Bit::Vector

1. CLASS METHODS

```
Version
    $version = Bit::Vector->Version();

Word_Bits
    $bits = Bit::Vector->Word_Bits();  #  bits in a machine word

Long_Bits
    $bits = Bit::Vector->Long_Bits();  #  bits in an unsigned long

new
    $vector = Bit::Vector->new($bits);  #  bit vector constructor

    @veclist = Bit::Vector->new($bits,$count);

new_Hex
    $vector = Bit::Vector->new_Hex($bits,$string);

new_Bin
    $vector = Bit::Vector->new_Bin($bits,$string);

new_Dec
    $vector = Bit::Vector->new_Dec($bits,$string);

new_Enum
    $vector = Bit::Vector->new_Enum($bits,$string);

Concat_List
    $vector = Bit::Vector->Concat_List(@vectors);
```

2. OBJECT METHODS

```
new
    $vec2 = $vec1->new($bits);  #  alternative call of constructor

    @veclist = $vec->new($bits,$count);

Shadow
    $vec2 = $vec1->Shadow();  #  new vector, same size but empty

Clone
    $vec2 = $vec1->Clone();  #  new vector, exact duplicate
```

```
Concat
    $vector = $vec1->Concat($vec2);

Concat_List
    $vector = $vec1->Concat_List($vec2,$vec3,...);

Size
    $bits = $vector->Size();

Resize
    $vector->Resize($bits);
    $vector->Resize($vector->Size()+5);
    $vector->Resize($vector->Size()-5);

Copy
    $vec2->Copy($vec1);

Empty
    $vector->Empty();

Fill
    $vector->Fill();

Flip
    $vector->Flip();

Primes
    $vector->Primes();  #  Sieve of Erathostenes

Reverse
    $vec2->Reverse($vec1);

Interval_Empty
    $vector->Interval_Empty($min,$max);

Interval_Fill
    $vector->Interval_Fill($min,$max);

Interval_Flip
    $vector->Interval_Flip($min,$max);

Interval_Reverse
    $vector->Interval_Reverse($min,$max);

Interval_Scan_inc
    if (($min,$max) = $vector->Interval_Scan_inc($start))

Interval_Scan_dec
    if (($min,$max) = $vector->Interval_Scan_dec($start))

Interval_Copy
```

```
    $vec2->Interval_Copy($vec1,$offset2,$offset1,$length);

Interval_Substitute
    $vec2->Interval_Substitute($vec1,$off2,$len2,$off1,$len1);

is_empty
    if ($vector->is_empty())

is_full
    if ($vector->is_full())

equal
    if ($vec1->equal($vec2))

Lexicompare (unsigned)
    if ($vec1->Lexicompare($vec2) == 0)
    if ($vec1->Lexicompare($vec2) != 0)
    if ($vec1->Lexicompare($vec2) <  0)
    if ($vec1->Lexicompare($vec2) <= 0)
    if ($vec1->Lexicompare($vec2) >  0)
    if ($vec1->Lexicompare($vec2) >= 0)

Compare (signed)
    if ($vec1->Compare($vec2) == 0)
    if ($vec1->Compare($vec2) != 0)
    if ($vec1->Compare($vec2) <  0)
    if ($vec1->Compare($vec2) <= 0)
    if ($vec1->Compare($vec2) >  0)
    if ($vec1->Compare($vec2) >= 0)

to_Hex
    $string = $vector->to_Hex();

from_Hex
    $vector->from_Hex($string);

to_Bin
    $string = $vector->to_Bin();

from_Bin
    $vector->from_Bin($string);

to_Dec
    $string = $vector->to_Dec();

from_Dec
    $vector->from_Dec($string);

to_Enum
    $string = $vector->to_Enum();  #  e.g. "2,3,5-7,11,13-19"
```

```
from_Enum
    $vector->from_Enum($string);

Bit_Off
    $vector->Bit_Off($index);

Bit_On
    $vector->Bit_On($index);

bit_flip
    $bit = $vector->bit_flip($index);

bit_test
contains
    $bit = $vector->bit_test($index);
    $bit = $vector->contains($index);
    if ($vector->bit_test($index))
    if ($vector->contains($index))

Bit_Copy
    $vector->Bit_Copy($index,$bit);

LSB (least significant bit)
    $vector->LSB($bit);

MSB (most significant bit)
    $vector->MSB($bit);

lsb (least significant bit)
    $bit = $vector->lsb();

msb (most significant bit)
    $bit = $vector->msb();

rotate_left
    $carry = $vector->rotate_left();

rotate_right
    $carry = $vector->rotate_right();

shift_left
    $carry = $vector->shift_left($carry);

shift_right
    $carry = $vector->shift_right($carry);

Move_Left
    $vector->Move_Left($bits);  #  shift left "$bits" positions

Move_Right
    $vector->Move_Right($bits);  #  shift right "$bits" positions
```

```
Insert
    $vector->Insert($offset,$bits);

Delete
    $vector->Delete($offset,$bits);

increment
    $carry = $vector->increment();

decrement
    $carry = $vector->decrement();

inc
    $overflow = $vec2->inc($vec1);

dec
    $overflow = $vec2->dec($vec1);

add
    $carry = $vec3->add($vec1,$vec2,$carry);
    ($carry,$overflow) = $vec3->add($vec1,$vec2,$carry);

subtract
    $carry = $vec3->subtract($vec1,$vec2,$carry);
    ($carry,$overflow) = $vec3->subtract($vec1,$vec2,$carry);

Neg
Negate
    $vec2->Neg($vec1);
    $vec2->Negate($vec1);

Abs
Absolute
    $vec2->Abs($vec1);
    $vec2->Absolute($vec1);

Sign
    if ($vector->Sign() == 0)
    if ($vector->Sign() != 0)
    if ($vector->Sign() <  0)
    if ($vector->Sign() <= 0)
    if ($vector->Sign() >  0)
    if ($vector->Sign() >= 0)

Multiply
    $vec3->Multiply($vec1,$vec2);

Divide
    $quot->Divide($vec1,$vec2,$rest);

GCD (Greatest Common Divisor)
```

```
    $vecgcd->GCD($veca,$vecb);
    $vecgcd->GCD($vecx,$vecy,$veca,$vecb);

Power
    $vec3->Power($vec1,$vec2);

Block_Store
    $vector->Block_Store($buffer);

Block_Read
    $buffer = $vector->Block_Read();

Word_Size
    $size = $vector->Word_Size();  #  number of words in "$vector"

Word_Store
    $vector->Word_Store($offset,$word);

Word_Read
    $word = $vector->Word_Read($offset);

Word_List_Store
    $vector->Word_List_Store(@words);

Word_List_Read
    @words = $vector->Word_List_Read();

Word_Insert
    $vector->Word_Insert($offset,$count);

Word_Delete
    $vector->Word_Delete($offset,$count);

Chunk_Store
    $vector->Chunk_Store($chunksize,$offset,$chunk);

Chunk_Read
    $chunk = $vector->Chunk_Read($chunksize,$offset);

Chunk_List_Store
    $vector->Chunk_List_Store($chunksize,@chunks);

Chunk_List_Read
    @chunks = $vector->Chunk_List_Read($chunksize);

Index_List_Remove
    $vector->Index_List_Remove(@indices);

Index_List_Store
    $vector->Index_List_Store(@indices);
```

```
Index_List_Read
    @indices = $vector->Index_List_Read();

Or
Union
    $vec3->Or($vec1,$vec2);
    $set3->Union($set1,$set2);

And
Intersection
    $vec3->And($vec1,$vec2);
    $set3->Intersection($set1,$set2);

AndNot
Difference
    $vec3->AndNot($vec1,$vec2);
    $set3->Difference($set1,$set2);

Xor
ExclusiveOr
    $vec3->Xor($vec1,$vec2);
    $set3->ExclusiveOr($set1,$set2);

Not
Complement
    $vec2->Not($vec1);
    $set2->Complement($set1);

subset
    if ($set1->subset($set2))  #  true if $set1 is subset of $set2

Norm
    $norm = $set->Norm();
    $norm = $set->Norm2();
    $norm = $set->Norm3();

Min
    $min = $set->Min();

Max
    $max = $set->Max();

Multiplication
    $matrix3->Multiplication($rows3,$cols3,
                    $matrix1,$rows1,$cols1,
                    $matrix2,$rows2,$cols2);

Product
    $matrix3->Product($rows3,$cols3,
              $matrix1,$rows1,$cols1,
              $matrix2,$rows2,$cols2);
```

```
Closure
    $matrix->Closure($rows,$cols);

Transpose
    $matrix2->Transpose($rows2,$cols2,$matrix1,$rows1,$cols1);
```

3. 使用举例

代码：

```perl
#!/usr/bin/perl

use Bit::Vector;

$vector = Bit::Vector->new_Hex(12,"cc");

$bin_str = $vector->to_Bin();

print "bin_str = $bin_str\n";

$hex_str = $vector->to_Hex();

print "hex_str = $hex_str\n";
```

运行结果：

```
bin_str = 000011001100
hex_str = 0CC
```

3.1 获取指定bit value

contains 与 bit_test功能相同

```perl
foreach $i (reverse(0..11)) {
    #$bit_value = $vector->contains($i);
    $bit_value = $vector->bit_test($i);
    print "$bit_value ";
}
print "\n";
```

注：不支持一次性获取多个bit，比如想获取 bit 7:0的值，那么需要分别获取bit7 ~ bit0的值，然后再拼凑到一起。作进制转换，创建一个8bit宽的bit vector, 先from_Bin然后再to_Hex，即获取到bit 7:0的hex值。

3.2 给指定bit赋值

```
1. $vector->Bit_Copy(0,1);
```

```
2. $vector->Bit_Copy(1,1);
3. $vector->Bit_Copy(8,1);
4. $hex_str = $vector->to_Hex();
5. print "$hex_str\n";
```

3.3 整个bit vector 取反

执行后整个bit vector序列全部取反

```
1. $vector->Flip();
2. $hex_str = $vector->to_Hex();
3. print "$hex_str\n";
```

3.4 指定bit取反

bit vector 指定bit取反，并且返回该bit取反后的值。

```
$bit = $vector->bit_flip(0);
print "bit = $bit\n";
$hex_str = $vector->to_Hex();
print "$hex_str\n";
```