Tcheburaschka - being third fastest - in search for FASTNESS... - FASTEST wildcard search function, FOUND! - Kirk J. Krauss is the man!

A short review by sanmayce@sanmayce.com, 2022-Nov-28

```
int Tcheburaschka_Wildcard_Iterative_Kaze_CaseSensitive (const char* mask, const char* name) {
	const char* maskSTACK;
	const char* nameSTACK;
#pragma nounroll
	for (name, mask; *name; ++name, ++mask) {
		if (*mask == '*') {
			goto Backtrack;
		//} else if ((*mask != '?') && (*name != *mask)) {
		  }
		  if ((*mask - '?') * (*name - *mask)) {
			return 0;
		}
	}
Backtrack:
#pragma nounroll
	for (nameSTACK = name, maskSTACK = mask; *nameSTACK; ++nameSTACK, ++maskSTACK) {
		if (*maskSTACK == '*') {
			mask = maskSTACK+1;
			if (!*mask) return 1;
			name = nameSTACK;
			goto Backtrack;
		//} else if ((*maskSTACK != '?') && (*nameSTACK != *maskSTACK)) {
		  }
		  if ((*maskSTACK - '?') * (*nameSTACK - *maskSTACK)) {
			name++;
			goto Backtrack;
		}
	}
	while (*maskSTACK == '*') ++maskSTACK;
	return (!*maskSTACK);
}
```

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

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):
https://qb64phoenix.com/forum/attachment.php?aid=1154

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

Function \ Compiler	CLANG 14.0.1, -O3	Intel's ICL 19.0.0, /O3	MinGW gcc 11.3.0, -O3
Dogan Kurt's 'Antimalware', 2016, Iterative (wild_iterative)	70.605000 s	102.610000 s	83.398000 s
Dogan Kurt's 'Antimalware', 2016, Iterative Optimised (wild_iterative_opt)	61.322000 s	74.243000 s	66.538000 s
Tcheburaschka_r3, 2022, (Tcheburaschka_Wildcard_Iterative_Kaze_CaseSensitive)	72.990000 s	76.161000 s	127.717000 s
JackHandy_Iterative, 2005, (IterativeWildcards)	80.053000 s	90.872000 s	70.156000 s
Kirk J. Krauss, 2014, DrDobbs (FastWildCompare)	44.113000 s	48.109000 s	51.018000 s
Alessandro Cantatore, 2003, (szWildMatch7)	98.729000 s	85.986000 s	121.965000 s
Nondeterministic Finite Automaton (wild_nfa)	162.561000 s	200.022000 s	176.440000 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.