0x0C. C - More malloc, free

C | Memory allocation

By: Julien Barbier

Weight: 1

Ongoing second chance project - started Oct 6, 2022 6:00 AM, must end by Oct 8, 2022 6:00 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:

- Do I cast the result of malloc? (/rltoken/uKhvfzpF3v8Be10NCZlQtA)

man or help:

- `exit (3)`
- `calloc`
- `realloc`

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/XQ_E28qyePVdJn1lrb_Dfg), **without the help of Google**:

General

- How to use the `exit` function
- What are the functions `calloc` and `realloc` from the standard library and how to use them

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`, `free` and `exit`. 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

Quiz questions

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

Question #0

(/)

To allocate enough space for an array of 10 integers (on a 64bit, Linux machine), I can use:

- malloc(64 * 10)
- malloc(10 * int)
- malloc(10 * sizeof(int))

Question #1

If I want to copy the string “Best School” into a new space in memory, I can use this statement to reserve enough space for it (select all valid answers):

- [x] malloc(sizeof(“Best School”))
- [ ] malloc(strlen(“Best School”))
- [ ] malloc(11)
- [x] malloc(12)
- [ ] malloc(sizeof(“Best School”) + 1)
- [x] malloc(strlen(“Best School”) + 1)

Question #2

malloc returns a pointer

- True
- False

Question #3

malloc returns an address

- True
- False

Question #4

What is wrong with this code:

(/)

```
int cp(void)
{
    char *s;

    s = malloc(12);
    strcpy(s, "Best School");
    return (0);
}
```

- [ ] You don't have enough space to store the copy of the string "Best School"
- [x] There is no comment
- [ ] You can't call `strcpy` with a string literal
- [x] malloc can fail so we should check its return value all the time before using the pointers returned by the function.

Question #5

You can do this:

```
free("Best School");
```

- ( ) Yes
- (•) No

Question #6

You can do this:

```
char str[] = "Best School";

free (str);
```

- ( ) Yes
- (•) No

Question #7

You can do this:

```
char *s;

s = strdup("Best School");
if (s != NULL)
{
    free(s);
}
```

- [x] Yes
- [ ] No

Question #8

The memory space reserved when calling `malloc` is on:

- [ ] The stack
- [x] The heap

Question #9

What will you see on the terminal?

```
int main(void)
{
    int *ptr;

    *ptr = 98;
    printf("%d\n", *ptr);
    return (0);
}
```

- [ ] 0
- [ ] 98
- [ ] It doesn't compile
- [x] Segmentation Fault

Tasks

0. Trust no one

mandatory

Write a function that allocates memory using `malloc` .

- Prototype: `void *malloc_checked(unsigned int b);`
- Returns a pointer to the allocated memory
- if `malloc` fails, the `malloc_checked` function should cause normal process termination with a status value of `98`

```
julien@ubuntu:~/0x0b. more malloc, free$ cat 0-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    char *c;
    int *i;
    float *f;
    double *d;

    c = malloc_checked(sizeof(char) * 1024);
    printf("%p\n", (void *)c);
    i = malloc_checked(sizeof(int) * 402);
    printf("%p\n", (void *)i);
    f = malloc_checked(sizeof(float) * 100000000);
    printf("%p\n", (void *)f);
    d = malloc_checked(INT_MAX);
    printf("%p\n", (void *)d);
    free(c);
    free(i);
    free(f);
    free(d);
    return (0);
}
julien@ubuntu:~/0x0b. more malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 0-main.c 0-malloc_checked.c -o a
julien@ubuntu:~/0x0b. more malloc, free$ ./a
0x1e39010
0x1e39830
0x7f31f6c19010
julien@ubuntu:~/0x0b. more malloc, free$ echo $?
98
julien@ubuntu:~/0x0b. more malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0C-more_malloc_free`
- File: `0-malloc_checked.c`

(/)

Done! Help Check your code

1. string_nconcat

mandatory

Write a function that concatenates two strings.

- Prototype: `char *string_nconcat(char *s1, char *s2, unsigned int n);`
- The returned pointer shall point to a newly allocated space in memory, which contains `s1`, followed by the first `n` bytes of `s2`, and null terminated
- If the function fails, it should return `NULL`
- If `n` is greater or equal to the length of `s2` then use the entire string `s2`
- if `NULL` is passed, treat it as an empty string

```
julien@ubuntu:~/0x0b. more malloc, free$ cat 1-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    char *concat;

    concat = string_nconcat("Best ", "School !!!", 6);
    printf("%s\n", concat);
    free(concat);
    return (0);
}
julien@ubuntu:~/0x0b. more malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 1-main.c 1-string_nconcat.c -o 1-string_nconcat
julien@ubuntu:~/0x0b. more malloc, free$ ./1-string_nconcat
Best School
julien@ubuntu:~/0x0b. more malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0C-more_malloc_free`
- File: `1-string_nconcat.c`

Done! Help Check your code

(/)

2. _calloc

mandatory

Write a function that allocates memory for an array, using `malloc` .

- Prototype: `void *_calloc(unsigned int nmemb, unsigned int size);`
- The `_calloc` function allocates memory for an array of `nmemb` elements of `size` bytes each and returns a pointer to the allocated memory.
- The memory is set to zero
- If `nmemb` or `size` is `0` , then `_calloc` returns `NULL`
- If `malloc` fails, then `_calloc` returns `NULL`

FYI: The standard library provides a different function: `calloc` . Run `man calloc` to learn more.

```
julien@ubuntu:~/0x0b. more malloc, free$ cat 2-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.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
 *
 * Return: Always 0.
 */
int main(void)
{
    char *a;

    a = _calloc(98, sizeof(char));
    strcpy(a, "Best");
    strcpy(a + 4, " School! :)\n");
    a[97] = '!';
    simple_print_buffer(a, 98);
    free(a);
    return (0);
}
julien@ubuntu:~/0x0b. more malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 2-main.c 2-calloc.c -o 2-calloc
```

```
julien@ubuntu:~/0x0b. more malloc, free$ ./2-calloc
0x42 0x65 0x73 0x74 0x20 0x53 0x63 0x68 0x6f 0x6f
0x6c 0x21 0x20 0x3a 0x29 0x0a 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x21
julien@ubuntu:~/0x0b. more malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0C-more_malloc_free`
- File: `2-calloc.c`

Done! Help Check your code

3. array_range

mandatory

Write a function that creates an array of integers.

- Prototype: `int *array_range(int min, int max);`
- The array created should contain all the values from `min` (included) to `max` (included), ordered from `min` to `max`
- Return: the pointer to the newly created array
- If `min > max`, return `NULL`
- If `malloc` fails, return `NULL`

```
julien@ubuntu:~/0x0b. more malloc, free$ cat 3-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.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(int *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
 *
 * Return: Always 0.
 */
int main(void)
{
    int *a;

    a = array_range(0, 10);
    simple_print_buffer(a, 11);
    free(a);
    return (0);
}
julien@ubuntu:~/0x0b. more malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 3-main.c 3-array_range.c -o 3-array_range
julien@ubuntu:~/0x0b. more malloc, free$ ./3-array_range
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
```

```
0x0a
julien@ubuntu:~/0x0b. more malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0C-more_malloc_free`
- File: `3-array_range.c`

Done! | Help | Check your code

4. _realloc

#advanced

Write a function that reallocates a memory block using `malloc` and `free`

- Prototype: `void *_realloc(void *ptr, unsigned int old_size, unsigned int new_size);`
- where `ptr` is a pointer to the memory previously allocated with a call to `malloc` : `malloc(old_size)`
- `old_size` is the size, in bytes, of the allocated space for `ptr`
- and `new_size` is the new size, in bytes of the new memory block
- The contents will be copied to the newly allocated space, in the range from the start of `ptr` up to the minimum of the old and new sizes
- If `new_size > old_size` , the "added" memory should not be initialized
- If `new_size == old_size` do not do anything and return `ptr`
- If `ptr` is `NULL` , then the call is equivalent to `malloc(new_size)` , for all values of `old_size` and `new_size`
- If `new_size` is equal to zero, and `ptr` is not `NULL` , then the call is equivalent to `free(ptr)` . Return `NULL`
- Don't forget to free `ptr` when it makes sense

FYI: The standard library provides a different function: `realloc` . Run `man realloc` to learn more.

```
julien@ubuntu:~/0x0b. more malloc, free$ cat 100-main.c
#include "main.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.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
 *
 * Return: Always 0.
 */
int main(void)
{
    char *p;
    int i;

    p = malloc(sizeof(char) * 10);
    p = _realloc(p, sizeof(char) * 10, sizeof(char) * 98);
    i = 0;
    while (i < 98)
    {
        p[i++] = 98;
    }
    simple_print_buffer(p, 98);
    free(p);
```

```
    return (0);
}
julien@ubuntu:~/0x0b. more malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 100-main.c 100-realloc.c -o 100-realloc
julien@ubuntu:~/0x0b. more malloc, free$ ./100-realloc
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
0x62 0x62 0x62 0x62 0x62 0x62 0x62 0x62
julien@ubuntu:~/0x0b. more malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0C-more_malloc_free`
- File: `100-realloc.c`

Done! Help Check your code

5. We must accept finite disappointment, but never lose infinite hope

#advanced

Write a program that multiplies two positive numbers.

- Usage: `mul num1 num2`
- `num1` and `num2` will be passed in base 10
- Print the result, followed by a new line
- If the number of arguments is incorrect, print `Error`, followed by a new line, and exit with a status of `98`
- `num1` and `num2` should only be composed of digits. If not, print `Error`, followed by a new line, and exit with a status of `98`
- You are allowed to use more than 5 functions in your file

You can use `bc` (`man bc`) to check your results.

```
julien@ubuntu:~/0x0b. more malloc, free$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 101-mul.c _putchar.c -o 101-mul
julien@ubuntu:~/0x0b. more malloc, free$ ./101-mul 10 98
980
julien@ubuntu:~/0x0b. more malloc, free$ ./101-mul 235234693269436436223446526546334
576437634765378653875874687649698659586695898579 286580343650843650834260831096791376082164086314308143086510846508164061340608316083108530861037690137096750671305865708327607320967309780146073697395678645086340863048074509730457034285809348250983420958324098503942850983425098342095834250983420958342534526741363923575589187997046452422615907476091498993541335055687577080701989306920124712185512283638941702255216631601001307425878158314387046118270789357784940867204055508948216034308548261234814532268988302522598879945232929028116992753216059081057377926651337612618248332113256902485974371969385156015068813868274000683912187818601667058605418678284322237297213673482412392922068159291496274311170208689056585352782844484721140846367741649962638649229509281867896067208474178402156294978940712959518351846413859141792380853313812015295333546716634344284086426775480775747808150030732119704867805688704303461042373101473485092019906795014369069932
julien@ubuntu:~/0x0b. more malloc, free$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0C-more_malloc_free`
- File: `101-mul.c`

Done? | Help | Check your code | >_ Get a sandbox