The Arena Library

0.2.1

Generated by Doxygen 1.7.6.1

Tue Apr 23 2013 22:31:26

Contents

Chapter 1

C Basic Library: Arena Library

Version

0.2.1

Author

Jun Woong (woong.jun at gmail.com)

Date

last modified on 2013-04-23

1.1 Introduction

This document specifies the Arena Library which belings to the C Basic Library. The basic structure is from David Hanson's book, "C Interfaces and Implementations." - I modified the original implementation to make it conform to the C standard and to enhance its readibility; for example a prefix is used more strictly in order to avoid the user namespace pollution.

Since the book explains its design and implementation in a very comprehensive way, not to mention the copyright issues, it is nothing but waste to repeat it here, so I finish this document by giving a brief motivation and introduction to the library; how to use the facilities is deeply explained in files that define them.

malloc() and other related functions from <stdlib.h> provide facilities for the size-based memory allocation strategy. Each invocation to allocation functions requires a corresponding invocation to deallocation functions in order to avoid the memory leakage problem. Under certain circumstances the size-based memory allocator is not the best way to manage storage. For example, consider a script interpreter that accepts multiple scripts and parses them for execution in sequence. During running a script, many data structures including trees from a parser and tables for symbols should be maintained

and those structures often require complicated patterns of memory allocation/deallocation. Besides, they should be correctly freed after the sciprt has been processed and before processing of the next sciprt starts off. In such a situation, a lifetime-based memory allocator can work better and simplify the patterns of (de)allocation. With a lifetime-based memory allocator, all storages allocated for processing a script is remembered somehow, and all the instances can be freed at a time when processing the script has finished. The Arena Library provides a lifetime-based memory allocator. Hanson's book also mentions constructing a graphic user interface (having various features like input forms, scroll bars and buttons) as an example where a lifetime-based allocator does better than a size-based one.

The Arena Library reserves identifiers starting with `arena_` and `ARENA_`, and imports the Assertion Library and the Exception Handling Library. Even if it does not depend on the Memory Management Library, it also uses the identifier `MEM_MAXALIGN`.

1.2 How to Use The Library

Basically, using the Arena Library to allocate storages does not quite differ from using malloc() or similars from <stdlib.h>. The differences are that every allocation function takes an additional argument to specify an arena, and that there is no need to invoke a deallocation function for each of allocated storage blocks; a function to free all storages associated with an arena is provided.

```
typeA *p = malloc(sizeof(*p));
...
typeB *q = malloc(sizeof(*p));
...
free(p);
free(q);
```

For example, suppose that `p` and `q` point to two areas that have been allocated at different places but can be freed at the same time. As the number of such instances increases, deallocating them gets more complicated and thus necessarily more error-prone.

```
typeA *p = ARENA_ALLOC(myarena, sizeof(*p));
...
typeB *q = ARENA_ALLOC(myarena, sizeof(*q));
...
ARENA_FREE(myarena);
```

On the other hand, if the allocator that the Arena Library offers is used, only one call to ARENA_FREE() frees all storages associated with the arena, `myarena`.

Applying to the problem mentioned to introduce a lifetime-based allocator, all storages for data structures used while a script is processed can be associated with an arena and get freed readily by ARENA_FREE() before moving to the next script.

As <stdlib.h> provides malloc() and calloc(), this library does ARENA_ALLOC() and ARENA_CALLOC() taking similar arguments. ARENA_FREE() does what free() does (actually, it does more as explained above). ARENA_NEW() creates an arena with

which allocated storages will be associated, and ARENA_DISPOSE() destroys an arena, which means that an arena itself may be reused repeatedly even after all storages with it have been freed by ARENA_FREE().

1.2.1 Some Caveats

As in the debugging version of the Memory Management Library, `MEM_MAXALIGN` indicates the common alignment factor; in other words, the alignment factor of pointers malloc() returns. If it is not given, the library tries to guess a proper value, but no guarantee that the guess is correct. Therefore, it is recommended to give a proper definition for `MEM_MAXALIGN` (via a compiler option like -D, if available).

Note that the Arena Library does not rely on the Memory Management Library. This means it constitutes a completely different allocator. Thus, the debugging version of the Memory Management Library cannot detect problems occurred in the storages maintained by the Arena Library.

1.3 Boilerplate Code

Using an arena starts with creating it:

```
arena_t myarena = ARENA_NEW();
```

As in the Memory Management Library, you don't need to check the return value of ARENA_NEW(); an exception named `arena_exceptfailNew` will be raised if the creation fails (see the Exception Handling Library for how to handle exceptions). Creating an arena is different from allocating a necessary storage. With an arena, you can freely allocate storages that belong to it with ARENA_ALLOC() or ARENA_CALLOC() as in:

```
sometype_t *p = ARENA_ALLOC(myarena, sizeof(*p));
othertype_t *q = ARENA_CALLOC(myarena, 10, sizeof(*q));
```

Again, you don't have to check the return value of these invocations. If no storage is able to be allocated, an exception, `arena_exceptfailAlloc` will be raised. - Due to problems in implementation, adjusting the size of the allocated storage is not supported; there is no ARENA_REALLOC() or ARENA_RESIZE() that corresponds to realloc().

There are two ways to release storages with an arena: ARENA_FREE() and ARENA_DISPOSE().

```
ARENA_FREE(myarena);
... myarena can be reused ...
ARENA_DISPOSE(myarena);
```

ARENA_FREE() deallocates any storage belonging to an arena, while ARENA_DISPOSE() does the same job and also destroy the arena to make it no more usable.

If you have a plan to use a tool like Valgrind to detect memory-related problems, see arena_dispose(); ARENA_DISPOSE() is a simple macro wrapper for arena_dispose() to keep the interface consistent.

1.4 Future Directions

1.4.1 Minor Changes

To mimic the behavior of calloc() specified by the standard, it is required for ARENA_CALLOC() to successfully return a null pointer when it cannot allocate storage of the requested size. Since this does not allow overflow, it has to carefully check overflow when calculating the total size.

1.5 Contact Me

Visit http://code.woong.org to get the latest version of this library. Only a small portion of my homepage (http://www.woong.org) is maintained in English, thus one who is not good at Korean would have difficulty when navigating most of other pages served in Korean. If you think the information you are looking for is on pages written in Korean, do not hesitate to send me an email to ask for help.

Any comments about the library are welcomed. If you have a proposal or question on the library just email me, and I will reply as soon as possible.

1.6 Copyright

Chapter 2

Todo List

Global arena_calloc (arena_t *, size_t c, size_t n, const char *, int)

Some improvements are possible and planned:

- the C standard requires calloc() return a null pointer if it cannot allocates storage of the size `c * n` in bytes, which allows no overflow in computing the multiplication. Overflow checking is necessary to mimic the behavior of calloc().

Global arena_calloc (arena_t *, size_t c, size_t n, const char *, int)

Some improvements are possible and planned:

- the C standard requires calloc() return a null pointer if it cannot allocates storage of the size `c * n` in bytes, which allows no overflow in computing the multiplication. Overflow checking is necessary to mimic the behavior of calloc().

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

File Documentation

4.1 arena.c File Reference

Source for Arena Library (CBL)

`#include <stddef.h>` `#include <stdlib.h>` `#include <string.h>` `#include "cbl/assert.h"` `#include "cbl/except.h"` `#include "arena.h"` Include dependency graph for arena.c:

Functions

- arena_t *() arena_new (void)

  creates a new arena.
- void() arena_dispose (arena_t **parena)

  disposes an arena.
- void *() arena_alloc (arena_t *arena, size_t n, const char *file, int line)

  allocates storage associated with an arena.
- void *() arena_calloc (arena_t *arena, size_t c, size_t n, const char *file, int line)

  allocates zero-filled storage associated with an arena.
- void() arena_free (arena_t *arena)

  deallocates all storages belonging to an arena.

Variables

- const except_t arena_exceptfailNew = { "Arena creation failed" }

  exception for arena creation failure.
- const except_t arena_exceptfailAlloc = { "Arena allocation failed" }

  exception for arena memory allocation failure.

4.1.1 Detailed Description

Source for Arena Library (CBL)

4.1.2 Function Documentation

4.1.2.1 void*() arena_alloc (arena_t * *arena*, size_t *n*, const char * *file*, int *line*)

allocates storage associated with an arena.

arena_alloc() allocates storage for an arena as malloc() or mem_alloc() does.

Possible exceptions: assert_exceptfail, arena_exceptfailAlloc

Unchecked errors: foreign data structure given for `arena`

Parameters

`in,out`	*arena*	arena for which storage to be allocated
`in`	*n*	size of storage requested in bytes
`in`	*file*	file name in which storage requested
`in`	*func*	function name in which storage requested (if C99 supported)
`in`	*line*	line number on which storage requested

Returns

storage allocated for given arena

Here is the caller graph for this function:

4.1.2.2 void*() arena_calloc (arena_t * *arena*, size_t *c*, size_t *n*, const char * *file*, int *line*)

allocates zero-filled storage associated with an arena.

arena_calloc() does the same as arena_alloc() except that it returns zero-filled storage.

Possible exceptions: assert_exceptfail, arena_exceptfailAlloc

Unchecked errors: foreign data structure given for `arena`

Parameters

`in,out`	*arena*	arena for which zero-filled storage is to be allocated
`in`	*c*	number of items to be allocated
`in`	*n*	size in bytes for one item
`in`	*file*	file name in which storage requested
`in`	*func*	function name in which storage requested (if C99 supported)
`in`	*line*	line number on which storage requested

Returns

zero-filled storage allocated for arena

Todo Some improvements are possible and planned:

- the C standard requires calloc() return a null pointer if it cannot allocates storage of the size `c * n` in bytes, which allows no overflow in computing the multiplication. Overflow checking is necessary to mimic the behavior of calloc().

Here is the call graph for this function:

4.1.2.3 void() arena_dispose (arena_t ** *parena*)

disposes an arena.

arena_dispose() releases storages belonging to a given arena and disposes it. Do not confuse with arena_free() which gets all storages of an arena deallocated but does not destroy the arena itself.

Note that storages belonging to `freelist` is not deallcated by arena_dispose() because it is possibly used by other arenas. Thus, a tool detecting the memory leakage problem might say there is leakage caused by the library, but that is intended not a bug.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `parena`

Parameters

`in,out`	*parena*	pointer to arena to dispose

Returns

nothing

Here is the call graph for this function:

4.1.2.4 void() arena_free (arena_t * *arena*)

deallocates all storages belonging to an arena.

arena_free() releases all storages belonging to a given arena.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `arena`

Parameters

`in,out`	*arena*	arena whose storages to be deallocated

Returns

nothing

Here is the caller graph for this function:

4.1.2.5 arena_t*() arena_new (void)

creates a new arena.

arena_new() creates a new arena and initialize it to indicate an empty arena.

Possible exceptions: arena_exceptfailNew

Unchecked errors: none

Returns

new arena created

4.2 arena.h File Reference

Header for Arena Library (CBL)

`#include <stddef.h> #include "cbl/except.h"` Include dependency graph for arena.h: This graph shows which files directly or indirectly include this file:

Defines

- #define ARENA_NEW() (arena_new())

  allocates a new arena.
- #define ARENA_DISPOSE(pa) (arena_dispose(pa))

  destroys an arena pointed to by `pa`.
- #define ARENA_ALLOC(a, n) (arena_alloc((a), (n), __FILE__, __LINE__))

  allocates storage whose byte length is `n` for an arena `a`.
- #define ARENA_CALLOC(a, c, n) (arena_calloc((a), (c), (n), __FILE__, __LINE__))

  allocates zero-filled storage of the size `c * n` for an arena `a`.
- #define ARENA_FREE(a) (arena_free(a))

  deallocates strorages belonging to an arena `a`.

Typedefs

- typedef struct arena_t arena_t

  represents an arena.

Functions

arena creating functions:

- arena_t * arena_new (void)
  creates a new arena.

memory allocating/deallocating functions:

- void * arena_alloc (arena_t *, size_t n, const char *, int)
  allocates storage associated with an arena.
- void * arena_calloc (arena_t *, size_t c, size_t n, const char *, int)
  allocates zero-filled storage associated with an arena.
- void arena_free (arena_t *arena)
  deallocates all storages belonging to an arena.

arena destroying functions:

- void arena_dispose (arena_t **)
  disposes an arena.

Variables

- const except_t arena_exceptfailNew
  exception for arena creation failure.
- const except_t arena_exceptfailAlloc
  exception for arena memory allocation failure.

4.2.1 Detailed Description

Header for Arena Library (CBL) Documentation for Arena Library (CBL)

4.2.2 Typedef Documentation

4.2.2.1 typedef struct arena_t arena_t

represents an arena.

`arena_t` represents an arena to which storages belongs.

4.2.3 Function Documentation

4.2.3.1 void* arena_alloc (arena_t * *arena*, size_t *n*, const char * *file*, int *line*)

allocates storage associated with an arena.

arena_alloc() allocates storage for an arena as malloc() or mem_alloc() does.

Possible exceptions: assert_exceptfail, arena_exceptfailAlloc

Unchecked errors: foreign data structure given for `arena`

Parameters

`in,out`	*arena*	arena for which storage to be allocated
`in`	*n*	size of storage requested in bytes
`in`	*file*	file name in which storage requested
`in`	*func*	function name in which storage requested (if C99 supported)
`in`	*line*	line number on which storage requested

Returns

storage allocated for given arena

Here is the caller graph for this function:

4.2.3.2 void* arena_calloc (arena_t * *arena*, size_t *c*, size_t *n*, const char * *file*, int *line*)

allocates zero-filled storage associated with an arena.

arena_calloc() does the same as arena_alloc() except that it returns zero-filled storage.

Possible exceptions: assert_exceptfail, arena_exceptfailAlloc

Unchecked errors: foreign data structure given for `arena`

Parameters

`in,out`	*arena*	arena for which zero-filled storage is to be allocated
`in`	*c*	number of items to be allocated
`in`	*n*	size in bytes for one item
`in`	*file*	file name in which storage requested
`in`	*func*	function name in which storage requested (if C99 supported)
`in`	*line*	line number on which storage requested

Returns

zero-filled storage allocated for arena

Todo Some improvements are possible and planned:

- the C standard requires calloc() return a null pointer if it cannot allocates storage of the size `c * n` in bytes, which allows no overflow in computing the multiplication. Overflow checking is necessary to mimic the behavior of calloc().

Here is the call graph for this function:

4.2.3.3 void arena_dispose (arena_t ** *parena*)

disposes an arena.

arena_dispose() releases storages belonging to a given arena and disposes it. Do not confuse with arena_free() which gets all storages of an arena deallocated but does not destroy the arena itself.

Note that storages belonging to `freelist` is not deallcated by arena_dispose() because it is possibly used by other arenas. Thus, a tool detecting the memory leakage problem might say there is leakage caused by the library, but that is intended not a bug.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `parena`

Parameters

`in,out`	*parena*	pointer to arena to dispose

Returns

nothing

Here is the call graph for this function:

4.2.3.4 void arena_free (arena_t * *arena*)

deallocates all storages belonging to an arena.

arena_free() releases all storages belonging to a given arena.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `arena`

Parameters

`in,out`	*arena*	arena whose storages to be deallocated

Returns

nothing

Here is the caller graph for this function:

4.2.3.5 arena_t* arena_new (void)

creates a new arena.

arena_new() creates a new arena and initialize it to indicate an empty arena.

Possible exceptions: arena_exceptfailNew

Unchecked errors: none

Returns

new arena created