Structs are used to create more complex datatypes. For example a circle can be represented with the following struct:

```
struct circle {
    double mx;
    double my;
    double radius;
}
```

Two floating point doubles to store the x and y coordinate of the middle point and a double to store the radius of the circle.

The individual members can be adressed using the . operator, so if you have `struct circle my_circle` its members can be adressed in such a manner:

```
    my_circle.mx = 0.4;
    my_circly.my = 0.33334;
    my_circle.radius = 4.3;
```

Geheugen

Een binary / executable heeft meerdere geheugen gebieden:

- Een .text gebied waar de instructies (code) staat.
- Een .bss en een .data gebied waar globale variabelen opgeslagen zijn.
- Stack (dynamisch) is een continu veranderend geheugen gebied dat gebruikt wordt voor functies om lokale variabelen in op te slaan, bij het aanroepen van een functie groeit de stack om ruimte vrij te maken voor die functie, bij het verlaten (returnen) van een functie wordt de vrijgemaakte ruimte weer vrijgegeven.
- Heap (dynamisch) is een geheugengebied dat groeit met de `malloc` (kort voor 'memory allocate') functie. En krimpt met de `free` functie.
- Geheugen van Shared Libraries (buiten scope)
- Overige geheugengebieden (buiten scope)

Stack

Gister hebben voornamelijk de stack gebruikt, dit zijn variabelen die lokaal in een functie zijn gedeclareerd. Elke functie heeft zijn eigen stackframe, die groeit als variabelen worden aangemaakt en krimpt als de functie verlaten wordt door een return statement.

Als we het geheugen zien als een lange array bytes, dan begint de stack rechts, bij het adres 0xfffffffffff, en groeit deze naar links, richting 0x000000000. Gister kon je dit bijvoorbeeld zijn in het programma `dag1/02_print_array_memory.c`.

Heap

De heap hebben we nog niet in gebruik gezien. Om de heap te gebruiken moet je geheugen op de heap alloceren. Dit wordt gedaan met de functie `malloc()`. Gealloceerd geheugen kan weer worden vrijgemaakt met de functie `free()`.

Pointers

Pointers zijn adressen van variabelen (opgeslagen in een variabele), zo'n adres *point* (wijst) naar een variabele, in dezelfde zin dat een huisnummer *point* (wijst) naar een huis in een straat.

Een pointer wordt als volgt gedeclareerd:

`<var_type>* <var_name>` of `<var_type> *<var_name>`, bijvoorbeeld: `int* ptr_a;` of `char *ptr_b`. De `*` wordt naar smaak achter het type of voor de naam gebruikt, dit leidt wel eens tot verwarring, maar de betekenis is hetzelfde. De regel `int *ptr_a` betekent: Maak een variabele `ptr_a` aan die een pointer is die verwijst naar een integer. Of een pointer naar een integer of een short integer verwijst is belangrijk om te weten. Immers een short integer is (doorgaans) 2 bytes groot en een integer (doorgaans) 4 bytes. Dit betekent dat als een variabele op adres 0x8000 staat, een integer op het geheugen gebied 0x8000 t/m 0x8003 zou staan, terwijl een short int op het geheugengebied 0x8000 t/m 0x8001 zou staan.

Linked Lists

Linked lists is een datastructuur die gebruikt wordt om reeksen variabelen in op te slaan. Een linked list bestaat uit 'nodes' met de volgende structuur:

```
struct node {
    struct node *next;
    int data;
}
```

Hierboven staat een node die gebruikt kan worden om een linked list te maken die reeksen integers opslaat. Deze heeft twee members: 1. `struct node *next` een pointer naar de volgende `struct node`. 2. `int data` een stukje waar we de daadwerkelijke data opslaan, in dit geval slaan we integers op.

Doorgaans werken we volledig met pointers als we met linked lists werken, dit is omdat we de nodes in de heap opslaan, dat betekent dat de variabelen dus niet lokaal in de functie (op de stack bestaan). Om een node aan te maken kan je de volgende code gebruiken:

```
struct node *create_node(int data)
{
    struct node *my_node = malloc(sizeof(struct node));
```

```
    (*my_node).next = NULL;
    (*my_node).data = data;
    return my_node;
}
```

Omdat structs vaak op de heap bestaan en telkens `(*struct).member` erg lelijke code geeft, kan je de `->` operator gebruiken. `struct->member` is een alias voor `(*struct).member`

De volgende code zou je kunnen gebruiken om een linked list zoals het voorbeeld hierboven uit te printen:

```
void print_list(struct node *lst)
{
    while( lst ) {
        printf("%d\n",lst->data);
        lst = lst->next;
    }
}
```