Case Study: Inline Functions Versus Normal Functions

I once worked on writing a word-processing program for a large computer manufacturer. We had a function `next_char` that was used to get the next character from the current file. It was used in thousands of places throughout the program. When we first tested the program with `next_char` written as a function, the program was unacceptably slow. Analyzing our program we found that 90 percent of the time was spent in `next_char`. So we changed it to an inline function. The speed doubled; however, our code size went up 40 percent and required a memory expansion card to work. So the speed was all right, but the size was unacceptable.

We finally had to write the routine as a function in hand-optimized assembly language to get both the size and the speed to acceptable levels.

Case Study: Optimizing a Color-Rendering Algorithm

I once was asked to optimize a program that did color rendering for a large picture. The problem was that the program took eight hours to process a single picture. This limited us to doing one picture a day.

The first thing I did was run the program on a machine with a floating-point accelerator. This brought the time down to about six hours. Next I got permission to use a high-speed RISC computer that belonged to another project but was currently sitting idle. That reduced the time to two hours.

I saved six hours solely by using faster machines. No code had changed yet.

Two fairly simple functions were being called only once from the innermost loop. Rewriting these functions as macros saved about 15 minutes.

Next I changed all the floating-point operations I could from floating-point to integer. The savings amounted to 30 minutes out of a 1:45 run.

Then I noticed the program was spending about 5 minutes reading an ASCII file containing a long list of floating-point numbers used in the conversion process. Knowing that `scanf` is an extremely expensive function, I cut the initialization process down to almost nothing by making the file binary. Total runtime was now down to 1:10.

By carefully inspecting the code and using every trick I knew, I saved another 5 minutes, leaving me 5 minutes short of my goal of an hour per run. At this point my project was refocused and the program put in mothballs for use at some future date.