1 Kennismaking met C

main(), types, operatoren, controlestructuren, printf , eenvoudige functies/procedures

Heb je de juiste instellingen gedaan in **Dev-C++** ? Zie richtlijnen installatie (Minerva).
Maak voor elke opgave en nieuw bronbestand (`Bestand > Nieuw > Broncode`) en sla dit op als **.c**-bestand; Maak géén projecten.

Oefening 1

Schrijf een programma dat volgende tekst op het scherm brengt (delay bij uitschrijven van de verschillende getallen is niet nodig).

```
Hello world!
10 9 8 7 6 5 4 3 2 1
START
```

Opmerkingen:

- Heb je de output `10 9 8 7 6 5 4 3 2 1` hardgecodeerd? Niet doen!
- Kan je het aftellen laten beginnen op 100? Gebruik een constante voor de startwaarde van het aftellen.

Oefening 2

Schrijf een programma dat alle (gehele) getallen van 0 tot en met 64 uitschrijft. Per regel komt zowel octale, decimale, als hexadecimale voorstelling van één getal. Zorg ervoor dat de getallen rechts gealligneerd zijn, dus zo:

```
  0    0    0
  1    1    1
  2    2    2
  3    3    3
...
 11    9    9
 12   10    a
 13   11    b
 14   12    c
 15   13    d
 16   14    e
 17   15    f
 20   16   10
 21   17   11
...
 77   63   3f
100   64   40
```

Oefening 3

Gegeven volgend programmafragment, als oplossing voor oefening 1. Dit levert de gevraagde output. Het levert echter op een examen strafpunten op. Waarom?

```
int i;
for(i=10; i>0; i--){
    if(i==10){
        printf("Hello world!\n");
    }
    printf("%d ",i);
    if(i==1){
        printf("\nSTART");
    }
}
```

Oefening 4

Gegeven de opgave *schrijf alle machten van 2 (***beginnend bij** $2^0 = 1$*), kleiner dan 10.000 uit.* De onderstaande code geeft niet de gevraagde uitvoer - hoe kan je dit heel snel detecteren? Pas de code ook aan zodat je de gevraagde uitvoer bekomt.

```
#include <stdio.h>
int main(){
    int macht = 1;
    while(macht < 10000){
        macht *= 2;
        printf("%d ",macht);
    }
    return 0;
}
```

Oefening 5

Deze code levert, in tegenstelling tot vorige oefening, wél de gevraagde output. Toch levert deze oplossing nog minder punten op dan de vorige oplossing.

Waarom?

```
#include <stdio.h>
int main(){
    int macht = 1;
    int i;
    for(i=0; i<20; i++){
        printf("%d ",macht);
        macht *= 2;
        if(macht > 10000){
            break;
        }
    }
    return 0;
}
```

Oefening 6

In deze opgave wordt de faculteit van positieve gehele getallen berekend.

1. Bereken de faculteit van de gehele getallen 2 tot en met 30 en schrijf de resultaten uit. Wat merk je op?
2. Bereken de faculteit met andere gegevenstype (geheel én reëel). Hoe exact is de berekende waarde voor de faculteit van grotere getallen?
3. Pas het programma aan zodat de computer een willekeurig geheel getal kiest tussen 2 en 20 (grenzen inbegrepen). Bereken de faculteit van dit getal.
   Gebruik de functie `rand()`; - meer informatie zoek je online.

Oefening 7

Bij het berekenen van de sinus van een gegeven hoek in radialen, gebruikt je computer of rekenmachine onderstaande reeksontwikkeling.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = \frac{1}{1!}x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots$$

1. Schrijf een programma dat de sinus berekent van 0.23 radialen door de eerste 10 termen van de reeksontwikkeling te berekenen. Opgelet: we kijken naar efficiëntie van je berekeningen! Vergelijk dit met het resultaat van de ingebouwde sinusfunctie uit de bibliotheek `math.h` - tot hoeveel decimalen is dit correct?
   Herneem de berekeningen voor 8.2 radialen, dan is de reeksontwikkeling niet nauwkeurig genoeg.
2. In de vorig oplossing heb je geen enkele zekerheid over de nauwkeurigheid van de reekssom. Pas je programma aan zodat je zeker bent dat de reekssom even correct is als het resultaat van de ingebouwde sinusfunctie.
3. Nu is de waarde van x hard-gecodeerd in het programma. Pas dit aan zodat x door het programma willekeurig gekozen wordt in het interval $[-3.200, 3.200[$. Het getal x bevat dus drie beduidende decimalen.

Het efficiënt implementeren van een reeksontwikkeling is niet evident. Vergelijk bijvoorbeeld het aantal berekeningen dat jouw code uitvoert met het aantal berekeningen van de voorbeeldoplossing. Heb je graag nog wat extra oefeningen op reeksontwikkelingen, dan vind je er op `https://nl.wikipedia.org/wiki/Taylorreeks` nog een hele resem (zie 'Ontwikkelingen'), die bovendien eenvoudig te controleren zijn. Zo kan je bijvoorbeeld de uitkomst die jouw zelfgeprogrammeerde reeksontwikkeling

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} x^{n+1} = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \dots$$

van `ln(1+x)` geeft, vergelijken met de uitkomst van `ln(1+x)`. Kies in dit geval `x` dan wel tussen -1 en 1 zoals de voorwaarde eist, en zoek uit hoe `ln` in C-taal geschreven wordt.

Vanaf de volgende oefeningen heb je theorieles 2 nodig: er worden functies gebruikt (zowel implementatie als oproepen van functies) - ook recursie komt aan bod.

Oefening 8

1. Schrijf een functie `cijfersom(x)` die van een gegeven geheel getal `x` de som van de cijfers berekent. Zo is `cijfersom(12345)` gelijk aan 15.
2. Maak een recursieve versie `cijfersom_rec(x)` die hetzelfde doet als `cijfersom(x)`.

Test de twee functies uit in een eenvoudig hoofdprogramma.

Oefening 9

Herneem oefening 6, maar los dit nu op met twee functies:

1. De functie `faculteit(x)` berekent de faculteit van een gegeven geheel getal `x`.
2. De functie `faculteit_rec(x)` doet een recursieve berekening.

Controleer in een eenvoudig hoofdprogramma.

Oefening 10

Om de grootste gemende deler (ggd) van twee getallen te berekenen, werd je allicht aangeleerd om de twee getallen eerst te ontbinden in priemfactoren, om dan de gemeenschappelijke factoren te ontdekken. Het kan ook anders, met het algoritme van Euclides dat gebruik maakt van volgende gelijkheid:

```
ggd(a , b) = ggd(b , a mod b);
```

De uitdrukking `a mod b` lees je als 'a modulo b' en stelt de rest van `a` bij deling door `b` voor. In C (en C++) gebruik je de notatie `%` in plaats van `mod`. Het algoritme van Euclides vervangt de getallen `a` en `b` (herhaaldelijk) door de getallen `b` en `a mod b`. Indien `a>b`, zullen `b` en `a mod b` kleiner zijn dan `a` en `b` - en dus werd het probleem vereenvoudigd. Dit vervangen stopt van zodra een van de getallen 0 is: `ggd(a,0) = a`.

Schrijf een recursieve functie `ggd(a,b)` die de grootste gemene deler van twee gehele getallen berekent. Test uit met

```
ggd(-6,-8)==2
ggd(24,18)==6
ggd( 0,-5)==5
ggd(6,-35)==1
```

Opmerking: Het is wel belangrijk dat je bij het uittesten duidelijk laat merken aan de gebruiker of de test geslaagd is (zodat hij zelf niet moet zitten narekenen of de gegeven uitkomst juist is). Meer over 'testing' komt in SO I aan bod - voor de studenten die deze cursus op hun curriculum hebben staan.
In deze oefening kan je best een procedure `controleer_ggd(a,b,res)` toevoegen die nagaat of het restultaat dat je bekomt met `ggd(a,b)` overeenkomt met het meegegeven verwachte resultaat en de uitslag van de test rapporteert.

Oefening 11

Wat doet deze code? Verklaar. Na theorieles 3 kan je de code zo omvormen, dat ze ook doet wat ze belooft.

```
void wissel(int a, int b){
     int hulp;
     printf("  Bij start van de wisselprocedure hebben we a=%d en b=%d.\n",a,b);
     hulp = a;
     a = b;
     b = hulp;
     printf("  Op het einde van de wisselprocedure hebben we a=%d en b=%d.\n",a,b);
}

int main(){
    int x, y;
    x = 5;
    y = 10;

    printf("Eerst hebben we x=%d en y=%d.\n",x,y);
    wissel(x,y);
    printf("Na de wissel hebben we x=%d en y=%d.\n",x,y);

    return 0;
}
```

Deze vraag beantwoord je eerst ZONDER de code uit te proberen. Daarna kan je jouw antwoord controleren, door de code te kopiëren, compileren en uit te voeren. Omdat het kopiëren vanuit een .pdf-bestand de kantlijnen niet behoudt, kan je kopiëren vanuit een andere bron. (Zeker nuttig als we je later langere code geven!) Op Minerva vind je een map met .tex-bestanden. Daar haal je het juiste bestand op (`opg_19_11_wissel.tex`, waarbij `19` staat voor jaargang 2018-2019 en `11` het nummer van de oefening is), en kopieer je het programma (te vinden tussen `\begin{verbatim}` en `\end{verbatim}`) in een .c-bestand.