(/)

You just released the optional tasks of this project. Have fun!

0x0B. C - malloc, free

C Memory allocation

By Julien Barbier

Weight: 1

Ongoing project - started Jul 26, 2022, must end by Jul 28, 2022 - you're done with 200% of tasks.

Checker was released at Jul 26, 2022 12:00 AM

An auto review will be launched at the deadline

Concepts

For this project, we expect you to look at this concept:

- Automatic and dynamic allocation, malloc and free (/concepts/62)

Resources

Read or watch:

- 0x0a - malloc & free - quick overview.pdf (/rltoken/7q6RmWq86XkUhvmlhrg9bg)
- Dynamic memory allocation in C - malloc calloc realloc free (/rltoken/pfGb2oVIYLO_1a8jtFGQYw) (*stop at 6:50*)

man or help:

- `malloc`
- `free`

Learning Objectives

(/)

At the end of this project, you are expected to be able to explain to anyone (/rltoken/f-MGO-Fu4KSrem3R6GkEyw), **without the help of Google**:

General

- What is the difference between automatic and dynamic allocation
- What is `malloc` and `free` and how to use them
- Why and when use `malloc`
- How to use `valgrind` to check for memory leak

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi` , `vim` , `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using `gcc` , using the options `-Wall -Werror -Wextra -pedantic -std=gnu89`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/holbertonschool/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/holbertonschool/Betty/blob/master/betty-doc.pl)
- You are not allowed to use global variables
- No more than 5 functions per file
- The only C standard library functions allowed are `malloc` and `free` . Any use of functions like `printf` , `puts` , `calloc` , `realloc` etc… is forbidden
- You are allowed to use _putchar (https://github.com/holbertonschool/_putchar.c/blob/master/_putchar.c)
- You don't have to push `_putchar.c` , we will use our file. If you do it won't be taken into account
- In the following examples, the `main.c` files are shown as examples. You can use them to test your functions, but you don't have to push them to your repo (if you do we won't take them into account). We will use our own `main.c` files at compilation. Our `main.c` files might be different from the one shown in the examples
- The prototypes of all your functions and the prototype of the function `_putchar` should be included in your header file called `main.h`
- Don't forget to push your header file

More Info

You do not have to learn about `calloc` and `realloc`.

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Hide quiz)

Question #0

What is Valgrind?

- [ ] A container service
- [x] It's a program to validate memory allocation
- [ ] It's a new step when I compile with `gcc`
- [ ] It's a program to test a C program in a specific environment

Question #1

How many bytes will this statement allocate?

```
malloc(sizeof(int) * 4)
```

- [ ] 4
- [ ] 8
- [x] 16
- [ ] 32

Question #2

How many bytes will this statement allocate?

```
malloc(sizeof(int) * 10)
```

- [x] 40
- [ ] 32
- [ ] 10

Question #3

How many bytes will this statement allocate?
(/)

```
malloc(sizeof(unsigned int) * 2)
```

- [ ] 2
- [ ] 4
- [x] 8

Question #4

How many bytes will this statement allocate?

```
malloc(sizeof(char) * 10)
```

- [x] 10
- [ ] 20
- [ ] 40

Question #5

How many bytes will this statement allocate?

```
malloc(10)
```

- [ ] 2
- [x] 10
- [ ] 4
- [ ] 40

Question #6

How many bytes will this statement allocate?

```
malloc((sizeof(char) * 10) + 1)
```

- [ ] 1
- [ ] 10
- [ ] 21
- [ ] 20
- [x] 11

Tasks

0. Float like a butterfly, sting like a bee

mandatory

Write a function that creates an array of chars, and initializes it with a specific char.

- Prototype: `char *create_array(unsigned int size, char c);`
- Returns `NULL` if size = `0`
- Returns a pointer to the array, or `NULL` if it fails

(/)

```
julien@ubuntu:~/0x0a. malloc, free$ cat 0-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * simple_print_buffer - prints buffer in hexa
 * @buffer: the address of memory to print
 * @size: the size of the memory to print
 *
 * Return: Nothing.
 */
void simple_print_buffer(char *buffer, unsigned int size)
{
    unsigned int i;

    i = 0;
    while (i < size)
    {
        if (i % 10)
        {
            printf(" ");
        }
        if (!(i % 10) && i)
        {
            printf("\n");
        }
        printf("0x%02x", buffer[i]);
        i++;
    }
    printf("\n");
}

/**
 * main - check the code for ALX School students.
 *
 * Return: Always 0.
 */
int main(void)
{
    char *buffer;

    buffer = create_array(98, 'H');
    if  (buffer == NULL)
    {
        printf("failed to allocate memory\n");
        return (1);
    }
    simple_print_buffer(buffer, 98);
    free(buffer);
    return (0);
}
```

```
julien@ubuntu:~/0x0a. malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 0-main.c 0-create_array.c -o a
julien@ubuntu:~/0x0a. malloc, free$ ./a
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
0x48 0x48 0x48 0x48 0x48 0x48 0x48 0x48
julien@ubuntu:~/0x0a. malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0B-malloc_free`
- File: `0-create_array.c`

Done! Help Check your code >_ Get a sandbox

1. The woman who has no imagination has no wings

mandatory

Write a function that returns a pointer to a newly allocated space in memory, which contains a copy of the string given as a parameter.

- Prototype: `char *_strdup(char *str);`
- The `_strdup()` function returns a pointer to a new string which is a duplicate of the string `str`. Memory for the new string is obtained with `malloc`, and can be freed with `free`.
- Returns `NULL` if str = NULL
- On success, the `_strdup` function returns a pointer to the duplicated string. It returns `NULL` if insufficient memory was available

FYI: The standard library provides a similar function: `strdup`. Run `man strdup` to learn more.

(/)

```
julien@ubuntu:~/0x0a. malloc, free$ cat 1-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * main - check the code for ALX School students.
 *
 * Return: Always 0.
 */
int main(void)
{
    char *s;

    s = _strdup("ALX SE");
    if (s == NULL)
    {
        printf("failed to allocate memory\n");
        return (1);
    }
    printf("%s\n", s);
    free(s);
    return (0);
}
julien@ubuntu:~/0x0a. malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 1
-main.c 1-strdup.c -o s
julien@ubuntu:~/0x0a. malloc, free$ ./s
ALX SE
julien@ubuntu:~/0x0a. malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0B-malloc_free`
- File: `1-strdup.c`

Done! Help Check your code >_ Get a sandbox

2. He who is not courageous enough to take risks will accomplish nothing in life

mandatory

Write a function that concatenates two strings.

- Prototype: `char *str_concat(char *s1, char *s2);`
- The returned pointer should point to a newly allocated space in memory which contains the contents of `s1`, followed by the contents of `s2`, and null terminated
- if `NULL` is passed, treat it as an empty string

- The function should return `NULL` on failure

(/)

```
julien@ubuntu:~/0x0a. malloc, free$ cat 2-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * main - check the code for ALX School students.
 *
 * Return: Always 0.
 */
int main(void)
{
    char *s;

    s = str_concat("Betty ", "Holberton");
    if (s == NULL)
    {
        printf("failed\n");
        return (1);
    }
    printf("%s\n", s);
    free(s);
    return (0);
}
julien@ubuntu:~/0x0a. malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 2
-main.c 2-str_concat.c -o c
julien@ubuntu:~/c/curriculum_by_julien/holbertonschool-low_level_programming/0x0a. m
alloc, free$ ./c | cat -e
Betty Holberton$
julien@ubuntu:~/c/curriculum_by_julien/holbertonschool-low_level_programming/0x0a. m
alloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0B-malloc_free`
- File: `2-str_concat.c`

Done! Help Check your code >_ Get a sandbox

3. If you even dream of beating me you'd better wake up and apologize

mandatory

Write a function that returns a pointer to a 2 dimensional array of integers.

- Prototype: `int **alloc_grid(int width, int height);`

(/)

- Each element of the grid should be initialized to `0`
- The function should return `NULL` on failure
- if `width` or `height` is `0` or negative, return `NULL`

(/)

```
julien@ubuntu:~/0x0a. malloc, free$ cat 3-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * print_grid - prints a grid of integers
 * @grid: the address of the two dimensional grid
 * @width: width of the grid
 * @height: height of the grid
 *
 * Return: Nothing.
 */
void print_grid(int **grid, int width, int height)
{
    int w;
    int h;

    h = 0;
    while (h < height)
    {
        w = 0;
        while (w < width)
        {
            printf("%d ", grid[h][w]);
            w++;
        }
        printf("\n");
        h++;
    }
}

/**
 * main - check the code for ALX School students.
 *
 * Return: Always 0.
 */
int main(void)
{
    int **grid;

    grid = alloc_grid(6, 4);
    if (grid == NULL)
    {
        return (1);
    }
    print_grid(grid, 6, 4);
    printf("\n");
    grid[0][3] = 98;
    grid[3][4] = 402;
    print_grid(grid, 6, 4);
    return (0);
```

```
}
julien@ubuntu:~/0x0a. malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 3
-main.c 3-alloc_grid.c -o g
julien@ubuntu:~/0x0a. malloc, free$ ./g
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

0 0 0 98 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 402 0
julien@ubuntu:~/0x0a. malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0B-malloc_free`
- File: `3-alloc_grid.c`

Done! Help Check your code >_ Get a sandbox

4. It's not bragging if you can back it up

mandatory

Write a function that frees a 2 dimensional grid previously created by your `alloc_grid` function.

- Prototype: `void free_grid(int **grid, int height);`
- Note that we will compile with your `alloc_grid.c` file. Make sure it compiles.

(/)

```
julien@ubuntu:~/0x0a. malloc, free$ cat 4-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * print_grid - prints a grid of integers
 * @grid: the address of the two dimensional grid
 * @width: width of the grid
 * @height: height of the grid
 *
 * Return: Nothing.
 */
void print_grid(int **grid, int width, int height)
{
    int w;
    int h;

    h = 0;
    while (h < height)
    {
        w = 0;
        while (w < width)
        {
            printf("%d ", grid[h][w]);
            w++;
        }
        printf("\n");
        h++;
    }
}

/**
 * main - check the code for ALX School students.
 *
 * Return: Always 0.
 */
int main(void)
{
    int **grid;

    grid = alloc_grid(6, 4);
    if (grid == NULL)
    {
        return (1);
    }
    print_grid(grid, 6, 4);
    printf("\n");
    grid[0][3] = 98;
    grid[3][4] = 402;
    print_grid(grid, 6, 4);
    free_grid(grid, 4);
```

```
    return (0);
}
julien@ubuntu:~/0x0a. malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 4
-main.c 3-alloc_grid.c 4-free_grid.c -o f
julien@ubuntu:~/0x0a. malloc, free$ valgrind ./f
==5013== Memcheck, a memory error detector
==5013== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5013== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5013== Command: ./f
==5013==
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

0 0 0 98 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 402 0
==5013==
==5013== HEAP SUMMARY:
==5013==     in use at exit: 0 bytes in 0 blocks
==5013==   total heap usage: 6 allocs, 6 frees, 1,248 bytes allocated
==5013==
==5013== All heap blocks were freed -- no leaks are possible
==5013==
==5013== For counts of detected and suppressed errors, rerun with: -v
==5013== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x0a. malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0B-malloc_free`
- File: `4-free_grid.c`

Done! Help Check your code >_ Get a sandbox

5. It isn't the mountains ahead to climb that wear you out; it's the pebble in your shoe

#advanced

Write a function that concatenates all the arguments of your program.

- Prototype: `char *argstostr(int ac, char **av);`
- Returns `NULL` if `ac == 0` or `av == NULL`
- Returns a pointer to a new string, or `NULL` if it fails

- Each argument should be followed by a `\n` in the new string

(/)

```
julien@ubuntu:~/0x0a. malloc, free$ cat 100-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * main - check the code for ALX School students.
 *
 * Return: Always 0.
 */
int main(int ac, char *av[])
{
    char *s;

    s = argstostr(ac, av);
    if (s == NULL)
    {
        return (1);
    }
    printf("%s", s);
    free(s);
    return (0);
}
julien@ubuntu:~/0x0a. malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 1
00-main.c 100-argstostr.c -o args
julien@ubuntu:~/0x0a. malloc, free$ ./args I will "show you" how great I am
./args
I
will
show you
how
great
I
am
julien@ubuntu:~/0x0a. malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0B-malloc_free`
- File: `100-argstostr.c`

Done! Help Check your code >_ Get a sandbox

6. I will show you how great I am

#advanced

(/)

Write a function that splits a string into words.

- Prototype: `char **strtow(char *str);`
- The function returns a pointer to an array of strings (words)
- Each element of this array should contain a single word, null-terminated
- The last element of the returned array should be `NULL`
- Words are separated by spaces
- Returns `NULL` if `str == NULL` or `str == ""`
- If your function fails, it should return `NULL`

(/)

```
julien@ubuntu:~/0x0a. malloc, free$ cat 101-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * print_tab - Prints an array of string
 * @tab: The array to print
 *
 * Return: nothing
 */
void print_tab(char **tab)
{
    int i;

    for (i = 0; tab[i] != NULL; ++i)
    {
        printf("%s\n", tab[i]);
    }
}

/**
 * main - check the code for ALX School students.
 *
 * Return: 1 if an error occurred, 0 otherwise
 */
int main(void)
{
    char **tab;

    tab = strtow("      ALX School         #cisfun      ");
    if (tab == NULL)
    {
        printf("Failed\n");
        return (1);
    }
    print_tab(tab);
    return (0);
}
julien@ubuntu:~/0x0a. malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 1
01-main.c 101-strtow.c -o strtow
julien@ubuntu:~/0x0a. malloc, free$ ./strtow | cat -e
ALX$
School$
#cisfun$
julien@ubuntu:~/0x0a. malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0B-malloc_free`

- File: `101-strtow.c`

(/)

Done! Help Check your code >_ Get a sandbox