A Two-Fisted Algorithm For Moving Data In Tight Places

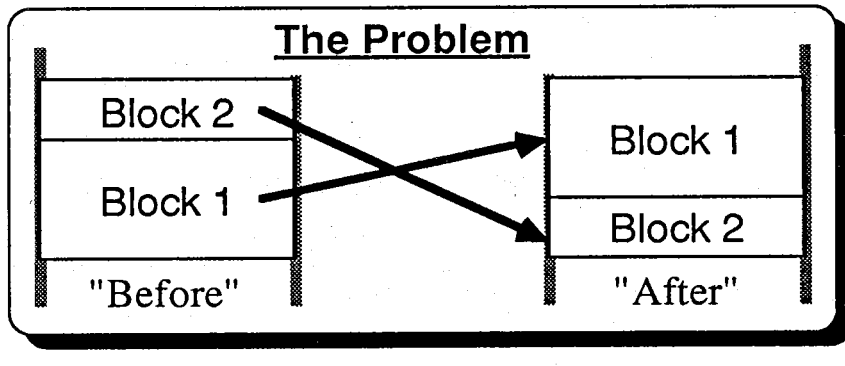
by Walt Pawley, Wump Research & Company

Some years ago, I invented this technique for exchanging the order of two arbitrarily sized, adjacent blocks of data for use in a text editor. I assumed, probably rightly, that the algorithm was well known; until recently, when informal discussions with numerous people knowledgeable about such matters led me to believe that it might not be. Unlike other algorithms that came up in these discussions, this "two-fisted" approach fetchs and stores each datum only once. While the overhead required to run the two-fisted algorithm is somewhat more than some others, it can greatly improve speed when the fetching and storing of the data is relatively costly. Applications include such things as text manipulation, heap management or disk de-thrashing. Forth code for the two-fisted block exhange algorithm is included as an appendix.

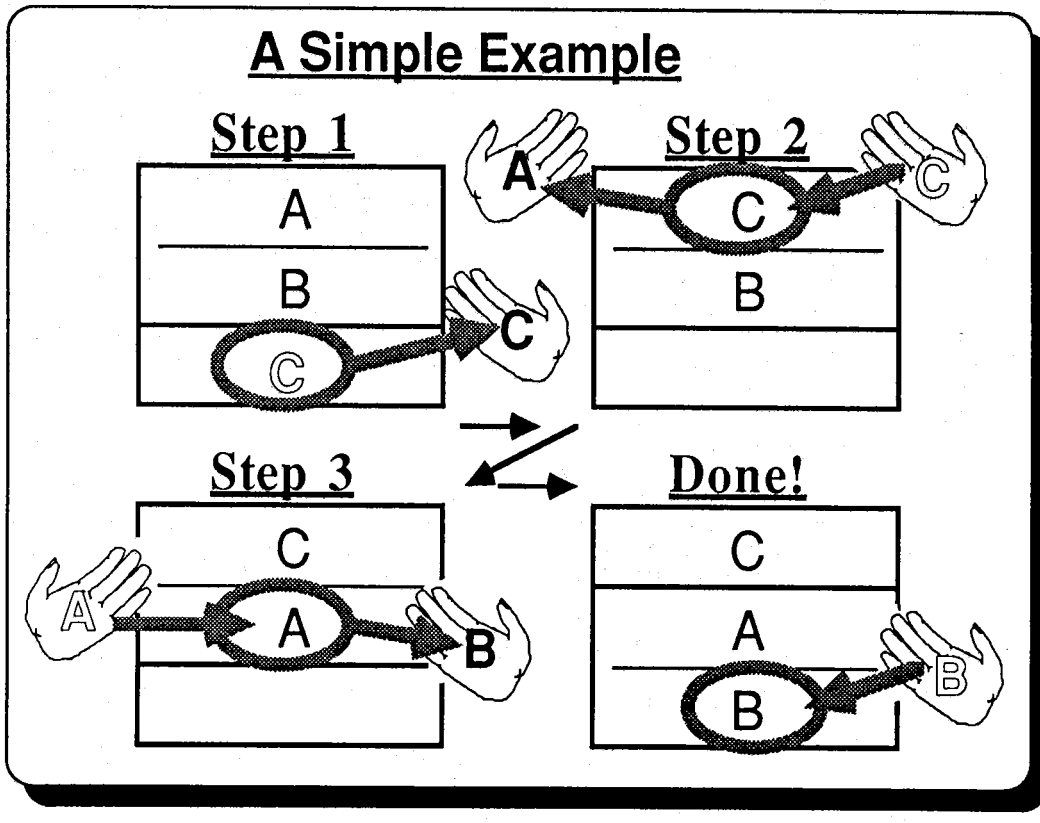
Many years ago I was working in a sawmill using a primitive developement system. I had to use a text editor named **TIDE**. TIDE's performance and design were at least as backwards as its name. TIDE was so exasperating that I vowed to write my own text editor, " ... in my spare time". I carefully designed my editor on paper, keeping copious notes in a looseleaf binder. I wanted to be able to pack meaningful data into "all" of memory and still be able to function. This necessitated some "clever" mechanisms including a garbage collection algorithm that came to me in a flash of inspiration. It was a stroke of genius. I was careful to record it in my notebook.

Well, it was about this time when events in the sawmill business erupted and erased all my spare time. I had to set work on my editor aside to do some work in the real world. My colleagues and I labored many months, but finally some of that ellusive spare time returned and I went back to my text editor right away. After all, I'd had to spend those months using TIDE! That's when I ran into a slight problem. My nice notebook had vanished! I discovered, also, that my inspirational stroke of genius had likewise disappeared. While most of the design for my editor came rushing back to me, to be frenetically recorded once again, I could not flash on garbage collection. In fact, for quite a while, I could not solve the garbage collection problem at all. But I did not give up and, after a while, the problem boiled down to this:

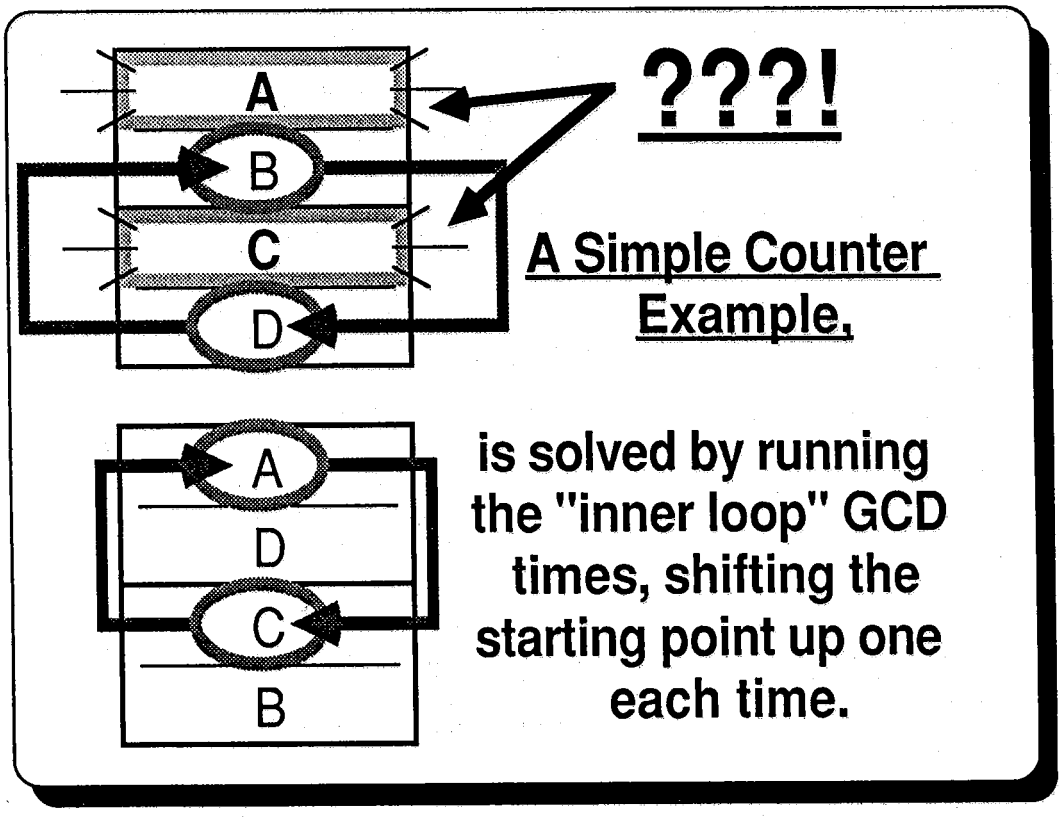
Given two blocks of data, adjacent to one another in memory and of arbitrary sizes, how can they be swapped without using large amounts of "unused" memory? It would also be nice if it could be done without using very much time.



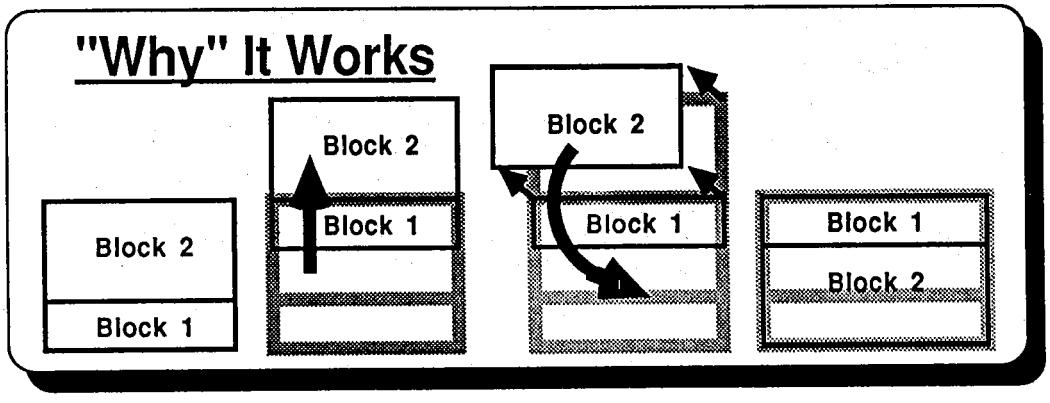
I did not get to spend much time on the second part of the problem, because I couldn't see how to solve the first part. I drew diagram after diagram and argued with myself about it day after day. I completely missed the beautifully simple algorithm that splits each block in half, exchanges bytes around the center of each block and then exchanges bytes around the center of the pair. This must be some sort of standard, because many people have subsequently told me about it. Anyway, one day while mired in my quandary, I was gesticulating about proposed events. I "picked up" the first byte in one fist , saying to myself, "It goes there." But there was already a byte there. So I "picked up" this in-the-way byte in my other fist to get it out of the way. Then I was able to put the first byte where it belonged (which was a good thing because I'd run out of hands). I knew where the second byte came from, and came to the conclusion that regardless of which of the two blocks it had come from, I could calculate where it had to go to be in the right place. Doing this, I found another byte in the way again. So I used my free fist to pick it up and get it out of the second byte's way. After a little reflection, this juggling act looked pretty good to me. All I had to do was use one fist to hold data out of the way while I moved data into place with the other fist until, finally, a byte ended up where the first byte came from. I tried all kinds of examples. It worked!



Or at least it seemed to work. After experimenting with all those examples that "proved" that it worked, I managed to generate a counter example. The bouncy path through the data didn't necessarily move all of the data by the time the first element got replaced. Again I didn't give up. I kept fooling with examples, not seeing the forest for the trees. Finally some of the nature of the problem osmosized through my thick skull. When the G.C.D. (Greatest Common Divisor) of the block sizes was not one, the path closed before all the data was moved. In addition, the rest of the data could be moved by tracing out the same path, shifted up one location each time. The number of paths to follow was the G.C.D. of the block sizes.



With this information in hand, I was able to successfully complete my text editor, which I still use to this day. I started thinking about how to write this paper on April Fool's Day. How appropriate! All these years I'd felt that this algorithm wasn't really obvious and, dispite the fact that I was confident that it worked, I never really understood "why" it worked. If you can imagine sliding both blocks of data up in memory until the first block is in its final resting place, you can see that the second block could then be lifted and placed in its new home. Therefore, the index, relative to the start of the first block, of any datum's destination is simply its source index, plus the size of the second block, taken modulo the sum of the block sizes.



```
(   ---     APPENDIX - THE "TWO-FISTED" BLOCK SWAPPING ALGORITHM   ---     )
( 20APR87    WMP   )

    0 VARIABLE SIZE2                ( THE SIZE OF THE UPPER BLOCK )
    0 VARIABLE SIZE1SIZE2+          ( THE SUM OF THE SIZES OF BOTH BLOCKS )
    0 VARIABLE THEBOTTOM            ( BEGINNING OF BLOCK 1 )
    0 VARIABLE THETOP               ( END OF BLOCK 2 )

: FIRSTFIST ( ... - BYTE ADDR    STARTS A LOOP )
    THEBOTTOM @ DUP C@ SWAP
;

: NEXTPLACE ( ADDR - ADDR          COMPUTES LOCATION IN SWAPPED BLOCK )
    SIZE2 @ +                       ( WHERE IT GOES, UNWRAPPED )
    THETOP @ OVER <                 ( CHECK FOR WRAP AROUND REQUIRED )
    IF
        SIZE1SIZE2+ @ -             ( WRAP THE ADDRESS AROUND )
    THEN
;

: SWAPFISTS   ( BYTE ADDR - BYTE ADDR     SWAP BYTE AT ADDR WITH BYTE ON STACK )
    DUP C@                          ( GET "IN-THE-WAY" BYTE )
    SWAP ROT OVER C!                ( PUT IN SWAPPED BYTE )
;

: GCD       ( X Y - GCD[X,Y]     ASSUMES BOTH X & Y ARE POSITIVE )
    BEGIN               ( THE G.C.D. IS THE GREATEST COMMON DIVISOR )
        OVER MOD -DUP           ( IS REMAINDER ZERO? )
    WHILE
        SWAP                    ( NO.  DIVIDE DIVISOR BY REMAINDER )
    REPEAT
;

: SWAPBLOCKS ( ADDR SIZE1 SIZE2 - ... SWAPS TWO ADJACENT BLOCKS )
    DUP                 SIZE2 !     ( INITIALIZE VARIABLES )
    + DUP         SIZE1SIZE2+ !
    OVER + 1-          THETOP !
                    THEBOTTOM !
    SIZE2 @ SIZE1SIZE2+ @ GCD 0   ( HOW MANY OUTER LOOPS )
    DO
        FIRSTFIST                   ( GET FIRST INNER LOOP BYTE )
        BEGIN                   ( THE INNER LOOP )
            NEXTPLACE               ( FIND OUT WHERE THIS BYTE GOES )
            THEBOTTOM @ OVER <      ( SEE IF INNER LOOP IS DONE )
        WHILE
            SWAPFISTS               ( PICK UP A BYTE.  PUT ONE DOWN )
        REPEAT
        C!                          ( PUT LAST INNER LOOP BYTE IN PLACE )
        1 THEBOTTOM +!              ( STARTING POINT FOR NEXT INNER LOOP )
    LOOP
;

( WRITTEN IN SS4TH - FORTH FOR INTERDATA/PERKIN ELMER/CONCURRENT COMPUTERS )
                    ( FROM WUMP RESEARCH & COMPANY )
```