```perl
#!/usr/bin/perl -w
#用perl语言按照文件大小列出当前目录中的文件
use strict;
print "This uses Hashtable\n";
Hashetable();
print "This dose not use Hashtable, use -s to get the size\n";
Nohashtable();

sub Hashetable{my @list = grep {-f $_} <*>;
#读取目录文件

my %f;
#定义哈希table

foreach my $file(@list){
    my $s=(stat($file))[7];
    $f{$file}=$s;
}
#遍历list中的文件，读取其大小并存储
my @k = sort { $f{$a} <=> $f{$b} || $a cmp $b } keys %f;
#根据value的大小比较并排序key,将key存在@k数组中

foreach my $file(@k){
    my $size=$f{$file};
    print $size,"\t",$file,"\n";
}
print "\n";
}
sub Nohashtable{
    my @files = grep {-f $_} <*>;
#  对文件按大小排序

@files = sort { -s $a <=> -s $b || $a cmp $b} @files;

#  输出文件名和大小
#因为-s操作可以直接得到文件的大小，所以实际上并不需要两个循环
foreach my $file (@files) {
    my $size = -s $file;
    printf "%s\t%s \n",$size,$file;
}
}
```

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

#统计学号邮箱里邮件的发件方有效email地址，按邮件多少
#排序，邮件数量相同的，按email地址逆排序，学号-03.pl

use strict;
use Mail::POP3Client;
use YAML qw(LoadFile);


my ($user, $pass) = LoadFile 'secret.txt';
my $pop = new Mail::POP3Client(
    HOST => 'mail.fudan.edu.cn',
    USER => $user,
    PASSWORD=> $pass,
    USESSL => 1,
    AUTH_MODE => 'PASS',
);

my %senders;

my $cnt = $pop->Count();

print "Found $cnt emails.\n";
# 打-1是用户密码错

for(my $i = 1; $i <= $cnt; $i++ ) {
# $i 为当前循环到的邮件编号，从1开始，一直到 $cnt 结束
    my $from;
    foreach($pop -> Head( $i )){
        chomp;
        if(/^From:\s+(.*)/i){
            $from = $1;
        }
    # 返回第 $i 封邮件的头部信息，该信息是一个字符串数组
    # ^From: 匹配字符串开头的 From:，表示该行文本必须以 From: 开头。
    # \s+ 匹配至少一个空白字符，这里用于匹配 From: 与邮件地址之间的空格。
    # (.*) 使用括号捕获了一个分组，表示匹配任意多个字符，
    # 这里的字符是除开头的单词（From:）和开头的空格之外的所有字符，该分组使用
* 表示匹配零个或多个字符，即匹配整个邮件地址及其后面可能包含的其他信息。
    # /i 表示正则表达式的模式修饰符，这里使用 i 修饰符表示匹配时忽略大小写。
    # and $from = $1; 如果正则表达式匹配成功，
    # 则将捕获到的文本存储在 $1 变量中，然后将其赋值给变量 $from。
    }
    if ($from =~ /<(.*)>/) {
    # 从\<匹配<，([^>]+) 使用括号捕获了一个分组，
    # 表示匹配任意多个非 > 字符，也就是尖括号内的文本内容，
    # 使用 + 表示至少匹配一个字符
    # \> 匹配 > 字符
    # 使用正则表达式匹配 $from 中的邮件地址
    my $email = $1;
    # 并将地址存储在 $email 变量中。
    $senders{$email}++;
    }
    else {
        # $from = ~s/^ +//;
        $senders{$from}++;
    }
}
my @keys = sort{ $senders{$b} <=> $senders{$a} || $b cmp $a }
keys %senders;
my $sum;

foreach my $key (@keys){
    my $t = $senders{$key};
    $sum += $t;
    printf("%-40s %d\n", $key, $t);
}
print "There are $sum mails in your mailbox";
1;


```

```perl
#!/usr/bin/perl -w
use strict;
use Chart::Lines;

my @x_values;
my @y_values;
my $flag = 0;
my $min_val = 0;
my $max_val = 0;
while (<>) {
  chomp;
  if (/^={3}\s=+$/) {

    $flag = 1;
    next;
  # 匹配3个=号一个空格还有至少第一个等号，匹配到就是到了数据行了
  # 直接快进到下一行
  # 这个时候把flag置为1
  }

  if ($flag) {
    my ($x, $y1, $y2) = split;
    # 如果数据有定义，那就push y 和与之对应的 x
    if(defined($y1)){
      push @y_values, $y1;
      push @x_values, $x;
    }
    if(defined($y2)){
      push @y_values, $y2;
      push @x_values, $x + 1;
    }

    # 取数据装进数组中
    if ($y1 < $min_val) {
      $min_val = $y1;
    }
    if ($y1 > $max_val) {
      $max_val = $y1;
    }
      if ($y2 < $min_val) {
      $min_val = $y2;
    }
    if ($y2 > $max_val) {
      $max_val = $y2;
    }
    # 为了方便表示取了最大最小值
  }
}
# 画图
my $chart = Chart::Lines->new(800, 600);
$chart->add_dataset(@x_values);
$chart->add_dataset(@y_values);
$chart->set('title' => 'Coefficient Curve',
            'legend' => 'bottom',
            'x_label' => 'X',
            'y_label' => 'Y',
            'min_val' => $min_val-($max_val-$min_val)/20,
            'max_val' => $max_val+($max_val-$min_val)/20,
            'skip_x_ticks' => 10);
$chart->png('20307130176.png');
1;
```

```
#!/usr/bin/perl
use strict;
use Time::HiRes qw(gettimeofday tv_interval);

=pod
1. 级数展开sin+cos|x(|x|<p)
2. 编写子程序sub tri，参数是x,n,返回(sin+cos)(x)的值，
级数展开到多项式的n次项
3. 主程序调用tri(2,$n)，其中$n=(1..10)，运行结果如下:
=cut

# 实际上每次进入循环都是上一个计算的结果乘一个算子，没必要算多次次方
# 可以进行优化
my ( $sin1, $cos1, $sin2, $cos2 );
$sin1 = sub {
    my ( $x, $n ) = @_;
    my $factor = -1;
    my $sum    = 0;
    for ( my $i = 1 ; $i <= $n ; $i += 2 ) {
        my $bot = ( $i * ( $i - 1 ) );
        $factor = -$factor / ( $bot == 0 ? 1 : $bot );
        $sum += $factor * $x**$i;

        # 参考多项式展开公式进行生成结果
    }
    $sum;
};
$sin2 = sub {
    my ( $x, $n ) = @_;
    my $sum  = 0;
    my $temp = -1 / $x;

    # 设置临时变量$temp，每次迭代时只是在$temp的基础上乘一个因子
    for ( my $i = 1 ; $i <= $n ; $i += 2 ) {
        my $bot = ( $i * ( $i - 1 ) );
        $temp = -$temp * $x * $x / ( $bot == 0 ? 1 : $bot );
```

```perl
        $sum += $temp;
    }
    $sum;
};
$cos1 = sub {
    my ( $x, $n ) = @_;
    my $factor = -1;
    my $sum    = 0;
    for ( my $i = 0 ; $i <= $n ; $i += 2 ) {
        my $bot = ( $i * ( $i - 1 ) );
        $factor = -$factor / ( $bot == 0 ? 1 : $bot );
        $sum += $factor * $x**$i;
    }
    $sum;
};
$cos2 = sub {
    my ( $x, $n ) = @_;
    my $sum  = 0;
    my $temp = -1 / ( $x * $x );

    # 设置临时变量$temp，每次迭代时只是在$temp的基础上乘一个因子
    for ( my $i = 0 ; $i <= $n ; $i += 2 ) {
        my $bot = ( $i * ( $i - 1 ) );
        $temp = -$temp * $x * $x / ( $bot == 0 ? 1 : $bot );
        $sum += $temp;
    }
    $sum;
};

sub tri1 {
    my ( $x, $n ) = @_;
    foreach (@{$n}) {
        print "$_\t", $sin1->( $x, $_ ) + $cos1->( $x, $_ ),
"\n";
    }
}

sub tri2 {
    my ( $x, $n ) = @_;
    foreach (@{$n}) {
        print "$_\t", $sin2->( $x, $_ ) + $cos2->( $x, $_ ),
"\n";
    }
```

```perl
}

my $n = [ 0 .. 10 ];
my $x = 2;

# 未优化
print "未进行优化：" . "\n";

# 记录开始时间
my $start_time = [gettimeofday];

tri1( $x, $n );

# 记录结束时间
my $end_time = [gettimeofday];

# 计算代码运行时间
my $elapsed_time = tv_interval( $start_time, $end_time );

# 输出结果
print "代码运行时间为 $elapsed_time 秒\n";

print "进行了简单的优化：" . "\n";

# 进行了简单的优化
# 记录开始时间
my $start_time = [gettimeofday];

tri2( $x, $n );

# 记录结束时间
my $end_time = [gettimeofday];

# 计算代码运行时间
my $elapsed_time = tv_interval( $start_time, $end_time );

# 输出结果
print "代码运行时间为 $elapsed_time 秒\n";

# 实际检验发现并没有多少优化
1;

```

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

# 科学地拟合函数，用到了内积
use strict;
use lib '.';
use Tk;
use inner;
use Data::Dumper;

our ($n,$target);
eval {require "task.inc"}
    or die "Error on task.inc\n$@";

# 向量减
sub subtract {
    my ($q,$v)=@_;
    my @r;
    my $len = scalar @{$q} - 1;
    for my $i (0..$len){
        push @r,$q->[$i]-$v->[$i];
    }
    return \@r;
}

# 向量标量乘法
sub scale {
    my ($scale,$v)=@_;
    my @new;
    foreach (@{$v}){
        push @new,$scale*$_;
    }
    return \@new;
}

# 产生正交基
sub gramschmidt {
    my $n = shift;
    my @v;   # 原始向量产生
    my @e;   # 存储正交向量
    for my $i (0 .. $n-1) {
      $v[$i] = [map{0} 0..$n-1];
      $v[$i][$i] = 1;
        my $q = [@{$v[$i]}];
        for my $j (0 .. $i-1) {
 $q=subtract($q,scale(inner(poly($v[$i]),poly($e[$j]))/inner(poly($e[$j]),poly($e[$j])),$e[$j]));
        }
        my $coeffs=sqrt(1/inner(poly($q),poly($q)));
        $q = scale($coeffs,$q);
        push @e, $q;
    }
    return \@e;
}
sub project {
    # 向量投影
    my ($target, $e) = @_;
    my @coeffs;
    my $len = scalar(@{$e}) - 1;
    for my $i (0 .. $len) {
        my $inner = inner($target, poly($e->[$i]));
        my $norm_squared = inner(poly($e->[$i]), poly($e->[$i]));
        my $scale = $inner / $norm_squared;
        push @coeffs, $scale;
    }
    # 求出内积然后除以原向量的内积得到投影向量
    return \@coeffs;
}
sub transcoord{
    # 向量转换回原空间，提取投影的系数，把对应的向量的系数相加得到多项式系数
    my ($coeffs, $e) = @_;
    my $len = scalar(@{$e}) - 1;
    my @new_coeffs = (map {0} $len);
    for my $i (0 .. $len) {
        my $scaled_e = scale($coeffs->[$i], $e->[$i]);
        # 产生一个按比例的原空间中的向量
        for my $j (0 .. $len) {
            $new_coeffs[$j] += $scaled_e->[$j];
            # 原空间中的向量对应系数加到对应的维度上
        }
    }
    return \@new_coeffs;
}

print "Here! gramschmidt"."\n";
```

```perl
my $e = gramschmidt($n);
foreach (@{$e}){
    foreach (@{$_}){
        printf("%.4f\t",$_);
    }
    print "\n";
}
print "Here! project"."\n";
my $m = project($target, $e);
for (@{$m}){
    printf("%.4f\t",$_);
}
print "\n";
print "Here! transcoord"."\n";
my $w = transcoord($m, $e);
for (@{$w}){
    printf("%.4f\t",$_);
}
print "\n";
my $err = norm(va($target,kv(-1,poly($w))));
print "Here! norm of error_vector:"."\n";
print $err,"\n";
plot(target=>$target,poly=>poly($w));
1;
```

```perl
#!/usr/bin/perl -w

# 从< >读入数学表达式，单变量x，如log(sin(2*x))，+-
# */()指数**浮点数、函数sin cos log exp abs sqrt
# 用适当的方法过滤输入，发现非法输入则报错
# 用eval在区间[-10,10]求值表达式，步长0.01，并作图
# 无效区间不作图，如log(x)，在x<=0的区间没有曲线
# 如果函数在整个[-10,10]上都无效，则报错
# 递交学号-07.pl，结果文件存入学号-07.png

# 使用方式：在同文件夹下准备一个txt文件，在powershell中使用
# Get-Content .\function.txt | perl .\20307130176-07.pl

use strict;
use Chart::Plot;

use constant DATASTART => -10;
use constant DATAEND => 10;

my $expr = <>;
chomp $expr;

# 去除开头的非字符
$expr =~ s/^[^a-zA-Z0-9()]+//;
print '$expr is '.$expr."\n";
# 替换变量
$expr =~ s/(?<!\bexp)\bx\b/\$x/g;
# print $expr."\n";
# print "sub{my \$x = shift;$expr}"."\n";

# 子函数代码块引用的构建
my $func = eval "sub {my \$x = shift; $expr}";
# print \$expr."\n";

# 检查表达式是否合法
if(ref $func ne 'CODE'){
    print $@;
    die "Syntax error!\n";
}

# 定义自变量和绘图的区间范围
my @x_all = map $_/100, 100*DATASTART..100*DATAEND;
my @x = ();
my @y = ();
# 计算函数值

# 图定义
my $fig = Chart::Plot->new(1000,700);
my $valid = 0;
foreach my $x_val (@x_all) {
    # 通过eval选取有效区间
    eval{
        my $y_val = $func->($x_val);
        push @y, $y_val;
        push @x, $x_val;
        };
        # 在这里添加数据集,
        # 到结尾或者到非连续区间，就会考虑在这个时候
        # 中断并先更新作图的数据集
    if($@ or $x_val == DATAEND){
        # 数据集有效
        if (@x != 0){
            # 数据集有效标识
            $valid = 1;
            $fig->setData([@x], [@y],'Blue Dashedline 
NoPoints');
            # 清空数据集
            @x=();
            @y=();
        }
    }
}

# 如果没有有效值就报错
if(!$valid){
    die "NO Valid number!\n";
}

# 绘制函数图
$fig->setGraphOptions('title' => '20307130176-07',);
open F, '>20307130176-07.png' or die;
binmode F;
print F $fig->draw('png');
close F;
1;

```

```
#!/usr/bin/perl -w
use strict;
use Data::Dumper;
use GraphViz;
use Tk;
use Tk::GraphViz;

# 每次读入一个整数表达式(只包含加法、乘
# 法、括号和整数，可含空格，不考虑单目
# 加)。先将表达式转化成树，树用递归方式
# 表示，每个节点表示成[op, node1, node2,
# node3…]，op可以是+、*，node可以是整
# 数或另一个节点。用Data::Dumper打印树，
# 用GraphViz结合Tk::GraphViz弹出窗体画
# 出多叉树，最后历遍树求表达式的值。

```

```perl

# 55 + (6 + 7 + 8*99 + 10)*12 + 23
# 由于对K叉树不熟，这里选择都用二叉树来实现
# 这个函数获得下一个token并返回当前位置的下标
sub getNextToken {
    my ($expression, $start_index) = @_;
    my $i = $start_index;
    # 超出表达式长度直接返回未定义
    if ($i >= length($expression)) {
        return (undef, $i);
    }
    # 从位置$i开始读一个字符，如果是符号就返回
    my $char = substr($expression, $i, 1);
    if ($char =~ /[\+\*\(\)]/) {
        return ($char, $i + 1);
    }
    # 从位置$i开始读字符，如果是数字就把数字读完并返回数字
    my $num_start = $i;
    while ($i < length($expression) && substr($expression, $i,
1) =~ /\d/) {
        $i++;
    }
    my $num = substr($expression, $num_start, $i - $num_start);
    return ($num, $i);
}

sub parseExpression {
    # 表达式以及初始下标
    my ($expression, $start_index) = @_;
    # 下标初始化
    $start_index //= 0;

    # 建立一个栈
    my @stack;
    # 获取一个token
    my ($token, $next_index) = getNextToken($expression,
$start_index);

    # token存在，分情况讨论
    while (defined($token)) {
        if ($token eq '(') {
            # 读到(，需要建立一个新的堆栈
            my ($sub_expression, $new_index) = parseExpression($expression, $next_index);
            push @stack, $sub_expression;
            $next_index = $new_index;
        } elsif ($token eq ')') {
            # 读到)，直接结束，这样就会回到(所在的地方结束堆栈
            last;
        } elsif ($token =~ /[\+\*]/) {
            # 读到+*就将符号加入
            push @stack, $token;
        } else {
            # 读到数字也加入堆栈
            push @stack, $token;
        }
        ($token, $next_index) = getNextToken($expression, $next_index);
    }
    # 堆栈建立完成之后，就是从下到上整理堆栈，因为乘号优先级高，所以先整理堆栈中所有的乘号
    my $i = 0;
    while ($i < scalar(@stack)) {
        if ($stack[$i] eq '*') {
            $stack[$i - 1] = ['*', $stack[$i - 1], splice(@stack, $i + 1, 1)];
            splice(@stack, $i, 1);
        } else {
            $i++;
        }
    }

    $i = 0;
    while ($i < scalar(@stack)) {
        if ($stack[$i] eq '+') {
            $stack[$i - 1] = ['+', $stack[$i - 1], splice(@stack, $i + 1, 1)];
            splice(@stack, $i, 1);
        } else {
            $i++;
        }
    }

    return $stack[0], $next_index;
}
```

```perl

my $expression = "99*12+0  +123+3*99+ 1";
$expression =~ s/\s+//g;
my ($tree, $index) = parseExpression($expression);
my $symbol;

$symbol = 'A';  # give each node a unique name
print Dumper $tree;
sub evaluateNode {
    my ($node) = @_;
    # 如果节点是一个数组的引用，就可以得到数组内的东西
    if (ref $node eq 'ARRAY') {
        my ($op, @value) = @$node;
        # 计算左右节点的值
        if ($op eq '+') {
            my $sum = 0;
            foreach(@value){
                $sum = evaluateNode($_) + $sum;
            }
            return $sum;
        } elsif ($op eq '*') {
            my $mul = 1;
            foreach(@value){
                $mul = evaluateNode($_) * $mul;
            }
            return $mul;
        }else {
            # 非法计算符
            die "Invalid operator: $op\n";
        }
    } else {
        return $node;
    }
}
print "\nFinal result=".evaluateNode($tree)."\n";

showTk(graph($tree));



sub showTk {
    my $g = shift;
    my $m = new MainWindow;
    my $gv = $m->GraphViz(-width => 600, -height => 600) ->
pack;
    $gv->show($g, fit=>0);
    MainLoop();
}

sub walk;
sub graph {
    my $tree = shift;
    my $g = GraphViz->new(width=>8, height=>8);
    $symbol = 'A';
    walk($tree, $g);
    open F, ">expr.png";
    binmode F;
    print F $g->as_png;
    close F;
    print "See also 'expr.png'.\n";
    return $g;
}

sub walk {
    my ($tree, $g) = @_;
    my $type = ref $tree eq 'ARRAY' ? $tree->[0] : 'I';
    my $res = $symbol;
    if ($type eq 'I') {
        $g->add_node($symbol++, label=>$tree);
        return $res;
    }

    my @list = @$tree;
    $g->add_node($symbol++, label => shift @list);
    $g->add_edge($res => walk($_, $g)) foreach @list;
    return $res;
}
1;
```

```perl
#!/usr/bin/perl -w
use strict;
use Time::HiRes qw(time sleep);
use Tk;
use Tk::PlotDataset;
use Tk::LineGraphDataset;
```

```perl
my $scriptname = $0;
my @num_iterations = (10..25);
my (@lib,@dss);
@lib = ('GMP','Calc','LTM','Pari');
foreach (@lib){
    my $lib = $_;
    my (@x,@y);
    foreach my $num_iterations(@num_iterations){
        my $start_time = time;
        system("perl 20307130176-09sqrt2.pl $num_iterations $lib");
        my $end_time = time;
        my $tspend=$end_time-$start_time;
        push @x,$num_iterations;
        push @y,$tspend;
        last if $tspend>4;
    }
    my $dataset = LineGraphDataset->new (
        -name => $lib,
        -xData => [@x],
        -yData => [@y],
        );
    push @dss, $dataset;
}
plot(\@dss);
sub plot{
    my $dss = shift;
    my $m = MainWindow->new;
    my $graph = $m->PlotDataset(
        -width => 800,
        -height => 500,
        )->pack;
    $graph->addDatasets(@{$dss});
    $graph->plot();
    MainLoop();
    1;
}
1;
```

```perl
#!/usr/bin/perl -w
use strict;
# my $num_iterations = 10;
# my $lib = "GMP";
# print $lib;
my ($num_iterations,$lib)=@ARGV;
myrequire($lib);
my $p = Math::BigInt->new('1');
my $q = Math::BigInt->new('1');
for (my $i = 0; $i < $num_iterations; $i++) {
    my $new_p = $p->bmul(2)->bmul($q);
    my $new_q = $q->bmuladd($q,$p->bmul($p));
    $p = $new_p;
    $q = $new_q;
}
1;

sub myrequire{
    my $lib = shift;
    require Math::BigInt;
    Math::BigInt->import(lib => $lib);
    # 在这里使用 GMP 库进行计算
};
```

```perl
#/usr/bin/perl -w
use strict;
use PDF::Create;

# 创建PDF文件
my $pdf = PDF::Create->new(
    'filename' => '20307130176.pdf',
    'Version'      => 1.2,
    'PageMode'     => 'UseOutlines',
    'Title'    => 'Math Exercises',
    'CreationDate' => [localtime],
);

# 添加一页
my $page = $pdf->new_page('MediaBox' => $pdf-
>get_page_size('A4'));
# 设置字体
my $font = $pdf->font('BaseFont' => 'Helvetica');

# 随机产生40道加减法题目
my @questions = ();
```

```perl
for (my $i = 1; $i <= 40; ) {
    my $num1 = int(rand(100)) + 1;
    my $num2 = int(rand(100)) + 1;
    my $operator = int(rand(2)) ? '+' : '-';
    my $answer = ($operator eq '+') ? ($num1 + $num2) : ($num1 - $num2);
    next if $answer < 0 or $answer >=100;
    push @questions, sprintf("%2d %s %-2d %2s %-2s", $num1, $operator, $num2, "=", "__");
    $i++;
}

# 在页面上添加题目
$page->string($font, 40, 175, 750,'Math Exercises');
my $x;
my $y = 700;
my $count = 1;
for (@questions){
    if($count == 5){
        $count = 1;
        $y = $y - 50;
    }
    $x = 100 * $count;
    $page->string($font, 14, $x, $y, $_);
    $count++;
}

# 保存PDF文件
$pdf->close();
1;
```

```perl
#!/usr/bin/perl -w
use strict;
use warnings;
use lib '.';
use Complex_20307130176;



my $m = Complex->new(1, 2);
my $n = Complex->new(3, 4);
my $c = $m + $n;
my $d = $m - $n;
my $e = $m * $n;
my $f = $m / $n;
my $g = conj($m);
my $h = abs($m);

print "$m + $n = $c\n";
print "$m - $n = $d\n";
print "$m * $n = $e\n";
print "$m / $n = $f\n";
print "conj($m) = $g\n";
print "abs($m) = $h\n";

$m->r(5);
$m->i(6);
print "a = $m\n";

my $x = Complex->new(2, 3);
my $y = Complex->new(2, -4);

print "$x == $y is ", $x == $y ? "true\n" : "false\n";

my @comp_arr = ($m, $n, $c, $d, $e, $f, $g, $x, $y);
print join("\n", map { abs($_) } @comp_arr), "\n";
my @sorted_comp_arr = sort { abs($a) <=> abs($b) } @comp_arr;
print "sorted by abs:\n";
print join("\n", map { "$_" } @sorted_comp_arr), "\n";
```

```perl
use strict;
use warnings;

# Writing by Zhiyu Zheng, May 2023.
# Define Complex Number and caculate.

sub conj {
    my ($self) = @_;
    return Complex->new($self->{r}, -$self->{i});
}

package Complex;

use overload
```

```perl
    '+' => \&add,
    '-' => \&subtract,
    '*' => \&multiply,
    '/' => \&divide,
    '""' => \&stringify,
    '=' => \&assign,
    '==' => \&equal,
    'abs' => \&abs;

sub new {
    my ($class, $real, $imaginary) = @_;
    my $self = bless { r => $real, i => $imaginary }, $class;
    return $self;
}

sub r {
    my ($self, $real) = @_;
    if (defined $real) {
        $self->{r} = $real;
    }
    return $self->{r};
}

sub i {
    my ($self, $imaginary) = @_;
    if (defined $imaginary) {
        $self->{i} = $imaginary;
    }
    return $self->{i};
}

sub abs {
    my ($self) = @_;
    return sqrt($self->{r}**2 + $self->{i}**2);
}

sub add {
    my ($self, $other) = @_;
    my $real = $self->{r} + $other->{r};
    my $imaginary = $self->{i} + $other->{i};
    return Complex->new($real, $imaginary);
}

```

```perl
sub subtract {
    my ($self, $other) = @_;
    my $real = $self->{r} - $other->{r};
    my $imaginary = $self->{i} - $other->{i};
    return Complex->new($real, $imaginary);
}

sub multiply {
    my ($self, $other) = @_;
    my $real = $self->{r} * $other->{r} -
               $self->{i} * $other->{i};
    my $imaginary = $self->{r} * $other->{i} +
                    $self->{i} * $other->{r};
    return Complex->new($real, $imaginary);
}

sub divide {
    my ($self, $other) = @_;
    my $denominator = $other->{r}**2 + $other->{i}**2;
    my $real = ($self->{r} * $other->{r} +
                $self->{i} * -$other->{i}) / $denominator;
    my $imaginary = ($self->{i} * $other->{i} -
                     $self->{r} * -$other->{r}) / $denominator;
    return Complex->new($real, $imaginary);
}

sub stringify {
    my ($self) = @_;
    return "($self->{r}, " . "$self->{i}j)";
}

sub assign {
    my $other = @_;
    my $real = $other->{r};
    my $imaginary = $other->{i};
    return Complex->new($real, $imaginary);
}

sub equal {
    my ($self, $other) = @_;
    return $self->{r} == $other->{r} &&
           $self->{i} == $other->{i};
}
```

```perl

1;
```

```perl
#!/usr/bin/perl -w
binmode( STDOUT, ":utf8" );
use strict;
use utf8;
use CGI qw/:standard/;
my @characters = qw/食 野 之 蒿 我 有 嘉 宾 德 音 孔 昭 视 民 不 恌/;
my @coding     = map { sprintf( "%04b", $_ ) } 0 .. 15;
my @line       = map { [] } 0 .. 3;
my %mapping    = map { $coding[$_] => $characters[$_] } 0 ..
$#coding;
my $CharacterCode;
my @choice = map { 'choice' . "$_" } 0 .. 3;
my @choiceValues;

# 将16字分成四行
foreach (@coding) {
    foreach my $i ( 0 .. 3 ) {
        if ( substr( $_, $i, 1 ) == '1' ) {
            push @{ $line[$i] }, $mapping{$_};
        }
    }
}

# 绑定choiceValue的值
foreach my $index ( 0 .. 3 ) {
    push @choiceValues, param( $choice[$index] );
}

print header( -charset => 'utf8' )
  . start_html( -title => "猜字游戏" )
  . start_form( 'GET', '/cgi-bin/20307130176-13.pl' )
  . h2('请在以下汉字中选择一个记住')
  . h3(@characters)
  . h3("你猜的字在以下的汉字中吗？");

# 显示问题以及对应的选项
foreach my $i ( 0 .. 3 ) {
    print p("请选择以下字中是否有你猜测的字?")
      . p( join( " ", @{ $line[$i] } ) )
      . radio_group(
        -name    => $choice[$i],
        -values  => [ 'yes', 'no' ],
        -default => 'no'
      );
}

print p() . submit( -name => 'submit', -value => "我想好了" );

# 判断选项结果对应的字
foreach my $index ( 0 .. 3 ) {
    $CharacterCode .= ( $choiceValues[$index] eq 'yes' ) ? '1' : '0';
}

print h3("你猜的字是：") . p("$mapping{$CharacterCode}") . end_form() . end_html;
1;
```