Documentation of `rubikrotation.pl` version 3.0

Nickalls RWD (dick@nickalls.org)
and
Syropoulos A (asyropoulos@yahoo.com)

Last revised: 25 September 2015
this file: `rubikrotationPL.tex`

Contents

1 Overview

This Perl program, is part of the LaTeX rubikrotation package. The two packages Rubikcube and Rubikrotation form the 'Rubik bundle' (available from `http://www.ctan.org/pkg/rubik`).

This program reads (as input) a formatted 'data' file (typically the file `rubikstate.dat` when used with the rubikcube package). The data-file defines the current state of the cube (using the keywords up, down, left, right, front, back), and includes special keywords (checkstate, rotation, random) which trigger the relevant subroutines to process the input data accordingly. Finally, the program writes the final state to a text-file (typically the file `rubikstateNEW.dat` when used with the Rubikcube package), and writes any error messages to the file `rubikstateERRORS.dat`.

Version 3 onwards uses the input and output filenames specified by the CALLing command-line, the usage being as follows:

```
rubikrotation  -i <input file>  [-o <out file>]
```

If no output filename is specified, then the default filename `rubikOUT.txt` is used. When used in conjunction with the Rubikcube package, then `rubikcube.sty` calls the program and sets the input filename as `rubikstate.dat` and the output filename as `rubikstateNEW.dat`.

2 The code

```
#!/usr/bin/env perl
##
use Carp;
use Fatal;
use warnings;
##
our $version = "3.0 (25 Sept 2015)";
##----------------------------
##
##  rubikrotation.pl
##  VERSION 3.0
##  Copyright 25 September, 2015,
##  RWD Nickalls (dick@nickalls.org}
##  A Syropoulos (asyropoulos@yahoo.com)
##
##--------------------------------
## changes in v3.0:
## 1) accepts command-line arguments for input (mandatory) and output (optional) filenames
##    default output filename is: rubikOUT.txt
## 2) included the symbols [ and ] to denote a rotation-name label (ie as well as *)
## 3) fixed some of the variable definitions (as highlighted by <use strict> pragma)
##--------------------------------
## changes in v2.3:
## 1) accepts a single  commandline argument (datafilename)
## 2) uses the standard modules Carp and Fatal (give extra  line info on error)
##--------------------------------
## changes in v2.2:
## 1) changed licence --> LatexPP
## 2) included random n errors in ERROR messages (lines 492--495)
## 3) included version number in error message
##-----------------------------
##
## This file is part of the LaTeX  rubikrotation package, and
## requires rubikcube.sty and rubikrotation.sty
##
## rubikrotation.pl is a Perl-5 program and free software:
```

```
## This program can be redistributed and/or modified under the terms
## of the LaTeX Project Public License Distributed from CTAN
## archives in directory macros/latex/base/lppl.txt; either
## version 1 of the License, or any later version.
##
## rubikrotation.pl is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
## GNU General Public License for more details.
##
##-------------------------------------------------------------------
## OVERVIEW
## This program is part of the rubikrotation package, and is complementary to
## the LaTeX rubikcube package. It processes Rubik rotation sequences  on-the-fly.
## The program reads a datafile (rubikstate.dat) output by the rubikcube package
## and writes the new state to the file rubikstateNEW.dat, which is then input
## by the TeX file. Further documentation  accompanies the rubikrotation package.
##
## Note that all possible state changing rotations of a 3x3x3 cube are
## either combinations of, or the inverse of, just 9 different rotations,
## three associated with each XYZ axis.
##-------------------------------------------------
```


2.1 Main

```
\begin{verbatim}
## This main module opens three files, and
##      sets up an array for collecting all errors (%error), and sets an error flag to "",
##      reads in the rubik state data file =rubikstate.dat (output by TeXfile),
##      and calls subs to write the TeX_OUT_FILE,
##      and finally closes all files.
## Each line of the input file consists of a comma separated list of arguments.
## The first argument in each line of the file rubikstate.dat is the rubikkeyword.
## Program is documented in the rubikrotation.pdf (see section ‘‘Overview’’)

our $source_file="";
our $out_file="rubikOUT.txt"; #default
our $argc=@ARGV;
```

```
our $commandLineArgs = join(" ", @ARGV);
our $showargs="\tcommandline args = $commandLineArgs\n";
our $usage="\tUsage: rubikrotation [-h|--help|-v|--version] -i <input file> [-o <out file>]\n";
our $rubikversion="\tVersion: this is rubikrotation version $version\n";
#
## check for correct number of commandline  arguments and allocate filenames
#
if ($argc == 0||$argc > 4 ){ # croak if  0 or  more than 4 arguments
        croak $rubikversion,$showargs,
        "\tWrong no of arguments\n",
        $usage;
   }
   else {
        SWITCHES:
        while($_ = $ARGV[0]) {
        shift;
        if (/^-h$/ || /^--help$/ ) {
             die $rubikversion,$usage,
             "\twhere,\n" .
             "\t[-h|--help]\tgives this help listing\n" .
             "\t[-v|--version]\tgives version\n" .
             "\t[-i]     \tcreates specified input file\n",
             "\t[-o]     \tcreates specified output file\n",
             "\tFor documentation see: rubikrotation.pdf,\n",
             "\trubikrotationPL.pdf and rubikcube.pdf.\n\n";
        }
        elsif  (/^-v$/ || /^--version$/ ) {die $rubikversion;}
        elsif (/^-i$/) {
              if (!@ARGV){
                       croak $showargs,
                       "\tNo input file specified!\n",
                       $usage;
                      }
              else {
                  $source_file = $ARGV[0],
                  shift;
                  }
        }
```

```
        elsif (/^-o$/) {
              if (!@ARGV) {
                 croak $showargs,
                    "\tNo output file specified!\n",
                    $usage;
                    }
              else {
                $out_file = $ARGV[0],
                shift;
              }
        }
        elsif (/^-\w+/) {
            croak $showargs,
            "\t$_: Illegal command line switch!\n",
            $usage;
        }
        else {
            croak $showargs,
            "\tmissing filenames or ? missing -i or -o switch!\n",
            $usage;
       }
   } # end of while
};   # end of else

print "This is rubikrotation version $version\n";
#================================
  open(IN_FILE, "<$source_file")  ||croak "\tCan't open source file: $source_file\n";
  open(TeX_OUT_FILE, ">$out_file")||croak "\tCan't open output file: $out_file\n";

  ## create error file (for append)
  open (ERROR_OUT_FILE,  ">>rubikstateERRORS.dat")||croak "ERROR: can't open file rubikstateERRORS.dat\n";


 ## use dots for Perl messages (I have used dashes for LaTeX messages in the .sty)
 ## gprint sub prints its argument (message) to both the screen and to the TeX_OUT_FILE

 gprint ("...PERL process.................................");
 gprint ("...program = rubikrotation.pl v$version");
```

```
## setup  global error parameters, so we can write all the errors to a file as an array
my %error       = (); # setup an array for error messages (was %)
my $erroralert  = ""; # error flag
my $errornumber = 0;  #set number of errors to zero

 gprint ("...reading the current rubik state (from File: $source_file)");

my $dataline = "";
my $rubikkeyword = "";
my $rotationcommand = "";
my @data;

 LINE: while (<IN_FILE>){
         next LINE if /^#/;  #skip comments
         next LINE if /^%/;  #skip comments
  next LINE if /^$/;  #skip blank lines
         $dataline = $_; # grab the whole line as a string
  chomp $dataline;  # remove the line-ending characters
  #$n++; # count the number of lines
  @data=split (/,/, $dataline); # create an array called data
  ## we have 10 fields (0--9)
  ## check for rubikkeyword= up,down,left,right,front,back,checkstate,rotation:
         gprint ("...$dataline");
  $rubikkeyword=$data[0];

  if ($rubikkeyword eq 'up') {
                               $Ult[0]=$data[1], $Umt[0]=$data[2],$Urt[0]=$data[3],
                               $Ulm[0]=$data[4], $Umm[0]=$data[5],$Urm[0]=$data[6],
                               $Ulb[0]=$data[7], $Umb[0]=$data[8],$Urb[0]=$data[9]
                               }

  if ($rubikkeyword eq 'down') {
                               $Dlt[0]=$data[1], $Dmt[0]=$data[2],$Drt[0]=$data[3],
                               $Dlm[0]=$data[4], $Dmm[0]=$data[5],$Drm[0]=$data[6],
                               $Dlb[0]=$data[7], $Dmb[0]=$data[8],$Drb[0]=$data[9]
                               }
  if ($rubikkeyword eq 'left') {
```

```
                                $Llt[0]=$data[1], $Lmt[0]=$data[2],$Lrt[0]=$data[3],
                                $Llm[0]=$data[4], $Lmm[0]=$data[5],$Lrm[0]=$data[6],
                                $Llb[0]=$data[7], $Lmb[0]=$data[8],$Lrb[0]=$data[9]
                                }
  if ($rubikkeyword eq 'right') {
                                $Rlt[0]=$data[1], $Rmt[0]=$data[2],$Rrt[0]=$data[3],
                                $Rlm[0]=$data[4], $Rmm[0]=$data[5],$Rrm[0]=$data[6],
                                $Rlb[0]=$data[7], $Rmb[0]=$data[8],$Rrb[0]=$data[9]
                                }

  if ($rubikkeyword eq 'front') {
                                $Flt[0]=$data[1], $Fmt[0]=$data[2],$Frt[0]=$data[3],
                                $Flm[0]=$data[4], $Fmm[0]=$data[5],$Frm[0]=$data[6],
                                $Flb[0]=$data[7], $Fmb[0]=$data[8],$Frb[0]=$data[9]
                                }
  if ($rubikkeyword eq 'back') {
                                $Blt[0]=$data[1], $Bmt[0]=$data[2],$Brt[0]=$data[3],
                                $Blm[0]=$data[4], $Bmm[0]=$data[5],$Brm[0]=$data[6],
                                $Blb[0]=$data[7], $Bmb[0]=$data[8],$Brb[0]=$data[9]
                                }

## if the rubikkeyword is ‘checkstate’  we just check the
## state and write the output data to a file.
          if ($rubikkeyword eq 'checkstate') {
                                        gprint ("...");
                                        $rotationcommand=$dataline; ## used in output message
                                        gprint ("...command=$rotationcommand");
                                        checkstate();
                                       };

## if the rubikkeyword is ‘rotation’  we first check to see if the second argument=random.
## if so, then we check that the third argument is an integer, if so --> random sub
## else  --> exit line  and get next line.
## finally we  write the output data to a file.

  if ($rubikkeyword eq 'rotation')
                  {
                    gprint ("...");
```

```
                    $rotationcommand=$dataline; ## used in output message
                    gprint ("...command=$rotationcommand");


                  ##-----random-----------------
                  ## if second argument = random,
                  ##   THEN we also need to check if third argument is an integer;
                  ## if so -->random sub.
                  ## if the 3rd argument is NOT an integer then reject line & get next input line
                  if ($data[1] eq 'random')
                        {
                           if ($data[2] =~/\D/) {
                               errormessage("[$data[2]] not an integer");
                               ##  we reject the whole line and look at next line in the file
                               next
                           }
                           else{
                               ## data[2] must be an integer (n), so we just do n random rotations
                               random($data[2])
                           };
                        }
                  else {
                        # the line must be an ordinary rotation sequence line, so send the sequence
                        # to the rotation sub,  BUT, need to first remove the
                        # rubikkeyword 'rotation'  from the begining of line (array)
                        # as we need to send ONLY the  sequence string to the rotation sub.

                        ## remove keyword rotation
                        shift (@data);

                        ## process @data string by the rotation sub
                        rotation(@data);
                        }
                  }
            ##------------------------------
            ## place additional keywords here
            ##------------------------------
                     }; ## end of while
```

```
 ## we have finished reading in all the lines from the source file,
 ## and doing all the rotations etc, so we now just write the new cube state
 ## to the output file = TeX_OUT_FILE (so LaTeX can read it)
 writestate();
 close; ##close all files
 exit;
##==============end of main==========================
```

2.2 Errormessage

```
sub errormessage{
## writes the argument as a standard error message to out file
  my $errormess        = $_[0];
  $erroralert          = "YES"; ## set error alert flag (for use in out message)
  $error[$errornumber] = "** ERROR: $errormess";
  $errornumber++; ## increment number
};
```


2.3 Gprint

```
##  prints argument (comments) to screen and also to TeX_OUT_FILE.
## The typeout commands will find its way into the log file when read by latex
## Important to include trailing % for messages written to the TeX_OUT_FILE
##    to stop extra <spaces> being seen by TeX.
sub gprint{
    my $gmess=$_[0];
    print  "$gmess\n";
    print (TeX_OUT_FILE    "\\typeout{$gmess}\%\n");
           };
```

2.4 Checkstate

```
sub checkstate{
### simple check to see if wrong no of colours being used etc
### uses the cubie colours as used by rubikcube package= ROYGBWX

 gprint ("...checking state of cube");
```

```
 my @cubies=($Ult[0],$Umt[0],$Urt[0],  $Ulm[0],$Umm[0],$Urm[0],  $Ulb[0],$Umb[0],$Urb[0],
             $Dlt[0],$Dmt[0],$Drt[0],  $Dlm[0],$Dmm[0],$Drm[0],  $Dlb[0],$Dmb[0],$Drb[0],
             $Llt[0],$Lmt[0],$Lrt[0],  $Llm[0],$Lmm[0],$Lrm[0],  $Llb[0],$Lmb[0],$Lrb[0],
             $Rlt[0],$Rmt[0],$Rrt[0],  $Rlm[0],$Rmm[0],$Rrm[0],  $Rlb[0],$Rmb[0],$Rrb[0],
             $Flt[0],$Fmt[0],$Frt[0],  $Flm[0],$Fmm[0],$Frm[0],  $Flb[0],$Fmb[0],$Frb[0],
             $Blt[0],$Bmt[0],$Brt[0],  $Blm[0],$Bmm[0],$Brm[0],  $Blb[0],$Bmb[0],$Brb[0]);

my $R=0,my $O=0,my $Y=0,my $G=0,my $B=0,my $W=0,my $X=0;

my $cubiecolour = "";

foreach $cubiecolour (@cubies)
      {
       if    ($cubiecolour eq R) {$R = $R+1}
       elsif ($cubiecolour eq O) {$O = $O+1}
       elsif ($cubiecolour eq Y) {$Y = $Y+1}
       elsif ($cubiecolour eq G) {$G = $G+1}
       elsif ($cubiecolour eq B) {$B = $B+1}
       elsif ($cubiecolour eq W) {$W = $W+1}
       elsif ($cubiecolour eq X) {$X = $X+1}
       else  {print " cubiecolour counting ERROR \n";}
     };
my $cubiesum=0;
$cubiesum = $R+$O+$Y+$G+$B+$W+$X;
gprint ("...cubiesum = $cubiesum (Red=$R, Or=$O, Ye=$Y, Gr=$G, Bl=$B, Wh=$W, X=$X)");

if ($cubiesum != 54) { errormessage("cubiesum not = 54")};
if ($R >9){ errormessage("No of Red cubies > 9 (=$R)")};
if ($O >9){ errormessage("No of Orange cubies > 9 (=$O)")};
if ($Y >9){ errormessage("No of Yellow cubies > 9 (=$Y)")};
if ($G >9){ errormessage("No of Green cubies > 9 (=$G)")};
if ($B >9){ errormessage("No of Blue cubies > 9 (=$B)")};
if ($W >9){ errormessage("No of White cubies > 9 (=$W)")};
};
##=================================================
```

2.5 Rotation

```
# no of arguments passed to the sub = $#_ (black book p 156)

sub rotation {
## here we process the array @data  (from main) consisting of all
## the rotation commands associated with
## a single RubikRotation command.

  my $m; #multiple associated with the char, eg D2 etc
  my $n = ($#_ +1);  ##total no of arguments passed
  my $originalchar="";
  my $j;
  my $numberofchars; ## length of a string
  my $nfrontchars;
  my $char = "";

     gprint ("...arguments passed to 'rotation' sub = @_ (n= $n)");
     foreach $char (@_) {
        $char =~s/^\s+//, $char=~s/\s+$//; ## clean leading and trailing white space
        ## grab a copy of the original char for use if  m Mod4=0
        $originalchar=$char;

    ## if argument has a leading * or [ or ] then it is a label (not a rotation)
    ## so jump to next one
   if (substr ($char,0,1)  =~  /[ * \[ \] ]/  ){
                                  gprint ("...$char is a label OK");
                                  next;
                                  };

  ## Need to detect any trailing integer associated with a command (eg rotation multiple, eg:U3, L2 etc)
  ## NOTES: since we are using mod4, we are only interested in trailing digit
  ## if trailing character is a digit, then
  ##    split char string into front chars (= $char) + trailing digit (= $m)
     $m = 1; # initialise m

  ## if terminal char is a digit (d) then let d --> m  and let frontstring --> char
  ## (Black book p 130 & 136)
```

```
## if the frontstring contains any digits then it will be rejected in the filter below anyway.

 if ( substr ($char,-1) =~ /(\d)/)
        {
        $m = $1; ## grab the trailing digit (only traps a single digit)
        $numberofchars = 0; #initialise it
        $numberofchars = length $originalchar;
        $nfrontchars   = $numberofchars-1;
        ## reassign the string except the terminal digit
        $char = substr($char,0,$nfrontchars);
        ## use MOD 4 since we are dealing with Rubik rotations
        $m = $m % 4;
        ## (if MOD 4 = 0 then nothing will happen as j starts at 1)
        ## but should generate an errormessage
        if ( $m == 0 ){
                   gprint ("...rotation $originalchar equiv 0 (NOT IMPLEMENTED)");
                   errormessage ("[$originalchar] in RubikRotation (4 MOD 4 = 0)");
                   next;
                  };
        };
    ## if  single trailing digit present, then we implement the rotation command m times.
    ## if more than one trailing digit
    ##     then the error is trapped at the end (as frontstring will not be recognised
    ##     ie will not be  in the following list, and hence will be trapped as an error, eg R3)
    ##
     if ($char eq "R")   {for($j=1;$j<=$m;$j++) { gprint ("...rotation R   OK"); &rrR}}
  elsif ($char eq "Rp")  {for($j=1;$j<=$m;$j++) { gprint ("...rotation Rp  OK (= rrR3)"); &rrRp}}
  elsif ($char eq "Rw")  {for($j=1;$j<=$m;$j++) { gprint ("...rotation Rw  OK (= rrR + rrSr)"); &rrRw}}
  elsif ($char eq "Rwp") {for($j=1;$j<=$m;$j++) { gprint ("...rotation Rwp OK (= rrRp + rrSrp)"); &rrRwp}}
  elsif ($char eq "Rs")  {for($j=1;$j<=$m;$j++) { gprint ("...rotation Rs  OK (= rrR + rrLp)"); &rrRs}}
  elsif ($char eq "Rsp") {for($j=1;$j<=$m;$j++) { gprint ("...rotation Rsp OK (= rrRp + rrL)"); &rrRsp}}
  elsif ($char eq "Ra")  {for($j=1;$j<=$m;$j++) { gprint ("...rotation Ra  OK (= rrR + rrL)"); &rrRa}}
  elsif ($char eq "Rap") {for($j=1;$j<=$m;$j++) { gprint ("...rotation Rap OK (= rrRp + rrLp)"); &rrRap}}
  ####
  elsif ($char eq "L")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation L   OK (= rrLp3)"); &rrL}}
  elsif ($char eq "Lp")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Lp  OK"); &rrLp}}
  elsif ($char eq "Lw")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Lw  OK (= rrLp3 + rrSrp)"); &rrLw}}
  elsif ($char eq "Lwp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Lwp OK (= rrLp + rrSr)"); &rrLwp}}
```

```
elsif ($char eq "Ls")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Ls  OK (= rrL + rrRp)"); &rrLs}}
elsif ($char eq "Lsp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Lsp OK (= rrLp + rrR)"); &rrLsp}}
elsif ($char eq "La")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation La  OK (= rrL + rrR)"); &rrLa}}
elsif ($char eq "Lap") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Lap OK (= rrLp + rrRp)"); &rrLap}}
####
elsif ($char eq "U")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation U   OK"); &rrU}}
elsif ($char eq "Up")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Up  OK (= rrU3)"); &rrUp}}
elsif ($char eq "Uw")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Uw  OK (= rrU + rrSu)"); &rrUw}}
elsif ($char eq "Uwp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Uwp OK (= rrUp + rrSup)"); &rrUwp}}
elsif ($char eq "Us")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Us  OK (= rrU + rrDp)"); &rrUs}}
elsif ($char eq "Usp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Usp OK (= rrUp + rrD)"); &rrUsp}}
elsif ($char eq "Ua")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Ua  OK (= rrU + rrD)"); &rrUa}}
elsif ($char eq "Uap") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Uap OK (= rrUp + rrDp)"); &rrUap}}
####
elsif ($char eq "D")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation D   OK (= rrDp3)"); &rrD}}
elsif ($char eq "Dp")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Dp  OK "); &rrDp}}
elsif ($char eq "Dw")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Dw  OK (= rrDp3 + rrSup)"); &rrDw}}
elsif ($char eq "Dwp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Dwp OK (= rrDp + rrSu)"); &rrDwp}}
elsif ($char eq "Ds")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Ds  OK (= rrD + rrUp)"); &rrDs}}
elsif ($char eq "Dsp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Dsp OK (= rrDp + rrU)"); &rrDsp}}
elsif ($char eq "Da")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Da  OK (= rrD + rrU)"); &rrDa}}
elsif ($char eq "Dap") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Dap OK (= rrDp + rrUp)"); &rrDap}}
####
elsif ($char eq "F")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation F   OK"); &rrF}}
elsif ($char eq "Fp")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Fp  OK (= rrF3)"); &rrFp}}
elsif ($char eq "Fw")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Fw  OK (= rrF + rrSf)"); &rrFw}}
elsif ($char eq "Fwp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Fwp OK (= rrFp + rrSfp)"); &rrFwp}}
elsif ($char eq "Fs")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Fs  OK (= rrF + rrBp)"); &rrFs}}
elsif ($char eq "Fsp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Fsp  OK (= rrFp + rrB)"); &rrFsp}}
elsif ($char eq "Fa")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Fa  OK (= rrF + rrB)"); &rrFa}}
elsif ($char eq "Fap") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Fap  OK (= rrFp + rrBp)"); &rrFap}}
####
elsif ($char eq "B")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation B   OK (= rrFp3)"); &rrB}}
elsif ($char eq "Bp")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Bp  OK"); &rrBp}}
elsif ($char eq "Bw")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Bw  OK (= rrFp3 + rrSfp)"); &rrBw}}
elsif ($char eq "Bwp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Bwp OK (= rrFp + rrSf)"); &rrBwp}}
elsif ($char eq "Bs")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Bs  OK (= rrB + rrFp)"); &rrBs}}
elsif ($char eq "Bsp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Bsp  OK (= rrBp + rrF)"); &rrBsp}}
```

```
elsif ($char eq "Ba")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Ba  OK (= rrB + rrF)"); &rrBa}}
elsif ($char eq "Bap") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Bap  OK (= rrBp + rrFp)"); &rrBap}}
####
elsif ($char eq "Su")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Su  OK "); &rrSu}}
elsif ($char eq "Sup") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sup OK (= rrSu3)"); &rrSup}}
elsif ($char eq "Sd")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sd  OK (= rrSup)"); &rrSd}}
elsif ($char eq "Sdp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sdp OK (= rrSu)"); &rrSdp}}
elsif ($char eq "Sl")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sl  OK (= rrSrp)"); &rrSl}}
elsif ($char eq "Slp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Slp OK (= rrSr)"); &rrSlp}}
elsif ($char eq "Sr")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sr  OK"); &rrSr}}
elsif ($char eq "Srp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Srp OK (= rrSr3)"); &rrSrp}}
elsif ($char eq "Sf")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sf  OK"); &rrSf}}
elsif ($char eq "Sfp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sfp OK (= rrSf3)"); &rrSfp}}
elsif ($char eq "Sb")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sb  OK (= rrSfp)"); &rrSb}}
elsif ($char eq "Sbp") {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sbp OK (= rrSf)"); &rrSbp}}
## XYZ stuff
## need to include lowercase x,y,x, and also u,d,l,r,f,b equivalents
elsif ($char eq "X" or $char eq "x" or $char eq "r")
          {for($j=1;$j<=$m;$j++) {gprint ("...rotation $char OK (= rrR + rrSr + rrLp)"); &rrR;&rrSr;&rrLp}}
elsif ($char eq "Xp" or $char eq "xp" or $char eq "l")
          {for($j=1;$j<=$m;$j++) {gprint ("...rotation $char OK (= rrRp + rrSrp + rrL)");&rrRp;&rrSrp;&rrL}}
elsif ($char eq "Y" or $char eq "y" or $char eq "u")
          {for($j=1;$j<=$m;$j++) {gprint ("...rotation $char OK (= rrU + rrSu + rrDp)"); &rrU;&rrSu;&rrDp}}
elsif ($char eq "Yp" or $char eq "yp" or $char eq "d")
          {for($j=1;$j<=$m;$j++) {gprint ("...rotation $char OK (= rrUp + rrSup + rrD)");&rrUp;&rrSup;&rrD}}
elsif ($char eq "Z" or $char eq "z" or $char eq "f")
          {for($j=1;$j<=$m;$j++) {gprint ("...rotation $char OK (= rrF + rrSf + rrBp)"); &rrF;&rrSf;&rrBp}}
elsif ($char eq "Zp" or $char eq "zp" or $char eq "b")
          {for($j=1;$j<=$m;$j++) {gprint ("...rotation $char OK (= rrFp + rrSfp + rrB)");&rrFp;&rrSfp;&rrB}}
## extras
elsif ($char eq "M")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation M   OK (= Sl) "); &rrSl}}
elsif ($char eq "Mp")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Mp  OK (= Sr) "); &rrSr}}
elsif ($char eq "E")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation E   OK (= Sd) "); &rrSd}}
elsif ($char eq "Ep")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Ep  OK (= Su) "); &rrSu}}
elsif ($char eq "S")   {for($j=1;$j<=$m;$j++) {gprint ("...rotation S   OK (= Sf) "); &rrSf}}
elsif ($char eq "Sp")  {for($j=1;$j<=$m;$j++) {gprint ("...rotation Sp  OK (= Sb) "); &rrSb}}
####
## check for missing rotation
```

```
    elsif ($char eq "")    {for($j=1;$j<=$m;$j++)
                               {gprint ("...rotation ,$char, ERROR  ? typo or missing rotation"),
                               errormessage("[,$char,] in RubikRotation: ? typo or missing rotation")}}
    ####
    ## rotation must be undefined
    else {
         gprint ("...rotation $originalchar NOT KNOWN");
         errormessage("[$originalchar] in RubikRotation\{\}");
         };
      }; # end of foreach
  };    # end of sub

##-----------------------------------------------
## aNOTE we have  defined (as rotation SUBs at the end) just 9 primary rotation transforms,
## rrR, rrSr, rrLp
## rrU, rrSu, rrDp
## rrF, rrSf, rrBp
##  and since all the remaining ones  are simply combinations of these 9
##  we now define all the other rotation subs in terms of these 9 primary moves.
##  do NOT use multiples here: write each rotation separately
## -------------
## derivative subs from R and Sr and Lp
sub rrRp{&rrR;&rrR;&rrR}; # (=rrR3)
sub rrRw{&rrR; &rrSr}; # (= rrR + rrSr)
sub rrRwp{&rrR;&rrR;&rrR;  &rrSr;&rrSr;&rrSr}; # (= rrRp + rrSrp)
sub rrRs{&rrR;&rrLp};
sub rrRsp{&rrRp;&rrL};
sub rrRa{&rrR;&rrL};
sub rrRap{&rrRp;&rrLp};
####
sub rrL{&rrLp;&rrLp;&rrLp}; # (= rrLp3)
sub rrLw{&rrLp;&rrLp;&rrLp;&rrSrp};   # (=rrLp3 + rrSrp)
sub rrLwp{&rrLp;&rrSr};
sub rrLs{&rrL;&rrRp};
sub rrLsp{&rrLp;&rrR};
sub rrLa{&rrL;&rrR};
sub rrLap{&rrLp;&rrRp};
##-------------
```

```
## derivative subs from U
sub rrUp{&rrU;&rrU;&rrU}; # (=rrU3)
sub rrUw{&rrU;&rrSu}; #
sub rrUwp{&rrUp;&rrSup};
sub rrUs{&rrU;&rrDp};
sub rrUsp{&rrUp;&rrD};
sub rrUa{&rrU;&rrD};
sub rrUap{&rrUp;&rrDp};
####
sub rrD{&rrDp;&rrDp;&rrDp}; # (= rrDp3)
sub rrDw{&rrDp;&rrDp;&rrDp;&rrSup};   # (=rrDp3 + rrSup)
sub rrDwp{&rrDp;&rrSu};
sub rrDs{&rrD;&rrUp};
sub rrDsp{&rrDp;&rrU};
sub rrDa{&rrD;&rrU};
sub rrDap{&rrDp;&rrUp};
##-------------
## derivative subs from F
sub rrFw{&rrF; &rrSf}; # (= rrF + rrSf)
sub rrFp{ &rrF;&rrF;&rrF}; # (=rrF3)
sub rrFwp{&rrF;&rrF;&rrF;  &rrSf;&rrSf;&rrSf}; # (= rrF3 + rrSf3)
sub rrFs{&rrF;&rrBp};
sub rrFsp{&rrFp;&rrB};
sub rrFa{&rrF;&rrB};
sub rrFap{&rrFp;&rrBp};
####
sub rrB{&rrBp;&rrBp;&rrBp}; # (= rrBp3)
sub rrBw{&rrBp;&rrBp;&rrBp; &rrSfp};   # (=rrBp3 + rrSfp)
sub rrBwp{&rrBp;&rrSf};
sub rrBs{&rrB;&rrFp};
sub rrBsp{&rrBp;&rrF};
sub rrBa{&rrB;&rrF};
sub rrBap{&rrBp;&rrFp};
####
## bring all the S versions together
sub rrSup{&rrSu;&rrSu;&rrSu}; # (=rrSu3)
sub rrSd{&rrSup}; # (=rrSup)
sub rrSdp{&rrSu}; # (=rrSu)
```

```
sub rrSl{&rrSrp}; # (=rrSrp)
sub rrSlp{&rrSr}; # (=rrSr)
sub rrSrp{&rrSr;&rrSr;&rrSr}; # (=rrSr3)
sub rrSfp{&rrSf;&rrSf;&rrSf}; # (=rrSf3)
sub rrSb{&rrSfp}; # (=rrSfp)
sub rrSbp{&rrSf}; # (=rrSf)
```

2.6 Random

```
# no of arguments passed to the sub = $#_ (black book p 156)
# parameters  passed = $_[0]

sub random{
## scramble  randomly using n rotations
## example command = RubikRotation{random,74}
## if no n given (second argument  = ""), then use default n=50
## if second argument is some string (not integer) then --> ERROR
##
## asign numbers to the minimal set of rotations to be used using a hash array list
## (perl 5 book page 68)
## ? maybe we should only use the 18 rotations mentioned in Rokicki 2013 paper?
## but here I have included all the S ones too.

my @rrlist= ("U", "Up", "Su", "Sup",
             "D", "Dp", "Sd", "Sdp",
             "L", "Lp", "Sl", "Slp",
             "R", "Rp", "Sr", "Srp",
             "F", "Fp", "Sf", "Sfp",
             "B", "Bp", "Sb", "Sbp");

my $rrlistnumber=$#rrlist;
print " rrlistnumber = $rrlistnumber\n";
# these are numbered 0--$rrlistnumber,

## let default no of random rotations for scrambling = 50
my $defaultn = 50;
my $maxn     = 200;
##  grab the integer passed  from the random() command in main
```

```
my $s = $_[0];
if ($s >= $maxn) {$s = $maxn;
                     gprint ("...WARNING: maximum n = 200");
                     errormessage ("random: max n =200 (n=200 was used)")}
    elsif ($s == 0) {$s = $defaultn;
                     gprint ("...WARNING: integer = 0 or not given: using default value 50");
                     errormessage ("random: n invalid (n=50 was used)")};

my @rr; ## array to hold all the random rotations
print " randomising the available rotations\n";


## set the seed for the randomisation (BlackBook p 235)
srand;

## now select s numbers at random (with replacement) from range 0--listnumber+1
## Since we are using int(rand x), and  using nos from 0--lastindex number,
## then max  rand vaue = (lastindexnumber -1).99999, the  integer of which
## = (lastindexnumber -1). Therefore we need to use the range 0--(lastindexnumber+1)
## in order to randomise all posibilities on  our list.

my $j;

for ($j = 1; $j <=$s; $j=$j+1)
    {
     my $p= int(rand ($rrlistnumber +1));
     print "Rotation = $p, $rrlist[$p] \n";
     ## push rotation code   $rrlist[$p] on to END of array @rr
     push (@rr, $rrlist[$p]);
    };

  ## we assume the user is starting from a solved cube (ie use the state given by user)
  gprint ("...scrambling Rubik cube using $s random rotations");
  ## now send the array off to the rotation sub
  rotation(@rr);
}
##==================subs================================
```

2.7 Writestate

```
sub writestate{
## this just writes the final state to the TeX_OUT_FILE (= rubikstateNEW.dat) will be read by latex.

print (TeX_OUT_FILE    "\%\% output datafile=$out_file\n");
print (TeX_OUT_FILE    "\%\% PERL prog=rubikrotation version $version\n");
print (TeX_OUT_FILE    "\\typeout{...writing new Rubik state to file $out_file}\%\n");
print (TeX_OUT_FILE    "\\RubikFaceUp\{$Ult[0]\}\{$Umt[0]\}\{$Urt[0]\}\{$Ulm[0]\}\{$Umm[0]\}\{$Urm[0]\}\{$Ulb[0]\}
    \{$Umb[0]\}\{$Urb[0]\}\%\n");
print (TeX_OUT_FILE    "\\RubikFaceDown\{$Dlt[0]\}\{$Dmt[0]\}\{$Drt[0]\}\{$Dlm[0]\}\{$Dmm[0]\}\{$Drm[0]\}\{$Dlb[0]\}
    \{$Dmb[0]\}\{$Drb[0]\}\%\n");
print (TeX_OUT_FILE    "\\RubikFaceLeft\{$Llt[0]\}\{$Lmt[0]\}\{$Lrt[0]\}\{$Llm[0]\}\{$Lmm[0]\}\{$Lrm[0]\}\{$Llb[0]\}
    \{$Lmb[0]\}\{$Lrb[0]\}\%\n");
print (TeX_OUT_FILE    "\\RubikFaceRight\{$Rlt[0]\}\{$Rmt[0]\}\{$Rrt[0]\}\{$Rlm[0]\}\{$Rmm[0]\}\{$Rrm[0]\}\{$Rlb[0]\}
    \{$Rmb[0]\}\{$Rrb[0]\}\%\n");
print (TeX_OUT_FILE    "\\RubikFaceFront\{$Flt[0]\}\{$Fmt[0]\}\{$Frt[0]\}\{$Flm[0]\}\{$Fmm[0]\}\{$Frm[0]\}\{$Flb[0]\}
    \{$Fmb[0]\}\{$Frb[0]\}\%\n");
print (TeX_OUT_FILE    "\\RubikFaceBack\{$Blt[0]\}\{$Bmt[0]\}\{$Brt[0]\}\{$Blm[0]\}\{$Bmm[0]\}\{$Brm[0]\}\{$Blb[0]\}
    \{$Bmb[0]\}\{$Brb[0]\}\%\n");

## now include any error messages generated
## (these are all  in an array waiting to be printed out)

if ($erroralert eq "YES")
      {
       ## write errors to a separate file (just for errors---we append the errrors to end of file)
       ## the error file (rubikstateERRORS.dat) was created by the TeX file
       my $ne;  #number of errors
       $ne=$#error; ## number of errors= largest index num since we started at zero
       ### do not attach error to a <checkstate> command
       if ($rotationcommand eq "checkstate") {}
          else {
                print (TeX_OUT_FILE     "\\typeout{** ERROR: command=$rotationcommand}\%\n");
                print (ERROR_OUT_FILE   "** ERROR: $rotationcommand\n");
               };
       ## last index number or array = $#arrayname (Black book p 62)
       my $k;
```

```
       for ($k=0; $k<=$ne; $k=$k+1)  {
                                       print (TeX_OUT_FILE      "\\typeout{$error[$k]}\%\n");
                                       print (ERROR_OUT_FILE    "$error[$k]\n");
                                     };
      };
print " Perl output file written OK\n";
}
##=========================================
```

2.8 rrR

```
## The following 9  (90 degree) rotation transformations are used
##   to generate all the rotations used in the 'rotation sub'
##   each of these is a permutation for both colours and numbers
##   of the cubie facelets.
## The following 9 subroutines are named as follows:
##   (about X-axis) rrR, rrSr, rrLp
##   (about Y-axis) rrU, rrSu, rrDp
##   (about Z-axis) rrF, rrSf, rrBp
##   see the rubikcube package documentation for full details regarding
##   rotation notation and commands.
## METHOD & NOTATION
## each sub (below) starts by making an array[0] for the cubie colour
##   and an array[1] for  the cubie number.
## Each  of the face rotations (rrR, rrLp, rrU, rrDp, rrF, rrBp) is involved with
##   two pairs of connected but different permutations/transformations as follows:
##   (a) one pair for the 12 Side cubies (arrays = @Xs0 (for Side colours), @Xs1 (for Side numbers)), and
##   (b) one pair for the 9 Face cubies (arrays = @Xf0 (for Face colours), @Xf1 (for Face numbers)).
## Each of the center slice rotations (rrSr, rrSu, rrSf)  is involved with just one pair of
##   permutations for the 12 Side cubies (arrays = @Xs0 (for Side colours), @Xs1 (for Side numbers)).
## We document only the side and face of the first sub (rrR) in detail, since the other subs are of similar form.
##=============================================
  sub rrR{

##  the R (slice + face) transform
## R = RIGHT, s = side; 0=colour, 1= number
## make the clockwise rotation permutation
## In this permutation the Front-right-bottom (Frb) (side)facelet rotates to
```

```
##    the new position of Up-right-bottom (Urb) (side)facelet.
##-----------SIDE-------
## 12 side cubie facelets in  arrays @Rs0 (colours) and @Rs1 (numbers)
## these are the initial positions
@Rs0=($Frb[0],$Frm[0],$Frt[0],
      $Urb[0],$Urm[0],$Urt[0],
      $Blt[0],$Blm[0],$Blb[0],
      $Drb[0],$Drm[0],$Drt[0]);

@Rs1=($Frb[1],$Frm[1],$Frt[1],
      $Urb[1],$Urm[1],$Urt[1],
      $Blt[1],$Blm[1],$Blb[1],
      $Drb[1],$Drm[1],$Drt[1]);

## now we reallocate the initial array elements to the new
##    post (90 degree clockwise) rotation position.
## Cube is viewed from FRONT.
## Positions of side facelets of Right slice are numbered 0-11 in clockwise direction,
##   (as seen from Right face) starting with Up-right-bottom facelet.
## First line example:
## variable $Urb[0] (Upface-right-bottom colour) <-- colour of first element in @Rs0 (=Frb[0])
## variable $Urb[1] (Upface-right-bottom number) <-- number of first element in @Rs1 (=Frb[1])
$Urb[0]=$Rs0[0]; $Urb[1]=$Rs1[0];
$Urm[0]=$Rs0[1]; $Urm[1]=$Rs1[1];
$Urt[0]=$Rs0[2]; $Urt[1]=$Rs1[2];

$Blt[0]=$Rs0[3]; $Blt[1]=$Rs1[3];
$Blm[0]=$Rs0[4]; $Blm[1]=$Rs1[4];
$Blb[0]=$Rs0[5]; $Blb[1]=$Rs1[5];

$Drb[0]=$Rs0[6]; $Drb[1]=$Rs1[6];
$Drm[0]=$Rs0[7]; $Drm[1]=$Rs1[7];
$Drt[0]=$Rs0[8]; $Drt[1]=$Rs1[8];

$Frb[0]=$Rs0[9];  $Frb[1]=$Rs1[9];
$Frm[0]=$Rs0[10]; $Frm[1]=$Rs1[10];
$Frt[0]=$Rs0[11]; $Frt[1]=$Rs1[11];
```

```
##-------------Right FACE---------------------
## R FACE (9 cubies in each array)
## (numbered in rows: 1,2,3/4,5,6/7,8,9 from top left(1) to bottom right(9))
## R=Right, f = face; 0=colour, 1= number
## do the Rface (90 degree) rotation transform
## here  the Right-left-bottom (Rlb) facelet rotates to the possn of Right-left-top (Rlt)
## we start with two arrays (one for colours @Rf0, one for numbers @Rf1)  with 9 elements each.
@Rf0=($Rlb[0], $Rlm[0], $Rlt[0],     $Rmb[0], $Rmm[0], $Rmt[0],     $Rrb[0], $Rrm[0], $Rrt[0]);
@Rf1=($Rlb[1], $Rlm[1], $Rlt[1],     $Rmb[1], $Rmm[1], $Rmt[1],     $Rrb[1], $Rrm[1], $Rrt[1]);

## now we reallocate the  array elements to the new
## post (90 degree clockwise) rotation  facelet position.
## Right face is viewed from RIGHT.
## First line example:
## variable $Rlt[0] (=Right-left-top colour) <-- colour of first element in @Rf0 (=Rlb[0])
## variable $Rlt[1] (=Right-left-top number) <-- number of first element in @Rf1 (=Rlb[1])
$Rlt[0]=$Rf0[0]; $Rlt[1]=$Rf1[0];
$Rmt[0]=$Rf0[1]; $Rmt[1]=$Rf1[1];
$Rrt[0]=$Rf0[2]; $Rrt[1]=$Rf1[2];

$Rlm[0]=$Rf0[3]; $Rlm[1]=$Rf1[3];
$Rmm[0]=$Rf0[4]; $Rmm[1]=$Rf1[4];
$Rrm[0]=$Rf0[5]; $Rrm[1]=$Rf1[5];

$Rlb[0]=$Rf0[6]; $Rlb[1]=$Rf1[6];
$Rmb[0]=$Rf0[7]; $Rmb[1]=$Rf1[7];
$Rrb[0]=$Rf0[8]; $Rrb[1]=$Rf1[8];

}
#============================
```

2.9 rrSr

```
sub rrSr {
## Sr = Right middle SLICE rotation (only 12 side facelets)
## modified from rrR (change the U,D,F, r --> m and Back Bl-->Bm; Rs--> ?Srs)
## change only the slice
## s = side; 0=colour, 1= number
```

```
## make the post rotation permutation

@SRs0=($Fmb[0],$Fmm[0],$Fmt[0],
       $Umb[0],$Umm[0],$Umt[0],
       $Bmt[0],$Bmm[0],$Bmb[0],
       $Dmb[0],$Dmm[0],$Dmt[0]);

@SRs1=($Fmb[1],$Fmm[1],$Fmt[1],
       $Umb[1],$Umm[1],$Umt[1],
       $Bmt[1],$Bmm[1],$Bmb[1],
       $Dmb[1],$Dmm[1],$Dmt[1]);


$Umb[0]=$SRs0[0]; $Umb[1]=$SRs1[0];
$Umm[0]=$SRs0[1]; $Umm[1]=$SRs1[1];
$Umt[0]=$SRs0[2]; $Umt[1]=$SRs1[2];

$Bmt[0]=$SRs0[3]; $Bmt[1]=$SRs1[3];
$Bmm[0]=$SRs0[4]; $Bmm[1]=$SRs1[4];
$Bmb[0]=$SRs0[5]; $Bmb[1]=$SRs1[5];

$Dmb[0]=$SRs0[6]; $Dmb[1]=$SRs1[6];
$Dmm[0]=$SRs0[7]; $Dmm[1]=$SRs1[7];
$Dmt[0]=$SRs0[8]; $Dmt[1]=$SRs1[8];

$Fmb[0]=$SRs0[9];  $Fmb[1]=$SRs1[9];
$Fmm[0]=$SRs0[10]; $Fmm[1]=$SRs1[10];
$Fmt[0]=$SRs0[11]; $Fmt[1]=$SRs1[11];

}
##===================
```

2.10 rrLp

```
sub rrLp{

## Left slice (side + face) anticlockwise rotation
## s = side; 0=colour, 1= number
```

```
##-------------side-----------
@LPs0=($Flb[0],$Flm[0],$Flt[0],
      $Ulb[0],$Ulm[0],$Ult[0],
      $Brt[0],$Brm[0],$Brb[0],
      $Dlb[0],$Dlm[0],$Dlt[0]);

@LPs1=($Flb[1],$Flm[1],$Flt[1],
      $Ulb[1],$Ulm[1],$Ult[1],
      $Brt[1],$Brm[1],$Brb[1],
      $Dlb[1],$Dlm[1],$Dlt[1]);


$Ulb[0]=$LPs0[0]; $Ulb[1]=$LPs1[0];
$Ulm[0]=$LPs0[1]; $Ulm[1]=$LPs1[1];
$Ult[0]=$LPs0[2]; $Ult[1]=$LPs1[2];

$Brt[0]=$LPs0[3]; $Brt[1]=$LPs1[3];
$Brm[0]=$LPs0[4]; $Brm[1]=$LPs1[4];
$Brb[0]=$LPs0[5]; $Brb[1]=$LPs1[5];

$Dlb[0]=$LPs0[6]; $Dlb[1]=$LPs1[6];
$Dlm[0]=$LPs0[7]; $Dlm[1]=$LPs1[7];
$Dlt[0]=$LPs0[8]; $Dlt[1]=$LPs1[8];

$Flb[0]=$LPs0[9];  $Flb[1]=$LPs1[9];
$Flm[0]=$LPs0[10]; $Flm[1]=$LPs1[10];
$Flt[0]=$LPs0[11]; $Flt[1]=$LPs1[11];


##---------------Left FACE-------------
## do the Lface transform (in rows: 1,2,3//4,5,6//7,8,9)
## f = face; 0=colour, 1= number
## NOTES: not same as for R

@LPf0=($Lrt[0], $Lrm[0], $Lrb[0],     $Lmt[0], $Lmm[0], $Lmb[0],     $Llt[0], $Llm[0], $Llb[0]);
@LPf1=($Lrt[1], $Lrm[1], $Lrb[1],     $Lmt[1], $Lmm[1], $Lmb[1],     $Llt[1], $Llm[1], $Llb[1]);

$Llt[0]=$LPf0[0]; $Llt[1]=$LPf1[0];
```

```
$Lmt[0]=$LPf0[1]; $Lmt[1]=$LPf1[1];
$Lrt[0]=$LPf0[2]; $Lrt[1]=$LPf1[2];

$Llm[0]=$LPf0[3]; $Llm[1]=$LPf1[3];
$Lmm[0]=$LPf0[4]; $Lmm[1]=$LPf1[4];
$Lrm[0]=$LPf0[5]; $Lrm[1]=$LPf1[5];

$Llb[0]=$LPf0[6]; $Llb[1]=$LPf1[6];
$Lmb[0]=$LPf0[7]; $Lmb[1]=$LPf1[7];
$Lrb[0]=$LPf0[8]; $Lrb[1]=$LPf1[8];
}
##================================
```

2.11 rrU

```
sub rrU{

## Up slice (side + face)
## do the Uside transform
## s = side; 0=colour, 1= number
## ----------SIDE--------------
@Us0=($Lrt[0],$Lmt[0],$Llt[0],
      $Brt[0],$Bmt[0],$Blt[0],
      $Rrt[0],$Rmt[0],$Rlt[0],
      $Frt[0],$Fmt[0],$Flt[0]);

@Us1=($Lrt[1],$Lmt[1],$Llt[1],
      $Brt[1],$Bmt[1],$Blt[1],
      $Rrt[1],$Rmt[1],$Rlt[1],
      $Frt[1],$Fmt[1],$Flt[1]);


$Brt[0]=$Us0[0]; $Brt[1]=$Us1[0];
$Bmt[0]=$Us0[1]; $Bmt[1]=$Us1[1];
$Blt[0]=$Us0[2]; $Blt[1]=$Us1[2];

$Rrt[0]=$Us0[3]; $Rrt[1]=$Us1[3];
$Rmt[0]=$Us0[4]; $Rmt[1]=$Us1[4];
```

```
$Rlt[0]=$Us0[5]; $Rlt[1]=$Us1[5];

$Frt[0]=$Us0[6]; $Frt[1]=$Us1[6];
$Fmt[0]=$Us0[7]; $Fmt[1]=$Us1[7];
$Flt[0]=$Us0[8]; $Flt[1]=$Us1[8];

$Lrt[0]=$Us0[9];  $Lrt[1]=$Us1[9];
$Lmt[0]=$Us0[10]; $Lmt[1]=$Us1[10];
$Llt[0]=$Us0[11]; $Llt[1]=$Us1[11];

##-------------Up FACE-------------------
## do the Rface transform (in rows: 1,2,3//4,5,6//7,8,9)
## f = face; 0=colour, 1= number
@Uf0=($Ulb[0], $Ulm[0], $Ult[0],     $Umb[0], $Umm[0], $Umt[0],     $Urb[0], $Urm[0], $Urt[0]);
@Uf1=($Ulb[1], $Ulm[1], $Ult[1],     $Umb[1], $Umm[1], $Umt[1],     $Urb[1], $Urm[1], $Urt[1]);

$Ult[0]=$Uf0[0]; $Ult[1]=$Uf1[0];
$Umt[0]=$Uf0[1]; $Umt[1]=$Uf1[1];
$Urt[0]=$Uf0[2]; $Urt[1]=$Uf1[2];

$Ulm[0]=$Uf0[3]; $Ulm[1]=$Uf1[3];
$Umm[0]=$Uf0[4]; $Umm[1]=$Uf1[4];
$Urm[0]=$Uf0[5]; $Urm[1]=$Uf1[5];

$Ulb[0]=$Uf0[6]; $Ulb[1]=$Uf1[6];
$Umb[0]=$Uf0[7]; $Umb[1]=$Uf1[7];
$Urb[0]=$Uf0[8]; $Urb[1]=$Uf1[8];
}
##==============================
```

2.12 rrSu

```
sub rrSu{
## middle slice rotation (side only 12 facelets)
## s = side; 0=colour, 1= number
## make the post rotation permutation
##-----------SIDE-------------------
@SUs0=($Lrm[0],$Lmm[0],$Llm[0],
```

```
      $Brm[0],$Bmm[0],$Blm[0],
      $Rrm[0],$Rmm[0],$Rlm[0],
      $Frm[0],$Fmm[0],$Flm[0]);

@SUs1=($Lrm[1],$Lmm[1],$Llm[1],
      $Brm[1],$Bmm[1],$Blm[1],
      $Rrm[1],$Rmm[1],$Rlm[1],
      $Frm[1],$Fmm[1],$Flm[1]);


$Brm[0]=$SUs0[0]; $Brm[1]=$SUs1[0];
$Bmm[0]=$SUs0[1]; $Bmm[1]=$SUs1[1];
$Blm[0]=$SUs0[2]; $Blm[1]=$SUs1[2];

$Rrm[0]=$SUs0[3]; $Rrm[1]=$SUs1[3];
$Rmm[0]=$SUs0[4]; $Rmm[1]=$SUs1[4];
$Rlm[0]=$SUs0[5]; $Rlm[1]=$SUs1[5];

$Frm[0]=$SUs0[6]; $Frm[1]=$SUs1[6];
$Fmm[0]=$SUs0[7]; $Fmm[1]=$SUs1[7];
$Flm[0]=$SUs0[8]; $Flm[1]=$SUs1[8];

$Lrm[0]=$SUs0[9];  $Lrm[1]=$SUs1[9];
$Lmm[0]=$SUs0[10]; $Lmm[1]=$SUs1[10];
$Llm[0]=$SUs0[11]; $Llm[1]=$SUs1[11];
}
##==============================
```

2.13 rrDp

```
sub rrDp{

## Dpwn Face anticlockwise rotation (side and face)
## s = side; 0=colour, 1= number
## make the post rotation permutation
##--------------SIDE----------------
@DPs0=($Lrb[0],$Lmb[0],$Llb[0],
       $Brb[0],$Bmb[0],$Blb[0],
```

```
       $Rrb[0],$Rmb[0],$Rlb[0],
       $Frb[0],$Fmb[0],$Flb[0]);

@DPs1=($Lrb[1],$Lmb[1],$Llb[1],
       $Brb[1],$Bmb[1],$Blb[1],
       $Rrb[1],$Rmb[1],$Rlb[1],
       $Frb[1],$Fmb[1],$Flb[1]);

$Brb[0]=$DPs0[0]; $Brb[1]=$DPs1[0];
$Bmb[0]=$DPs0[1]; $Bmb[1]=$DPs1[1];
$Blb[0]=$DPs0[2]; $Blb[1]=$DPs1[2];

$Rrb[0]=$DPs0[3]; $Rrb[1]=$DPs1[3];
$Rmb[0]=$DPs0[4]; $Rmb[1]=$DPs1[4];
$Rlb[0]=$DPs0[5]; $Rlb[1]=$DPs1[5];

$Frb[0]=$DPs0[6]; $Frb[1]=$DPs1[6];
$Fmb[0]=$DPs0[7]; $Fmb[1]=$DPs1[7];
$Flb[0]=$DPs0[8]; $Flb[1]=$DPs1[8];

$Lrb[0]=$DPs0[9];  $Lrb[1]=$DPs1[9];
$Lmb[0]=$DPs0[10]; $Lmb[1]=$DPs1[10];
$Llb[0]=$DPs0[11]; $Llb[1]=$DPs1[11];

##---------------Down FACE-------------------
## f = face; 0=colour, 1= number

@DPf0=($Dlt[0], $Dlm[0], $Dlb[0],     $Dmt[0], $Dmm[0], $Dmb[0],     $Drt[0], $Drm[0], $Drb[0]);
@DPf1=($Dlt[1], $Dlm[1], $Dlb[1],     $Dmt[1], $Dmm[1], $Dmb[1],     $Drt[1], $Drm[1], $Drb[1]);

$Dlb[0]=$DPf0[0]; $Dlb[1]=$DPf1[0];
$Dmb[0]=$DPf0[1]; $Dmb[1]=$DPf1[1];
$Drb[0]=$DPf0[2]; $Drb[1]=$DPf1[2];

$Dlm[0]=$DPf0[3]; $Dlm[1]=$DPf1[3];
$Dmm[0]=$DPf0[4]; $Dmm[1]=$DPf1[4];
$Drm[0]=$DPf0[5]; $Drm[1]=$DPf1[5];
```

```
$Dlt[0]=$DPf0[6]; $Dlt[1]=$DPf1[6];
$Dmt[0]=$DPf0[7]; $Dmt[1]=$DPf1[7];
$Drt[0]=$DPf0[8]; $Drt[1]=$DPf1[8];
}
##============================
```

2.14 rrF

```
sub rrF{

## do the Fside transform (side and face)
## s = side; 0=colour, 1= number
## -----------SIDE-----------------
@Fs0=($Lrb[0],$Lrm[0],$Lrt[0],
      $Ulb[0],$Umb[0],$Urb[0],
      $Rlt[0],$Rlm[0],$Rlb[0],
      $Drt[0],$Dmt[0],$Dlt[0]);

@Fs1=($Lrb[1],$Lrm[1],$Lrt[1],
      $Ulb[1],$Umb[1],$Urb[1],
      $Rlt[1],$Rlm[1],$Rlb[1],
      $Drt[1],$Dmt[1],$Dlt[1]);

$Ulb[0]=$Fs0[0]; $Ulb[1]=$Fs1[0];
$Umb[0]=$Fs0[1]; $Umb[1]=$Fs1[1];
$Urb[0]=$Fs0[2]; $Urb[1]=$Fs1[2];

$Rlt[0]=$Fs0[3]; $Rlt[1]=$Fs1[3];
$Rlm[0]=$Fs0[4]; $Rlm[1]=$Fs1[4];
$Rlb[0]=$Fs0[5]; $Rlb[1]=$Fs1[5];

$Drt[0]=$Fs0[6]; $Drt[1]=$Fs1[6];
$Dmt[0]=$Fs0[7]; $Dmt[1]=$Fs1[7];
$Dlt[0]=$Fs0[8]; $Dlt[1]=$Fs1[8];

$Lrb[0]=$Fs0[9];  $Lrb[1]=$Fs1[9];
$Lrm[0]=$Fs0[10]; $Lrm[1]=$Fs1[10];
$Lrt[0]=$Fs0[11]; $Lrt[1]=$Fs1[11];
```

```
## -------Front FACE--------------------
## f = face; 0=colour, 1= number

@Lf0=($Flb[0], $Flm[0], $Flt[0],     $Fmb[0], $Fmm[0], $Fmt[0],     $Frb[0], $Frm[0], $Frt[0]);
@Lf1=($Flb[1], $Flm[1], $Flt[1],     $Fmb[1], $Fmm[1], $Fmt[1],     $Frb[1], $Frm[1], $Frt[1]);

$Flt[0]=$Lf0[0]; $Flt[1]=$Lf1[0];
$Fmt[0]=$Lf0[1]; $Fmt[1]=$Lf1[1];
$Frt[0]=$Lf0[2]; $Frt[1]=$Lf1[2];

$Flm[0]=$Lf0[3]; $Flm[1]=$Lf1[3];
$Fmm[0]=$Lf0[4]; $Fmm[1]=$Lf1[4];
$Frm[0]=$Lf0[5]; $Frm[1]=$Lf1[5];

$Flb[0]=$Lf0[6]; $Flb[1]=$Lf1[6];
$Fmb[0]=$Lf0[7]; $Fmb[1]=$Lf1[7];
$Frb[0]=$Lf0[8]; $Frb[1]=$Lf1[8];
}
##==============================
```


2.15 rrSf

```
sub rrSf{

## do the Fs  transform (side only)
## s = side; 0=colour, 1= number
##----------SIDE---------------
@SFs0=($Lmb[0],$Lmm[0],$Lmt[0],
       $Ulm[0],$Umm[0],$Urm[0],
       $Rmt[0],$Rmm[0],$Rmb[0],
       $Drm[0],$Dmm[0],$Dlm[0]);

@SFs1=($Lmb[1],$Lmm[1],$Lmt[1],
       $Ulm[1],$Umm[1],$Urm[1],
       $Rmt[1],$Rmm[1],$Rmb[1],
       $Drm[1],$Dmm[1],$Dlm[1]);
```

```
$Ulm[0]=$SFs0[0]; $Ulm[1]=$SFs1[0];
$Umm[0]=$SFs0[1]; $Umm[1]=$SFs1[1];
$Urm[0]=$SFs0[2]; $Urm[1]=$SFs1[2];

$Rmt[0]=$SFs0[3]; $Rmt[1]=$SFs1[3];
$Rmm[0]=$SFs0[4]; $Rmm[1]=$SFs1[4];
$Rmb[0]=$SFs0[5]; $Rmb[1]=$SFs1[5];

$Drm[0]=$SFs0[6]; $Drm[1]=$SFs1[6];
$Dmm[0]=$SFs0[7]; $Dmm[1]=$SFs1[7];
$Dlm[0]=$SFs0[8]; $Dlm[1]=$SFs1[8];

$Lmb[0]=$SFs0[9];  $Lmb[1]=$SFs1[9];
$Lmm[0]=$SFs0[10]; $Lmm[1]=$SFs1[10];
$Lmt[0]=$SFs0[11]; $Lmt[1]=$SFs1[11];
}
##==============================
```


2.16 rrBp

```
sub rrBp{

## Back rotation anticlockwise (side + face)
## do the Bp side transform
## s = side; 0=colour, 1= number
## --------------Side-----------------
@BPs0=($Llb[0],$Llm[0],$Llt[0],
       $Ult[0],$Umt[0],$Urt[0],
       $Rrt[0],$Rrm[0],$Rrb[0],
       $Drb[0],$Dmb[0],$Dlb[0]);

@BPs1=($Llb[1],$Llm[1],$Llt[1],
       $Ult[1],$Umt[1],$Urt[1],
       $Rrt[1],$Rrm[1],$Rrb[1],
       $Drb[1],$Dmb[1],$Dlb[1]);

$Ult[0]=$BPs0[0]; $Ult[1]=$BPs1[0];
$Umt[0]=$BPs0[1]; $Umt[1]=$BPs1[1];
```

```
$Urt[0]=$BPs0[2]; $Urt[1]=$BPs1[2];

$Rrt[0]=$BPs0[3]; $Rrt[1]=$BPs1[3];
$Rrm[0]=$BPs0[4]; $Rrm[1]=$BPs1[4];
$Rrb[0]=$BPs0[5]; $Rrb[1]=$BPs1[5];

$Drb[0]=$BPs0[6]; $Drb[1]=$BPs1[6];
$Dmb[0]=$BPs0[7]; $Dmb[1]=$BPs1[7];
$Dlb[0]=$BPs0[8]; $Dlb[1]=$BPs1[8];

$Llb[0]=$BPs0[9];  $Llb[1]=$BPs1[9];
$Llm[0]=$BPs0[10]; $Llm[1]=$BPs1[10];
$Llt[0]=$BPs0[11]; $Llt[1]=$BPs1[11];

##-----------------Back FACE-------------
## do the B face transform (in rows: 1,2,3/4,5,6/7,8,9)
## f = face; 0=colour, 1= number
@BPf0=($Brb[0], $Brm[0], $Brt[0],     $Bmb[0], $Bmm[0], $Bmt[0],     $Blb[0], $Blm[0], $Blt[0]);

@BPf1=($Brb[1], $Brm[1], $Brt[1],     $Bmb[1], $Bmm[1], $Bmt[1],     $Blb[1], $Blm[1], $Blt[1]);

$Brt[0]=$BPf0[0]; $Brt[1]=$BPf1[0];
$Bmt[0]=$BPf0[1]; $Bmt[1]=$BPf1[1];
$Blt[0]=$BPf0[2]; $Blt[1]=$BPf1[2];

$Brm[0]=$BPf0[3]; $Brm[1]=$BPf1[3];
$Bmm[0]=$BPf0[4]; $Bmm[1]=$BPf1[4];
$Blm[0]=$BPf0[5]; $Blm[1]=$BPf1[5];

$Brb[0]=$BPf0[6]; $Brb[1]=$BPf1[6];
$Bmb[0]=$BPf0[7]; $Bmb[1]=$BPf1[7];
$Blb[0]=$BPf0[8]; $Blb[1]=$BPf1[8];
}
##====================end==================================
```

—— END ——