

Beginning Perl Programming (X52.9543.001)**An Implementation of Homework 6**

Here's **one** implementation of Homework #6. The **bold** items indicate changes from map_info1.pl:

map_info2.pl

```
#!/App/Perl/bin/perl -w
#
# map_info2.pl
#
# Perl script to output the information for a map grid location.
#
# (c) Copyright 2000 by Clint Goss, All Rights Reserved.

use strict 'vars';

# Pick up our options

use Getopt::Std;
use vars qw($opt_d $opt_D $opt_f $opt_h $opt_p $opt_s $opt_v);
if (! getopts("dDf:hp:s:v")) { &usage(); exit; }

sub usage {
    print <<END_OF_HELP;
usage: $0 [options]

Produce the information for a map grid location.

Options:
-d          Turn on moderate debugging
-D          Turn on very verbose debugging
-f FILE    Specify the map grid database file. Default is map_grid.db
-h          Print this help message
-p RxC     Position (required!).
            The R and C are Row and column positions (zero-based).
            For example, -p 0x17 indicates the first row, 18th column.
-s RxC    Size of map to be displayed, in #ROWSx#COLS format.
            The default is -s 2x2.
-v          Turn on verbose reporting mode
END_OF_HELP
}

if ($opt_h) { &usage(); exit; }

# Parameters to this program

my ($debug) = $opt_D ? 2 : ($opt_d ? 1 : 0);
my ($mapGridFilename) = $opt_f ? $opt_f : "map_grid.db";
my ($verbose) = $opt_v;

# Parse the required -p position parameter

my ($position) = $opt_p;
if (! $position) { &usage(); exit; }
my ($rowPosition, $colPosition) = split ("x", $position);
if (! $rowPosition) { &usage(); exit; }
if (! $colPosition) { &usage(); exit; }
```

```

# Parse the options -s size parameter

my ($size) = $opt_s ? $opt_s : "2x2";
my ($sizeRows, $sizeCols) = split ("x", $size);
if (! $sizeRows) { &usage(); exit; }
if (! $sizeCols) { &usage(); exit; }

# Open the input file.

open (MAP_GRID, "<$mapGridFilename") || die "Can't open $mapGridFilename: $!";

# Pick up the number of rows.

my ($header);
read (MAP_GRID, $header, 2) || die "Can't read $mapGridFilename: $!";
my ($rowCount, $colCount) = unpack ("cc", $header);

# Standard sizes in the file.

my ($headerByteCount) = 10;
my ($bytesPerFileName) = 20;

# Loop over all rows and column to extract the file names.

my ($r, $c);
foreach $r (0 .. $sizeRows-1) {

    my (@rowFileNames);

    foreach $c (0 .. $sizeCols-1) {

        push @rowFileNames,
            &mapFileName ($rowPosition + $r, $colPosition + $c,
                $rowCount, $colCount);
    }

    # Produce the output for this row of file names

    printf "Map files: %s\n", join (" ", @rowFileNames);
}

# Map movement references
# Strategy: Calculate the furthest distant map grid reference which could
# be off the map. If it is, emit an "off-map" printout.
#
# Output the movement grid references for each of the direction of movement.
# Note that the -half movement will actually move less than half a map movement,
# in the case where $rowCount or $colCount are odd.

&emitMovement ("Left", rowCount => $rowCount, colCount => $colCount,
    furthestRow => $rowPosition + $sizeRows - 1,
    furthestCol => $colPosition - $sizeCols,
    moveRow => $rowPosition,
    moveCol => $colPosition - $sizeCols);

&emitMovement ("Right", rowCount => $rowCount, colCount => $colCount,
    furthestRow => $rowPosition + $sizeRows - 1,
    furthestCol => $colPosition + $sizeCols - 1 + $sizeCols,
    moveRow => $rowPosition,
    moveCol => $colPosition + $sizeCols);

```

```

if ($sizeCols > 1) {
    &emitMovement ("Left-half", rowCount => $rowCount, colCount => $colCount,
        furthestRow => $rowPosition + $sizeRows - 1,
        furthestCol => $colPosition - int($sizeCols/2),
        moveRow => $rowPosition,
        moveCol => $colPosition - int($sizeCols/2));

    &emitMovement ("Right-half", rowCount => $rowCount, colCount => $colCount,
        furthestRow => $rowPosition + $sizeRows - 1,
        furthestCol => $colPosition + int($sizeCols/2) - 1 + $sizeCols,
        moveRow => $rowPosition,
        moveCol => $colPosition + int($sizeCols/2));
}

&emitMovement ("Up", rowCount => $rowCount, colCount => $colCount,
    furthestRow => $rowPosition - $sizeRows,
    furthestCol => $colPosition,
    moveRow => $rowPosition - $sizeRows,
    moveCol => $colPosition);

&emitMovement ("Down", rowCount => $rowCount, colCount => $colCount,
    furthestRow => $rowPosition + $sizeRows - 1 + $sizeRows,
    furthestCol => $colPosition,
    moveRow => $rowPosition + $sizeRows,
    moveCol => $colPosition);

if ($sizeRows > 1) {
    &emitMovement ("Up-half", rowCount => $rowCount, colCount => $colCount,
        furthestRow => $rowPosition - int($sizeRows/2),
        furthestCol => $colPosition,
        moveRow => $rowPosition - int($sizeRows/2),
        moveCol => $colPosition);

    &emitMovement ("Down-half", rowCount => $rowCount, colCount => $colCount,
        furthestRow => $rowPosition + $sizeRows - 1 + int($sizeRows/2),
        furthestCol => $colPosition,
        moveRow => $rowPosition + int($sizeRows/2),
        moveCol => $colPosition);
}

close MAP_GRID;
exit;

# Read a single file name at a given row and column position from MAP_GRID.

sub mapFileName {
    my ($rowPos, $colPos, $rowCount, $colCount) = @_;

    # Handle requests for file names which are off the map.

    if (&offMap ($rowPos, $colPos, $rowCount, $colCount)) {
        return "off-map";
    }

    # Move the file pointer to the start of the file name.

    seek (MAP_GRID,
        $headerByteCount +
        (($rowPos*$colCount) + $colPos) * $bytesPerFileName,

```

```

    0);

# Read the file name and strip any blanks.

my ($filename);
read (MAP_GRID, $filename, $bytesPerFileName)
    || die "Can't read $mapGridFilename: $!";

$filename =~ s/^\s+//;
$filename =~ s/\s+$//;
return $filename;
}

# Routine which reports whether a given map reference is on the map or not.

sub onMap {
    my ($rowPos, $colPos, $rowCount, $colCount) = @_;

    # Check if any of the map bounds have been exceeded

    if (($rowPos < 0) || ($rowPos >= $rowCount) ||
        ($colPos < 0) || ($colPos >= $colCount)) {

        return undef;
    }

    return 1;
}

# Useful convenience routine

sub offMap { return !(&onMap (@_)); }

# Utility routine to print a given movement reference, as long as it is
# on the map. Please see the comments for the SIX REQUIRED NAMED ARGUMENTS
# in the body of the routine below.

sub emitMovement {
    my ($tag, %params) = @_;

    # REQUIRED count of rows and columns

    my ($rowCount) = $params{rowCount};
    my ($colCount) = $params{colCount};

    # The most extreme row and column position which could be off the map.

    my ($furthestRow) = $params{furthestRow};
    my ($furthestCol) = $params{furthestCol};

    # The map reference to go to for the movement being requested.

    my ($moveRow) = $params{moveRow};
    my ($moveCol) = $params{moveCol};

    # Test if the most extreme position is on the map,
    # and emit the appropriate output.

    if (&onMap ($furthestRow, $furthestCol, $rowCount, $colCount)) {
        printf "%s: %dx%d\n", $tag, $moveRow, $moveCol;
    }
}

```

```
    } else {  
        printf "%s: off-map\n", $tag;  
    }  
}
```