# C standard library should have a fuzzy way of comparing memory blocks

Rafael Santiago de Souza Netto <rafael.santiago@tutanota.com>

April 3rd, 2021

**Document:** n2684
**Audience:** WG14
**Proposal Category:** New Features
**Target Audience:** General Developers

**Abstract:**

`C` features couple of useful memory handling functions. The sense of utility behind those sharp-and-simple building blocks functions made them a kind of implicit standard on modern programming. Functions to set, copy, move and compare memory areas were replicated on newer languages since then. However, dynamic programming and IA growth has been creating a lack into the `C` memory toolbox: it lacks a fuzzy way of comparing bits from memory areas. Seeking to filling up this related gap the following proposal introduces the `memfzcmp` function.

## The problem

The lack of a native way of stochastically compare two memory blocks, sometimes can lead to clumsy workarounds and/or bloated solutions.

Imagine you need to filter some words based on a specific radix by offering some suggested topics. The default way of doing it with pure `C` would be normalize the input and then use `strstr(curr_topic, provided_word) || strstr(provided_word, curr_topic).`

Imagine you want to compare a data section from two different binary files in order to guess up if them came from the same source or not. Maybe it could be done by using some kind of perceptual hashing against the original one but if C was already featured with some way of comparing those bits, it would be much simpler and maybe it would produce a result near of a presented by a solution with several 3rd party dependencies.

## Proposal

The idea is to introduce `memfzcmp` as the fuzzy equivalent of `memcmp` in the standard library. This fuzzy compare function could be implemented by using the well-known Jaccard Coefficient as follows in *Listing 1*.

```c
double memfzcmp(const void *s1, const void *s2, size_t n) {
        double f[2][2] = { 0, 0, 0, 0 };
        const unsigned char *s1_bp = NULL, *s2_bp = NULL, *s1_bp_end = NULL;
        double jcd_coeff = 0.0;

        if (s1 == NULL || s2 == NULL || n == 0) {
                goto memfzcmp_epilogue;
        }

        if (memcmp(s1, s2, n) == 0) {
                jcd_coeff = 1.0;
                goto memfzcmp_epilogue;
        }

#define memfzcmp_get_bit_n(byte, bit) ( (((byte) >> (bit)) & 0x1) )

        s1_bp = (const unsigned char *)s1;
        s1_bp_end = (const unsigned char *)s1 + n;

        s2_bp = (const unsigned char *)s2;

        while (s1_bp != s1_bp_end) {
                f[memfzcmp_get_bit_n(s1_bp[0], 0)][memfzcmp_get_bit_n(s2_bp[0], 0)] += 1;
                f[memfzcmp_get_bit_n(s1_bp[0], 1)][memfzcmp_get_bit_n(s2_bp[0], 1)] += 1;
                f[memfzcmp_get_bit_n(s1_bp[0], 2)][memfzcmp_get_bit_n(s2_bp[0], 2)] += 1;
                f[memfzcmp_get_bit_n(s1_bp[0], 3)][memfzcmp_get_bit_n(s2_bp[0], 3)] += 1;
                f[memfzcmp_get_bit_n(s1_bp[0], 4)][memfzcmp_get_bit_n(s2_bp[0], 4)] += 1;
                f[memfzcmp_get_bit_n(s1_bp[0], 5)][memfzcmp_get_bit_n(s2_bp[0], 5)] += 1;
                f[memfzcmp_get_bit_n(s1_bp[0], 6)][memfzcmp_get_bit_n(s2_bp[0], 6)] += 1;
                f[memfzcmp_get_bit_n(s1_bp[0], 7)][memfzcmp_get_bit_n(s2_bp[0], 7)] += 1;
                s1_bp++;
                s2_bp++;
        }

        jcd_coeff = f[1][1] / (f[0][1] + f[1][0] + f[1][1]);

#undef memfzcmp_get_bit_n

memfzcmp_epilogue:

        return jcd_coeff;
}
```

*Listing 1: Proposed implementation of memfzcmp function*

## A fuzzy-diff sample by using memfzcmp

A well-simple example of how to implement a statistical diff to compare two files is presented in *Listing 2*.

## Conclusion

Since `memfzcmp` follows the simplicity idea behind memory functions, users would be able to virtually compare any kind of data statistically. From fundamental data types or structures until raw chunks of external data. As a result `C` language comparing capabilities would be enhanced by making `C` also able to natively solve more modern programming requirements/dilemmas urged by our current dynamic and data-driven world (2021). Where our computer programs must be responsive enough – in terms of reactions and performance – to external and unnormalized raw data as much as possible. As instance we could take the demand created by image, voice and signal processing that has been shifted the idea behind data equality in *Computer Programming* – sometimes only analyze it in boolean terms it is not enough anymore. `C` while a well-established programming language should provide some fresh ways of accomplish those current necessities in order to keep relevant and self-sufficient as it has been proving itself during these years.

A standard way of doing it would be important in order to ensure the best performance from a user solution to another. While a memory handling function, `memfzcmp` accomplishes all C function requirements for a memory function from C's memory toolbox: short, sharp, simple and totally generic.

```
int main(int argc, char **argv) {
	int exit_code = EXIT_FAILURE;
	FILE *fp = NULL;
	unsigned char *buf[2] = { NULL, NULL };
	size_t buf_size[2] = { 0, 0 };
	size_t b = 0;
	struct stat st = { 0 };
	double fsim = 0.0;

	if (argc >= 3) {
		for (b = 0; b < 2; b++) {
			if (stat(argv[b + 1], &st) != 0) {
				fprintf(stderr, "error: unable to access `%s`.\n", argv[b + 1]);
				goto epilogue;
			}

			buf_size[b] = st.st_size;

			if ((fp = fopen(argv[b + 1], "rb")) == NULL) {
				fprintf(stderr, "error: unable to open `%s`.\n", argv[b + 1]);
				goto epilogue;
			}

			buf[b] = (unsigned char *)malloc(buf_size[b]);

			if (buf[b] == NULL) {
				fprintf(stderr, "error: not enough memory.\n");
				goto epilogue;
			}

			fread(buf[b], 1, buf_size[b], fp);
			fclose(fp);
		}

		fsim = memfzcmp(buf[0], buf[1],
				(buf_size[0] < buf_size[1]) ? buf_size[0] : buf_size[1]);
		fprintf(stdout, "`%s` and `%s` are %.2lf%% similar.\n", argv[1], argv[2], fsim * 100);

		exit_code = EXIT_SUCCESS;
		fp = NULL;
	} else {
		fprintf(stderr, "use: %s <filepath> <filepath>\n", argv[0]);
	}

epilogue:

	if (fp != NULL) {
		fclose(fp);
	}

	if (buf[0] != NULL) {
		free(buf[0]);
	}

	if (buf[1] != NULL) {
		free(buf[1]);
	}

	return exit_code;
}
```

*Listing 2: Diff program by using the proposed memfzcmp function.*