CPROUC/2021-2022

Jesse op den Brouw

CPROUC

Functies

DE HAAGSE HOGESCHOOL

Functies

- Als een programma langer wordt, dan wordt het al snel onoverzichtelijk.
- Sommige stukken code komen misschien meerdere keren in het programma voor, bijvoorbeeld het inlezen van een positief getal.
- Dit kopiëren van code zorgt ervoor dat het programma slecht onderhoudbaar wordt.
- Als er namelijk een wijziging in deze code moet worden doorgevoerd dan moeten ook alle kopieën worden aangepast.
- Ook zijn delen uit zo'n lang programma niet eenvoudig te gebruiken in een ander programma.

Functies

- We kunnen deze problemen oplossen door het gebruik van *functies*.
- Een logisch bij elkaar behorend stuk code wordt dan ergens apart (in een functie) geplaatst.
- Deze functie kunnen we vervolgens naar believen *aanroepen* om de code van de functie uit te voeren.
- Een Engelse term van de code van een functie wordt de *body* genoemd.
- We kunnen gegevens aan de functie meegeven om zo de bruikbaarheid te vergroten.
- Een functie kan een gegeven teruggeven. Ook dit vergroot de bruikbaarheid.

Algemene gedaante

- De algemene gedaante van een functie is:

```
returntype functienaam(parameter, parameter, ...)
{
  body van de functie
}
```

- Parameters mogen weggelaten worden → gebruik `void`
- Return-type mag `void` zijn: er wordt niks teruggegeven

Functies zonder gegevens

- We kunnen een functie definiëren zonder gegevensoverdracht

```
void sla3regelsover(void)
{
   printf("\n\n\n");
}
```

- Aanroepen met:

```
sla3regelsover();  /* sla in totaal 6 regels over */
sla3regelsover();
```

Functies zonder gegevens

- We kunnen een functie aanroepen vanuit main:

```
int main(void)
{
   printf("Hallo");
   sla3regelsover();
   printf("Wereld!");
   return 0;
}
```

- We kunnen een functie meerdere malen aanroepen.

Functies met parameters

- We kunnen aan een functie gegevens meegeven

```
void slaregelsover(int aantal)  /* aantal wordt een parameter genoemd */
{
   int teller;
   for (teller = 0; teller < aantal; teller = teller + 1) {
      printf("\n");
   }
}
```

Functies met parameters

- We kunnen een functie aanroepen vanuit main:

```
int main(void)
{
   printf("Hallo");
   slaregelsover(4);    /* 4 wordt een argument genoemd */
   printf("Wereld!");
   return 0;
}
```

Functies met returnwaarde

- Een functie kan één gegeven teruggeven
- Standaard datatypes: `char`, `int`, `float`, `double` (mag met long etc.)
  - Later nog: structures en pointers

- De executie van een functie met een returnwaarde *moet* eindigen met een `return`-statement en een waarde
  - Compiler geeft een waarschuwing/foutmelding als dat niet gebeurt
- Dat hoeft niet aan het (tekstuele) eind van de functie te zijn

Functies met parameters en returnwaarde

- Voorbeeld van een functie met twee parameters en een returnwaarde

```
int mul(int a, int b) /* a en b zijn parameters */
{
   int c = a * b;
   return c;
}
```

- Aanroepen met:

```
int z = mul(x, y); /* x en y zijn argumenten */
```

Functies met parameters en returnwaarde

- Voorbeeld van een functie met twee parameters en een returnwaarde

```
int main(void)
{
   int x, y, z;
   scanf("%d%d", &x, &y);
   z = mul(x, y);   /* x en y zijn argumenten */
   printf("%d", z);
   return 0;
}
```

Maximum bepalen

- Een return-statement hoeft niet aan het (tekstuele) einde van de functie te staan:

```
int max2(int a, int b) {
  if (a > b) {
     return a;
  } else {
     return b;
  }
}
```

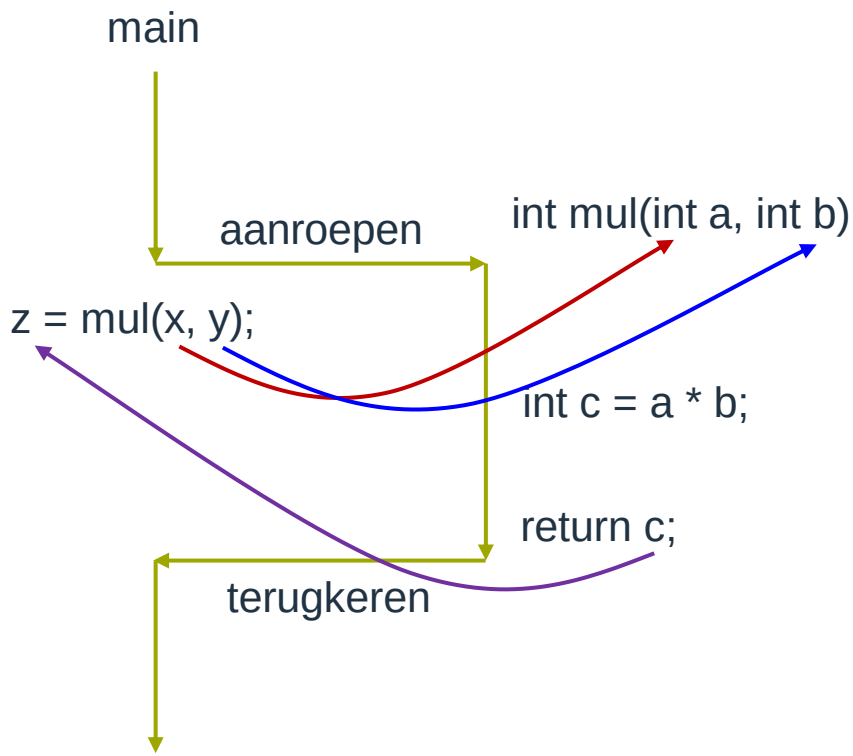
Maximum bepalen

- Een return-statement hoeft niet aan het (tekstuele) einde van de functie te staan:

```
int max2(int a, int b) {
  if (a > b) {
     return a;
  }
  return b;
}
```

Call by Value

- De argumenten die bij een functie-aanroep worden meegegeven, worden gekopieerd naar de bijbehorende parameters.
- Dit wordt *Call by Value* genoemd.
- De functie krijgt als parameters dus kopieën mee en “weet” niet de plek in het geheugen van de argumenten.
- Teruggave wordt via een kopie van c gedaan.



Verwisselen

- In veel sorteerprogramma's komt het verwisselen van gegevens voor

```
void wissel(int a, int b) {

  int hulpje = a;
  a = b;
  b = hulpje;
}
```

- Aanroepen met

```
wissel(x, y);
```

Verwisselen

- In veel sorteerprogramma's komt het verwisselen van gegevens voor

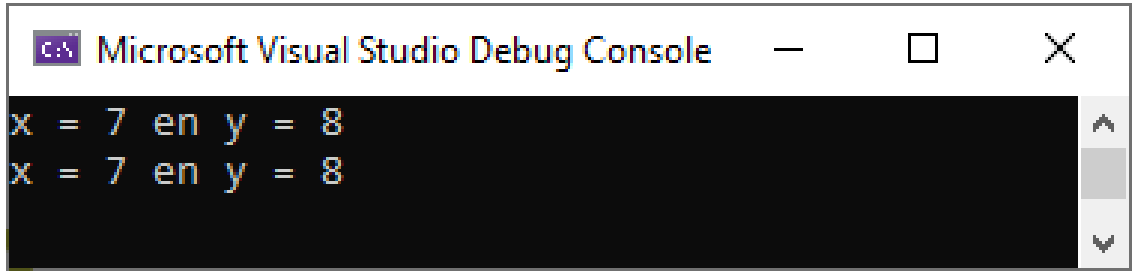
```
int main(void) {

    int x = 7, y = 8;

    printf("x = %d en y = %d\n", x, y);
    wissel(x, y);
    printf("x = %d en y = %d\n", x, y);
    return 0;
}
```

Verwisselen

- Uitvoer van het programma:



- x en y zijn niet verwisseld!
- Dat kan ook niet want de functie krijgt *kopieën* mee van x en y!

Functiedeclaratie

- In veel gevallen wordt een functie *na* `main` gedefinieerd.
- Tijdens compilatie van `main` is de functie nog niet “gezien” door de compiler.
- Een aanroep van de functie in main zal leiden tot een waarschuwing of foutmelding.
- Door gebruik te maken van een *functiedeclaratie* (ook wel *functieprototype* genoemd), kunnen we de compiler informeren over de functie.

Functiedeclaratie

- Een voorbeeld:

```
void slaregelsover(int aantal);  /* declaratie */

int main(void) {
   slaregelsover(4);
}

void slaregelsover(int aantal) {  /* definitie */
   /* body */
}
```

Wiskundige functies

- De wiskundige bibliotheek (math library) kent een groot aantal wiskundige functies.
- Om ze te gebruiken moet het header-bestand math.h geladen worden.

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

- De functies krijgen één of twee `double`-argumenten mee en geven een `double` terug.

Wiskundige functies

- `sin(x)` - sinus van x, x in radialen
- `cos(x)` - cosinus van x, x in radialen
- `tan(x)` - tangens van x, x in radialen
- `asin(x)` - arcsin, $\sin^{-1}$, x tussen -1 en 1 (inclusief)
- `acos(x)` - arccos, $\cos^{-1}$, x tussen -1 en 1 (inclusief)
- `atan(x)` - arctan, $\tan^{-1}$, voor alle x

- `sqrt(x)` - vierkantswortel van x, $x \geq 0$
- `exp(x)` - e^x, $e \approx 2{,}718281828$, voor alle x
- `log(x)` - ln x, natuurlijke logaritme, $x > 0$
- `pow(x, y)` - x^y, $x > 0$ (er zijn meer begrenzingen)

let’s change