utstring: dynamic string macros for C

Troy D. Hanson <tdh@tkhanson.net>
v2.3.0, February 2021

Here's a link back to the https://github.com/troydhanson/uthash[GitHub project page].

Introduction

A set of basic dynamic string macros for C programs are included with uthash in `utstring.h`. To use these in your own C program, just copy `utstring.h` into your source directory and use it in your programs.

```
  #include "utstring.h"
```

The dynamic string supports operations such as inserting data, concatenation, getting the length and content, substring search, and clear. It's ok to put binary data into a utstring too. The string <<operations,operations>> are listed below.

Some utstring operations are implemented as functions rather than macros.

Download

To download the `utstring.h` header file,
follow the links on https://github.com/troydhanson/uthash to clone uthash or get a zip file,
then look in the src/ sub-directory.

BSD licensed

This software is made available under the
link:license.html[revised BSD license].
It is free and open source.

Platforms

The 'utstring' macros have been tested on:

 * Linux,
 * Windows, using Visual Studio 2008 and Visual Studio 2010

Usage

Declaration

The dynamic string itself has the data type `UT_string`. It is declared like,

```
  UT_string *str;
```

New and free

The next step is to create the string using `utstring_new`. Later when you're done with it, `utstring_free` will free it and all its content.

Manipulation

The `utstring_printf` or `utstring_bincpy` operations insert (copy) data into the string. To concatenate one utstring to another, use `utstring_concat`. To clear the content of the string, use `utstring_clear`. The length of the string is available from `utstring_len`, and its content from `utstring_body`. This

```
evaluates to a `char*`.  The buffer it points to is always null-terminated.
So, it can be used directly with external functions that expect a string.
This automatic null terminator is not counted in the length of the string.

Samples
~~~~~~~

These examples show how to use utstring.

.Sample 1
-------------------------------------------------------------------------------
#include <stdio.h>
#include "utstring.h"

int main() {
    UT_string *s;

    utstring_new(s);
    utstring_printf(s, "hello world!" );
    printf("%s\n", utstring_body(s));

    utstring_free(s);
    return 0;
}
-------------------------------------------------------------------------------

The next example demonstrates that `utstring_printf` 'appends' to the string.
It also shows concatenation.

.Sample 2
-------------------------------------------------------------------------------
#include <stdio.h>
#include "utstring.h"

int main() {
    UT_string *s, *t;

    utstring_new(s);
    utstring_new(t);

    utstring_printf(s, "hello " );
    utstring_printf(s, "world " );

    utstring_printf(t, "hi " );
    utstring_printf(t, "there " );

    utstring_concat(s, t);
    printf("length: %u\n", utstring_len(s));
    printf("%s\n", utstring_body(s));

    utstring_free(s);
    utstring_free(t);
    return 0;
}
-------------------------------------------------------------------------------

The next example shows how binary data can be inserted into the string. It also
clears the string and prints new data into it.

.Sample 3
-------------------------------------------------------------------------------
#include <stdio.h>
#include "utstring.h"
```

```
int main() {
    UT_string *s;
    char binary[] = "\xff\xff";

    utstring_new(s);
    utstring_bincpy(s, binary, sizeof(binary));
    printf("length is %u\n", utstring_len(s));

    utstring_clear(s);
    utstring_printf(s,"number %d", 10);
    printf("%s\n", utstring_body(s));

    utstring_free(s);
    return 0;
}
--------------------------------------------------------------------------------

[[operations]]
Reference
---------
These are the utstring operations.

Operations
~~~~~~~~~~

[width="100%",cols="50<m,40<",grid="none",options="none"]
|===============================================================================
| utstring_new(s) | allocate a new utstring
| utstring_renew(s) | allocate a new utstring (if s is `NULL`) otherwise clears
it
| utstring_free(s) | free an allocated utstring
| utstring_init(s)  | init a utstring (non-alloc)
| utstring_done(s) | dispose of a utstring (non-alloc)
| utstring_printf(s,fmt,...) | printf into a utstring (appends)
| utstring_bincpy(s,bin,len) | insert binary data of length len (appends)
| utstring_concat(dst,src) | concatenate src utstring to end of dst utstring
| utstring_clear(s) | clear the content of s (setting its length to 0)
| utstring_len(s) | obtain the length of s as an unsigned integer
| utstring_body(s) | get `char*` to body of s (buffer is always null-terminated)
| utstring_find(s,pos,str,len) | forward search from pos for a substring
| utstring_findR(s,pos,str,len) | reverse search from pos for a substring
|===============================================================================

New/free vs. init/done
~~~~~~~~~~~~~~~~~~~~~~
Use `utstring_new` and `utstring_free` to allocate a new string or free it.  If
the UT_string is statically allocated, use `utstring_init` and `utstring_done`
to initialize or free its internal memory.

Substring search
~~~~~~~~~~~~~~~~
Use `utstring_find` and `utstring_findR` to search for a substring in a
utstring.
It comes in forward and reverse varieties. The reverse search scans from the end
of
the string backward. These take a position to start searching from, measured
from 0
(the start of the utstring).  A negative position is counted from the end of
the string, so, -1 is the last position.  Note that in the reverse search, the
initial position anchors to the 'end' of the substring being searched for;
e.g., the 't' in 'cat'. The return value always refers to the offset where the
substring 'starts' in the utstring.  When no substring match is found, -1 is
returned.
```

```
For example if a utstring called `s` contains:

  ABC ABCDAB ABCDABCDABDE

Then these forward and reverse substring searches for `ABC` produce these
results:

  utstring_find(  s, -9, "ABC", 3 ) = 15
  utstring_find(  s,  3, "ABC", 3 ) =  4
  utstring_find(  s, 16, "ABC", 3 ) = -1
  utstring_findR( s, -9, "ABC", 3 ) = 11
  utstring_findR( s, 12, "ABC", 3 ) =  4
  utstring_findR( s,  2, "ABC", 3 ) =  0

"Multiple use" substring search
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
The preceding examples show "single use" versions of substring matching, where
the internal Knuth-Morris-Pratt (KMP) table is internally built and then freed
after the search. If your program needs to run many searches for a given
substring, it is more efficient to save the KMP table and reuse it.

To reuse the KMP table, build it manually and then pass it into the internal
search functions. The functions involved are:

 _utstring_BuildTable  (build the KMP table for a forward search)
 _utstring_BuildTableR (build the KMP table for a reverse search)
 _utstring_find        (forward search using a prebuilt KMP table)
 _utstring_findR       (reverse search using a prebuilt KMP table)

This is an example of building a forward KMP table for the substring "ABC", and
then using it in a search:

 long *KPM_TABLE, offset;
 KPM_TABLE = (long *)malloc( sizeof(long) * (strlen("ABC")) + 1));
 _utstring_BuildTable("ABC", 3, KPM_TABLE);
 offset = _utstring_find(utstring_body(s), utstring_len(s), "ABC", 3,
KPM_TABLE );
 free(KPM_TABLE);

Note that the internal `_utstring_find` has the length of the UT_string as its
second argument, rather than the start position. You can emulate the position
parameter by adding to the string start address and subtracting from its length.

Notes
~~~~~

1. To override the default out-of-memory handling behavior (which calls `exit(-
1)`),
   override the `utstring_oom()` macro before including `utstring.h`.
   For example,

  #define utstring_oom() do { longjmp(error_handling_location); } while (0)
  ...
  #include "utstring.h"

// vim: set nowrap syntax=asciidoc:
```