Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Gestructureerd programmeren in C

GESPRG: Functies

DE HAAGSE HOGESCHOOL

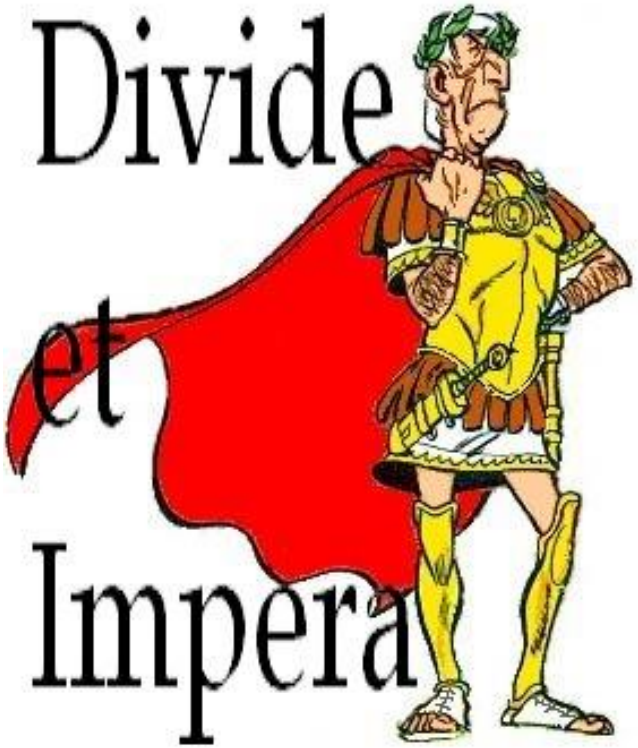
Programmeren

- Welke basisbewerkingen zijn er?
  - Lezen en schrijven (invoer en uitvoer)
  - Onthouden (variabelen)
  - Rekenen
  - Herhalen
  - Beslissen

  - Delegeren (verdeel en heers ➔ functies)
  - Structureren (array en struct)

DE HAAGSE HOGESCHOOL

Verdeel en heers



DE HAAGSE HOGESCHOOL

Problemen bij grotere prog's

- Programma wordt heel lang
  ➔ programma opsplitsen in delen maakt het duidelijker.

- Sommige stukken code komen meerdere keren in het programma voor (b.v. inlezen van een positief getal + controle).
  ➔ slecht voor de onderhoudbaarheid.

- Deel van een programma is niet eenvoudig te gebruiken in een ander programma.
  ➔ slecht voor de herbruikbaarheid.

DE HAAGSE HOGESCHOOL

Functionele decompositie

- Programma opdelen in stukken genaamd functies.
- Deze functies kunnen meerdere keren worden aangeroepen.
- Goed voor aanpasbaarheid en herbruikbaarheid.

DE HAAGSE HOGESCHOOL

Voorbeeld

```
int main(void) {
…
    printf("\n");
    printf("\n");
    printf("\n");
…
    printf("\n");
    printf("\n");
    printf("\n");
…
    return 0;
}
```

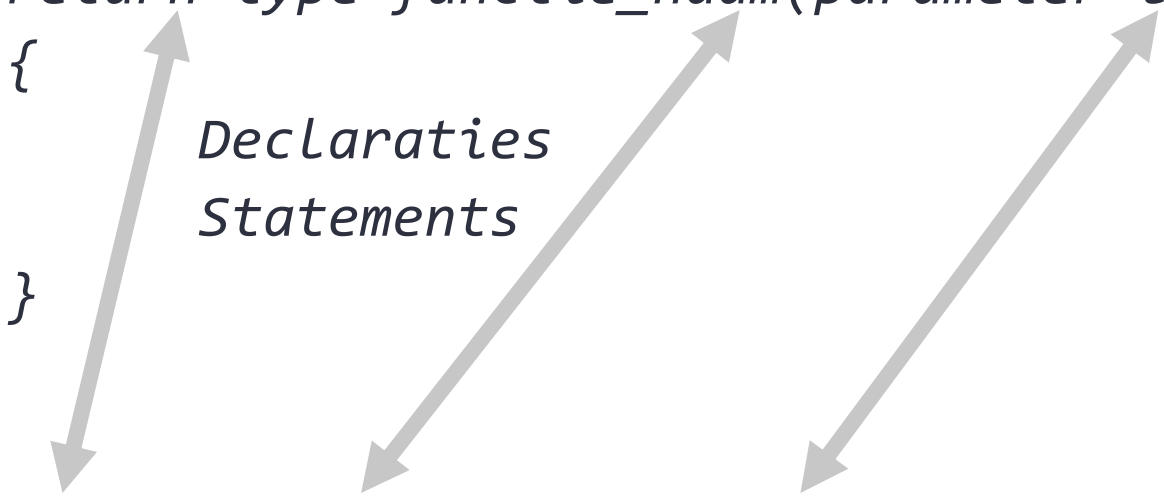
Sla 3 regels over

Sla 3 regels over

Schrijf een functie om 3 regels over te slaan

DE HAAGSE HOGESCHOOL

Opbouw van een functie




```
int functie_naam(int parameter) {
      int onzin = 0;
      onzin = onzin*parameter;
      return onzin;
}
```

DE HAAGSE HOGESCHOOL

Voorbeeld

```
void sla3RegelsOver(void) {
    printf("\n");
    printf("\n");
    printf("\n");
}

int main(void) {
…
    sla3RegelsOver();
…
    sla3RegelsOver();
…
    return 0;
}
```

Functie **definitie**

void betekent leeg
Deze functie geeft niets terug en heeft geen parameters (zie verderop)

Functie **aanroep**

Functie **aanroep**

DE HAAGSE HOGESCHOOL

Voorbeeld

```
int main(void) {
    void sla3RegelsOver(void);
…
    sla3RegelsOver();
…
    sla3RegelsOver();
…
    return 0;
}

void sla3RegelsOver(void) {
    printf("\n");
    printf("\n");
    printf("\n");
}
```

Functie **declaratie** (functie **prototype**)

Functie aanroep

Functie aanroep

Functie definitie

DE HAAGSE HOGESCHOOL

Voorbeeld

```
int main(void) {
…
    printf("\n");
    printf("\n");
    printf("\n");
…
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
…
    return 0;
}
```

Sla 3 regels over

Sla 4 regels over

Schrijf een functie om een **aantal** regels over te slaan

DE HAAGSE HOGESCHOOL

Voorbeeld

```
void slaRegelsOver(int aantal) {
    int teller;
    for (teller = 0; teller < aantal; teller = teller + 1) {
        printf("\n");
    }
}

int main(void) {
…
    slaRegelsOver(3);
…
    slaRegelsOver(4);
…
    return 0;
}
```

parameter

lokale variabele

Bij **aanroep** van de functie wordt de **waarde** van het argument **gekopieerd** naar de parameter (*call by value*)

argument

argument

DE HAAGSE HOGESCHOOL

Voorbeeld (zo moet het niet!)

```
int aantal;

void slaRegelsOver(void) {
    int teller;
    for (teller = 0; teller < aantal; teller = teller + 1) {
        printf("\n");
    }
}

int main(void) {
…
    aantal = 3; slaRegelsOver();
…
    aantal = 4; slaRegelsOver();
…
```

globale variabele

Waarom is het verkeerd om globale variabelen te gebruiken?

Voorbeeld

- Lees een geheel getal en controleer op een minimale en maximale waarde.

```
int leesGetal(int min, int max) {
    int getal;
    do {
        printf("Geef een getal [%d..%d]: ", min, max);
        scanf("%d", &getal);
    }
    while (getal < min || getal > max);
    return getal;
}

int main(void) {
    int toetscijfer = leesGetal(1, 10);
…
```

Na **afloop** van de functie wordt de **waarde** van het return statement **gekopieerd** naar de functieaanroep (*return by value*)

Huiswerk

- Bestudeer boek/dictaat:
  - Paragrafen 5.1, 5.2, 5.4, 5.5

- Maak opdrachten:
  - Schrijf een functie die twee parameters heeft (*a* en *b*) en die de waarde *a*b* retourneert.
  - 2 van https://www.w3resource.com/c-programming-exercises/function/index.php

DE HAAGSE HOGESCHOOL

main

- Elk C-programma heeft altijd een “speciale” functie: main.
- Het programma wordt gestart door het Operating System (Windows, OS-X, Linux) bij de eerste instructie in main.
- Bij terugkeer naar het Operating System moet een return-waarde worden meegegeven. In de regel is dat het getal 0, dat aangeeft dat alles goed is verlopen. Elk ander getal geeft een foutmelding of een waarschuwing aan.

```
int main(void) {  /* main krijgt geen parameters mee */
    return 0;
}
```

DE HAAGSE HOGESCHOOL

Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Gestructureerd programmeren in C

GESPRG Scope, lifetime, recursie

DE HAAGSE HOGESCHOOL

Scope versus Lifetime

- Elke variabele heeft een:
  - Scope (bereik) ➔ waar kun je de variabele gebruiken (tijdens het coderen van het programma)
  - Lifetime (levensduur) ➔ wanneer bestaat de variabele (tijdens het uitvoeren van het programma)

	Scope	Lifetime
Globaal	Hele programma (kan verborgen zijn)	Bestaat alleen als het programma wordt uitgevoerd
Lokaal	Functie	Bestaat alleen als de functie wordt uitgevoerd

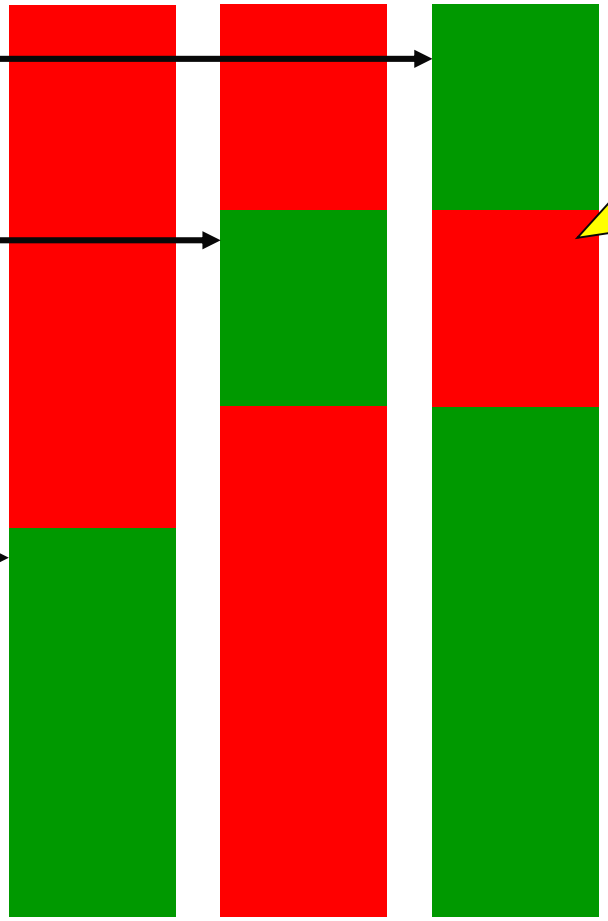
Een parameter is een lokale variabele die bij de functieaanroep wordt geïnitialiseerd.

Scope

```
int a;

void func(void) {
    int a, b;
    …
}

int main(void) {
    int b;
    …
    func();
    …
    return 0;
}
```



De globale a is verborgen (door de lokale a).

In scope

Out of scope

DE HAAGSE HOGESCHOOL

Lifetime

```
int a;

void func(void) {
    int a, b;
    …
}

int main(void) {
    int b;
    …
    func();
    …
    return 0;
}
```

Reserveer geheugenruimte voor globale a

Reserveer geheugenruimte voor lokale a en b in func

Geef geheugenruimte vrij van lokale a en b in func

Reserveer geheugenruimte voor lokale b in main

Geef geheugenruimte vrij van lokale b in main

Geef geheugenruimte vrij voor globale a

DE HAAGSE HOGESCHOOL

Recursieve functies

- Roept zichzelf aan
  - In de functie definities staat een functie-aanroep naar de functie zelf.
- Heeft altijd een *stopconditie*
  - Waarom?
- Een recursieve functie roept zichzelf een *x* aantal keren aan.
  - Wat wordt er dan herhaald?
- In een recursieve functie zitten bijna nooit, herhalingslussen (`while`, `do-while`, `for`).

Faculteit

- Bereken n! = 1 * 2 * 3 * … * n

Iteratieve implementatie

```
int faculteit(int n) {
    int i, res = 1;
    for (i = 1; i <= n; i = i + 1) {
        res = res * i;
    }
    return res;
}
```

$$n! = \begin{cases} 1 \text{ als } n = 0 \\ n * (n - 1)! \text{ als } n > 0 \end{cases}$$

Recursieve implementatie

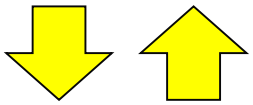
```
int faculteit(int n) {
    if (n > 0)
        return n * faculteit(n - 1);
    return 1;
}
```

Wat is beter ?

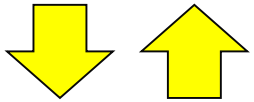
DE HAAGSE HOGESCHOOL

Recursieve aanroep (voorbeeld)

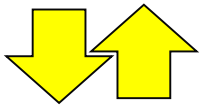
```
int j = 24;
```



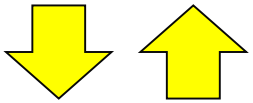
```
return 24;
```



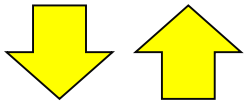
```
return 6;
```



```
return 2;
```



```
return 1;
```



```
return 1;
```

```
int faculteit(int n) {
    if (n > 0)
        return n * faculteit(n - 1);
    return 1;
}
```

DE HAAGSE HOGESCHOOL

n boven k

- Bereken n boven k
  - $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Wat is beter ?

Iteratieve implementatie

```
int n_boven_k(int n, int k) {
    return faculteit(n) / (faculteit(k) * faculteit(n - k));
}
```

- n boven k
  - $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

Recursieve implementatie

```
int n_boven_k(int n, int k) {
    if (k == 0 || k == n)
        return 1;
    return n_boven_k(n - 1, k - 1) + n_boven_k(n - 1, k);
}
```

DE HAAGSE HOGESCHOOL

Binomiaalcoëfficiënt

- Op hoeveel manieren kun je uit **n** objecten er zonder terugleggen **k** kiezen?

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad voor\ 0 \leq k \leq n$$

Recursieve versie:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

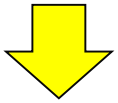
$$\binom{n}{k} = 0 \quad voor\ k < 0\ of\ k > n$$

$$\binom{n}{k} = 1 \quad voor\ k = 0\ en\ k = n$$

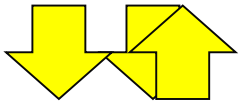
Recursieve aanroep (voorbeeld)

```
int n_boven_k(int n, int k) {
    if (k == 0 || k == n) return 1;
    return n_boven_k(n - 1, k - 1) + n_boven_k(n - 1, k);
}
```

```
int j = n_boven_k(4, 2);
```

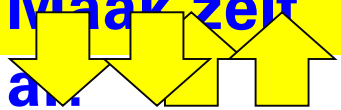


```
return 3 + n_boven_k(3, 2);
```

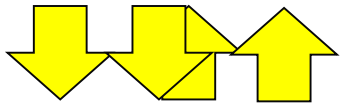


```
return 1 + n_boven_k(2, 1); oven_k(2, 1);
```

Maak zelf af.



```
return 1 + return 1 + 1; 1); oven_k(1, 1);
```



```
ret return 1;
```

DE HAAGSE HOGESCHOOL

Domein

- Faculteit
  - Domein = [0..12]

- n boven k
  - Domein versie met faculteit = [0..12]
  - Domein recursieve versie = [0..33]

In dit geval is de recursieve versie dus “beter”.

Huiswerk

- Bestudeer boek/dictaat:
  - 5.12 (**geen** static storage class), 5.13, 5.14
- Maak opdrachten:
  - 4 van paragraaf 6.5 uit Mark Burgess, *The C programming Tutorial*. (is te vinden op Blackboard) (Gebruik hierbij de debugger om de scope van de variabelen te zien)
- Schrijf een **recursieve** functie `int fib(int n)` die het n^{de} getal uit de fibonacci rij berekent. Zie: http://en.wikipedia.org/wiki/Fibonacci_number

Huiswerk

- Schrijf een **recursieve** functie `int fib(int n)` die het n^{de} getal uit de fibonacci rij berekent. Zie: http://en.wikipedia.org/wiki/Fibonacci_number

- Bestudeer C boek:
  - paragrafen 5.6, 5.7 en 5.15.

DE HAAGSE HOGESCHOOL