

Re: Non-restoring binary square root and convergence

Source: <http://sci.tech-archive.net/Archive/sci.math/2005-03/3924.html>

From: Clint Olsen (*clint_at_olsen.net*)

Date: 03/11/05

Date: Fri, 11 Mar 2005 16:12:32 -0600

Sorry it took so long to followup on this. I have an implementation which works but exhibits the behavior I described in the original post (non-zero difference in even input).

Notes:

Compiling:

```
% cc -o sqrt -Wall -pedantic -g sqrt.c
```

If you don't want to see the debug messages, use `-DNDEBUG` as well.

About the code:

The code is written so it doesn't make any assumptions about the width of an unsigned long. It estimates this by taking the width of character and then multiplying by the sizeof the parameterized type: `sqrt_t`. I made no attempt to calculate the value bits of an unsigned long (see the ANSI C spec if you don't know what this means). It's not really necessary for this experiment, in real life this would produce the correct result for width. The odd initialization of shift was to account for some oddball machine that had an odd number of bits in an unsigned long.

I took the algorithm from "Algorithms for Extracting Square Roots and Cube Roots", Peng, 1981.

Running:

```
% sqrt 9 16
```

Usually you want to perform 16 iterations to produce a correct result for a machine with 32-bit unsigned longs. However, for the purposes of generating pseudo-random sequences on primes, I was interested in the behavior if you specified something more arbitrary, like:

```
% sqrt 2 40
```

sci.math: Re: Non-restoring binary square root and convergence

I'd appreciate any comments about correctness and efficiency if you have any.

Thanks,

-Clint

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

typedef unsigned long sqrt_t;

int main(int argc, char *argv[])
{
    sqrt_t a; /* target of sqrt */
    sqrt_t root = 0; /* root */
    sqrt_t diff; /* running difference */
    sqrt_t a_i, b_i;
    unsigned width = CHAR_BIT * sizeof(sqrt_t);
    int shift = width & 1 ? width - 1 : width - 2;
    int i;

    if (argc == 3) {
        a = strtoul(argv[1], NULL, 10);
        i = strtoul(argv[2], NULL, 10);

        /* pass 1 */
        a_i = a >> shift & 3;

        diff = a_i - 1;

        if (diff < a_i) /* overflow check */
            root = 1;

        for (shift -= 2, --i; i > 0; i--, shift -= 2) {

            if (shift >= 0) {
                a_i = a >> shift & 3;
            } else {
                a_i = 0;
            }

            b_i = diff << 2 | a_i;

            if (root & 1)
                diff = b_i - (root << 2 | 1);
            else
                diff = b_i + (root << 2 | 3);

            root <<= 1;
        }
    }
}
```

```
if (diff < b_i)
    root |= 1;

#ifdef NDEBUG
printf("root: %lu, d: %lu, shift: %d\n", root, diff, shift);
#endif
}
printf("%lu\n", root);
} else {
    fprintf(stderr, "usage: %s <target> <iterations>\n", argv[0]);
}

return EXIT_SUCCESS;
}
```