Объектное программирование. Перегрузка операторов

(пособие В.А. Клячин "Автоматизированы методы обработки текстовой информации")

1.2.9 Перегрузка операторов

Перегрузка операторов определяет способ обработки результата операции, если в качестве операндов присутствуют объекты некоторых классов, не являющиеся базовыми типами. Для перегрузки операторов применяется прагма

```
use overload;
```

Например,

```
package Paket;

use overload '+'=>\&add,
             '>'=>"more",
             'abs'=>sub{return @_};
```

Здесь в первом случае указана ссылка на процедуру, которая будет вызвана, если попытаться сложить два объекта класса Paket. Такие процедуры мы будем называть обработчиками. Для бинарных операторов обработчик вызывается, если первый операнд или второй операнд являются объектами класса, если для первого не задан обработчик. Так что можно написать:

```
$ob+12;
```

или

```
12+$ob;
```

Для объектов разных классов вызывается обработчик левого операнда.

При выполнении перегрузки соответствующий обработчик получает три аргумента. Первые два – его операнды. Для унарных операторов – первый операнд, второй undef – неопределен.

Третий параметр указывает на то, могут ли быть переставлены аргументы местами. Если они могут меняться местами, то третий параметр имеет значение true. Приведем пример класса с перегрузкой операторов:

```
package ClipByte;

use overload '+'=> \&clip_add,
             '-'=>\&clip_sub;

sub myprint { my $class = shift;

my $v=$$class; print $v => "\n"; }

sub new {

my $class = shift;

my $value = shift;

return bless \$value => $class;

 }

sub clip_add
 {
```

```
my ($x,$y)=@_;
my ($value)=ref($x)?$$x:$x;
$value += ref($y)?$$y:$y;
$value= 255 if $value > 255;
$value =0 if $value <0;
return bless \$value => ref($x);
 }
sub clip_add
 {
my ($x,$y)=@_;
my ($value)=ref($x)?$$x:$x;
$value -= ref($y)?$$y:$y;
$value= 255 if $value > 255;
$value =0 if $value <0;
return bless \$value => ref($x);
 }
package main;
$byte1=ClipByte->new(200);
$byte2=ClipByte->new(100);
$byte3=$byte1+$byte2;
$byte4=$byte1-$byte2;
$byte5=150-$byte2;
$byte3->myprint;
 $byte4->myprint;
```

Приведем список перегружаемых операторов в Perl:

```
Арифметические +, -, *, /, %, **
Логические !
Поразрядные & | << >>
Присваивания += -= /= %= **= ++ --
Сравнения < > == <= >= != lt le gt ge eq  ne cmp
Математические atan2 cos sin exp log sqrt
```

Наконец, приведем пример перегрузки преобразования в строку:

```
package Mans;

use overload q("")=> \&as_string;

sub new {

my $class = shift;

return bless { @_ } => $class;

 }

 sub as_string
 {

my $self = shift; my ($key,$value,$result);
while(($key,$value)= each %$self)
{ $result.="$key => $value\n"; } return $result;
 }

package main;

$obj=Mans->new("user1"=>"login1",

"user2"=>"login2","user3"=>"login3");

print "$obj";
```

В данном примере вместо ожидаемого вывода вроде HASH(0x23654) мы увидим

```
user1 => login1

user2 => login2

user3 => login3
```

1.2.10 Генерация HTML-таблиц

Здесь мы рассмотрим пример, в котором по значениям в ячейках таблицы автоматически генерируется HTML-код этой таблицы. Мы реализуем соответствующий класс Table.

```
package Table;

sub create{ -- процедура создания таблицы
my ($myself, $ref)=@_;
$ref=[];

bless $ref, $myself;
}
#Вывод таблицы на экран
sub print
{
my $this=shift;

for (@{$this})
{
if(defined $_){
for my $name (keys %{$_})
{
print $_->{$name}, "\t";
}
print "\n";
}

}
}
```

```
#Загрузка данных в таблицу из массива
sub load
{
my $self=shift;
my $table=shift;


for(@{$table})
{
push @{$self},$_;
}

}
# добавить строку
sub add_row
{

my $self=shift;
my $row=shift;

push @{$self},$row;
return;

}
#Выбрать строку таблицы
sub get_row
{
my $self=shift;
my $num=shift;
return $self->[$num];

}
#Удалить строку таблицы
sub del_row
{
my $self=shift;
my $num=shift;
$self->[$num]=undef;
return;
}
# Перевести таблицу в HTML-формат
sub html
{

my $self=shift;
my $border=shift;
my $str="<table border=$border>";

for my $row (@{$self})
{
$str.="<tr>";
for(keys %{$row})
{
$str.="<td>";
$str.=$row->{$_};
$str.="</td>\n";
}
$str.="</tr>";
}
$str.="<table>";

return $str;

}
```

```
# Упорядочить столбцы
sub ordered_html
{

my $self=shift;
my $border=shift;
my $order=shift;
my $str="<table border=$border>";

for my $row (@{$self})
{
$str.="<tr>";
for(@{$order})
{
$str.="<td>";
$str.=$row->{$_};
$str.="</td>";
}
$str.="</tr>\n";
}
$str.="<table>";

return $str;

}
# Сохранить таблицу в файл
sub save_html
{
my $self=shift;
my $border=shift;

$border=1 unless $border;
my $order=shift;
$order=[keys %{$self->[0]}] unless $order;
my $file=shift; $file="index.html" unless $file;
open(OUT,">",$file) or die;
print OUT $self->ordered_html($border,$order);
close OUT;
return;
}

1;
```

Таким образом, теперь задача автоматической генерации HTML-таблицы решается весьма просто:

```
use Table;
#Данные таблицы мы представляем в виде массива хешей
$table=[
{name=>Ivanov, email=>'iv@as.ru',region=>122},
{name=>Petrov, email=>'pet@rasd.su',region=>121},
{name=>Sergeev, email=>'serge@mail.ru',region=>167}
];

$new_row={name=>Sidorov, email=>'sidr@mail.ru',region=>163};

$t=Table->create(); -- создаем объект таблицы
$t->load($table); загружаем в него нашу таблицу
$t->add_row($new_row); -- добавляем строку
$t->print; -- печатаем
print "-------\n";
$t->del_row(1); -- удаляем строку
$t->print; -- печатаем


@order=(name,email,region); -- задаем порядок
                               следования столбцов
```

```
print $t->ordered_html(1,\@order); создаем html-таблицу

$t->save_html(' 0',\@order,"save.html"); -- сохраняем
                                            ее в файл.
```

1.2.15 Поддержка кодировок

В этом разделе мы рассмотрим модуль Encode для работы с русскоязычным текстом. Русские буквы не входят в первую половину таблицы ASCII символов и поэтому в разных кодовых таблицах имеют разный цифровой код. Так, если слово *хлеб* представить 16-ричными кодами в кодировке windows cp-1251, то это будет

```
0xf5, 0xeb, 0xe5, 0xe1.
```

То же слово в кодировке DOS cp-866 выглядит по-другому:

```
0xe5, 0xab, 0xa5, 0xa1.
```

Теперь представим, что текст в некотором файле на русском языке закодирован в кодировке cp-1251, а мы его попытаемся открыть редактором в кодировке cp-866. Например, если текст содержит слово *хлеб*, то в редакторе оно будет выглядеть так:

```
їыхс
```

Указанный выше модуль позволяет программно изменить кодировку текста. Рассмотрим пример. Пусть файл text.txt будет такого содержания:

```
Самое дорогое у человека --  это жизнь.
Она дается ему один раз, и прожить ее надо
так, чтобы не было мучительно больно за бесцельно
прожитые годы ...
```

Теперь осуществим вывод этого файла на консоль DOS, использующее кодировку cp-866. Получим

```
рьюх фюЁююух є ўхыютхър Ц ⌧Єю цшчэ№. эр фрхЄё  хьє юфшэ Ёрч,
ш яЁююшЄ№ хх эрфю
Єръ, ўЄюс√  √      √ √, ...
```

Исправим ситуацию. Для этого воспользуемся модулем Encode:

```
use Encode;

while(<>)
{
Encode::from_to($_,'cp1251','cp866');
print;
}
```

Аналогично поступают и с другими таблицами кодов (koi8,utf8, и т. п.).