Vorlesungsskript zu „Vertiefung Programmieren“ Allocators

Dozent: Dipl.-Inf. (FH) Andreas Schmidt

Allocators

Allocators

Speicherallokation mit einem „Allocator“

Dynamische Speicherallokation mit den klassischen `malloc()/free()` Konstrukt hat die Nachteile, dass:

- Speicherfragmentierung entstehen kann
- Determinismus verloren geht

Dennoch kann eine dynamische Speicherverwaltung in Embedded-Systeme diverse Vorteile bieten.

- Code kann einfacher/lesbarer gestaltet werden
- APIs können generischer gehalten werden
- Wiederverwendung von Speicherallokation vermeidet wiederholte Implementierung von statischer Speicherallokation.

Quelle: https://www.gingerbill.org/article/2019/02/08/memory-allocation-strategies-002/

DHBW
Duale Hochschule
Baden-Württemberg

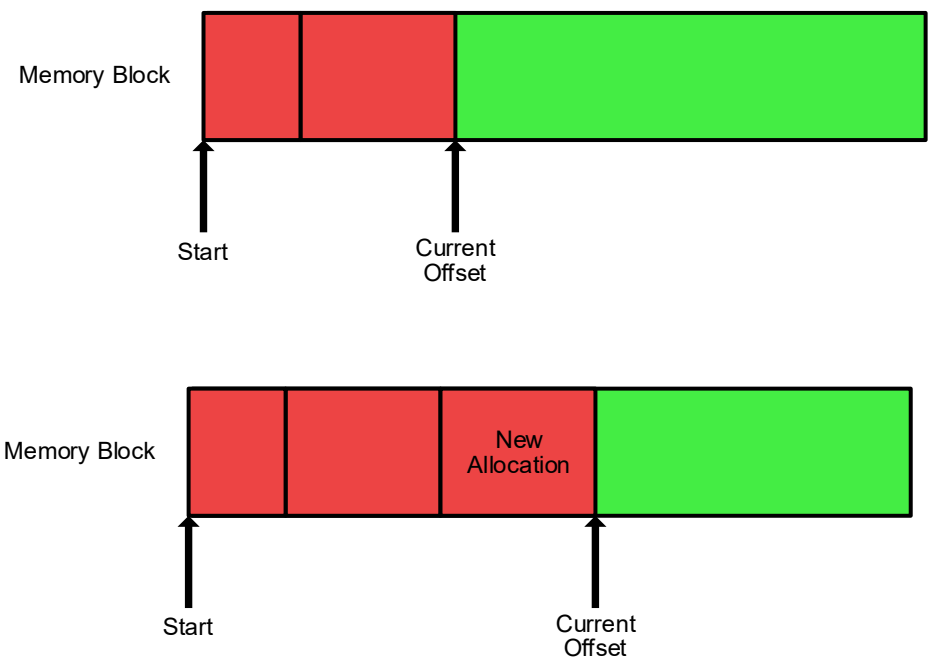
Allocators

Arena/Region Allocator

Arena/Region Allocator

Lineare Speicherallokation mit dem Arena/Region Allocator

Mit Hilfe eines Arena/Region Allocator kann Speicher linear, also in aufeinander folgenden Bereichen, reserviert werden.



Key Facts:

- Allokation von unterschiedliche großen Speicherblöcken ist möglich
- Keine Freigabe von einzelnen Blöcken vorgesehen
- Freigabe erfolgt nach dem Alles oder nichts Prinzip
- Laufzeit einer Allokation ist O(1)
- Sehr einfache Implementierung, da nur ein Offset-Wert für die Realisierung benötigt wird.

Quelle: https://www.gingerbill.org/article/2019/02/08/memory-allocation-strategies-002/

Arena/Region Allocator

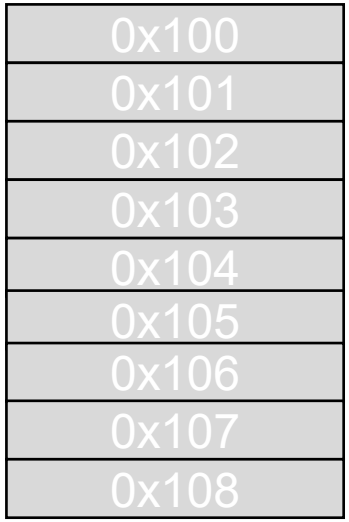
Exkurs: Alignment

```
uintptr_t align_forward(uintptr_t ptr, size_t align) {
    uintptr_t p, a, modulo;

    assert(is_power_of_two(align));

    p = ptr;
    a = (uintptr_t)align;
    // Same as (p % a) but faster as 'a' is a power of two
    modulo = p & (a-1);

    if (modulo != 0) {
        // If 'p' address is not aligned, push the address to the
        // next value which is aligned
        p += a - modulo;
    }
    return p;
}
```



Pointer-Address = Pointer-Address + (Alignment – Modulo)

Beispiel für 4 Byte Alignment

➔ Alignment=4

➔ Module = 0x102 mod 4 = 2

Aligned-Pointer = 0x102 + (4 – 2) = 0x102 + 2 = 0x104

Quelle: https://www.gingerbill.org/article/2019/02/08/memory-allocation-strategies-002/

Arena/Region Allocator

Exkurs: Alignment

```
// Same as (p % a) but faster as 'a' is a power of two
modulo = p & (a-1);
```

0x100	0b100000000
0x101	0b100000001
0x102	0b100000010
0x103	0b100000011
0x104	0b100000100
0x105	0b100000101
0x106	0b100000110
0x107	0b100000111
0x108	0b100001000

0x101 mod 4 = 0x01

```
  0b100000001 (0x101)
& 0b000000011 (0x003)
-------------------------
  0b000000001 (0x001)
```

Bitweise UND-Verknüpfung ermöglicht Modulo Berechnung, wenn der Operand auf einem Exponenten zur Basis 2 (2^n) basiert

Beispiel für 4 Byte Alignment

➔ Alignment a=4

➔ a – 1 = 0x03 = 0b000000011

Quelle: https://www.gingerbill.org/article/2019/02/08/memory-allocation-strategies-002/

Arena/Region Allocator

Lineare Speicherallokation mit dem Arena/Region Allocator

```c
typedef struct Arena Arena;
struct Arena {
    unsigned char *buf;
    size_t         buf_len;
    size_t         prev_offset; // This will be useful for later on
    size_t         curr_offset;
};

void *arena_alloc_align(Arena *a, size_t size, size_t align) {
    // Align 'curr_offset' forward to the specified alignment
    uintptr_t curr_ptr = (uintptr_t)a->buf + (uintptr_t)a->curr_offset;
    uintptr_t offset = align_forward(curr_ptr, align);
    offset -= (uintptr_t)a->buf; // Change to relative offset

    // Check to see if the backing memory has space left
    if (offset+size <= a->buf_len) {
        void *ptr = &a->buf[offset];
        a->prev_offset = offset;
        a->curr_offset = offset+size;

        // Zero new memory by default
        memset(ptr, 0, size);
        return ptr;
    }
    // Return NULL if the arena is out of memory (or handle differently)
    return NULL;
}

#ifndef DEFAULT_ALIGNMENT
#define DEFAULT_ALIGNMENT (2*sizeof(void *))
#endif

// Because C doesn't have default parameters
void *arena_alloc(Arena *a, size_t size) {
    return arena_alloc_align(a, size, DEFAULT_ALIGNMENT);
}
```

Berechnung inkl. Alignment des Offsets

Allokationsvorgang und verschieben des aktuellen Offsets der Arena

Quelle: https://www.gingerbill.org/article/2019/02/08/memory-allocation-strategies-002/

DHBW
Duale Hochschule
Baden-Württemberg

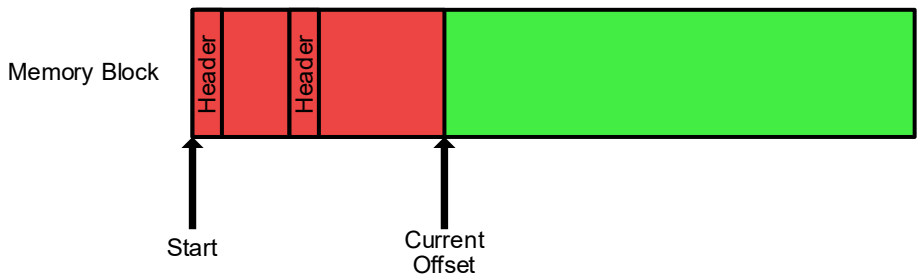
Allocators

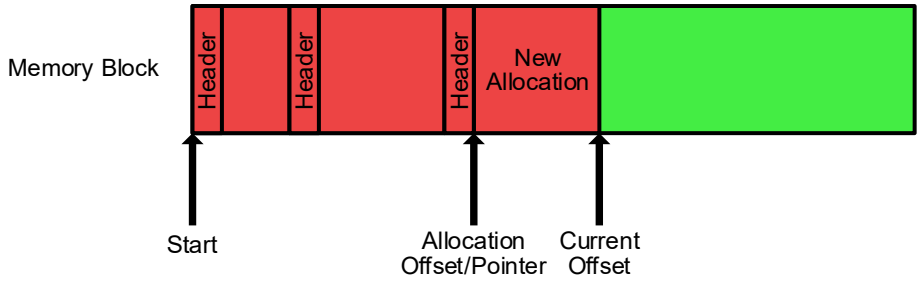
Stack Allocator

Stack Allocator

Speicherallokation mit dem Stack Allocator

Mit Hilfe eines Stack Allocator kann Speicher linear, also in aufeinander folgenden Bereichen, reserviert werden. Eine Freigabe erfolgt nach dem LIFO Prinzip





Key Facts:

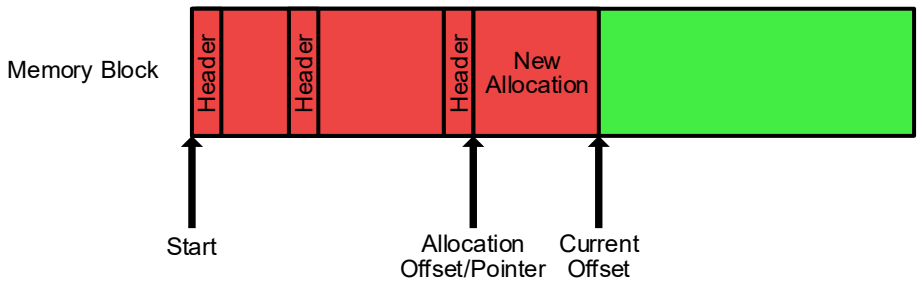
- Allokation von unterschiedliche großen Speicherblöcken ist möglich
- Freigabe von einzelnen Blöcken nach dem LIFO vorgesehen
- Laufzeit einer Allokation und Freigabe ist O(1)

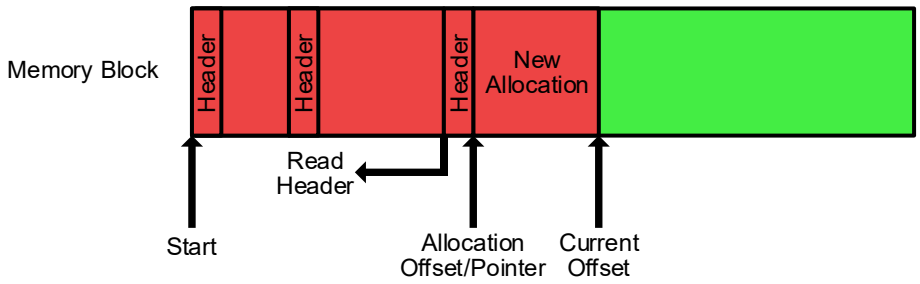
Quelle: https://www.gingerbill.org/article/2019/02/15/memory-allocation-strategies-003/

Stack Allocator

Speicherallokation mit dem Stack Allocator

Mit Hilfe eines Arena/Region Allocator kann Speicher linear, also in aufeinander folgenden Bereichen, reserviert werden.





Freigabe eines Blocks nutzt den „Header“ vor einem Allocations-Block um den Offset-Pointer nach „hinten“ zu verschieben.

Quelle: https://www.gingerbill.org/article/2019/02/15/memory-allocation-strategies-003/

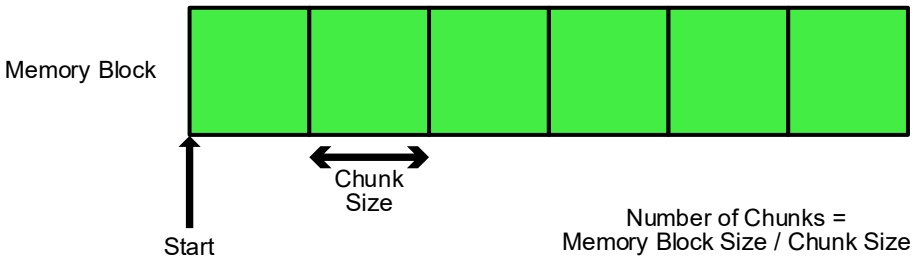
Allocators

Pool Allocator

Pool Allocator

Speicherallokation mit dem Pool Allocator

Mit Hilfe eines Pool Allocator kann Speicher von fixer Größe reserviert werden.



Key Facts:

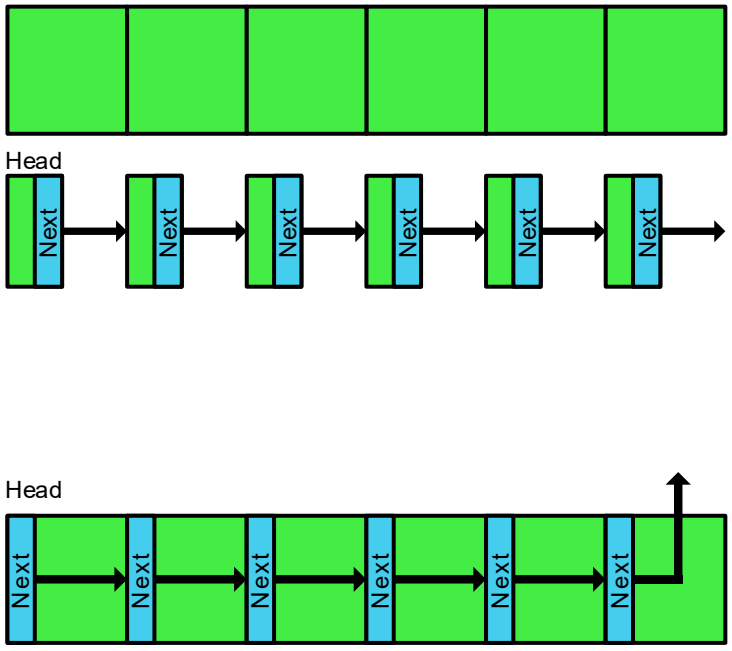
- Allokation von unterschiedliche großen Speicherblöcken ist **nicht** möglich. Alle Speicherblöcke haben eine identische Größe
- Freigabe von einzelnen Blöcken möglich
- Laufzeit einer Allokation und Freigabe ist O(1)

Quelle: https://www.gingerbill.org/article/2019/02/16/memory-allocation-strategies-004/

Pool Allocator

Speicherallokation mit dem Pool Allocator

Mit Hilfe eines Pool Allocator kann Speicher von fixer Größe reserviert werden.



Jeder Speicherblock (Chunk) besitzt die gleiche Größe

Mit Hilfe einer einfach verketteten Liste (Linked-List) werden die freien Speicherblöcke verkettet.

Initial hat die sog. Free List einen Next-Pointer auf jeden Chunk innerhalb des Speicherbereichs

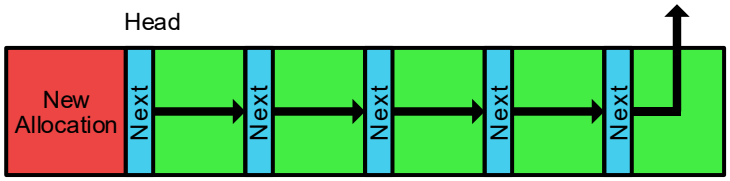
Die „Listen-Header“ werden ebenfalls im Chunk gespeichert

Quelle: https://www.gingerbill.org/article/2019/02/16/memory-allocation-strategies-004/

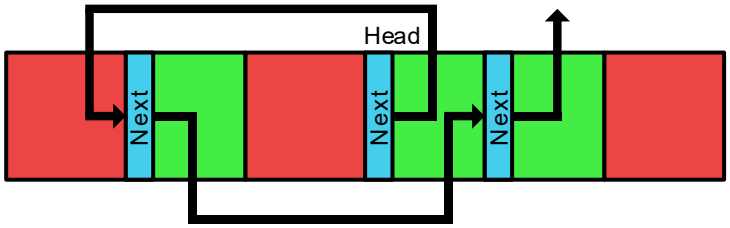
Pool Allocator

Speicherallokation mit dem Pool Allocator

Mit Hilfe eines Pool Allocator kann Speicher von fixer Größe reserviert werden.



Um einen Speicherblock zu allokieren, muss nur der Head-Eintrag auf den nächsten Speicherblock gesetzt werden.



Um einen Speicherblock freizugeben, muss wird der freizugebende Block als neuer Listen-Head gesetzt

Quelle: https://www.gingerbill.org/article/2019/02/16/memory-allocation-strategies-004/