Dynamic Memory Allocation in C

Dhruv Grover | Pranav Bisht | Sujal Singh

The What

What is dynamic memory allocation?

- Dynamic memory allocation is the process of allocating memory at **runtime**.
- Dynamically allocated memory is allocated in a pool of memory called the **heap** (it has nothing to do with the heap data structure) which can grow as required.
- The amount of space required does not have to be known by the compiler in advance.
- To read or write to dynamically allocated memory, **pointers** must be used.
- Allocated block of memory can also be **resized** or **freed** after allocation at runtime.

The Why

Why do we need dynamic memory?

Statically allocated memory has some limitations, such as:

- Exact size and type of storage must be known at **compile time**.
- Since statically allocated memory cannot be resized at runtime
  - It can cause **wastage of memory** if the amount of memory used is less than the memory allocated at compile time.
  - It cannot handle the scenario wherein more memory is required at runtime than initially declared during compilation.

And so, a need for another type of memory allocation arises.

The Why

Dynamically allocated memory solves these problems:

- Unlike static memory, dynamic memory does not require the type and size to be declared during compilation.
- If the amount of space required in a dynamically allocated block increase or reduces, the allocated block of memory can be re-allocated to match the new space requirements. This decreases wastage of memory.

Note

It is important to note that using dynamically allocated memory doesn't automatically solve undefined behaviour, the allocated memory still has to be manually reallocated if the required amount of memory changes.

The How

The Standard Library (`<stdlib.h>`)

The standard library in C provides some functions that handle dynamic memory allocation, these are:

- `malloc()`
- `calloc()`
- `realloc()`
- `free()`

Let us look at these in depth individually.

The How

`malloc()`

- Short for "memory allocate".
- It is used to dynamically allocate a block of memory of specified size.
- Returns a void pointer to the starting address of the allocated block of memory.
- It can also return `NULL` if it is unable to find the requested space in memory.
- Example:

```
int *ptr = (int*) malloc(2 * sizeof(int));
```

The How

`calloc()`

- Short for "clear allocate".
- It is similar to `malloc()` except that it ensures that the allocated block of memory contains zero initialized bytes.
- Example:

```
int *ptr = (int*) calloc(2, sizeof(int));
```

The How

`realloc()`

- Short for "re-allocate".
- It is used to resize an already allocated block of memory without losing the old data.
- If `realloc()` can't find the space in memory where the currently allocated block exists, it will copy it to a new location where the requested amount of space is available.
- Example:

```
int *ptr = (int*) realloc(ptr, 3 * sizeof(int));
```

The How

`free()`

- It is used to free up memory that was dynamically allocated by using `malloc()` or `calloc()`
- It helps in preventing wastage of memory.
- Example:

```c
#include <stdlib.h>

void main() {
    // Allocate some memory using malloc
    int *ptr = (int*) malloc(2 * sizeof(int));

    // Free memory allocated with malloc()
    free(ptr);
}
```

The End