Malloc,Calloc,Free Implementation using Global Array

Generated by Doxygen 1.8.11

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 meta Struct Reference

Structure declaration of the metadata block used for memory allocations. Used internally to implement linked list of allocated memory blocks.

`#include <functions.h>`

Data Fields

- struct meta * **next**
- int **free**
- size_t **size**

3.1.1 Detailed Description

Structure declaration of the metadata block used for memory allocations. Used internally to implement linked list of allocated memory blocks.

The documentation for this struct was generated from the following file:

- functions.h

Chapter 4

File Documentation

4.1 functions.c File Reference

Function definitions for dynamic memory management.

```
#include "functions.h"
```

Functions

- static metadata * search_freespace (size_t size)

  The function which searches in the list of memory blocks which fits the requested size. Used during a malloc() or calloc() calls to get the first fit.
- static void fuse (metadata *ptr)

  The function which fuses two adjacent free blocks into single free block. Used during a call to free()
- static void split (metadata *ptr, size_t size)

  The function which splits a large memory block into two smaller block based on the threshold value. Used during a call to malloc() and calloc().
- static int verifyBlockAddress (metadata *to_free, metadata **prev)

  The function which checks the validity of the block starting address. Used during a call to free() and thus avoids memory corruption.
- void * my_malloc (size_t size)

  Allocates requested bytes of memory from the available global array.
- void * my_calloc (size_t n, size_t size)

  Allocates requested blocks of memory from the available global array and initializes its contents to 0.
- void my_free (void *ptr)

  Frees the memory pointed to by ptr.
- void * my_realloc (void *ptr, size_t size)

  Reallocates the current memory to a new memory location according to requested size. Expands or shrinks the allocated memory by the give size.
- void deframent_my_heap (void)

  Defragments the global array by fusing adjacent free blocks.
- size_t free_space_in_my_heap (void)

  Computes the available free space in the global array.
- void print_memory_contents (void)

  Prints the state of the global array with memory block size and associated status (free or not free). Used for verification of testcases.

Variables

- static char buffer [BUFF_SIZE] ={0}

  The buffer array used for memory allocation.
- static metadata ∗ heap_base =(metadata ∗)buffer

  Points to the base address of the array.
- static metadata ∗ last =NULL

  Keeps tracks of the tail pointer of the linked list of allocated block.
- static size_t freespace =BUFF_SIZE

  The variable keep track of freespace in the array.
- static int **first** =1

4.1.1 Detailed Description

Function definitions for dynamic memory management.

This contains the function definitons for the dynamic memory allocations and deallocations including utility functions.

Author

Freeze Francis

4.1.2 Function Documentation

4.1.2.1 void deframent_my_heap (void)

Defragments the global array by fusing adjacent free blocks.

Parameters

ptr	The pointer the current memory.
size	The new size.

Returns

The pointer to the reallocated memory.

4.1.2.2 size_t free_space_in_my_heap (void)

Computes the available free space in the global array.

Parameters

void	

Returns

The available free space in bytes.

4.1.2.3 static void fuse (metadata * *ptr*) [static]

The function which fuses two adjacent free blocks into single free block. Used during a call to free()

Parameters

ptr	pointer to the first free block

Returns

void

4.1.2.4 void* my_calloc (size_t *n*, size_t *size*)

Allocates requested blocks of memory from the available global array and initializes its contents to 0.

Parameters

n	The number of blocks to allocated.
size	The size of each block in bytes.

Returns

The pointer to the allocated memory.

4.1.2.5 void my_free (void * *ptr*)

Frees the memory pointed to by ptr.

Parameters

ptr	The pointer the memory to be freed.

Returns

void

4.1.2.6 void* my_malloc (size_t *size*)

Allocates requested bytes of memory from the available global array.

Parameters

size	The number of bytes to allocated.

Returns

The pointer to the allocated memory.

4.1.2.7 void* my_realloc (void * *ptr*, size_t *size*)

Reallocates the current memory to a new memory location according to requested size. Expands or shrinks the allocated memory by the give size.

Parameters

ptr	The pointer the memory to be reallocated.
size	The new size.

Returns

The pointer to the reallocated memory.

4.1.2.8 void print_memory_contents (void)

Prints the state of the global array with memory block size and associated status (free or not free). Used for verification of testcases.

Parameters

void	

Returns

void

4.1.2.9 static metadata * search_freespace (size_t *size*) [static]

The function which searches in the list of memory blocks which fits the requested size. Used during a malloc() or calloc() calls to get the first fit.

Parameters

size	The requested size

Returns

pointer to the free block which fits the size

4.1.2.10 static void split (metadata * *ptr*, size_t *size*) [static]

The function which splits a large memory block into two smaller block based on the threshold value. Used during a call to malloc() and calloc().

Parameters

ptr	pointer to the block to be split

Returns

void

4.1.2.11 static int verifyBlockAddress (metadata * *to_free*, metadata ** *prev*) [static]

The function which checks the validity of the block starting address. Used during a call to free() and thus avoids memory corruption.

Parameters

to_free	pointer to the block to be freed

Returns

validity (0-invalid 1-valid)

4.1.3 Variable Documentation

4.1.3.1 char buffer[BUFF_SIZE] ={0} [static]

The buffer array used for memory allocation.

4.1.3.2 metadata* heap_base =(metadata *)buffer [static]

Points to the base address of the array.

4.1.3.3 metadata* last =NULL [static]

Keeps tracks of the tail pointer of the linked list of allocated block.

4.2 functions.h File Reference

Function prototypes for dynamic memory management.

```
#include <unistd.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
```

Data Structures

- struct meta

  Structure declaration of the metadata block used for memory allocations. Used internally to implement linked list of allocated memory blocks.

Macros

- #define **THRESHOLD** sizeof(int)+sizeof(metadata)
- #define **BUFF_SIZE** 1000000

Typedefs

- typedef struct meta **metadata**

Functions

- void * my_malloc (size_t size)

  Allocates requested bytes of memory from the available global array.
- void * my_calloc (size_t n, size_t size)

  Allocates requested blocks of memory from the available global array and initializes its contents to 0.
- void * my_realloc (void *ptr, size_t size)

  Reallocates the current memory to a new memory location according to requested size. Expands or shrinks the allocated memory by the give size.
- void my_free (void *ptr)

  Frees the memory pointed to by ptr.
- void deframent_my_heap (void)

  Defragments the global array by fusing adjacent free blocks.
- size_t free_space_in_my_heap (void)

  Computes the available free space in the global array.
- void print_memory_contents (void)

  Prints the state of the global array with memory block size and associated status (free or not free). Used for verification of testcases.

4.2.1 Detailed Description

Function prototypes for dynamic memory management.

This contains the prototypes for the dynamic memory allocations and deallocations including macros,structure declarations.

Author

Freeze Francis

4.2.2 Function Documentation

4.2.2.1 void deframent_my_heap (void)

Defragments the global array by fusing adjacent free blocks.

Parameters

ptr	The pointer the current memory.
size	The new size.

Returns

The pointer to the reallocated memory.

4.2.2.2 size_t free_space_in_my_heap (void)

Computes the available free space in the global array.

Parameters

void	

Returns

The available free space in bytes.

4.2.2.3 void* my_calloc (size_t *n*, size_t *size*)

Allocates requested blocks of memory from the available global array and initializes its contents to 0.

Parameters

n	The number of blocks to allocated.
size	The size of each block in bytes.

Returns

The pointer to the allocated memory.

4.2.2.4 void my_free (void * *ptr*)

Frees the memory pointed to by ptr.

Parameters

ptr	The pointer the memory to be freed.

Returns

void

4.2.2.5 void* my_malloc (size_t *size*)

Allocates requested bytes of memory from the available global array.

Parameters

size	The number of bytes to allocated.

Returns

The pointer to the allocated memory.

4.2.2.6 void* my_realloc (void * *ptr*, size_t *size*)

Reallocates the current memory to a new memory location according to requested size. Expands or shrinks the allocated memory by the give size.

Parameters

ptr	The pointer the memory to be reallocated.
size	The new size.

Returns

The pointer to the reallocated memory.

4.2.2.7 void print_memory_contents (void)

Prints the state of the global array with memory block size and associated status (free or not free). Used for verification of testcases.

Parameters

void	

Returns

void

4.3 main.c File Reference

Driver program for dynamic memory allocations functions. A mixture of calls to function are made and after each call the memory layout is printed in the format Block-size v/s free-status(0-not free 1-free) The amount of free space left is also printed.

```
#include "functions.h"
```

Functions

- int **main** ()

4.3.1 Detailed Description

Driver program for dynamic memory allocations functions. A mixture of calls to function are made and after each call the memory layout is printed in the format Block-size v/s free-status(0-not free 1-free) The amount of free space left is also printed.

Author

Freeze Francis

Index