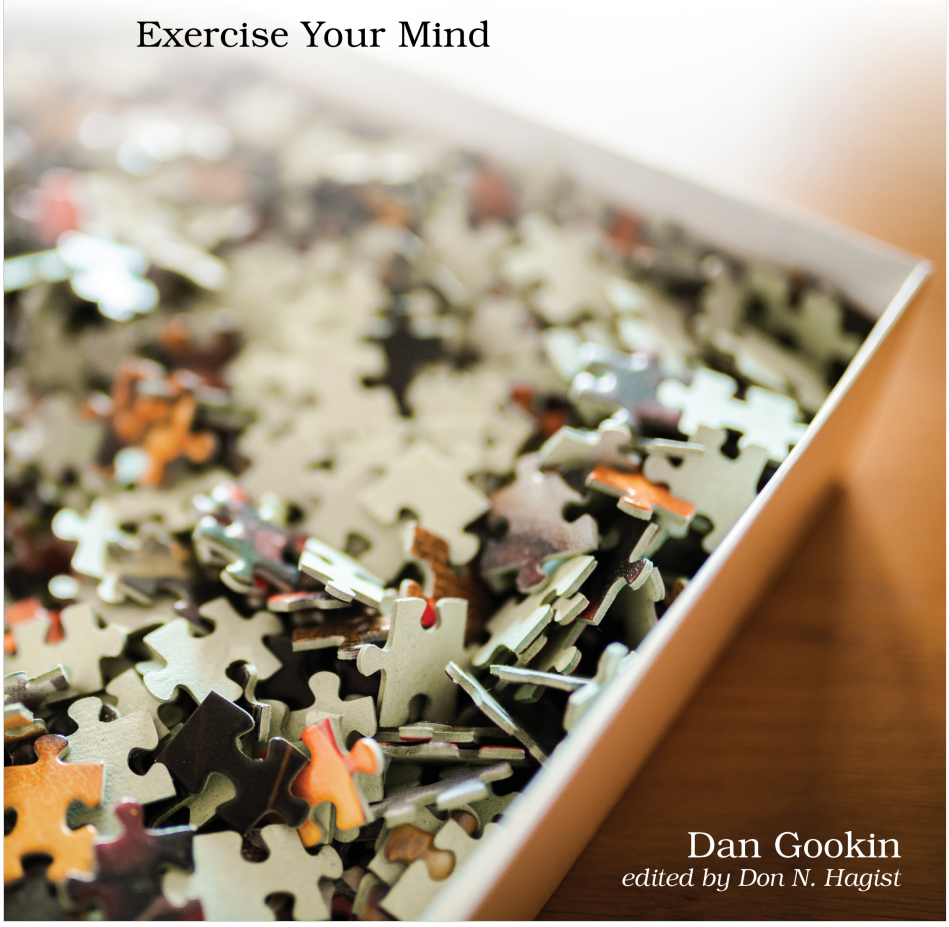
The Pragmatic Programmers

C Brain Teasers

Exercise Your Mind



Dan Gookin
edited by Don N. Hagist

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Puzzle 18

Superhero's Secret Identity

```
#include <stdio.h>

#define SIZE 5

int main()
{
        int values[SIZE] = {2, 3, 5, 8, 13};
        int *v, x;

        /* initialize the pointer */
        v = values;

        for( x=0; x<SIZE; x++ )
        {
                printf("%2d = %2d\n",
                                values[x],
                                *(v+x)
                        );
        }

        return(0);
}
```

Guess the Output

Try to guess what the output is before moving to the next page.

The code uses both array and pointer notation to output the first five Fibonacci numbers:

```
 2 =  2
 3 =  3
 5 =  5
 8 =  8
13 = 13
```

Discussion

Arrays and pointers are similar but not exactly interchangeable. This relationship explains why many beginning C programmers use arrays as a replacement for pointers. Doing so can get you into trouble. But the point of this chapter is to show the easy conversion method between array and pointer notation.

Pointers are variables that hold a memory location, an address. A pointer can be considered a base, like an array's name. In the sample code, both `values[]` and `v` reference the same chunk of memory. The similarity between array notation and pointer notation to reference the integer values appears in the `printf()` statement: `values[x]` translates to `*(v+x)`.

In both instances, variable `x` refers to an offset. It's an element number in array notation and an address offset in pointer notation: `v+x` is the base of the memory chunk (`v`) plus a given number of integer increments (`x`). This address is wrapped in parentheses and dereferenced by the `*` operator.

Tips and Traps

The relationship between array notation and pointers works only in one direction. You cannot assign a memory chunk address held in a pointer variable to an array. For example, assuming `values[]` is an array and `v` is a pointer:

```
values = v;
```

If you attempt such an assignment, the compiler gets all huffy and tosses an error in your direction.

However, it's possible to use array notation in a function where a pointer is passed as an argument.

```
void output(int *v)
{
        printf("%d\n", v[0]);
}
```

If the output() function accepts an integer pointer v, you can use v[] notation in the function to reference the memory chunk's data. Above, pointer v references a spot in memory containing integer values. Within the function, v[0] represents the first integer stored at memory location v. You could also use *(v+0) with the same result.

Further Reading

The relationship between pointers and arrays

https://www.w3schools.com/c/c_pointers_arrays.php

Comparing pointers and arrays

https://www.codingninjas.com/codestudio/library/difference-between-arrays-and-pointers

Good Q&A on pointers and arrays

https://c-faq.com/aryptr/