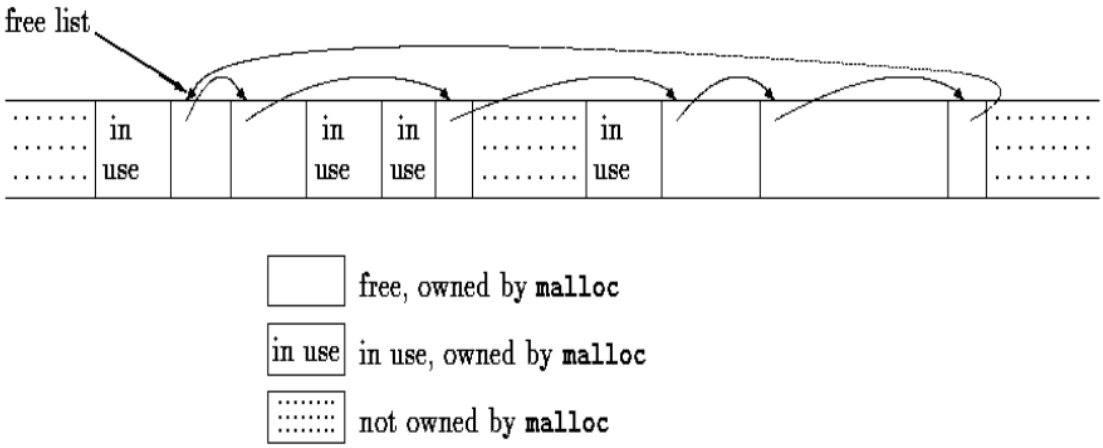
#uni/semester3/Betriebssysteme/chapter14/chapter-notes

➜ stack memory, auch automatic memory genannt, wird von compiler verwaltet
➜ heap memory, wird vom Programmierer verwaltet
➜ sizeof ist ein compile-time operator und kein functioncall. D. h. der Wert ist bei compile time bekannt.
➜ ein function call passiert bei run time.
➜ ein cast ist wenn return type von malloc zu double pointer gemacht wird. Der cast wird eigentlich nicht gebraucht
➜ free akzeptiert kein size argument, d.h. the memory-allocation library muss den allocated region tracken.
➜ neue Sprachen haben ein automatic memory management, da es oft problematisch wird. Malloc wird zu new. Free wird zu garbage collector
➜ segmentation fault bekommt man wenn Spericher nicht allocated wurde
➜ buffer overflow wenn man nicht genug memory allokiert
➜ unitiliazed read ist wenn man den Speicher reserviert hat, aber nicht initialisiert
➜ memory leak wenn man vergisst den Speicher freizugeben
➜ dangling pointer, wenn man free macht, aber weiter den Pointer verwendet.
➜ double free wenn man memory mehrmals freigibt.
➜ invalid free wenn man free nicht mit einem malloc pointer aufruft, sondern mit einem anderen Wert.
➜ break ist der location vom Ende des Heaps des Programms. Man ändert diese mit dem System call bro
➜ anonymous memory (mit mmap gemacht)ist ein Speicherbereich, das mit keiner Datei assoziiert ist sondern mit swap space. Dieser Bereich kann dann wie Heap gehandelt und verwaltet werden.
➜ read Novark et al. [N+07];

Seite 134
https://hikage.freeshell.org/books/theCprogrammingLanguage.pdf
➜ der Speicherraum der malloc verwaltet kann eventuell nicht benachbart sein. Deshalb befindet sich der freie Speicher in einer Liste von freien Blocks. Jeder Block beinhaltet eine Größe, ein Pointer zu nächsten Block und der Speicherraum selber. Die Blocks sind sortiert mit steigende Speicheradresse und der letzte Block (größte Adresse) zeigt auf den ersten.

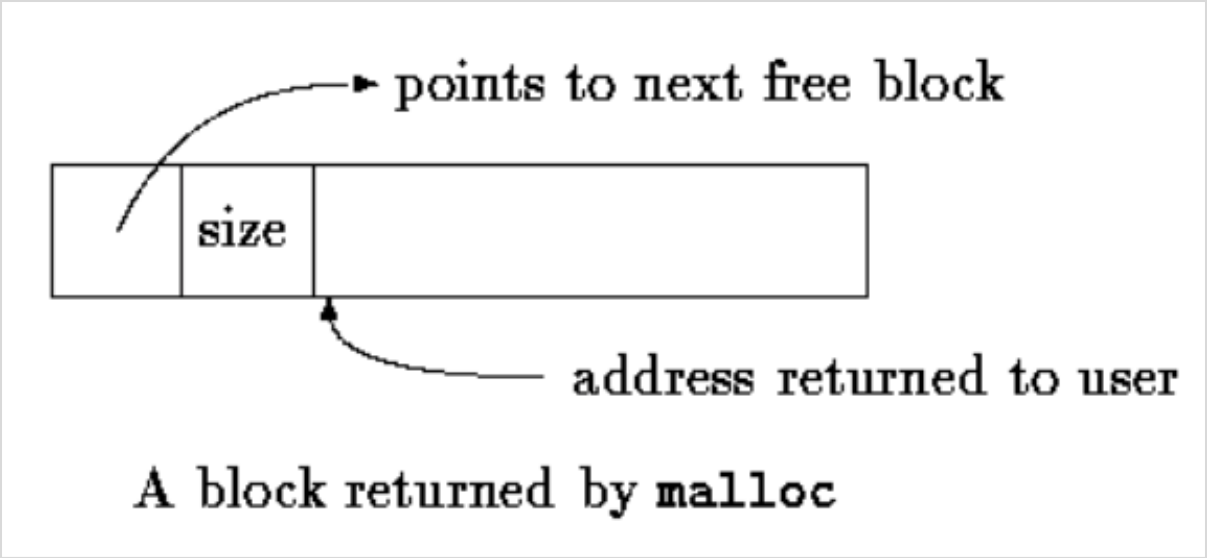


➜ Wenn ein Request gemacht wird, sucht es in der free list nach einem genug großen Block. Dieser Algorithmus heisst first fit. Best fit im Gegensatz sucht nach dem kleinsten Block das die Bedingung erfüllt. Wenn der Block genau die Größe hat, dann wird er zurückgegeben. Wenn der Block zu Größer ist, dann wird er gesplittert und der Speicher der reicht, zurueckgegeben. Wenn kein Block gefunden wird, dann muss ein anderer chunck vom OS in die free list rein gemacht werden.
➜ Beim free() wird auch in die free list gesucht, um den Block am besten Platz reinzutun. Wenn der Block selber benachbart ist mit freie Blocks, dann werden diese zu eins gemacht. Somit wird Fragmentierung verhindert.
➜ Speicher von malloc muss richtig aligned sein, damit Objekte gespeichert werden können. Jedes System hat ein most restrictive type. Wenn er an einer Adresse gespeichert werden kann, dann können andere auch. Diese sind meistens double, int, long, etc. Das wird durch ein union header ermöglicht. Wobei Align die most restrictive type ist. Alle Blocks sind eine vielfache von der header Größe.

```
typedef long Align;    /* for alignment to long boundary */

union header {         /* block header */
    struct {
        union header *ptr; /* next block if on free list */
        unsigned size;     /* size of this block */
    } s;
    Align x;           /* force alignment of blocks */
};

typedef union header Header;
```



➜ das size Feld ist notwendig, weil die Blocks verwaltet von malloc sind vielleicht nicht benachbart. Man kann nicht pointer arithmetic verwenden um die Größen zu berechnen.
➜ wenn malloc nach einem freien Block sucht, macht er da weiter wo er letztes mal einen Block gefunden wurden. Somit wird die free list homogeneuous gehalten.
➜ morecore() in malloc ruft sbrk() auf.
➜ unix system call sbrk(n) gibt ein pointer zu n mehrere Bytes Speicher. Gibt -1 wenn kein Speicher. NULL koennte ein besseres Design sein können, anstatt die -1. Mann muss -1 zu pointer konvertieren char*, um sie mit der return value von sbrk zu vergleichen.

Seite 317
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.2318&rep=rep1&type=pdf

Unterschied malloc, calloc: c - Difference between malloc and calloc? - Stack Overflow

```
#include <stdlib.h>

void *malloc(size_t size);

void *calloc(size_t nobj, size_t size);

void *realloc(void *ptr, size_t newsize);
```

All three return: non-null pointer if OK, NULL on error

```
void free(void *ptr);
```

The pointer returned by the three allocation functions is guaranteed to be suitably aligned so that it can be used for any data object. For example, if the most restrictive alignment requirement on a particular system requires that doubles must start at memory locations that are multiples of 8, then all pointers returned by these three functions would be so aligned.

➜ Wenn man vor oder nach dem allocated Bereich schreibt, kann man den record-information von eigenem Block oder anderem Block ändern und somit korruptieren. (Heap buffer overflow)

http://web4.cs.columbia.edu/~junfeng/10fa-e6998/papers/memcheck.pdf

➜ Memcheck, tracks alle bytes von memory, und aktualisiert diese wenn malloc oder free gerufen wird. So kann man detektieren, wenn Zugriff auf unzugriffbares memory passiert

➜ Tracks alle heap blocks erstellt mit malloc und new. Somit können schlechte oder wiederholte frees erkannt werden.

➜ Schaut dass Argumente bei Funktionen wie strcpy oder memcpy sich nicht überlappen.

➜ tracks jeden Bit von Daten in Registern und Speicher.