by Gary Logan

Over a quarter century ago, I memorized a Rubik's cube solution from "The Simple Solution to Rubik's Cube" by James G. Nourse.

In a nutshell, this solution is in five steps:

- Top edges
- Top corners
- Middle edges
- Bottom corners
- Bottom edges

This took me a few months. The next step, of course, was to model the solution on a computer. More to the point, I set out to write a solution in the language I knew best. APL.

What is needed is a way to represent the physical turns of the cube faces, a way to determine the current state of the cube and a way to display the state as the solution progresses.

The first attempts to define the face turns were not very productive. An attempt to impose a coordinate system was soon discarded. Then a multi-dimensional array approach was discarded. Both attempts were tossed for the same reason: they lacked simplicity.

The next approach turned out to be successful beyond my expectations. It developed from a very low tech idea. I wanted to number each of the fifty-four squares and represent the solved cube by a vector of integers.

From adhesive tape, 54 pieces were cut. Large enough to number with a pen yet small enough to still see the color of the square. Each of the pieces were attached to the cube and the squares numbered from 1 to 54. The number order is arbitrary but the scheme chosen was to number each face from left to right and top to bottom. The faces to number were chosen in Up, Left, Front, Right, Back, Down order. All this numbering was done on a solved cube. See Figure 1. below.

Three turns for each face were defined, along with three rotations of the entire cube. This led to twenty-one definitions with the following names:

```
up un u2 lp ln l2 fp fn f2 rp rn r2 bp bn b2
dp dn d2 l90 r90 r180
```

Each move was defined by starting with a solved cube, making the move and writing down the numbers on each

face. In this way, the 21 move vectors form a matrix with 21 rows and 54 columns.

The following expression will define the move, fp, of the front face in a positive direction.

```
      s←s[m[7;]]
```

Where s is the cube state and m is the matrix of moves. In words, the new state is assigned the current state indexed by a row of the move matrix.

This is the only computer code needed to make the moves necessary to solve the cube.

Most cube solutions use a sequence of basic face turns and cube rotations to achieve specific results along the way to a solution. Any sequence of moves can be defined in the same way the basic moves were defined. The solution under discussion uses fifteen defined sequences. For completeness, the names are:

```
 s21 s22 s23 s31 s32 s33 s41 s42 s43 s44 s51 s52 s53
 s54 s55
```

These defined sequences added fifteen rows to the move matrix. No other moves are necessary to solve the cube. For example, s42 is defined by

```
      fn dn rn dp rp fp dp
```

and executed by the expression:

```
      s←s[m[29;]]
```

It is very nice that an arbitrary sequence of moves can be executed with the same expression used for a simple face turn.

We now discuss an overview of the five steps to the solution. The first three steps are done one target square at a time. The last two steps deal with a set of cubes as a group.

Top edges: S1
Rotate the cube until the square at position 8 is incorrect. Determine target square (2,4,6 or 8). Locate the target square which can be in any one of 24 places. Use a matrix, tab1, with 24 rows that has the possible places for the target square in the first two columns and basic moves in the remaining columns to place the square where it belongs. For the first step no defined sequences are needed. Suppose the target square is in position 33, we find a row of tab1 with 33r2fnr2 and place the square with moves r2 fn r2.

Top corners: S2
Rotate the cube until the square at position 9 is incorrect. Determine the target square (1,3,7 or 9). Locate the target square which can be in 24 places (8 corners, 3 squares per corner). Matrix tab2 contains the moves to solve each possibility. The basic strategy is to move the target square to the lower right corner, then choose s21, s22 or s23 to move the square to position 9. The choice depends on the position of the target square.

Middle edges: S3
Try to find one of the target squares (15,24,33 or 42) on the bottom layer and use the proper row of tab3 to place the square in position 24. If no target square is on the bottom layer, use s33 to drop it down. Matrix tab3 has 8 rows to handle each possibility for the target square on the bottom layer. The basic strategy is to move the square from position 26 to 24 with s31 or from position 51 to 24 with s32.

Bottom corners: S4
This step has two parts, first the corners are correctly placed then oriented in part two. With a move of the down face the corners can be aligned such that exactly two or four corners correct. [At this point, there are 15,552 cases for the bottom layer.] When two corners are correct, use s41 to exchange adjacent corners or s42 to exchange diagonal corners. Moves are in tab41. [Now 2,592 bottom cases remain.]

The corners are oriented with s43 and s44. There are 7 distinct patterns formed by the bottom corner colors with 4 rotations of each pattern. Matrix tab42 has 28 rows to deal with each possibility.

Bottom edges: S5
There are 96 possible cases and matrix tab5 has solutions for each.

At this point, any scrambled cube is solved with the steps described. The key APL programs to do this are in the appendix below. The comments have been removed to save space and the solution tables are not shown.

The third objective is to display the cube on the screen and provide a way to control the moves leading to a solution. This feature depends on the screen interaction facility provided by the various APL vendors and will not be discussed for that reason. Except to say, it was easy to make a one-to-one correspondence with the cube state and the color of the fields defined on the screen.

The programs were written as a personal exercise with simplicity of solution in mind, not performance. It takes around 49 seconds to scramble and solve the cube 10,000 times. The chip is a ten year old Pentium III.

Figure 1.

```
                Back
                45 44 43
                42 41 40
                39 38 37
      Left      Up       Right
      16 13 10  01 02 03  30 33 36
      17 14 11  04 05 06  29 32 35
      18 15 12  07 08 09  28 31 34
                Front
                19 20 21
                22 23 24
                25 26 27
                Down
                46 47 48
                49 50 51
                52 53 54
```

Appendix:

MIX

```
execute tabdescribe[?16⍴18;⍳12]

S1;a
L0:
forward
a←(s[8 6 2 4]≠8 6 2 4)⍳1
→(a>4)/0
⍞,(4 4⍴'    l90 r180r90 ')[a;]
a←(24 2↑tab1)∧.=2 0⍕s⍳(8 6 2 4)[a]
execute 5 2⍴¯10↑,a⌿tab1
→L0

S2;a
L0:
forward
a←(s[9 7 3 1]≠9 7 3 1)⍳1
→(a>4)/0
⍞,(4 4⍴'    r90 l90 r180')[a;]
a←(24 2↑tab2)∧.=2 0⍕s⍳(9 7 3 1)[a]
execute 4 3⍴¯12↑,a⌿tab2
→L0

S3;a
L0:
forward
a←(24 33 42 15∈s[be])⍳1
```

```
→(a=5)/L1
⍞,(4 4ρ'    l90 r180r90 ')[a;]
a←s[be]⍳(24 33 42 15)[a]
execute 2 3ρ¯6↑,tab3[a;]
→L0
L1:
forward
a←(s[24 33 42 15]≠24 33 42 15)⍳1
→(a=5)/0
⍞,(4 4ρ'    l90 r180r90 ')[a;]
s33
→L0

S4;a
forward
a←46 48 54 52⍳s[18 25 46 27 34 48 36 43 54 45 16 52]
a←(24 4↑tab41)∧.=4↑1 0⍒(a<5)/a
execute 3 4ρ¯12↑,a⌿tab41
a←4 4ρ46 48 54 52 48 54 52 46 54 52 46 48 52 46 48 54
a←∨/s[tab4bc]∧.=a
execute 4 4ρ16↑,a⌿tab42

S5;a
forward
a←1 0⍒be⍳s[be]
a←(96 8↑tab5)∧.=a
execute 5 4ρ¯20↑,a⌿tab5
forward

execute rΔ;Δ
⍞⎕FX((¯1↑⍴rΔ)↑'Δ'),[⎕IO]rΔ

forward
⍞,(s[23]=14 23 32 41)⌿4 4ρ'l90     r90 r180'
```