

# Re: Need help on string manipulation

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2006-03/msg04320.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-03/msg04320.html)

---

- *From:* [liljencrantz@xxxxxxxxxx](mailto:liljencrantz@xxxxxxxxxx)
  - *Date:* 27 Mar 2006 05:56:52 -0800
- 

WaterWalk skrev:

Hello, I'm currently learning string manipulation. I'm curious about what is the favored way for string manipulation in C, especially when strings contain non-ASCII characters. For example, if substrings need be replaced, or one character needs be changed, what shall I do? Is it better to convert strings to UCS-32 before manipulation?

But on Windows, `wchar_t` is 16 bits which isn't enough for characters which can't be simply encoded using 16 bits.

On Linux, I hear `wchar_t` is 32 bit. Maybe on Linux, strings can be simply converted to `wchar_t` and then handle them without worrying? I'm not sure.

Characters represented by `wchar_t` must use one `wchar_t` per character, unlike characters using `char`, which may use a multibyte encoding. The actual size and encoding of `wchar_t` is undefined, and e.g. Dragonfly BSD uses different encodings of `wchar_t` depending on the encoding of `char` strings. If Windows uses a 16-bit `wchar_t`, you will be unable to use some newer Unicode characters, if this is a problem for you, then avoid `wchar_t`. You will not have this problem under Linux, since `glibc` uses the UCS4, which is 31-bit.

Things like being able to use `[]` to access a character with a specific index, being able to use `int:s` to iterate over a string and being able to examine a specific character without worrying about if it's a multibyte character makes life `_much_` easier.

What is a "good" way to handle all this mess? Are there any good examples? I'll be very thankful for your help.

I have written a non-trivial program called `fish` (It's a commandline

## Re: Need help on string manipulation

shell for Unix, kind of like bash or zsh) that uses wide character strings internally, you can download it from <http://roo.no-ip.org/fish/>.

The lessons I've learned from this:

\* Converting from char strings to `wchar_t` is not hard, but while C string handling functions have wide character equivalents, most other functions don't. Mixing narrow and wide character strings in the same program is a nightmare, you will end up with a maze of spaghetti that you will never be able to untangle. Don't. The best way to get around this long-term is to use a wrapper library around the functions you want. I have written a wrapper around some common Unix functions like `open`, `fopen`, `stat`, `access`, `realpath`, etc.. You can simply borrow the `wutil.c` and `wutil.h` files from fish to use with your own project if you wish, it should be easy to extend to more functions. (Provided you aren't writing commercial software – fish is GPL:ed)

\* If your program faces the user a lot, it is likely that you will be exposed to data in the wrong character set, e.g. it is not uncommon to have some filenames in Latin-1 even if your system should use UTF-8. I handle this in a way that breaks on systems that don't use a Unicode representation for `wchar_t`, but that works with 16-bit Unicode encodings. Specifically, I have a special set of conversion routines that takes bytes that failed to convert correctly and map them to a byte range inside the Unicode private use area. That way, I have a reversible representation of invalid characters, which means I can do e.g. wildcarding on filenames with invalid characters. The biggest hurdle with this method is that you have to make sure to always use your 'magical' conversion routines and not the ones supplied by the system. I hope that eventually all wide character sets will provide a private use area of some sort, it is a very useful feature.

\* Using `getopt` and `gettext` for option parsing and `i18n` also took some work, but it was doable. I used search-and-replace on the `getopt` source to make 'wgetopt', and wrote a wrapper around `getopt`.

\* Memory usage may increase by a factor of 4 or more. 90% of the allocated memory in fish is used to store strings, so the memory usage increase is significant. Because there are conversions taking place, additional memory is required to store both the narrow and wide version of a string at once.

In the end, I still think it is often worth using wide character strings, since you avoid the huge hassle of handling what could be multibyte encodings of strings. You might also consider using a string handling library that does all the evil string handling for you, though. Some things will be easier that way – others harder.

Good luck. I have found Unicode in C to be hard – but not undoable.

Re: Need help on string manipulation

—  
Axel

.