пример:

```
const  x: integer = 100;
```

Замечание: в диалекте Фри Паскаля (т.е. без опции **–Mtp)** еще удобнее, можно выполнять инициализацию непосредственно в разделе переменных, например:

```
var <имя переменной>:<ее тип>=<константное выражение>;
```

Пример:

```
var  c: char = #83;
```

4 Дополнительные операторы, процедуры и функции

halt - останов программы
exit - выход из тела той процедуры или функции, где встретился этот оператор
break – досрочный выход из for, while или repeat- цикла на следующий по порядку оператор
continue – немедленное прекращение выполнения текущей итерации for, while или repeat-цикла с попыткой перехода на новый шаг цикла
inc(x) - *x:=x+1* (*x* - числовая переменная)
dec(x) - *x:=x-1* (*x* - числовая переменная)
random(n) - функция, при каждом обращении к которой выдается новое случайное целое число из [0, n-1].

5 Расширенные возможности оператора CASE

В списке констант каждого варианта можно указывать диапазон констант. В конце оператора *case* допустима *else*-часть, на которую передается управление, если в предыдущих вариантах нет подходящей константы. Пример:

```
   case k of
1,4..8,15: x:=5; {выполняется при k=1, 4, 5, 6, 7, 8 или 15}
9: x:=4;         {выполняется при k=9}
10..12: inc(x);  {выполняется при k=10, 11 или 12}
else x:=0        {выполняется при остальных значениях k}
   end
```

6 Строки переменной длины (тип string)

Во Фри Паскале можно использовать строки в том смысле, как они определены в стандарте языка (как упакованные символьные массивы). Однако во Фри Паскале имеются более удобные строки переменной длины (далее – просто "строки"), размер которых может меняться в процессе выполнения программы. Эти строки считаются относящимися к стандартному типу (точнее, типам) *string*.

6.1 *Описание строк и представление их в памяти*:

```
var S:string[m];       {m - максимальная длина строки S (m<=255)}
```

Если *m*=255, то часть [255] можно опустить:

```
var T:string;      {эквивалентно: var T:string[255];}
```

Строка *S* представляется в памяти как символьный массив из *m*+1 байтов, которые индексируются от 0 до *m*. В байте *S*[0] хранится текущая длина (*k*, *k*<=*m*) строки, а в байтах *S*[1], ..., *S*[*k*] - сами символы строки (элементы *S*[*k*+1] и т.д. считаются не относящимися к текущему значению строки). Доступ к элементам строки осуществляется с помощью индексных переменных *S*[*i*], которые рассматриваются как переменные типа *char*. Переменная *S*[0] обеспечивает доступ к текущей длине строки, но следует учитывать, что и эта переменная имеет тип *char*; поэтому узнать длину строки можно с помощью выражения *ord*(S[0]), однако лучше воспользоваться стандартной функцией *length* (см. ниже).

6.2 *Строки-константы*

Явно заданные строки записываются как '$c_1c_2...c_n$', где c_i - символы и $n \geq 0$. Допускается пустая строка (''). Строка из одного символа (например, 'a') является одновременно и величиной типа *char*.

6.3 *Операции над строками*

1) *Присваивание*: S:=<строковое выражение>

Особый случай здесь: если длина присваиваемой строки больше максимальной длины строки *S*, то лишние справа символы отбрасываются. Пример:

```
var S1:string[10]; S2:string[5];
     ...
S1:='12345678';  {значение S1 - строка '12345678'}
S2:=S1;          {значение S2 - строка '12345'}
```

2) *Конкатенация (сцепление) строк*: S1+S2+...+Sk

Результат - строка, полученная последовательным выписыванием символов указанных строк. Если получилось более 255 символов, то 256-й и последующие символы отбрасываются.

3) *Сравнение строк на =, <>, <, <=, > и <=*

Длины сравниваемых строк могут быть различными. Например: 'abc'>'ab' - истина.

6.4 *Стандартные функции для работы со строками*

length(S) - текущая длина строки S

pos(SS,S) - номер позиции в строке *S*, с которой начинается первое вхождение строки *SS* в *S*, или 0, если *SS* не входит в строку *S* Примеры:

```
pos('*','+-*/') → 3
```

`pos('ac','abcde')` → `0`

copy(S,p,n) - равно `S[p..p+n-1]`, т.е. выдается копия части строки *S* из *n* символов начиная с *p*-го.

Особые случаи: *p*>255 - ошибка

p>length(*S*) - выдается пустая строка

p+n>length(*S*) - выдается `S[p..length(S)]`

6.5 *Некоторые стандартные процедуры для работы со строками*

Delete(S,p,n) - `S:=S[1..p-1]+S[p+n..length(S)]`, т.е. из строки *S* удаляется *n* символов начиная с *p*-го.

Особые случаи: *p*>255 - ошибка

p>length(S) - *S* не меняется

p+n>length(S) - `S:=S[1..p-1]`

Insert(SS,S,p) - вставка подстроки *SS* в строку *S* начиная с *p*-й позиции. Пример:

```
S:='abcde'; insert('XX',S,3) -> S='abXXcde'
```

Особые случаи: *p*>255 - ошибка

p>length(*S*) - `S:=S+SS`

если длина результата больше максимальной длины *S*, то лишние справа символы отбрасываются

Str(x,S) - аналог *write(x)* при числовом *x*, но значение *x* (как последовательность символов) не выводится, а записывается в строку *S*. Параметр *x* должен иметь тот же вид, что и в процедуре *write*: *e*, или e:*n*, или *e:n:d*.

6.6 *Строковые функции, параметры-строки*

В языке Фри Паскаль разрешены функции, значениями которых являются строки. При описании таких функций их тип указывается только именем типа:

```
type T=string[80];
function F1(...):T; ...
function F2(...):string; ...
```

Формальные параметры-строки также описываются только именем типа:

```
procedure P(var a:T; b:string); ...
```

Если параметр-строка вызывается по значению, то в качестве фактического параметра может быть указана строка любой длины (если она слишком длинная, то лишние справа символы будут отброшены). Если же параметр-строка вызывается по ссылке, то типы формального и фактического параметра должны быть идентичными. Из этого, в частности, следует, что у этих строк должны быть одинаковые максимальные длины; однако это ограничение можно обойти (оно не будет проверяться), если в начале программы указать директиву `{$V-}`.

7 Процедурные типы

В стандартном Паскале для подпрограмм, кроме параметров-значений и параметров-переменных, предусмотрены еще два вида параметров: параметры-процедуры и параметры-функции. Это удобно для записи на Паскале обобщенных алгоритмов, в которых входными являются не только простые данные, но и от функции или другие алгоритмы (процедуры).

Пусть, например, нужно описать функцию `Root(f,a,b,eps)`, которая методом деления отрезка пополам находит с точностью `eps` корень уравнения f(x)=0 на интервале (a,b). Заметим, что первый параметр в Root должен быть функцией. В основной программе можно описать несколько функций, реализующих некоторые математические функции, например f1(x)=exp(x)+1; f2(x)=sin(x); f3(x)=4-x и указывать их в качестве фактического параметра при обращении к Root. Таким образом, с помощью одной и той же функции Root можно решать различные уравнения. Другой пример – описать процедуру `Plot(f, a,b)`, которая с помощью звездочек '*' печатает график функции f на отрезке [a,b]. Одну и ту же процедуру можем использовать для печати графиков различных функций.

Во Фри Паскале передача процедур и функций в качестве параметров имеет свои особенности: вводятся так называемые процедурные типы, которые указывают, какой вид подпрограммы (процедуру или функцию) можно использовать в качестве параметра и с какими параметрами должны быть эти подпрограммы. Объявление процедурного типа похоже на заголовок подпрограммы, но в отличие от заголовка не указывается имя подпрограммы.
Функции и процедуры, которые будут передаваться в качестве параметров, должны компилироваться с директивой компилятора {$F+} (вместо этого можно указывать директиву far; после заголовка) для получения полного адреса программ. Они не должны быть вложенными в другие подпрограммы. Стандартные процедуры и функции передавать в качестве параметров нельзя. (Чтобы обойти это ограничение можно описать подпрограмму-«оболочку», внутри которой происходит обращение к стандартной подпрограмме).

Пример.

```
type
  Func = function (x:real): real;
      {Тип: Функция вещественного типа с одним параметром-значением}
  Proc = procedure (var a, b: integer);
      {Тип: Процедура с двумя параметрами-переменными целого типа}


{$F+}
     function f1 (x:real): real;  begin  f1:=exp(x)+1  end;
     function f2 (x:real): real;  begin  f2:=sin(x)  end;
     function f3 (x:real): real;  begin  f3:=4-x  end;
```

В программе можно ввести переменные процедурных типов:

```
var
   p: Proc;
   f: Func;
```

В разделе операторов переменным `p` и `f` можно присваивать имена конкретных процедур и функций подходящего типа. Например, после присваивания `f:=f2` вызов `f(x)` будет означать обращение к функции `f2` с параметром `x`. Если далее выполнить присваивание `f:=f3`, то теперь тот же самый вызов `f(x)` означает обращение к `f3`.

8 Модули

В стандартном Паскале любая программа состоит из одной единицы трансляции – текста, начинающегося ключевым словом `program` и завершающегося точкой после ключевого слова `end`. Такой подход к трансляции удобен для создания небольших по объему программ (например, учебных), но не удобен для индустриального программирования, поскольку объем промышленной программы, как правило, велик, и над ней обычно трудятся несколько человек одновременно. Кроме того, процедуры и функции, разработанные для одной программы, часто годятся и для других программ. Нужно средство, чтобы воспользоваться в новой программе уже готовыми процедурами, функциями и другими программными элементами, а не включать их заново в виде текстовых фрагментов. Таким средством в языке Фри Паскаль является *модуль*.

Модуль — это автономно транслируемая программная единица для хранения типов, констант, переменных, процедур и функций. Работа над удачно спроектированной программой из нескольких модулей может быть организована параллельно: каждый разработчик занимается своим модулем. Такая организация, а также возможность повторного использования модулей из других программ позволяют существенно сокращать сроки изготовления программы.

Структура модуля

Модуль имеет следующую структуру:
- Заголовок модуля

```
unit ⟨имя⟩
```

- Интерфейс модуля

```
interface
  ⟨интерфейсная часть⟩
```

- Исполнительная часть (реализация)

```
implementation
  ⟨исполнительная часть⟩
```

- Секция инициализации

```
begin
  ⟨инициализирующая часть⟩
```

- Признак конца модуля

```
end.
```

Все разделы модуля, кроме секции инициализации, являются обязательными и должны располагаться в указанном порядке. В качестве имени модуля используется индентификатор. Имя должно быть уникальным (отличаться от других), например:

`unit` Ratio; {арифметические действия с рациональными числами}

Модуль записывается в файл с расширением .pas, имя которого совпадает с именем модуля: ratio**.**pas.

Интерфейсная часть

В интерфейсной части объявляются (описываются) константы, типы, переменные, процедуры и функции, которые могут быть использованы основной программой (или другим модулем). Правила описания такие же, как для раздела описаний основной программы.

При объявлении процедур и функций в интерфейсной части указывается только их заголовок, например:

```
unit Ratio;
interface
 type
   rational= record
                numerator: integer;
              denominator: 1..Maxint;
             end;

procedure AddR(x,y: rational; var z: rational);
 {сложение z:=x+y}

procedure MulR(x,y: rational; var z: rational);
 {умножение z:=x*y}
```

Сущности, описанные в интерфейсной части, могут использоваться и в других частях модуля.

Исполнительная часть

Исполнительная часть содержит реализацию (описание) подпрограмм, объявленных в интерфейсной части.

```
implementaion

 procedure AddR(x,y: rational; var z: rational);
 {сложение z:=x+y}
 begin
   z.numenator:=x.numerator*y.denominator+
                y.numerator*x.denominator;
   z.denominator:= x.denominator * y.denominator
 end;

 procedure MulR(x,y: rational; var z: rational);
 {умножение z:=x*y}
 begin
    z.numenator:=x.numerator * y.numerator;
   z.denominator:= x.denominator * y.denominator
 end;
```

В исполнительной части также могут описываться вспомогательные типы, константы, переменные, процедуры и функции, недоступные интерфейсной части и другим модулям. Например, мы можем добавить в исполнительную часть модуля `Ratio` процедуру `simplify(x,y)`, которая приводит дробь `x/y` к несократимому виду, т.е. делит x и y на их наибольший общий делитель, и использовать эту процедуру для реализации операций над рациональными числами, чтобы уменьшить вероятность переполнения (превышение `Maxint`).

Секция инициализации

В секции инициализации находится последовательность операторов. Эта последовательность выполняется один раз, в момент запуска программы, до передачи управления основной программе. В этой секции обычно указываются подготовительные действия (инициализация переменных, открытие нужных файлов, установление связи по коммуникационным каналам и т.п.), например:

```
begin
  assign(f1, 'myfile.dat');
  rewrite(f);
{конец секции инициализации}

end.
```

Если никаких начальных действий не требуется, секция инициализации опускается.

Использование модуля

Чтобы использовать подпрограммы, типы, константы, переменные, описанные в интерфейсной части модуля, в основной программе перед разделом описаний следует указать раздел объявления используемых модулей. Он состоит из одного предложения, начинающегося служебным словом `uses`, после которого через запятую указываются имена модулей.

Программе становятся доступны имена, описанные в интерфейсах перечисленных модулей, например:

```
program Myprog;
uses Raitio;
var z1,z2, z3: Rational;
begin
…
 AddR(z1,z2,z3);
…
end.
```

Модуль может использоваться не только в основной программе, но и в других модулях. Можно указывать раздел объявления используемых модулей перед разделом описаний в интерфейсной и/или исполнительной части модуля.
Важно следить за тем, чтобы модули, указываемые в интерфейсной части, не имели циклических ссылок друг на друга, например: A использует B, B использует C, C использует A. Иначе компилятор не сможет установить нужные связи. Если модуль объявлен в интерфейсной части, то в исполнительной части его указывать не следует. Чаще всего используемые модули объявляются в исполнительной части.

Компиляция модулей

Результат компиляции модуля помещается в одноименный файл с расширением `.tpu` (Turbo Pascal Unit). Чтобы компилировать программу, использующую модуль Ratio.pas, файл Ratio.tpu должен существовать к моменту компиляции. Фри Паскаль кроме основного режима компиляции COMPILE, предоставляет еще два режима, удобные для компиляции многомодульных программ:

1) BUILD. В этом режиме существующие tpu-файлы игнорируются. Для каждого объявленного в предложении `uses` модуля ищется соответствующий pas-файл и заново создается tpu-файл.
2) MAKE. В отличие от режима BUILD, перекомпиляция модуля происходит, только если tpu-файл отсутствует или если вносились изменения в соответствующий pas-файл.

Стандартные модули

С помощью модулей во Фри Паскале организованы библиотеки стандартных подпрограмм и данных. Имеется следующие стандартные модули: System, Crt, Graph и другие. Модуль Graph выделен в отдельный tpu-файл, System и Crt входят в состав библиотечного файла turbo.tpl.
Модуль System автоматически подключается к любой программе, все остальные модули надо указывать в списке после ключевого слова `uses`. В модуль System входят все процедуры и функции стандартного Паскаля (например, trunc) и другие встроенные процедуры и функции (inc, dec, getdir) и др.
В модуле Crt сосредоточены процедуры и функции, обеспечивающие управление текстовым режимом работы экрана (так называемая работа с консолью: перемещение курсора в произвольную позицию, изменение цвета выводимых символов и окружающего их фона, создание окна, чтение с клавиатуры без «эха» и управление звуком).
Модуль Graph содержит обширные набор типов, констант, процедур и функций для управления графическим режимом работы экрана.