5 Structs, malloc

Oefening 32

1. Schrijf een functie `lees()` die een lijn (bestaande uit 1 of meerdere woorden) inleest vanop het klavier, en een nieuwe C-string teruggeeft. Je weet niet hoe lang de tekst is, maar na oproep van de functie neemt de C-string niet meer geheugenplaats in dan strikt noodzakelijk. Indien de tekst langer is dan 1000 karaktertekens, breek je het af na het 1000^e teken en kuis je de resterende tekst op. Test dit uit door de constante 1000 aan te passen - uiteraard.
   Zorg er ook voor dat op het einde van de resulterende C-string geen newline-karakter staat.
2. Test je functie uit in het onderstaande (half-afgewerkte) hoofdprogramma (haal eerst het latex-bestand af van Minerva, zodat je geen code hoeft over te tikken).
   Vervolledig het hoofdprogramma met 1 regel code.

```
int main(){
    int i;
    for(i=0; i<5; i++){
        char * tekst = lees();
        printf("Ik las in %s.\n",tekst);
    }
    return 0;
}
```

Oefening 33

Werk verder op vorige oefening. Gebruik de functie `lees()` - enkele aanpassingen zijn echter noodzakelijk.

1. Schrijf een functie `lees_meerdere()` die maximaal 6 lijnen tekst inleest vanop het klavier, tot de gebruiker de regel `STOP` intikt. Alle lijnen tekst (behalve `STOP`) worden teruggegeven. Merk op: de functie vraagt hoogstens `MAXAANTAL=6` lijnen tekst aan de gebruiker.
   Voorzie net voldoende plaats voor elke tekst, en net voldoende plaats voor alle teksten (dat zijn er maximaal 6). Na de laatste tekst bewaar je een nullpointer.
2. Test je functie uit in een hoofdprogramma: lees in en schrijf alles weer uit. Uiteraard laat je geen 1000 karaktertekens toe, dat kan je niet controleren. Stel die constante in op 4. Denk eraan om bij het uitschrijven elke tekst te laten volgen door een speciaal karakter (bijvoorbeeld `'!'`), zodat je kan nagaan of er niet teveel witruimte achteraan je tekst plakt.

Oefening 34

Loop niet te snel in deze oefeningen, volg de opdracht nauwgezet. (Dus geen functies/procedures waar het niet staat.)

1. Definieer een struct `Deeltal` met drie velden: `waarde`, `aantal_delers` en `delers`. De `waarde` zal een geheel getal bevatten; `aantal_delers` geeft aan hoeveel delers (tussen 1 en `waarde`-1) dit getal heeft; `delers` is een pointer naar `int`(s), hier zullen de delers te vinden zijn.
2. Werk dit deel uit in het hoofdprogramma.
   Maak lokaal een variabele van het type `Deeltal` aan. Vul op de volgende regels de drie velden van deze variabele in: `waarde` wordt 6, `aantal_delers` wordt 3 en de drie delers zijn 1, 2 en 3. Gebruik hiervoor geen lus.
3. Schrijf een procedure `schrijf_ints(...)` die een array van `int`s meekrijgt, en deze gehele getallen naast elkaar uitschrijft met een liggend streepje tussen.
4. Gebruik de bovenstaande procedure in de procedure `schrijf_deeltal(d)` die een deeltal (=getal en zijn delers) uitschrijft. Voor het getal 6 zou er komen:

   ```
   6  1-2-3
   ```

   Merk op: C laat geen hergebruik van functienamen toe (C++ wel)! Heb je in de schrijf-procedure(s) een kopie gemaakt, of ben je zuiniger geweest?
5. Schrijf de functie `aantal_delers_van(x)` die telt hoeveel delers het geheel getal `x` heeft. (Overloop hiervoor alle getallen van `1` tot en met `x/2`.)
6. Schrijf een functie `delers_van(x,aantal)` die een pointer teruggeeft naar een array die alle delers bevat van het geheel getal `x`. De parameter `aantal` bevat het aantal delers van `x`. Test uit door in je hoofdprogramma de delers van 6 niet handmatig in te geven.
7. Schrijf een procedure `lees_deeltal(g)` die de velden van een deeltal invult: het veld `waarde` wordt hierbij ingelezen vanop het toetsenbord; de andere velden worden automatisch berekend (gebruik de voorgaande functies).
8. Schrijf een procedure `lees_deeltallen(t,aantal)` die een `aantal` deeltallen inleest en bijhoudt in de array `t`. Gebruik bovenstaande procedure (uiteraard).
9. Schrijf een procedure `schrijf_deeltallen(t,aantal)` die een array `t` van deeltallen uitschrijft. Overloop de array met indexering, en gebruikt een hulpprocedure om elk deeltal apart uit te schrijven.
10. Test voorgaande procedures uit in een hoofdprogramma: lees één deeltal in en schrijf het uit. Vraag de gebruiker nadien om een aantal op te geven, waarna je hem/haar exact zoveel deeltallen vraagt. Deze schrijf je ook uit.
11. Schrijf een functie `zoek(waarde, t, aantal)` die een pointer teruggeeft naar het deeltal met waarde `waarde` (type `int`) in de array `t` van deeltallen. (Wat geef je terug indien het gezochte niet aanwezig is?) Denk na over de types in de parameterlijst.
12. Schrijf een of meerdere procedures die memory-leaks voorkomen. Aantal en aard van de parameters bepaal je zelf. Tip: voor je de geheugenplaats van een deeltal weer vrijgeeft aan het geheugen, schrijf je de waarde van dat deeltal uit. Zo kan je zelf nakijken of alle deeltallen zijn vrijgegeven.

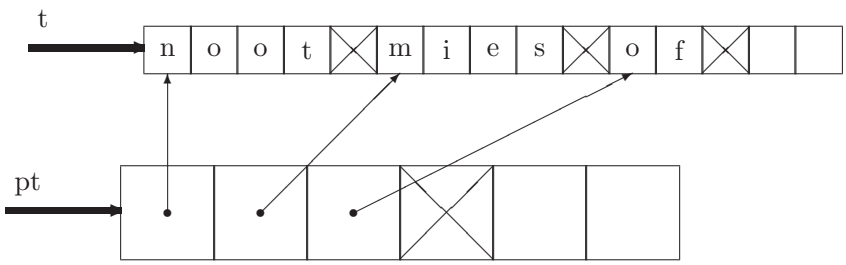
Oefening 35

In deze oefening lezen we een aantal (nl. 10) woorden in, en slaan deze op in één lange array van karaktertekens. Het hoofdprogramma is deels gegeven (haal eerst het latex-bestand af van Minerva, zodat je geen code hoeft over te tikken):

```
#include <stdio.h>
#include <string.h>

#define AANTAL_WOORDEN            10
#define GEMIDDELDE_LENGTE_WOORDEN  7
#define TOTALE_LENGTE_ARRAY   ...   /*gebruik vorige constanten om dit te berekenen */

int main(){
    char* pt[AANTAL_WOORDEN+1];  /* zodat je ook nog een nullpointer kan toevoegen
                                    op het einde van de pointertabel */
    char t[TOTALE_LENGTE_ARRAY];

    pt[0] = t;

    printf("Geef %d woorden in:\n", AANTAL_WOORDEN);
    lees(pt);     /* leest alle woorden in                  */
    schrijf(pt);  /* schrijft alle woorden onder elkaar uit */

    return 0;
}
```

We kunnen toch aan alle woorden apart, als we (zie schets hieronder):

1. elk woord netjes afsluiten met een nullkarakter (dat zal automatisch gebeuren als we `scanf(...)` juist gebruiken), **EN**
2. de start van elk woord onthouden door er een pointer naar te laten wijzen, **EN**
3. toegang hebben tot de onderste tabel uit de schets (toegang tot de bovenste tabel is dan niet meer nodig).



Nog één opmerking: de onderste tabel heeft als laatste element een nullpointer. Dat zal ervoor zorgen dat de schrijfprocedure niet per se moet weten hoe lang de uit te schrijven (pointer)array is.

Schrijf de procedures `lees(...)` en `schrijf(...)`.

Ga voorlopig nog niet na of de gebruiker geen woorden opgeeft die te lang zijn.

Extra
Er werd nog niet expliciet nagegaan of de gebruiker geen woorden opgeeft die te lang zijn. Dit kan je op twee manieren inbouwen.

1. Leg bij elk woord dat je inleest dezelfde beperking op; lees bijvoorbeeld niet meer dan `GEMIDDELDE_LENGTE_WOORDEN` in.
2. Leg bij elk woord dat je inleest een variabele beperking op; namelijk het aantal elementen dat nog vrij is in de array met lettertekens.

De eerste optie heeft het nadeel dat je allicht veel 'overschot' (niet-gebruikte elementen) hebt in de array t. De tweede optie heeft het nadeel dat de gebruiker als eerste woord een heel lang woord kan ingeven - waarna er geen plaats meer is voor de volgende. Pas de oefening daarom als volgt aan, gebruik makend van de tweede manier om in te lezen:

Schrijf een programma dat woorden inleest, tot de gegeven array `t` van lengte `LENGTE_ARRAY_T` vol is (het max aantal worden ligt dus niet vast). Laat de gebruiker telkens weten hoe lang het op te geven woord maximaal mag zijn. Geeft de gebruiker een woord op waardoor de tabel overvol raakt, dan zorg je ervoor dat enkel het eerste deel van dat woord bewaard wordt.