CODE REVIEW

Doing a qsort function to compare strings, case insensitive

Asked 2 years, 9 months ago Modified 2 years, 9 months ago Viewed 813 times

3

How does the following C program look to compare to strings and return a case-insensitive sort? The following is what I have so far:

```
int main(void)
{
    char* strings[4] = {"Onus", "deacon", "Alex", "zebra"};
    for (int i=0; i<4; i++) printf("%d. %s\n", i+1, strings[i]);
    qsort(strings, 4, 8, scmp);
}

int scmp(const void *p1, const void *p2)
{

    // being a string (pointer-to-char)
```

```
with {[len_s1]=0} in VLA ?
```

```
the sort value
```

Another version extracting out the `str_lower` function would be:

```
void lower_string(char buffer[], const char* str, size_t len)
{
    char str_lower[len+1];
    str_lower[len] = 0;
    for (int i=0; str[i] != 0; i++)
        str_lower[i] = tolower(str[i]);
}
int scmp(const void *p1, const void *p2)
{
    const char* s1 = *(char**) p1;
    const char* s2 = *(char**) p2;

    char s1_lower[strlen(s1)+1];
    lower_string(s1_lower, s1, strlen(s1));

    char s2_lower[strlen(s2)+1];
    lower_string(s2_lower, s2, strlen(s2));

    return strcmp(s1_lower, s2_lower);
}
```

c

Share Improve this question Follow

edited Mar 5, 2021 at 6:50

asked Mar 5, 2021 at 6:40

David542
409 1 4 9

2 Answers

Sorted by: Highest score (default)

Firstly, thank you for providing a test program. We could make the test even better, if we make it self-checking:

```
"zebra"};
```

acters, too ("should `"A-team"` sort

f equal and almost-equal strings; not

ed with `gcc -std=c17 -Wall -Wextra -`

`ounds -Wstrict-prototypes -`

- We missed a few necessary includes for the comparison function:

```
#include <ctype.h>
#include <stdint.h>
#include <string.h>
```

And for the test program:

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

- We're assigning `char*` pointers using string literals. Whilst C allows this for historical reasons, it's dangerous, and should be avoided (because assignment through such pointers is UB, and the compiler can't spot that for you). It's easy to fix that:

  ```
  const char* strings[4] = {"Onus", "deacon", "Alex", "zebra"};
  ```

And one that GCC can't spot:

```
qsort(strings, 4, 8, scmp);
//               ^^^
```

We can't assume that a `const char*` is a particular size on the target platform. Whilst this value may be correct where you are, it's not true everywhere. Luckily, C provides the `sizeof` operator to help:

```
qsort(strings, 4, sizeof strings[0], scmp);
```

Looking in detail at the comparison function, it does a lot of extra work to make a copy of each input string (we have to traverse each string up to three times - once to find its length, once to copy and convert case, and once to compare).

l (and should) write a case-insensitive
ıd downcase only those characters we

```
        return 1;
```

```
        }
        ++s1;
        ++s2;
    }
    if (*s2) {
            /* s2 is longer */
            return -1;
    }
    /* compare equal */
    return 0;
}

/* Adapter for qsort */
int scmp(const void *p1, const void *p2)
{
    const char *const *sp1 = p1;
    const char *const *sp2 = p2;
    return ci_strcmp(*sp1, *sp2);
}
```

We can make the body of the comparer more compact (but functionally identical) with a little cleverness:

```
int ci_strcmp(const char *s1, const char *s2)
{
    for (;;) {
        int c1 = tolower(*s1++);
        int c2 = tolower(*s2++);
        int result = (c1 > c2) - (c1 < c2);
        if (result || !c1) return result;
    }
}
```

As mentioned in another answer, we need to use `unsigned char` with `tolower()` :

```
int ci_strcmp(const unsigned char *s1, const unsigned char *s2)
{
    for (;;) {
```

1

answered Mar 5, 2021 at 8:16

Toby Speight
72.1k 14 90 246

1 An annoying attribute about standard *string* functions is that their functional interface is `char *` , but the internal workings are as if `unsigned char *` . Unfortunately calling `my_strfoo1(unsigned char *)` with a `char *` can give a warning, hence often a `my_strfoo2(char *)` obliges a pointer conversion within. As much string processing is ASCII only - this issue is not common and left unaddressed. – chux - Reinstate Monica Mar 5, 2021 at 17:57

4

How does the following C program look to compare to strings and return a case-insensitive sort?

In addition to @Toby Speight fine answer:

Order consideration

Folding values to lower case gives a different sort than folding to uppercase.

Consider `"a", "A", "_"` . What order is expected? Is `'_'` before or after letters?

Function description should specify its intent.

Non-ASCII values

Values outside the 0-127 are tricky as they may be negative. `tolower()` is not defied well for most negative values.

```
const char* s1 = *(char**) p1;
...
s1_lower[i] = tolower(s1[i]); // Potential UB
```

Instead handle as `unsigned char *` .

```
                                                p1;
```

ıg between lower and upper case.

ı a Unicode one.

Candidate caseless compare

```
int scmpi(const void *p1, const void *p2) {
  const unsigned char* s1 = *(unsigned char**) p1;
  const unsigned char* s2 = *(unsigned char**) p2;

  // Keep loop simple
  while ((tolower(*s1) == tolower(*s2)) && *s2) {
    s1++;
    s2++;
  }
  int ch1 = tolower(*s1);
  int ch2 = tolower(*s2);
  return (ch1 > ch2) - (ch1 < ch2);
}
```

Speedier??

I found `my_strcmp_caseless()` about twice as fast, back-in-the-day; useful in a string heavy application.

Share Improve this answer Follow   edited Mar 5, 2021 at 18:06   answered Mar 5, 2021 at 14:15
chux - Reinstate Monica
30.5k 2 32 79

@David542 You might like Additional pitfalls to watch out for when doing case insensitive compares. – chux - Reinstate Monica Mar 5, 2021 at 18:10

thanks for this great answer. What would be an example of how a string could have an `unsigned char` in it? – David542 Mar 6, 2021 at 3:02

1 @David542 `unsigned char example[] = { 255, 0 };` . In C: "A string is a contiguous sequence of characters terminated by and including the first null character." – chux - Reinstate Monica Mar 6, 2021 at 3:33