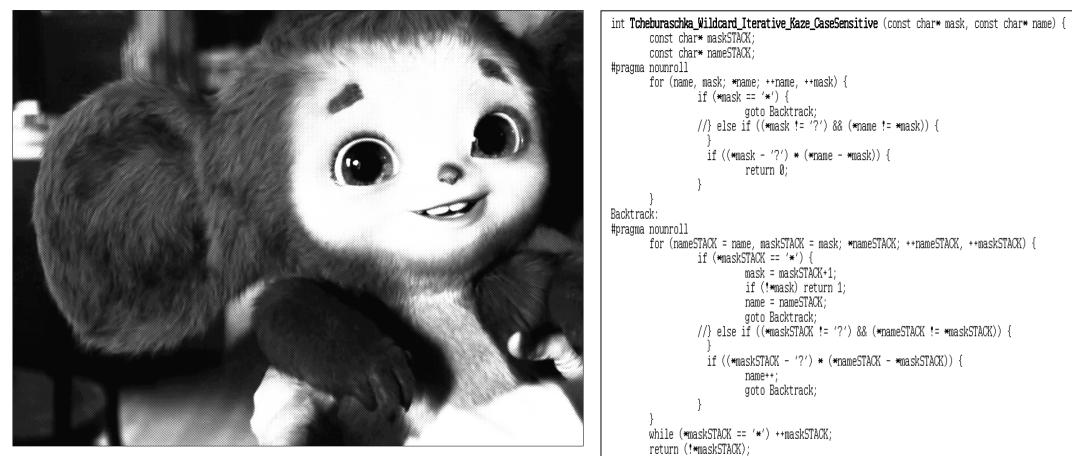
Tcheburaschka - being third fastest - in search for FASTNESS... - FASTEST wildcard search function, FOUND! - Kirk J. Krauss is the man! A short review by sammayce@sammaycammayce@sammayce@sammayce@samma



It's time for etude extraordinaire - the work of Mr.Krauss - the inventor of FASTEST wildcard matching. <u>https://github.com/kirkjkrauss/MatchingWildcards</u>

Until last night, was not aware how ahead etudes of wildcard matching have gone, first I found the superb webpage of Dogan Kurt, he in a nutshell did all, acknowledging the previous works, offering a benchmark, and offering the second fastest function. https://dogankurt.com/wildcard.html

I took his 'wildtest' benchmark and added Jack Handy's, Alessandro Cantatore and Mr.Krauss' functions, and my Tcheburaschka (based on the work of Alessandro).

The test machine's specifications are the following: OS: Windows 10 64 bit, CPU: AMD Zen2 Ryzen7 4800H. I compiled them with MinGW gcc version 11.3.0 with -O3 flag, also with CLANG 14.0.1 and Intel's ICL 19.0.0, both with -O3. We have 87 test patterns and 95 test strings; this combination makes 8265 unique pattern-string pairs. We repeat calls 1,000,000 times to get a better time resolution, and therefore every function's called **8,265,000,000 times**. You can download the test code here, **WILDTEST_wildcard_benchmark_Dogan_Kurt_modified_by_Kaze.zip** 353 KB (362,230 bytes): <u>https://qb64phoenix.com/forum/attachment.php?aid=1154</u>

[CPU: AMD Zen2 Ryzen7 4800H, @2.9GHz, Max. Boost Clock Up to 4.2GHz]

+	+ CLANG 14.0.1, -03	Intel's ICL 19.0.0, /03	MinGW gcc 11.3.0, -O3
<pre>/ Dogan Kurt's 'Antimalware', 2016, Iterative (wild_iterative) / Dogan Kurt's 'Antimalware', 2016, Iterative Optimised (wild_iterative_opt) / Tcheburaschka_r3, 2022, (Tcheburaschka_Wildcard_Iterative_Kaze_CaseSensitive) / JackHandy_Iterative, 2005, (IterativeWildcards) / Kirk J. Krauss, 2014, DrDobbs (FastWildCompare) / Alessandro Cantatore, 2003, (szWildMatch7) / Nondeterministic Finite Automaton (wild_nfa)</pre>	70.605000 s 61.322000 s	102.610000 s 74.243000 s 76.161000 s 90.872000 s 48.109000 s 85.986000 s	83.398000 s 66.538000 s 127.717000 s 70.156000 s <u>51.018000 s </u> 121.965000 s

[Note1: All functions returned 1,075,000,000 Matches - that is TRUEs, kinda means they passed the quality test, no, really, I printed all the 1's and 0's after each run - the sequences matched.] [Note2: It is well-known that Maximum Turbo Modes are maintained for some 15-30 seconds, so it is good that each function takes 30+ seconds, to emulate some 8 billion real-world searches.]

It takes a can of beer to contemplate and discern what-is-what, the fluctuations across compilers are scary. Yet, Mr.Krauss kicks asses left and right.

Thanks a lot, Mr.Krauss, for inspirational work of yours. It is both superuseful and really eye-opening how unappreciated the SPEED/CRAFTSMANSHIP is.