1 Kennismaking met C

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

---

Oefening 1

```
#include <stdio.h>
#define AANTAL 15    /* te kiezen */

int main(){
    int i;
    printf("Hello world!\n");
    for(i=AANTAL; i>0; i--){
        printf("%d ",i);
    }
    printf("\nSTART");
    return 0;
}

/*  MERK OP
 *
 *  DECLARATIE VAN i
 *  C eist dat de teller buiten de lus gedeclareerd wordt. Te betreuren,
 *  maar in latere versies van C ook rechtgezet. Van zodra we in C++
 *  werken, wordt dit als NOT DONE beschouwd (en gequoteerd).
 *  Ook de plaats van declaratie is op de C-manier: vooraan in het
 *  programma. Eveneens NOT DONE beschouwd zodra we met C++ bezig zijn.
 */
```

Oefening 2

```
#include <stdio.h>

int main(){
    int i;
    for(i=0; i<=64; i++){
        printf("%4o %4d %4x\n",i,i,i);
    }
    return 0;
}
```

Oefening 3

```
/*
Testen op de teller binnen een for-lus is NOT DONE,
tenzij in het onwaarschijnlijke geval dat de test een ingewikkelde
berekening op die teller vraagt.
Maar testen of je aan het begin dan wel aan het einde van de lus zit,
DOE JE NIET.
Ga maar eens na wat deze overbodige testen voor de efficientie van je
programma betekenen, als we niet van 10 tot 1 maar van
100000000000000000000000000000 tot 1 laten aftellen!
*/
```

Oefening 4

```
/*  int macht = 1;
    while(macht < 10000){
        macht *= 2;
        printf("%d ",macht);  // TE LAAT
    }

De volgorde van derde en vierde regel moet omgewisseld worden.
Dit is heel eenvoudig te vermijden, door volgende tip toe te passen:


De VIER delen van de while-lus zijn:

DEEL 1       klaarzetten van het 'item' waarop getest wordt
DEEL 2       while (voorwaarde waar item aan moet voldoen om
                    door te gaan in de lus){
   DEEL 3            // hier komen alle zaken die wel herhaald moeten
                     // worden, maar niet tot DEEL 4 behoren
   DEEL 4            // opnieuw klaarzetten van het 'item' waarop getest
                     // wordt (dit kan eventueel meerdere regels beslaan)
             }

De JUISTE VOLGORDE om deze vier delen te schrijven zijn als volgt:

Eerst schrijf je DEEL 2.
Uit de opgave blijkt namelijk heel makkelijk hoelang je mag doorgaan met
herhalen, dus volgt er ook snel het 'item' waarop getest wordt.

Dan schrijf je DEEL 1 EN DEEL 4.
Deel 1 zet het item klaar (deftige initialisatie);
deel 4 is daar (dikwijls) quasi een kopie van.
Je zet deel 4 ook meteen ONDERAAN de lus.

Dan schrijf je DEEL 3.
Alles wat moet herhaald worden, maar niet tot deel 4 behoort, komt VOOR
deel 4, tussen de accolades.
*/
```

Oefening 5

```
/*
Als je een for-lus gebruikt, geef je de lezer het sein dat je op voorhand
weet hoe dikwijls je de code gaat uitvoeren.
Als je dan toch al van plan was om tussendoor van gedachten te veranderen
(zie de 'break' in de 'if'), had je dat beter op voorhand duidelijk
gemaakt door een (correcte) while-lus te gebruiken.

Dat is ook de bestaansreden van BEIDE lussen:
de for-lus gebruik je als het aantal herhalingen op voorhand vastligt,
de while-lus gebruik je in de andere gevallen.

Bovendien: de bovengrens 20 in de for-lus kan ondergedimensioneerd zijn
als je de grens 10000 aanpast.
Dat duidt er nogmaals op dat deze oplossing niet foolproof is.
(Ja, ze zou correcter met constanten kunnen werken,
maar dan nog blijft de opmerking over de 'break' die niet netjes is!)
*/
```

Oefening 6

```
/* versie 1 is correct tot 12 !  */
#include <stdio.h>
int main(){
    long int fac = 1;
    int n = 30;
    int i;
    for(i=2 ; i<=n ; i++){
        fac *= i;
        printf("%d! = %lld \n",i,fac);
    }
    return 0;
}

/* versie 2 - met unsigned long long is het resultaat tot 20 !*/
#include <stdio.h>
int main(){
    unsigned long long int fac = 1;
    int n = 30;
    int i;
    for(i=2 ; i<=n ; i++){
        fac *= i;
        printf("%d! = %lld \n",i,fac);
    }
    return 0;
}

/* met double is niet veel verder correct want er zijn te weinig beduidende cijfers,
    dus het resultaat is afgerond ! */
#include <stdio.h>
int main(){
    double fac = 1;
    int n = 30;
    int i;
    for(i=2 ; i<=n ; i++){
        fac *= i;
        printf("%d! = %.0f \n",i,fac);
    }
    return 0;
}

/* computer kiest getal */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(){
    unsigned long long int fac = 1;
    int i, n;
    srand(time(NULL)); /* Laat je dit weg, dan genereert het programma bij elke run
        telkens hetzelfde getal.
                        Probeer uit!  */
    n = rand()%19 + 2;
    for(i=2 ; i<=n ; i++){
        fac *= i;
    }
    printf("%d! = %lld \n",n,fac);
    return 0;
}
```

Oefening 7

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

/* LET OP! om zo min mogelijk zaken te herberekenen, bewaar je telkens de laatst
 *    gebruikte term (bvb -x^3/3!) en leid je daaruit de  volgende term af.
 *
 *    Anders zou je veel berekeningen opnieuw doen.  Vergelijk:
 *        om x^5/5! rechtstreeks uit te rekenen, heb je volgende bewerkingen nodig:
 *                    x*x*x*x*x / (1*2*3*4*5)
 *
 *        om x^5/5! af te leiden uit -x^3/3! (het 'vorige resultaat'),
 *             heb je volgende bewerkingen nodig:
 *             ('vorige resultaat') * (-x*x)/(4*5)
 *
 * Dit verschil vergroot als je verder in de reeksontwikkeling bent.
 */

/********************************** versie 1 *********************/

int main(){
    double x = 0.23;
    double term = x;
    double som;
    int n = 2 * 10;
    int i;

    som = x;
    for(i=2 ; i<n ; i += 2){
        term *= -x*x / (i*(i+1));
        som += term;
    }
    printf("sin(%.2f) =  %.16f \n",x,som);
    printf("controle     %.16f \n",sin(x));
    return 0;
}

/********************************** versie 2 *********************/

/* double heeft 16 beduidende cijfers en je zoekt een waarde tussen -1 en 1
termen die kleiner zijn dan 1e-16 zullen geen invloed meer hebben op de som*/

#define EPS 1e-16
int main(){
    double x = 0.23;
    double term = x;
    double som = term;
    int i=2;

    while (fabs(term)>EPS){
        term *= -x*x / (i*(i+1));
        som += term;
        i +=2;
    }

    printf("sin(%.2f) =  %.16f \n",x,som);
    printf("controle     %.16f \n",sin(x));
    return 0;
}
```

```
/********************************** versie 3 *********************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define EPS 1e-16
int main(){
    double x, term, som;
    int i;

    srand(time(NULL));
    x = (rand()%6400-3200)/1000.0;
   /* let op! vraag je 4 cijfers na de komma, dan krijg je geen gelijkmatige
      randomverdeling, omdat de bovengrens van rand() kleiner is dan 64000 !! */

    term = x;
    som = x;
    i = 2;
    while (fabs(term)>1e-16){
        term *= -x*x/(i*(i+1));
        som += term;
        i +=2;
    }

    printf("sin(%.2f) = %.15f \n",x,som);
    printf("controle    %.15f \n",sin(x));
    return 0;
}

/* De rand-functie van C is niet geschikt voor doorgedreven statistische
    toepassingen!!
   Het bereik is te klein, en er valt wel wat aan te merken op de kwaliteit.
   In C++ heb je nieuwe bibliotheken die meer garanties bieden.
*/
```

Oefening 8

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int cijfersom(int x){
     int som = 0;
     while(x>0){
          som += x%10;
          x /= 10;
     }
     return som;
}

/* 'rec' staat voor 'recursief' */
int cijfersom_rec(int x){
     if(x<10) return x;
     return x%10 + cijfersom_rec(x/10);
}
```

```
int main(){
     int getal;

     srand(time(NULL));

     /* rand() geeft willekeurig positief getal kleiner dan RAND_MAX (=32767)
        We willen elke getal met 5 cijfers genereren*/
     getal = 10000 + (rand()+rand()+ rand())%90000;

     printf("De cijfersom van %d     is %d  \n",getal, cijfersom_herhaald(getal));
     printf("De cijfersom_rec van %d is %d \n",getal,cijfersom_rec(getal));

     return 0;
}
```

Oefening 9

```
#include <stdio.h>

unsiged long long int faculteit (int x){
    int i;
    unsigned long long int fac = 1;
    for(i = 2; i<= x; i++){
          fac *= i;
    }
    return fac;
}

unsiged long long int faculteit_rec(int x){
    if(x<2) return 1;
    return x*faculteit_rec(x-1);
}


int main(){
    int i;
    for(i=2 ; i<=20 ; i++){
        printf("%d ! = %lld = %lld\n", i , faculteit(i),faculteit_rec(i));
    }
    return 0;
}

/* MERK OP:
 * Voor wie zich de vraag stelt: de snelheid van de rechtstreekse
 * berekening en de recursieve zijn niet merkbaar verschillend.
 * Enkel als de stack (= geheugenruimte die bijhoudt waar het
 * programma gebleven was met de recursie) vol begint te raken, zou je
 * vertraging kunnen zien. Maar tegen dan zijn we al lang de limiet voor
 * de overflow gepasseerd...
 *
 * Wil je tijdsverschillen echt afmeten tegen elkaar, dan zou je een
 * grote herhaling moeten inbouwen, en timen. Dit is hier echter niet
 * gevraagd.
 */
```

Oefening 10

```
#include <stdio.h>

int ggd(int a, int b){
    if(a<0 || b<0){
        return ggd(abs(a),abs(b));
    }
    if(a==0 || b==0){
        return a+b;    /* een van beide is nul; de ggd is de andere */
    }
    return ggd(b,a%b); /* indien a<b, zal dit gelijk zijn aan ggd(b,a)
                          en kan de recursie vanaf die stap de getallen
                          echt verkleinen */
}


/*Procedure om de ggd te testen, en dit te rapporteren*/
void controleer_ggd(int a, int b, int result){
     printf("ggd(%d,%d) is %0sok\n",a,b,(ggd(a,b)!=result? "NIET ":""));
}

int main(){
    controleer_ggd(24,-36,12);
    controleer_ggd(24,1,1);
    controleer_ggd(-9,-24,9);
    controleer_ggd(-6,-8,2);
    controleer_ggd(24000,5000,1000);
    controleer_ggd(0,-5,5);
    return 0;
}

/* Merk op:
   de waarden voor het uittesten (zowel a,b als uitkomst) zouden we beter
   in een array bewaren, zodat het controleren in een lus geschreven kan
   worden. Dat kan je doen na volgende theorieles.   */
```

Oefening 11

```
/*
Wat hier op te merken viel:

blijkbaar KOPIEERT C de waarde van de meegegeven parameters naar de
variabelen die klaarstaan in de parameterlijst van de functie/procedure;
in het hoofdprogramma blijven de oorspronkelijke waarden dus ongewijzigd.
*/
```