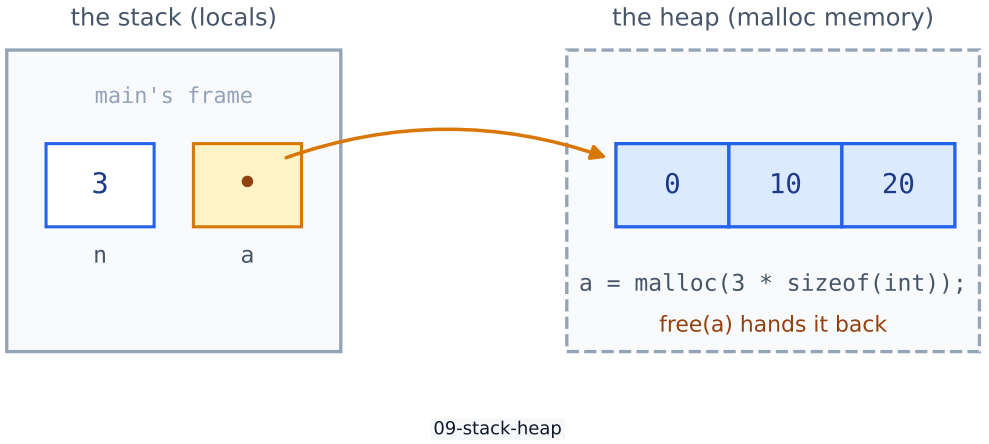
动态内存

C Reference

malloc, free & realloc

`malloc` 在运行时从堆 (heap) 上分配 (allocate) 内存, 并返回一个指向它的指针。`realloc` 调整这块内存的大小;`free` 把它归还。每个 `malloc` 都需要一个对应的 `free`, 否则就会发生内存泄漏 (memory leak)。这些函数在 `<stdlib.h>` 里。



09-stack-heap

局部变量住在栈上;*malloc* 的内存住在堆上, 直到你 *free* 它

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int n = 3;
    int *a = malloc(n * sizeof(int));    // room for 3 ints
    if (a == NULL) return EXIT_FAILURE;
    for (int i = 0; i < n; i++) a[i] = i * 10;

    int *grown = realloc(a, 4 * sizeof *a); // positive size
    if (grown == NULL) {
        free(a);
        return EXIT_FAILURE;
    }
    a = grown;
    a[3] = 30;

    for (int i = 0; i < 4; i++) printf("%d ", a[i]);
    printf("\n");                           // 0 10 20 30

    free(a);                                // hand the memory back
    return 0;
}
```

- `calloc(n, size)` 像 `malloc` 一样分配数组，并且将其初始化为零。

常见错误

- 每个 `malloc` 都要有对应的 `free`; 忘记 free 会内存泄漏。
- `free` 之后再用那块内存 (use-after-free) 是严重的错误。
- 使用内存前, 先检查 `malloc` 没有返回 `NULL`。

对于正数大小，失败的 `realloc` 返回 `NULL` 且保留原有分配不变：在替换唯一的指针前请先检查临时结果。成功时保留旧/新字节数中较小的值；新增单元格未初始化。此示例避免了零大小请求和溢出问题。普通的非静态局部变量具有自动存储期，通常实现于栈上；但 C 并不强制要求这种布局。