

HOOFDSTUK 3

STRUCTS & DYNAMISCH GEHEUGENBEHEER

Inhoud

- **Structs**
 - Wat is een struct?
 - Definitie van een struct
 - Declaratie struct-variabelen
 - Bewerkingen op structs
 - Pointer naar struct
 - Structs en functies
- Dynamisch geheugenbeheer

Wat is een struct?

- Worden gebruikt om **gegevens** (die logisch samenhangen) te **groeperen**
- De componenten mogen van **verschillende types** zijn
⇒ **HETEROGEEN** type.
- De programmeur mag zelf een naam geven aan het **nieuwe** (samengestelde) **datatype**

Definitie van een struct

```
struct naam {  
    type_1 naam_1;  
    type_2 naam_2;  
    ...  
};
```

Voorbeeld:

```
struct punt {  
    double x,y;  
};
```

↑ niet vergeten!!

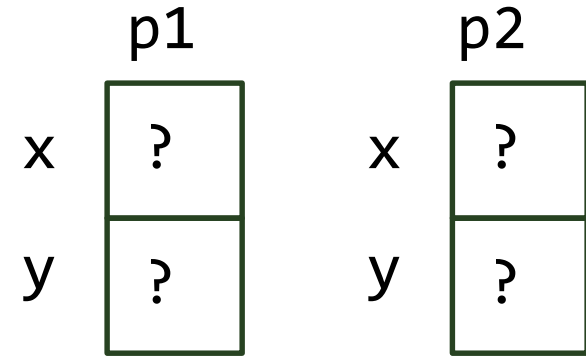
- legt blauwdruk vast van toekomstige variabelen van het type
 struct naam
- *type_1, type_2, ...* : willekeurig (inclusief pointers, arrays, structs, ...)
- *naam_1, naam_2, ...* : zijn **velden** van de struct
- Waar? Vóór gebruik (meestal na compiler directieven)

Declaratie struct-variabelen

```
struct naam var1, var2, ...;
```

Voorbeeld:

```
struct punt p1, p2;
```



Alternatief: gecombineerd met struct-definitie:

```
struct naam {  
    type_1 naam_1;  
    type_2 naam_2;  
    ...  
} var1, var2, ...;
```

Voorbeeld:

```
struct punt {  
    double x,y;  
} p1, p2;
```

globale variabelen!!

Gebruik typedef

```
typedef struct naam naam;
```

```
naam var1, var2, ...;
```

```
struct punt {  
    double x,y;  
};  
typedef struct punt punt;  
  
punt p1, p2;
```

Gebruikelijke notatie in C:

```
typedef struct {  
    type_1 naam_1;  
    type_2 naam_2;  
    ...  
} naam;
```

```
typedef struct {  
    double x,y;  
} punt;  
  
punt p1, p2;
```

Bewerkingen op structs

1. Toegang tot velden

- Gebruik **.** **operator** om velden in te vullen of de inhoud ervan op te vragen
- *naam_struct_variabele.naam_veld* mag overal gebruikt worden waar een uitdrukking van hetzelfde type toegelaten is!

- Voorbeeld:

```
p1.x = 3.5;  
printf("y-coord p1: ");  
scanf("%lf",&p1.y);  
printf("p1 = {%.2lf,%.2lf}",p1.x, p1.y);
```

inlezen/uitschrijven struct:
veld per veld
(of via zelfgeschreven
lees/schrijf-functie)

2. Toekenningsoperator =

Voorbeeld:

`p2 = p1;`

Kopieert variabele p1 naar p2

3. Vergelijken van structs

Voorbeeld:

~~`if (p1 == p2) { ... }`~~

`if (p1.x == p2.x && p1.y == p2.y) { ... }`

Structs veld per veld met
elkaar vergelijken!!

4. Initialisatie bij declaratie

- cfr. syntax voor het initialiseren van een array
- schrijf initialisatie-uitdrukkingen voor elk veld tussen { }
- bij expliciete initialisatie: alle niet gespecificeerde velden worden op 0 geplaatst
- Voorbeelden:

```
punt p2 = {3.5, -2.9};
```

```
punt p3 = {7.2};
```

```
punt t1[5] = {{1.2, -1.2}, {5.1, 4.6}};
```

```
punt t2[3] = {0}; /* alles 0 */
```

```
p2 = {6.2, 3.8};
```

Hoe kan je de volledige array t1 naar het scherm printen?

vb_struct.c

Oefening

Gegeven:

```
typedef struct {  
    double x, y;  
} punt;
```

```
typedef struct {  
    punt mp;  
    double straal;  
} cirkel;
```

Gevraagd:

Lees eerst 10 cirkels in en daarna een punt.

Schrijf daarna alle cirkels uit waarvan het punt binnen de cirkel ligt (op de rand is buiten).

Pointer naar struct

- Voorbeeld:

```
punt p = {2.2, 3.1};
```

```
punt *pp = &p;
```

- Toegang tot gegevens:

$(*struct_pointer).veld \equiv struct_pointer->veld$

Voorbeeld:

```
printf("p = {%.21f,%.21f}", (*pp).x, (*pp).y);
```

```
printf("p = {%.21f,%.21f}", pp->x, pp->y);
```

Let op: $(*pp).x \neq *pp.x$

Structs en functies

1. Struct als parameter

- Wordt behandeld als elk ander type
 - ⇒ **pass-by-value**: er wordt een kopie genomen
- Om kopie te vermijden: pointer naar struct als parameter
- Voorbeeld:

```
double afstand(punt p1, punt p2) {  
    return sqrt((p1.x-p2.x)*... + ...;    }
```

=> beter:

```
double afstand(const punt *p1, const punt *p2) {  
    return sqrt((p1->x-p2->x)*... + ...; }
```

2. (pointer naar) struct als resultaat

- Wordt behandeld als elk ander type
⇒ resultaat wordt gekopieerd naar oproepende context
- Resultaat van functie kan ook pointer naar struct zijn
(niet array van struct)

- Oefening:

Schrijf een functie `zoek_punt(t, n, p)` die in de gegeven array `t` (die `n` cirkels bevat) op zoek gaat naar de eerste cirkel waarvan het middelpunt gelijk is aan het punt waarnaar `p` wijst. Geef een pointer naar deze cirkel terug of de nullpointer indien geen dergelijke cirkel gevonden wordt.

Inhoud

- Structs
 - Wat is een struct?
 - Definitie van een struct
 - Declaratie struct-variabelen
 - Bewerkingen op structs
 - Pointer naar struct
 - Structs en functies
- **Dynamisch geheugenbeheer**

Dynamisch geheugenbeheer

1. Probleemsituatie

Zoek alternatieven voor onderstaande functie die een punt inleest.

```
punt lees_punt() {  
    punt p;  
    printf("x-coordinaat: "); scanf("%lf",&p.x);  
    printf("y-coordinaat: "); scanf("%lf",&p.y);  
    return p;  
}
```

Oproep: `punt p = lees_punt();`

Alternatief 1: pointer naar struct als parameter

```
void lees_punt(punt *p) {  
    printf("x-coordinaat: ");  
    scanf("%lf",&p->x);  
    printf("y-coordinaat: ");  
    scanf("%lf",&p->y);  
}
```

Oproep:

```
punt *p; lees_punt(p);  
punt p; lees_punt(&p);
```


Alternatief 2: pointer naar struct als resultaat

```
punt* lees_punt() {  
    punt p;  
    printf("x-coordinaat: ");  
    scanf("%lf",&p.x);  
    printf("y-coordinaat: ");  
    scanf("%lf",&p.y);  
    return &p;  
}
```

WARNING: function returns
address of local variable

Oproep:

~~punt *p = lees_punt();~~

Alternatief 2: correcte oplossing:

```
punt* lees_punt() {  
    punt *p = (punt *) malloc(sizeof(punt));  
    printf("x-coordinaat: ");  
    scanf("%lf",&p->x);  
    printf("y-coordinaat: ");  
    scanf("%lf",&p->y);  
    return p;  
}
```

reserveer op de heap
geheugen voor 1 punt

casten mag, maar moet niet

geef na gebruik
geheugen vrij

vb_struct_res.c

Oproep:

```
punt *p = lees_punt(); ... ; free(p);
```

2. Methodes dynamisch geheugenbeheer

- Methodes uit <stdlib.h>
- Toewijzen geheugen:

```
void* malloc(size_t totaal_aantal_bytes)
void* calloc(size_t aantal, size_t aantal_bytes)
```

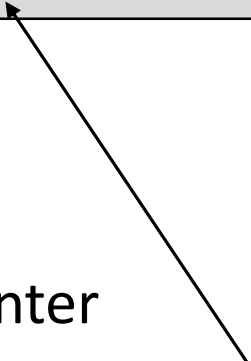
- resultaat: type **void*** (evt. casten) of **NULL** (indien mislukt)
- malloc versus calloc?
 - $\text{malloc}(n*N) \leftrightarrow \text{calloc}(n,N)$
 - calloc **initialiseert** geheugen (op 0)
- maak code systeemafhankelijk:
 voorbeeld: `malloc(n*sizeof(double))`

- **Vrijgeven geheugen:**

```
void free(void *toegewezen_pointer)
```

- fout indien
 - argument al vrijgegeven
 - argument geen toegewezen pointer

pointer bekomen met
malloc, calloc of realloc



- **Hergebruik geheugen:**

```
void* realloc(void *toegewezen_pointer,  
              size_t totaal_aantal_bytes)
```



- poogt geheugenblok te hergebruiken voor totaal_aantal_bytes
- inhoud begin van blok blijft behouden

- Voorbeeld:

```
char *p = (char *) malloc(11 * sizeof(char));
if (p == NULL) printf("malloc error\n");
else {
    strcpy(p, "1234567890");
    char *nieuw = realloc(p, 20);
    if (nieuw) {
        strcat(nieuw, " langer");
        puts(nieuw);
        free(nieuw);
    }
    else free(p);
}
```

3. Veelgemaakte fouten en problemen

- **[Not checking for allocation failures.]**
- **Memory leak:** block of memory that was allocated, but will never be freed. If all pointers to that block have gone out of scope or were assigned to point elsewhere, the application will never be able to free that block.
- **Logical errors:** memory usage after a call to free (**dangling pointer**) or before a call to malloc (**wild pointer**), calling free twice ("double free"), ...
- **Memory fragmentation:** na veelvuldig alloceren en dealloceren kunnen bepaalde kleine stukjes geheugen niet meer gebruikt worden omdat ze te klein zijn.

4. Voorbeeld

- Opgave:

Lees een positief geheel getal n in, gevolgd door n gehele getallen en schrijf daarna deze n gehele getallen opnieuw uit.

- Oplossing: **vb_geheugenbeheer.c**

5. Oefening

Schrijf een functie `herhaal(s, n)` die een nieuwe string van de gepaste grootte aanmaakt, die n keer de gegeven string s bevat, en deze string als resultaat teruggeeft.

Oefening

Gegeven:

```
typedef struct {  
    int leeftijd;  
    char *naam;  
} persoon;
```

Gevraagd:

Schrijf een procedure `lees_persoon(p)` die van een persoon de leeftijd en de naam (max. 80 letters, eventueel meerdere woorden) inleest en opslaat in `p`.