# Memory Allocation

Morten Kühnrich
mokyhn@cs.aau.dk

## Implementing `myalloc`, `myfree` and `myrealloc`

# Introduction

- A memory allocator (MA) manages the memory.
- A process may perform the following operations
  - Request memory: `void *mymalloc(size_t s);`
  - Free memory: `void myfree(void *p);`
  - Get more (or less) memory: `void *myrealloc(void *p, size_t s);`

# Functional specification

```
void *mymalloc(size_t s);
```

- A call to `mymalloc` returns a pointer to a free memory part of size `s` -bytes.
- If there is not room enough – return the `NULL` pointer.
- **Caution** The user might supply strange numbers for `s`, such as 0 or 17. Your MA should be able to handle that.

# Functional specification

```
void myfree(void *p);
```

- The function releases the allocated memory chunk with start adress `*p`.

- **Caution** Running `myfree` on some pointer `*p` without a previous allocation for `*p` is not defined. You can do whatever you like.

# Functional specification

```
void *myrealloc(void *p, size_t s);
```

- The function extends (or shrinks) an allocated memory chunk at `*p` to a size of `s` bytes.

- When extending a chunk it should not change the memory content of the previosly allocated area.

# Functional specification

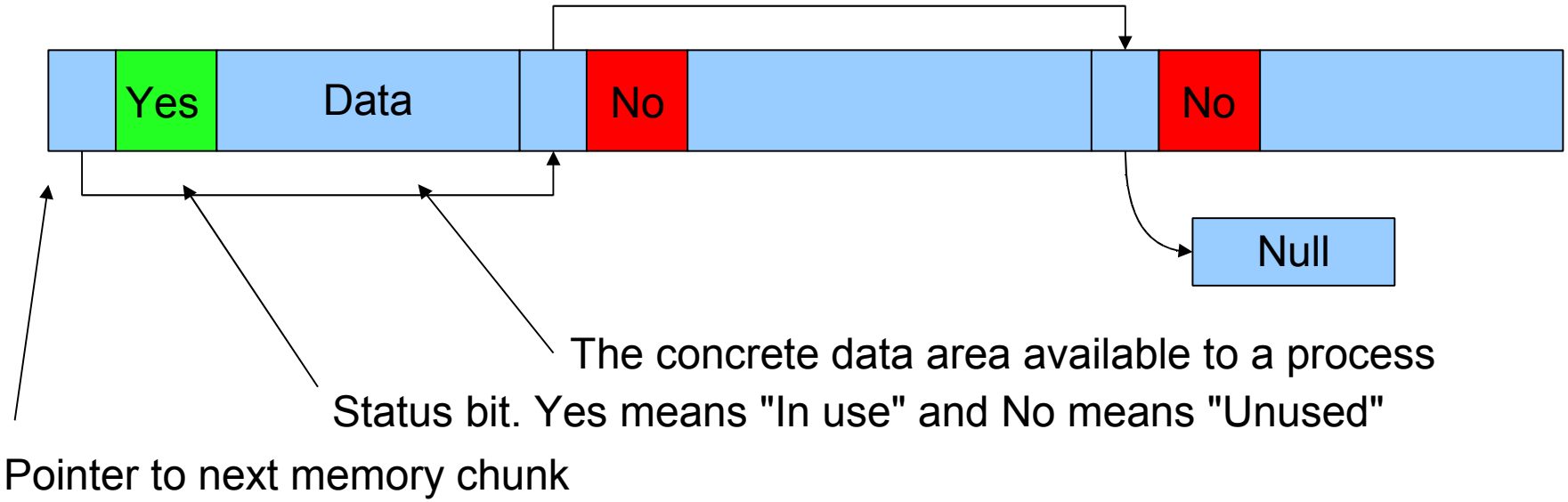
```
void *myrealloc(void *p, size_t s);
```

- If the new size of the memory chunk can be obtained though an extension of the present chunk, do so.
- **If** the new size of the memory chunk requires movement of the chunk **then** free the space for the previous allocation and return a pointer to a new area
- If the space cannot be allocated, return `NULL`.

# Implementation ideas

- You are advised to use a linked list.

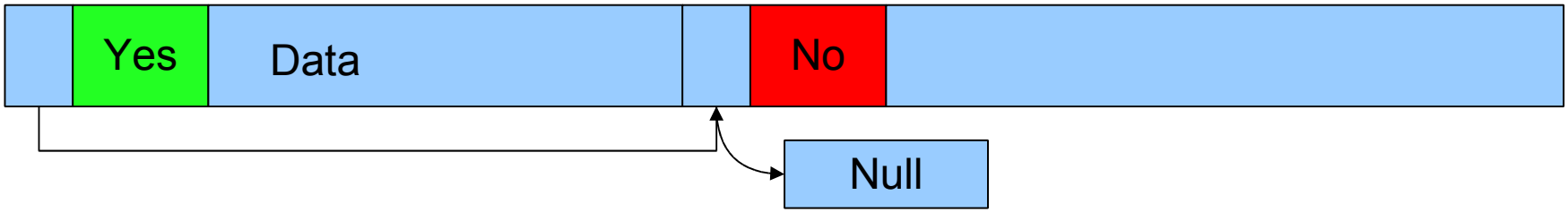
## Memory layout



# Implementation ideas..

- Initial memory layout

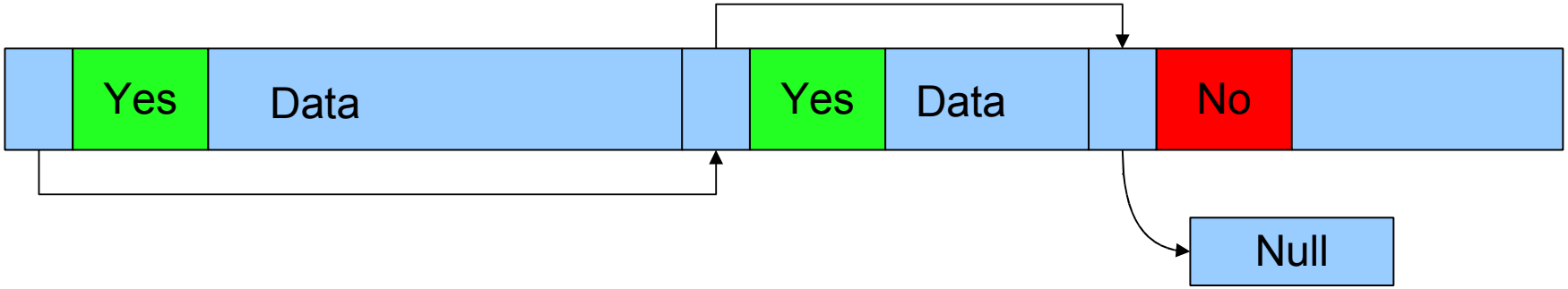

- The first allocation



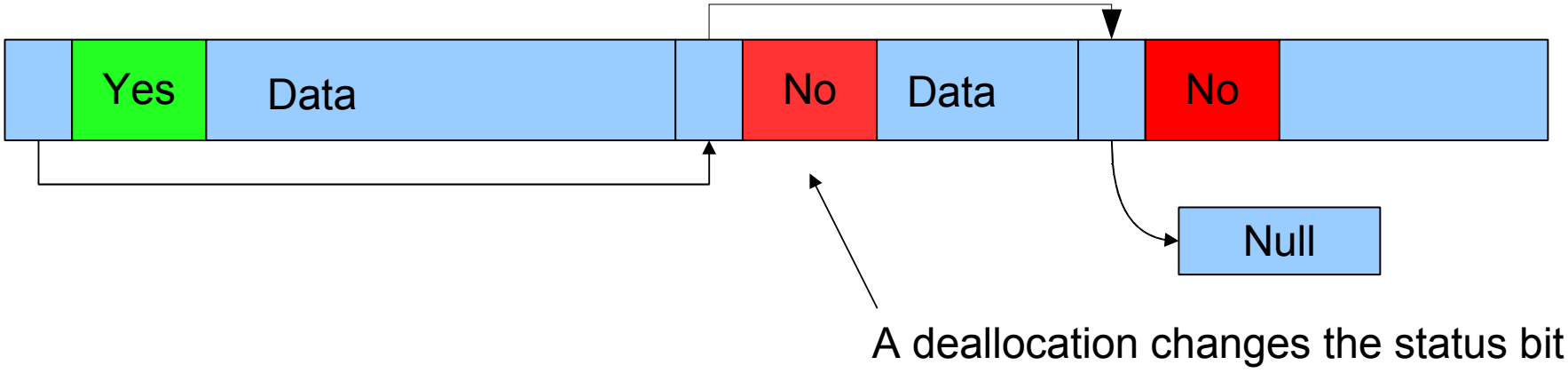

  - Memory is divided in two chunks

# Implementation ideas....

- On allocation more

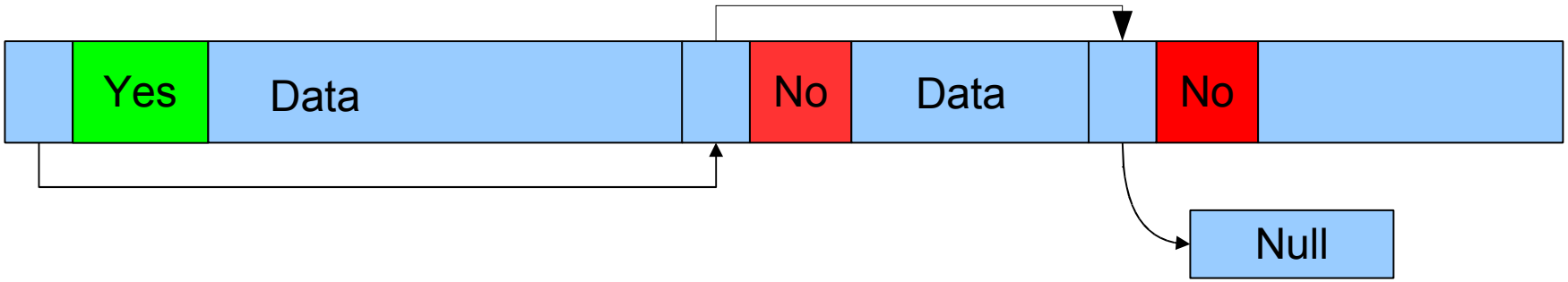



- A deallocation

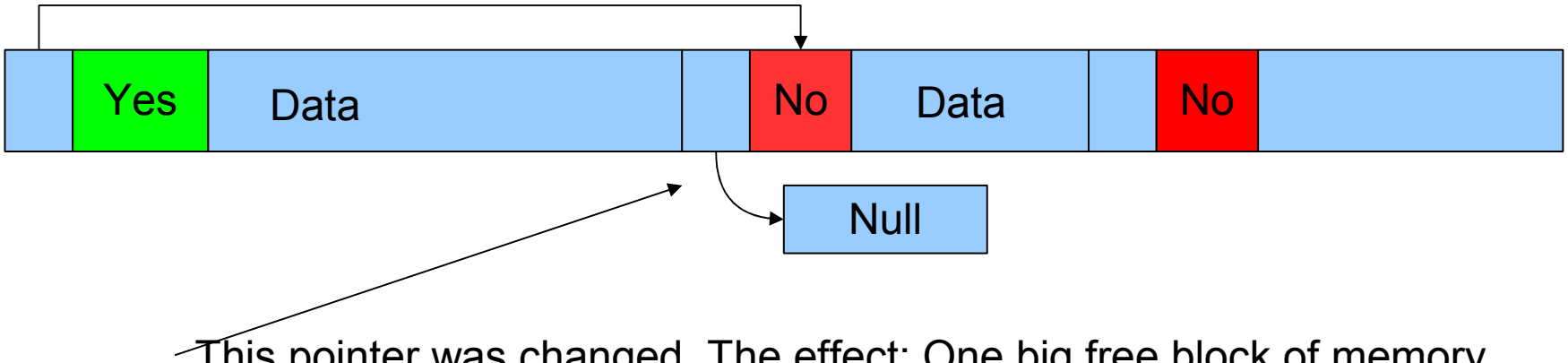


# Implementation ideas.....

- The previous memory layout repeated

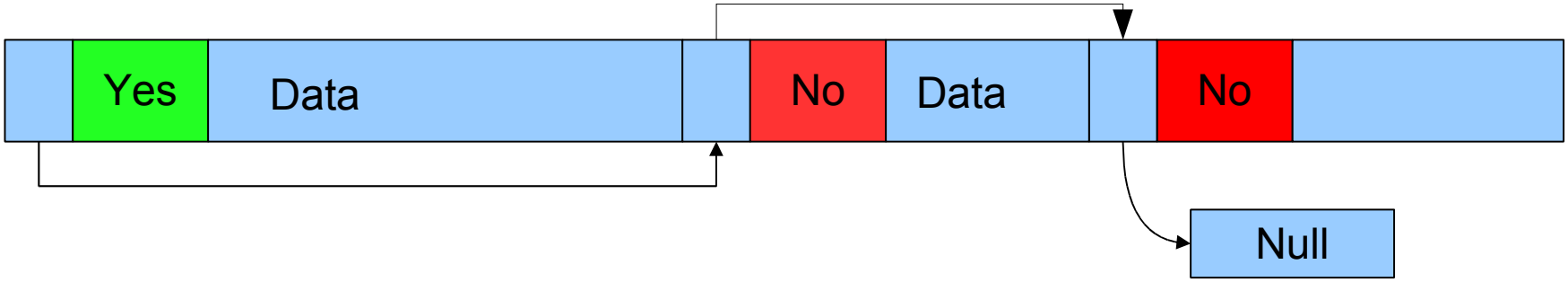



- A memory cleanup

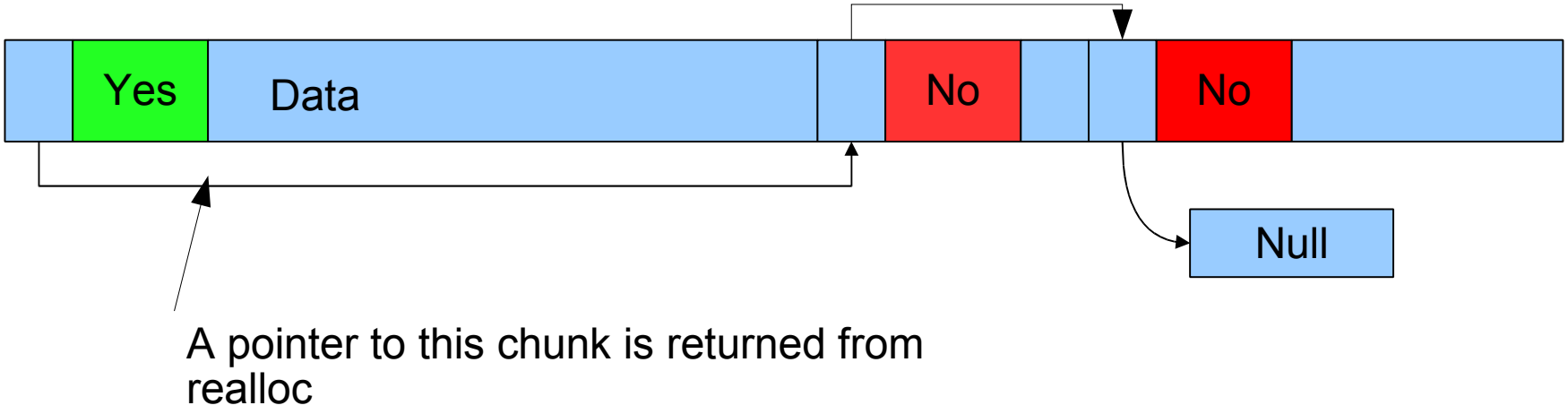


# Implementation ideas.....

- Given the memory layout

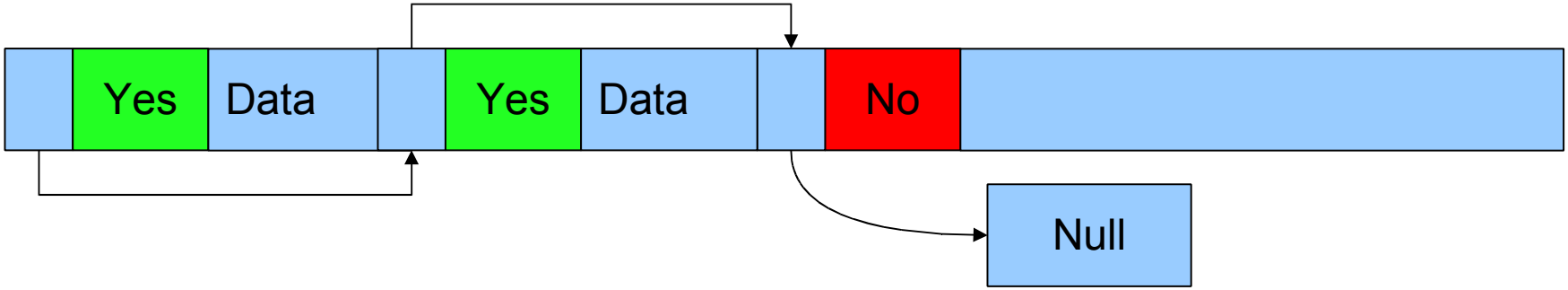



- do a reallocation (simple)

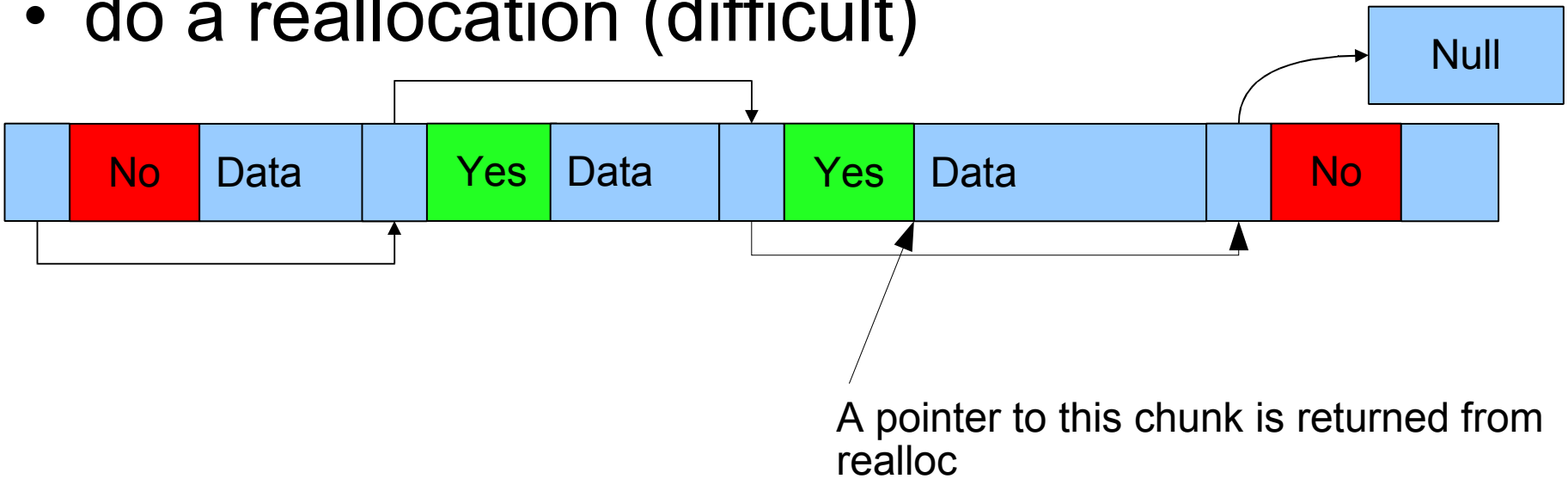


# Implementation ideas.....

- Given the memory layout




- do a reallocation (difficult)



# In pratice

- You should implement a simple cleanup function, otherwise you might end up having trouble.
- The cleanup collapses consecutive chunks which are free.
- Since different threads might allocate and deallocate memory there is a mutual exclusion problem.
- You implementation should therefore be *thread safe.*

# In pratice..

- Use the code templates from the webpage
- It should pass the test by running `testalloc.c`
- Do you own tests of `myrealloc.`
- Use buddy blocks if time permits and you want to.

# Output from a test session

```
Welcome to the test program ver 1.0
    On this architecture, an integer is of size 4 bytes
    On this architecture, an size_t is of size 8 bytes
    Good luck...
    The size of the header is 16
    Beginning basic test
    ---part 1
    ---part 2
    ---part 3
    ---part 4
    ---part 5
    ---part 6
    Basic test passed
    Beginning stress test
    Doing cycle 0 out of 25
    Doing cycle 5 out of 25
    Doing cycle 10 out of 25
    Doing cycle 15 out of 25
    Doing cycle 20 out of 25
    Stress test passed
    Beginning thread test
    Cycle 1 of 5
    Cycle 2 of 5
    Cycle 3 of 5
    Cycle 4 of 5
    Cycle 5 of 5
    Thread test passed
    Congratulations, all tests passed
```