````
utstack: intrusive stack macros for C
=====================================
Arthur O'Dwyer <arthur.j.odwyer@gmail.com>
v2.3.0, February 2021

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

Introduction
------------
A set of very simple stack macros for C structures are included with
uthash in `utstack.h`.  To use these macros in your own C program, just
copy `utstack.h` into your source directory and use it in your programs.

  #include "utstack.h"

These macros support the basic operations of a stack, implemented as
an intrusive linked list. A stack supports the "push", "pop", and "count"
operations, as well as the trivial operation of getting the top element
of the stack.

Download
~~~~~~~~
To download the `utstack.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 'utstack' macros have been tested on:

 * Linux,
 * Mac OS X,
 * Windows, using Visual Studio 2008 and Visual Studio 2010

Usage
-----

Stack (list) head
~~~~~~~~~~~~~~~~~
The stack head is simply a pointer to your element structure. You can name it
anything. *It must be initialized to `NULL`*. It doubles as a pointer to the
top element of your stack.

  element *stack = NULL;

Stack operations
~~~~~~~~~~~~~~~
The only operations on a stack are O(1) pushing, O(1) popping, and
O(n) counting the number of elements on the stack. None of the provided
macros permit directly accessing stack elements other than the top element.

To increase the readability of your code, you can use the macro
`STACK_EMPTY(head)` as a more readable alternative to `head == NULL`,
and `STACK_TOP(head)` as a more readable alternative to `head`.

[width="100%",cols="50<m,40<",grid="none",options="none"]
````

| | |
|---|---|
| STACK_PUSH(stack,add); | push `add` onto `stack` |
| STACK_POP(stack,elt); | pop `stack` and save previous top as `elt` |
| STACK_COUNT(stack,tmp,count); | store number of elements into `count` |
| STACK_TOP(stack) | return `stack` |
| STACK_EMPTY(stack) | return `stack == NULL` |

The parameters shown in the table above are explained here:

stack::
  The stack head (a pointer to your element structure).
add::
  A pointer to the element structure you are adding to the stack.
elt::
  A pointer that will be assigned the address of the popped element. Need not be initialized.
tmp::
  A pointer of the same type as `elt`. Used internally. Need not be initialized.
count::
  An integer that will be assigned the size of the stack. Need not be initialized.

## Example

This example program reads names from a text file (one name per line), and pushes each name on the stack; then pops and prints them in reverse order.

.A stack of names

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utstack.h"

#define BUFLEN 20

typedef struct el {
    char bname[BUFLEN];
    struct el *next;
} el;

el *head = NULL; /* important- initialize to NULL! */

int main(int argc, char *argv[]) {
    el *elt, *tmp;

    char linebuf[sizeof el->bname];
    int count;
    FILE *file = fopen("test11.dat", "r");
    if (file == NULL) {
        perror("can't open: ");
        exit(-1);
    }

    while (fgets(linebuf, sizeof linebuf, file) != NULL) {
        el *name = malloc(sizeof *name);
        if (name == NULL) exit(-1);
        strcpy(name->bname, linebuf);
        STACK_PUSH(head, name);
    }
    fclose(file);
    STACK_COUNT(head, elt, count);
    printf("%d elements were read into the stack\n", count);

    /* now pop, print, and delete each element */
    while (!STACK_EMPTY(head)) {
        printf("%s\n", STACK_TOP(head)->bname);
        STACK_POP(head, elt);
        free(elt);
    }

    return 0;
}
--------------------------------------------------------------------------------

[[flex_names]]
Other names for next
~~~~~~~~~~~~~~~~~~~~
If the element structure's `next` field is named something else, a separate
group
of macros must be used. These work the same as the regular macros, but take the
field name as an extra parameter.

These "flexible field name" macros are shown below. They all end with `2`. Each
operates the same as its counterpart without the `2`, but they take the name of
the `next` field as a trailing argument.

[width="100%",cols="50<m,40<",grid="none",options="none"]
|===============================================================================
|STACK_PUSH2(stack,add,next);         | push `add` onto `stack`
|STACK_POP2(stack,elt,next);          | pop `stack` and save previous top as
`elt`
|STACK_COUNT2(stack,tmp,count,next);  | store number of elements into `count`
|===============================================================================


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