**Design Document**

**for**

**Sudoku Solver**

# Revisions

|  |  |  |
| --- | --- | --- |
| **Date(DDMMYYYY)** | **Description** | **Author** |
| 03122016 | Initial version | Teo Tse Tsong |
|  |  |  |

# Glossary

|  |  |
| --- | --- |
| **Term** | **Meaning** |
| DIM | Dimension. As a start for Sudoku, DIM = 3. |
| box | A DIM x DIM square of integers. |
| puzzle | A puzzle is a DIM x DIM square of boxes. |
| candidate | A candidate is a box of numbers which is a possible solution for a designated box. |
| search space | A search space is a list of candidates A puzzle solution requires the search of a DIM x DIM array of search spaces for a set of candidates that do not violate the rules of the puzzle. |
| map | A scratch pad DIM x DIM array space where candidates can be tested to find a solution. |
|  |  |

A box of numbers, DIM = 3 :



A solved puzzle, DIM = 3



**Permutations**

Given two numbers, say 2 and 3, how many way are there to arrange them ? It is straightforward – only two. 2,3 and 3,2.

This is the most basic of permutations. We might even call it a single unit of permutation. There are two numbers and two ways to arrange them.

What about 3 numbers, say 2,3,4 ?

It gets a little trickier :

|  |  |  |  |
| --- | --- | --- | --- |
| Sequence | Permutations | | |
| 1 | 2 | 3 | 4 |
| 2 | 2 | 4 | 3 |
| 3 | 3 | 2 | 4 |
| 4 | 3 | 4 | 2 |
| 5 | 4 | 2 | 3 |
| 6 | 4 | 3 | 2 |

There are a total of 6 (permutations) different ways to arrange the numbers. In fact, there are always *n*! permutations for a set of *n* numbers (3x2 in this case). It can also be seen from the coloured rows (a pair at a time) that only the last two numbers switch places. This again is our unit of permutation. Between each change of colours the most significant digit changes. In fact if we always reference the original sequence, we see that in sequences 1, 3 and 5, the first digit cycles through the entire number sequence. First a 2, then a 3, then lastly 4. In sequences 1, 3 and 5, it seems the first digit from the original sequence of 2,3,4 swaps with a subsequent digit. First 3(in sequence 3) then 4(in sequence 5).

This mechanism forms the basis for writing the code to implement permutation. In fact the structure is recursive.

We take the example of 4 numbers – 2,3,4 and 5.

The permutations are :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sequence** | | **Permutations** | | | |
| Permutate 3,4,5  With 2 in front | 1 | 2 | 3 | 4 | 5 |
| 2 | 2 | 3 | 5 | 4 |
| 3 | 2 | 4 | 3 | 5 |
| 4 | 2 | 4 | 5 | 3 |
| 5 | 2 | 5 | 3 | 4 |
| 6 | 2 | 5 | 4 | 3 |
| Permutate 2,4,5  with 3 in front | 7 | 3 | 2 | 4 | 5 |
| 8 | 3 | 2 | 5 | 4 |
| 9 | 3 | 4 | 2 | 5 |
| 10 | 3 | 4 | 5 | 2 |
| 11 | 3 | 5 | 2 | 4 |
| 12 | 3 | 5 | 4 | 2 |
| Permutate 2,3,5  with 4 in front | 13 | 4 | 2 | 3 | 5 |
| 14 | 4 | 2 | 5 | 3 |
| 15 | 4 | 3 | 2 | 5 |
| 16 | 4 | 3 | 5 | 2 |
| 17 | 4 | 5 | 2 | 3 |
| 18 | 4 | 5 | 3 | 2 |
| Permutate 2,3,4  with 5 in front | 19 | 5 | 2 | 3 | 4 |
| 20 | 5 | 2 | 4 | 3 |
| 21 | 5 | 3 | 2 | 4 |
| 22 | 5 | 3 | 4 | 2 |
| 23 | 5 | 4 | 2 | 3 |
| 24 | 5 | 4 | 3 | 2 |

There are 4! = 4 x 3 x 2 x 1 = 24 permutations or different ways of arranging 4 numbers. Again we see our unit permutation of two numbers (rows of same colour), and for each group of six rows, only the last three digits permutates. At the end of each group of six rows, the most significant digit swaps with one of the other 3 digits and the cycle starts afresh. This repetitive structure can be implemented using a recursive function call in ‘C’.

int array[4] ={2,3,4,5};

…

Permutate(array, 4,0);

…

void Permutate(int array[], int ninput, int currindex)

{

copy array[] to temp[]

if more than 2 numbers to permutate (based on currindex)

loop i = currindex to ninput

swap temp[i] with temp[currindex]

Permutate(temp, ninput, currindex)

end loop

else

record current sequence

record current sequence with last two elements swapped

}