
Programmeermethoden

Functies — vervolg

Walter Kosters en Jonathan Vis

week 5: 2–6 oktober 2023

www.liacs.leidenuniv.nl/~kosterswa/pm/

Een eenvoudige void-functie:

```
// kort infoblokje
void infoblokje ( ) {
    cout << "Hallo allemaal ..." << endl;
    cout << "Enzovoorts ..." << endl;
} //infoblokje

int main ( ) {
    ...
    infoblokje ( );
    ...
} //main
```

Let op het inspringen.

Een functie die zijn parameters omwisselt:

```
void wissel (int & x, int & y) { // call by reference: &
    int hulp = x;
    x = y;
    y = hulp;
} //wissel
```



Met aanroep:

```
int a = 33, x = 88;
cout << a << " en " << x << endl; // 33 en 88
wissel (a,x);
cout << a << " en " << x << endl; // 88 en 33
```

Tel x en y op (als return-waarde resp. in z):

```
int telop (int x, int y) {  
    return x + y;  
} //telop                aanroep: som = telop (8,13);  
void telop2 (int x, int y, int & z) {  
    z = x + y;  
} //telop2              aanroep: telop2 (8,13,som);
```

Tel breuken $\frac{x_1}{x_2}$ en $\frac{y_1}{y_2}$ op in $\frac{z_1}{z_2}$:

```
void telbreukenop (int x1, int x2, int y1, int y2,  
                  int & z1, int & z2) {  
    z1 = x1 * y2 + y1 * x2;  
    z2 = x2 * y2;  
} //telbreukenop
```

[vierde werkcollege](#)

Gevraagd: zet de cijfers van een getal op aparte regels, het laatste cijfer eerst:

```
// hier moet commentaar staan, dus:  
// zet cijfers van getal > 0 omgekeerd op aparte regels  
void cijfers (int getal) {  
    while ( getal != 0 ) {  
        cout << getal % 10 << endl; // of tel de cijfers ...  
        getal = getal / 10;  
    }//while  
}//cijfers
```

Neem aan dat `getal` minstens 1 is.

Een getal omkeren (196 → 691) of het aantal cijfers van een getal tellen (6171 → 4) gaat analoog!

Geen strings, geen arrays!

Bij **top-down** maak je een functie als je deze nodig hebt, bij **bottom-up** bedenk je deze daarvoor al.

Voorbeeld: machtverheffen, $y = x^7$. Bij bottom-up gebruik je `pow` uit `<cmath>` of uit "zelf.h", bij top-down maak je:

```
// bereken x tot de n-de voor n >= 0
int machtsverheffen (int x, int n) {
    int i; // tellertje
    int res = 1; // om resultaat in op te bouwen
    for ( i = 1; i <= n; i++ ) { res = res * x; }//for
    return res;
}//machtsverheffen
```

Daarna zet je dit misschien alsnog in "zelf.h".

PS En met 5 vermenigvuldigingen x^{15} berekenen?

```
void john (int x, int y) { ... }//john
```

```
int paul (double x, bool b) { ... }//paul
```

```
void george ( ) { ... }//george
```

```
bool ringo (int & getal) { ... }//ringo
```

```
int main ( ) { ... }//main
```

De functie george mag de functies george (“recursie”), john en paul gebruiken = aanroepen, maar *niet* de functie ringo! Wil je dat toch, dan moet je boven george een **prototype** `bool ringo (int & getal);` toevoegen.

Dus bij

```
bool ringo (int & getal); // prototype ringo
```

```
void george ( ) { ... }//george
```

```
bool ringo (int & getal) { ... }//ringo
```

```
int main ( ) { ... }//main
```

mogen ringo en george elkaar beide(n) aanroepen! Dankzij het prototype van ringo (let op de ;) mag george de eigenlijk verderop gedefinieerde ringo toch gebruiken.

En om misverstanden te vermijden: functies worden binnen functies aangeroepen, maar *na elkaar* en niet binnen elkaar gemaakt.

Functies hebben verschillende soorten parameters:

- globaal — gelden overal
- lokaal — gelden alleen binnen een functie (of ...)
- formeel — staan in functie-heading
- actueel — bij aanroep van een functie
- call by value — geef waarde door
- call by reference — geef (adres van) variabele door

```
void hoogop (int x) { x = x + 10; cout << x; }//hoogop
```

```
void maaknul (int t) { t = 0; cout << t; }//maaknul
```

```
int x, m, q;  
x = 7; hoogop (x); cout << x;           17 7  
m = 3; hoogop (m+8); cout << m;        21 3  
q = 5; maaknul (q); cout << q;         0 5  
maaknul (42);                           0
```

Er wordt alleen een *waarde* doorgegeven, en wel van de *actuele* parameter aan de *formele* parameter; er wordt dus een “lokale kopie” gemaakt, wat tijd en ruimte kost.

Functies — vervolg **Voorbeeld — call by reference**

```
void hoogop (int & x) { x = x + 10; cout << x; }//hoogop
```

```
void maaknul (int & y) { y = 0; cout << y; }//maaknul
```

```
int x, m, q;  
x = 7; hoogop (x); cout << x;           17 17  
m = 3; hoogop (m+8); // VERBODEN!!!  
q = 5; maaknul (q); cout << q;        0 0  
maaknul (42); // VERBODEN!!!
```

Er wordt nu een *adres* (een *pointer*) doorgegeven. De *actuele* parameter kan nu wel veranderen. De *actuele* parameter mag geen “rare” expressie als $m+8$ of 42 zijn. Er wordt alleen een *adres* gekopieerd.

En met een **globale variabele** erbij:

```
int globaal; // globale variabele, geldt overal, vermijden
```

```
int doewat (char kar, double & getal) {  
    // kar is call by value, getal call by reference  
    int lokaal; // locale variabele, geldt binnen doewat  
    ...  
} //doewat
```

```
void nogeen ( ) {  
    double lokaal; // locale variabele, geldt binnen nogeen  
    cout << doewat (globaal, lokaal) << endl;  
    // waarde van globaal gaat naar kar (met casting)  
    // lokaal "is" hetzelfde als getal uit doewat  
} //nogeen
```

```
int a; int b;
void kwadraat (int a) { // call by value
    a = pow (a,2); // uit <cmath>, oftewel a * a
    b++;
    cout << "0: " << a << " en " << b << endl;
} //kwadraat
```

Nu doen we:

```
(1) a = 5; b = 13; kwadraat (a);
    cout << "1: " << a << " en " << b << endl;
(2) a = 2; b = 7; kwadraat (b);
    cout << "2: " << a << " en " << b << endl;
```

Dat levert

```
0: 25 en 14
1: 5 en 14
```

```
0: 49 en 8
2: 2 en 8
```

```
int a; int b;
void kwadraat (int & a) { // call by reference
    a = pow (a,2); // uit <cmath>, oftewel a * a
    b++;
    cout << "0: " << a << " en " << b << endl;
} //kwadraat
```

Nu doen we:

```
(3) a = 5; b = 13; kwadraat (a);
    cout << "3: " << a << " en " << b << endl;
(4) a = 2; b = 7;  kwadraat (b);
    cout << "4: " << a << " en " << b << endl;
```

Dat levert

```
0: 25 en 14
3: 25 en 14
```

```
0: 50 en 50
4: 2 en 50
```

```
void alias (int r, int & s) {
    int t;
    t = 3;
    r = r + 2;
    s = s + r + t;
    t = t + 1;
    r = r - 3;
    cout << r << " " << s << " " << t << endl;
} //alias
...
t = 12; alias (t,t); cout << t << endl;
```

t = s	r	t'
12		
	12	?
		3
	14	
29		
		4
	11	

↓ tijd

Dit levert: 11 29 4 en 29.

En met een & voor r: 28 28 4 en 28.

Een functie mag zichzelf (in)direct aanroepen: **recurisie**.

```
int som (int n) { // berekent 1 + 2 + ... + n    versie 1
    int i, res = 0;
    for ( i = 1; i <= n; i++ ) res += i;
    return res;
}//som
```

```
int somrecursief (int n) { // idem, recursief    versie 2
    if ( n == 0 ) return 0;
    else return n + somrecursief (n-1);
}//somrecursief
```

```
int somslimGauss (int n) { // en nog eens ...    versie 3
    return ( n * ( n + 1 ) ) / 2;
}//somslim
```


De ggd kan ook recursief berekend worden:

```
int ggdrecursief (int x, int y) {  
    if ( y == 0 ) return x;  
    else return ggdrecursief (y,x % y);  
}//ggdrecursief
```



Je gebruikt eigenlijk:

$$\text{ggd}(x, y) = \begin{cases} x & \text{als } y = 0 \\ \text{ggd}(y, x \bmod y) & \text{als } y \neq 0 \end{cases}$$

Voor meer over recursie, zie later.

Voor de **tweede programmeeropgave** moet je een C++-programma schrijven dat een gegeven file codeert of decodeert met **run-length encoding**.

```
Eet_meer_zeeegels
ABC11123ddd\efG\\1
```

moet worden:

```
Eet_me2r_10ze3gels
ABC\13\2\3d3\\efG\\3\1
```

Hier is `␣` een spatie.

Klopt **Collatz** voor 6171?

www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php

1. coderen
test dat goed met voorbeeldfiles
gebruik zo weinig mogelijk put's en get's
2. decoderen (\approx coderen)
3. daarna, of juist eerder, de Collatz-controle (INT_MAX!)
4. en tot slot details, tellers, . . . , en het verslag

✓ $\leq \approx 250$ regels

Houd het kort! Gebruik geschikte functies; zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc4.php

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc5.php

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc6.php

Met `char kar = invoer.get ();` probeer je het eerste karakter uit `invoer` te halen; met `invoer.eof ()` verifieer je of je aan het einde van de `invoer` staat/stonde.

Met `uitvoer.put (kar);` schrijf je `kar` achteraan de `uitvoer`. Dat kan ook met `uitvoer << kar; .`

Let op het verschil tussen `kar = invoer.get ();` en `invoer >> kar; .` Die tweede slaat “whitespace” (waaronder spaties en regelovergangen) over!

En `kar = cin.get ();` wacht op het eerste karakter vanaf het toetsenbord (met ooit een “enter”).

Stel dat iemand karakters (char's, waaronder cijfers) op je afstuurt, en je daar een getal van moet maken. Hoe doe je dat?

Gebruik `int getal = 0;`, en herhaal:

```
if ( '0' <= kar && kar <= '9' )  
    getal = 10 * getal + ( kar - '0' );  
else  
    ...
```



```
qwerty7392abc    de---12fghijklmnopq
```



getal is 73 en kar is '9'

getal wordt 739

Deze void-functie manipuleert een file invoer:

```
void manipuleer (ifstream & invoer) {
    char prevkar = '\n', kar = invoer.get ( );
    while ( ! invoer.eof ( ) ) {
        if ( ... )                // GEEN get's,
            ...                    // GEEN while's
        if ( ... )                // GEEN strings,
            ... put ...           // ...
        prevkar = kar;
        kar = invoer.get ( );
    }//while
}//manipuleer
```

Deze void-functie telt **niet-lege regels** in een file invoer:

```
void telNietLegeRegels (ifstream & invoer, int & tel) {
    char prevkar = '\n', kar = invoer.get ( );
    while ( ! invoer.eof ( ) ) {
        if ( prevkar != '\n' && kar == '\n' )
            tel++;
        prevkar = kar;
        kar = invoer.get ( );
    }//while
};//telNietLegeRegels
```

Het aantal wordt opgeteld bij de oorspronkelijke “oude” waarde van `tel`. De actuele parameter bij die aanroep wordt dus gewijzigd.

Het is meestal verstandig invoer, rekenwerk en uitvoer door verschillende functies te laten verrichten.

Deze int-functie telt ook **niet-lege regels** in een file invoer:

```
int telNietLegeRegels (ifstream & invoer) {
    char prevkar = '\n', kar = invoer.get ( );
    int tel = 0; // lokaal tellertje
    while ( ! invoer.eof ( ) ) {
        if ( prevkar != '\n' && kar == '\n' )
            tel++;
        prevkar = kar;
        kar = invoer.get ( );
    }//while
    return tel; // <===== int functie!
}//telNietLegeRegels
```

Denk aan het openen en sluiten van de file(s).

Soms problemen met regelovergangen Windows/Linux:

\r\n respectievelijk **\n**. (... ios::in | ios::binary ...)

- werk aan de tweede programmeeropgave — de deadline is op maandag 16 oktober 2023, 18:00 uur
- vorm tweetallen!
- lees Savitch Hoofdstuk 3 en 4
- lees dictaat Hoofdstuk 3.6 en 3.7
- www.liacs.leidenuniv.nl/~kosterswa/pm/