

Allocator Assignment

Generated by Doxygen 1.8.12

Contents

1	Pattern Assignment	1
1.1	the setup	1
1.1.1	STEP 1:either use make compile(which run the production versiion) or make debug which runs the debug version.	1
1.1.2	STEP 2: run ./main to load the program into the main memory.	1
2	Bug List	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	Meminfo Struct Reference	9
5.1.1	Detailed Description	9
5.1.2	Member Data Documentation	9
5.1.2.1	free	9
5.1.2.2	next	9
5.1.2.3	size	9

6 File Documentation	11
6.1 log.h File Reference	11
6.1.1 Detailed Description	11
6.2 main.c File Reference	11
6.2.1 Detailed Description	12
6.2.2 Function Documentation	12
6.2.2.1 main()	12
6.3 malloc.c File Reference	12
6.3.1 Detailed Description	13
6.3.2 Function Documentation	13
6.3.2.1 check_for_enough_space(Meminfo *last, size_t s)	13
6.3.2.2 free_space_in_my_heap(void)	13
6.3.2.3 get_free_block(size_t s)	14
6.3.2.4 merge_free_block(Meminfo *left, Meminfo *right)	14
6.3.2.5 my_calloc(size_t num, size_t size)	14
6.3.2.6 my_free(const void *ptr)	14
6.3.2.7 my_malloc(size_t size)	15
6.3.2.8 my_realloc(void *ptr, size_t size)	15
6.3.3 Variable Documentation	15
6.3.3.1 free_space	15
6.3.3.2 head	16
6.3.3.3 heap	16
6.4 malloc.h File Reference	16
6.4.1 Detailed Description	17
6.4.2 Macro Definition Documentation	17
6.4.2.1 ALIGN	17
6.4.2.2 FALSE	17
6.4.2.3 MAX_SIZE	17
6.4.2.4 TRUE	17
6.4.3 Function Documentation	17
6.4.3.1 free_space_in_my_heap(void)	17
6.4.3.2 my_calloc(size_t num, size_t size)	18
6.4.3.3 my_free(const void *ptr)	18
6.4.3.4 my_malloc(size_t size)	18
6.4.3.5 my_realloc(void *ptr, size_t size)	19
Index	21

Chapter 1

Pattern Assignment

1.1 the setup

1.1.1 STEP 1: either use `make compile` (which runs the production version) or `make debug` which runs the debug version.

1.1.2 STEP 2: run `./main` to load the program into the main memory.

Chapter 2

Bug List

File [log.h](#)

No known Bugs.

File [main.c](#)

No Known Bugs

File [malloc.c](#)

No Known bugs.

File [malloc.h](#)

no knowe bugs

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Meminfo	This structure contains the declaratiuon of the metadata structure	9
-------------------------	--	---

Chapter 4

File Index

4.1 File List

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

log.h	This file contains all the debug macros	11
main.c	This is the driver program for the lib functions	11
malloc.c	All the functions of the allocator	12
malloc.h	This file contains all the memory operations prototype of my library	16

Chapter 5

Class Documentation

5.1 Meminfo Struct Reference

this structure contains the declaratiuon of the metadata structure.

```
#include <malloc.h>
```

Public Attributes

- `size_t` [size](#)
- `Meminfo *` [next](#)
- `int` [free](#)

5.1.1 Detailed Description

this structure contains the declaratiuon of the metadata structure.

5.1.2 Member Data Documentation

5.1.2.1 `int` `Meminfo::free`

flag to check whether the block is free

5.1.2.2 `Meminfo *` `Meminfo::next`

holds the address the address of the next metadata

5.1.2.3 `size_t` `Meminfo::size`

Size of the block

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

- [malloc.h](#)

Chapter 6

File Documentation

6.1 log.h File Reference

This file contains all the debug macros.

Macros

- `#define LOG(msg)`

6.1.1 Detailed Description

This file contains all the debug macros.

This debug macros when enabled in debug mode print the debug messages into the stderr.

Author

Prakash

Bug No known Bugs.

6.2 main.c File Reference

this is the dirver program for the lib functions

```
#include <stdio.h>
#include <stdint.h>
#include "malloc.h"
```

Functions

- `int main ()`
this main method for the driver program.

6.2.1 Detailed Description

this is the driver program for the lib functions

this file provides a testcase to call the function in the library and print heap size after some operation for verification.

Author

Prakash

Bug No Known Bugs

6.2.2 Function Documentation

6.2.2.1 int main ()

this main method for the driver program.

In this implementation we have assumed that heap size is 16k which is defined in the [malloc.h](#) file and each metadata block takes 24 bytes. First it creates 4 integer memory followed by 200 characters and 4 integers using `my_calloc`. So the total free memory will be $65536 - (16 * 2 + 200 + 24 * 3) = 65232$. After that `c` is reallocated to 100 bytes which causes a split in the 200 byte block into a 100 byte block which is used and a 100 byte block which is free. So the size should be 65332 bytes but the actual free space will be 65308 because the 100 byte free block has covered a metadata of 24 bytes. Now when `free` on `c` is called then it frees the 100 byte block for `c` and it is merged with the next 100 byte block. So the size now is 65442. Now when `free` on `a` is called it frees 16 bytes and it merges with the next 200 byte free block getting the total memory size to 65472 bytes.

6.3 malloc.c File Reference

contains all the functions of the allocator.

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "log.h"
#include "malloc.h"
```

Functions

- static [Meminfo](#) * [get_free_block](#) (size_t s)
checks whether free block of the mentioned size is available or not.
- static int [check_for_enough_space](#) ([Meminfo](#) *last, size_t s)
checks whether a block has sufficient memory.
- static void [merge_free_block](#) ([Meminfo](#) *left, [Meminfo](#) *right)
merge to memory block.
- void * [my_malloc](#) (size_t size)
This function allocates memory from the 64KB Array.
- void * [my_calloc](#) (size_t num, size_t size)
this function allocates initialized memory in 64 KB Array.
- void [my_free](#) (const void *ptr)
This function free the memory allocated by `my_malloc` and `my_calloc`.
- void * [my_realloc](#) (void *ptr, size_t size)
This function's reallocates the previous allocated chunk with new size.
- uint32_t [free_space_in_my_heap](#) (void)
this function returns the current free memory of the heap excluding the metadata.

Variables

- static uint8_t `heap` [`MAX_SIZE`]
- static `Meminfo` * `head` = NULL
- static uint32_t `free_space` = `MAX_SIZE`

6.3.1 Detailed Description

contains all the functions of the allocator.

This contains all the implementations of the allocators along with the various static utility function which can only be accessed inside the file.

Author

Prakash

Bug No Known bugs.

6.3.2 Function Documentation

6.3.2.1 static int check_for_enough_space (`Meminfo` * *last*, size_t *s*) [static]

checks whether a block has sufficient memory.

Parameters

<i>last</i>	last segment.
<i>s</i>	size.

Returns

int true or false.

6.3.2.2 uint32_t free_space_in_my_heap (void)

this function returns the current free memory of the heap excluding the metadata.

Parameters

<i>Void</i>	
-------------	--

Returns

uint32_t free size of the heap.

6.3.2.3 static Meminfo * get_free_block (size_t s) [static]

checks whether free block of the mentioned size is available or not.

Parameters

<i>s</i>	size required.
----------	----------------

Returns

[Meminfo](#) * pointer to the metadata block.

6.3.2.4 static void merge_free_block (Meminfo * left, Meminfo * right) [static]

memrge to memory block.

Parameters

<i>left</i>	left block to be merged.
<i>right</i>	right block.

Returns

void.

6.3.2.5 void* my_calloc (size_t num, size_t size)

this function allocates intialized memory in 64 KB Array.

This function creates the initialized memory using malloc first then using memset to intialize it to zero.

Parameters

<i>num</i>	number of elements.
<i>size</i>	size of each element.

Returns

void * pointer to the data block.

6.3.2.6 void my_free (const void * ptr)

This function free the memory allocated by my_malloc and my_calloc.

This function frees up the block allocated by the allocation functions. if the address is not a part of this memory it simply does nothing. finally if there are adjacent block's it merges them to form a new block.

Parameters

<i>ptr</i>	pointer to the memory.
------------	------------------------

Returns

Void

6.3.2.7 void* my_malloc (size_t size)

This function allocates memory from the 64KB Array.

This function allocates memory in the 64KB uint_8 array by keeping track of the metadata block followed by the data block for each request and returns NULL if some error happens or it runs out of memory. if there is free block which is large enough then it can split the block into two depending on the size of the block that will remain after allocation.

Parameters

<i>size</i>	size to be allocated.
-------------	-----------------------

Returns

void * pointer to the data block.

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

This function's reallocates the previous allocated chunk with new size.

This pointer to invalid memory is given it return null because to give the user a error message that it cannot be created. if size 0 is passed free is called. if a size less than the current size is called then it is allocated and if possible the block is splitted.

Parameters

<i>ptr</i>	pointer to the data
<i>size</i>	new size.

Returns

Void * pointer to the increased size block or null.

6.3.3 Variable Documentation**6.3.3.1 uint32_t free_space = MAX_SIZE [static]**

free size of the heap

6.3.3.2 Meminfo* head = NULL [static]

initial head pointer for the linked list

6.3.3.3 uint8_t heap[MAX_SIZE] [static]

this is the max sizeof the heap* 64KB by default

6.4 malloc.h File Reference

This file contains all the memory operations prototype of my library.

Classes

- struct [Meminfo](#)
this structure contains the declaratiuon of the metadata structure.

Macros

- #define [MAX_SIZE](#) 65536
- #define [FALSE](#) 0
- #define [TRUE](#) 1
- #define [ALIGN](#)(size) (((size-1)/4)*4)+4)

Typedefs

- typedef struct [Meminfo](#) **Meminfo**

Functions

- void * [my_malloc](#) (size_t size)
This function allocates memory from the 64KB Array.
- void * [my_calloc](#) (size_t num, size_t size)
this function allocates intialized memory in 64 KB Array.
- void [my_free](#) (const void *ptr)
This function free the memory allocated by my_malloc and my_calloc.
- void * [my_realloc](#) (void *ptr, size_t size)
This function's reallocates the previous allocated chunk with new size.
- uint32_t [free_space_in_my_heap](#) (void)
this function returns the current free memory of the heap excluding the metadata.

6.4.1 Detailed Description

This file contains all the memory operations prototype of my library.

This contains all the prototypes of the memory operation with metadata structure.

Author

Prakash

Bug no knowe bugs

6.4.2 Macro Definition Documentation

6.4.2.1 `#define ALIGN(size) (((size-1)/4)*4)+4)`

this is fo aligning the space for 4 bytes

6.4.2.2 `#define FALSE 0`

this is set when block is not free

6.4.2.3 `#define MAX_SIZE 65536`

this is the max sizeof the heap

6.4.2.4 `#define TRUE 1`

this is set when block is free

6.4.3 Function Documentation

6.4.3.1 `uint32_t free_space_in_my_heap (void)`

this function returns the current free memory of the heap excluding the metadata.

Parameters

<i>Void</i>	
-------------	--

Returns

`uint32_t` free size of the heap.

6.4.3.2 void* my_calloc (size_t num, size_t size)

this function allocates intialized memory in 64 KB Array.

This function creates the initialized memory using malloc first then using memset to intialize it to zero.

Parameters

<i>num</i>	number of elements.
<i>size</i>	size of each element.

Returns

void * pointer to the data block.

6.4.3.3 void my_free (const void * ptr)

This function free the memory allocated by my_malloc and my_calloc.

This function frees up the block allocated by the allocation functions. if the address is not a part of this memory it simply does nothing. finally if there are adjacent block's it merges them to from a new block.

Parameters

<i>ptr</i>	pointer to the memory.
------------	------------------------

Returns

Void

6.4.3.4 void* my_malloc (size_t size)

This function allocates memory from the 64KB Array.

This function allocates memory in the 64KB uint_8 array by keeping track of the metadata block followed by the data block for each request and returns NULL if some error happens or it runs out of memory.if there is free block which is large enough then it can split the blockk into two depending on the size of the block that will remain after allocation.

Parameters

<i>size</i>	size to be allocated.
-------------	-----------------------

Returns

void * pointer to the data block.

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

This function's reallocates the previous allocated chunk with new size.

This pointer to invalid memory is given it return null because to give the user a error message that it cannot be created. if size 0 is passed free is called. if a size less than the current size is called then it is allocated and if possible the block is splitted.

Parameters

<i>ptr</i>	pointer to the data
<i>size</i>	new size.

Returns

Void * pointer to the increased size block or null.

Index

ALIGN
 malloc.h, 17

check_for_enough_space
 malloc.c, 13

FALSE
 malloc.h, 17

free
 Meminfo, 9

free_space
 malloc.c, 15

free_space_in_my_heap
 malloc.c, 13
 malloc.h, 17

get_free_block
 malloc.c, 13

head
 malloc.c, 15

heap
 malloc.c, 16

log.h, 11

MAX_SIZE
 malloc.h, 17

main
 main.c, 12

main.c, 11
 main, 12

malloc.c, 12
 check_for_enough_space, 13
 free_space, 15
 free_space_in_my_heap, 13
 get_free_block, 13
 head, 15
 heap, 16
 merge_free_block, 14
 my_calloc, 14
 my_free, 14
 my_malloc, 15
 my_realloc, 15

malloc.h, 16
 ALIGN, 17
 FALSE, 17
 free_space_in_my_heap, 17
 MAX_SIZE, 17
 my_calloc, 17
 my_free, 18
 my_malloc, 18
 my_realloc, 18
 TRUE, 17

Meminfo, 9
 free, 9
 next, 9
 size, 9

merge_free_block
 malloc.c, 14

my_calloc
 malloc.c, 14
 malloc.h, 17

my_free
 malloc.c, 14
 malloc.h, 18

my_malloc
 malloc.c, 15
 malloc.h, 18

my_realloc
 malloc.c, 15
 malloc.h, 18

next
 Meminfo, 9

size
 Meminfo, 9

TRUE
 malloc.h, 17