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

# Gestructureerd programmeren in C

**GESPRG: Pointers**

DE HAAGSE HOGESCHOOL

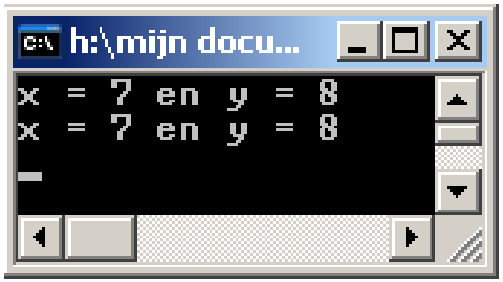
# Verwisselen

Eerste poging

- Schrijf een functie waarmee twee int variabelen verwisseld kunnen worden.

```
void wissel(int a, int b) {
    int hulpje = a;
    a = b;
    b = hulpje;
}

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);
    getchar(); return 0;
}
```

Niet goed!




DE HAAGSE

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

# Pointers

- C kent naast de “gewone” variabelen ook pointer variabelen.

DE HAAGSE HOGESCHOOL

# Geheugen

- Het geheugen van een computer bestaat uit bytes (groepje van 8 bits). Elk byte heeft zijn eigen adres (een nummer).
- Een variabele is opgeslagen in het geheugen vanaf een bepaald adres.
- Het type van de variabele bepaalt het aantal bytes dat nodig is om de waarde van de variabele op te slaan.

DE H

HOGESCHOOL

# Adres opvragen

- Je kunt het adres van een variabele opvragen met behulp van de unaire operator `&`.

Adres van getal, zodat `scanf` de ingelezen waarde op dat adres kan opslaan.

```
int getal;
scanf("%d", &getal);
```

- Je kunt het aantal bytes dat een variabele in beslag neemt opvragen met de unaire operator `sizeof`.

(merk op dat `sizeof` een keyword is)

DE HAAGSE HOGESCHOOL

# Adres opvragen

```c
#include <stdio.h>

int global = 4;

int main(void) {
    double local = 7.2;

    printf("global staat op adres %p\n", &global);
    printf("global is %d bytes groot\n", sizeof global);
    printf("local  staat op adres %p\n", &local);
    printf("local  is %d bytes groot\n", sizeof local);

    getchar();
    return 0;
}
```

Gebruik conversiekarakter %p voor het printen van een pointer-adres


Adres wordt in het **hexadecimale** talstelsel geprint (32-bits compiler)

DE HAAGSE HOGESCHOOL

# Pointer declaratie

- Een pointer wordt als volgt gedeclareerd

```
char *p;    /* p is een pointer naar een char */
int *p;     /* p is een pointer naar een int */
float *p;   /* p is een pointer naar een float */
double *p; /* p is een pointer naar een double */

void *p;    /* p is een generieke pointer */
```

# Pointer dereference

- Je kan een pointer als volgt gebruiken:

```
int *p;             /* p is een pointer naar een int */
…
p = &i;             /* p wijst naar variabele i */
…
*p = *p + 1;        /* verhoog de variabele waar p heen wijst */
(*p) = (*p) + 1;    /* verhoog de variabele waar p heen wijst */
```

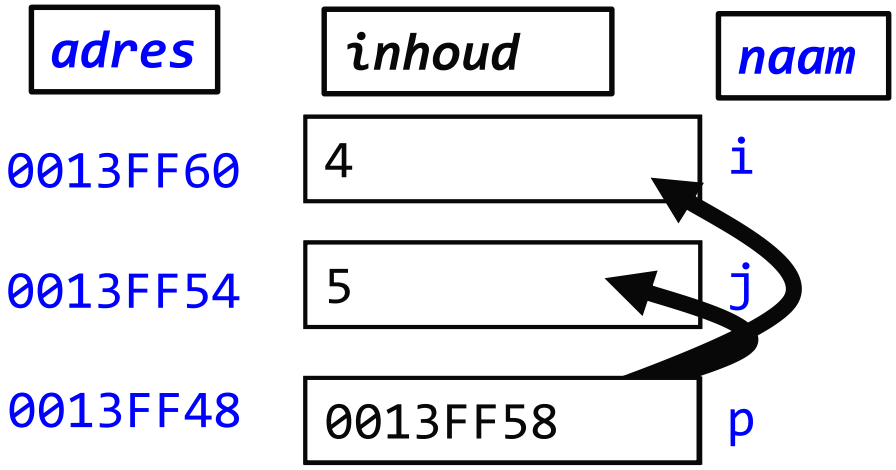
- Dit wordt *pointer dereference* genoemd. De asterisk (“sterretje”) wordt de dereference operator genoemd.

# Pointer

- Je kunt het adres van een variabele opslaan in een pointer variabele.

*p ➔ waar p naar wijst

```
int i = 3, j = 17;
int *p;
 p = &i;
*p = *p + 1;
 p = &j;
*p = i + 1;
```




p is een int pointer die wordt geïnitialiseerd met het adres van i

# Let op

- De volgende code kan verwarrend zijn:

```
int *p = &i;
```

- Wordt hier het adres van i toegekend aan de inhoud waar de pointer p naar wijst?
- NEE
- Deze code bestaat uit twee delen:

```
int *p; /* declaratie */
p = &i; /* initialisatie */
```

DE HAAGSE HOGESCHOOL

# Pointer aanpassen

- Je kan met pointers rekenen.

```
int *p;         /* p is een pointer naar een int */
…
p = &i;         /* p wijst naar variabele i */
…
p = p + 1;      /* pointer p wijst naar volgende int */
                /* pointer p wordt dus met 4 verhoogd */
```

- Let op: pointers aanpassen kan gevaarlijk zijn!!

DE HAAGSE HOGESCHOOL

# Oefening met pointers

- Doe deze oefening niet op een laptop/pc maar **met pen en papier**!

*Code:*

```
int i = 30, j = 33;
1. int *p2 = &j;
2. int *p = &i;
3. *p = *p + 13;
4. p = &j;
5. *p = i + 1;
6. *p = 15;
7. p = p2;
8. *p = i;
9. *p = 2*(*p2);
```

```
adres i = 0025F984
adres j = 0025F978
1. p2   = 0025F978
2. p    = 0025F984
3. *p   = 43
4. p    = 0025F978
5. *p   = 44
6. *p   = 15
7. p    = 0025F978
8. *p   = 43
9. *p   = 86
```

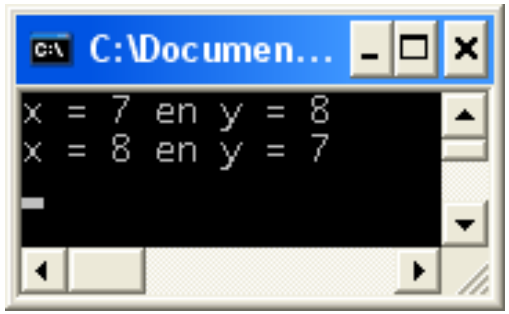
# Verwisselen

Tweede poging

- Schrijf een functie waarmee twee int variabelen verwisseld kunnen worden.

```
void wissel(int *p, int *q) {
    int hulpje = *p;
    *p = *q;
    *q = hulpje;
}

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);
    getchar(); return 0;
}
```

*call by reference*




Wel goed!

DE HAAGSE HOGESCHOOL

# Size does matter

- Bij een 32-bits compiler zijn de pointers 4 bytes groot zodat maximaal 4 GB kan worden geadresseerd.
- Bij een 64-bits compiler zijn de pointers 8 bytes groot zodat maximaal 16 EB (exa-bytes) kan worden geaddresseerd.
- Code::Blocks is standaard uitgerust met een 32-bits compiler. Het is mogelijk een 64-bits compiler te installeren.
- Moderne Linux-varianten voor Intel/AMD zijn standaard uitgerust met een 64-bits compiler.

DE HAAGSE HOGESCHOOL

# What's in the name

- Een aantal functies (`fopen`, `malloc`) geeft een zogenoemde *NULL-pointer* terug als een operatie niet kan worden uitgevoerd.

- In een NULL-pointer zijn alle bits 0*). Daar kan op getest worden.

```
int *p;

p = some_function(void);
if (p == NULL) {      /* NULL is defined in C */
    panic(void);
}
```

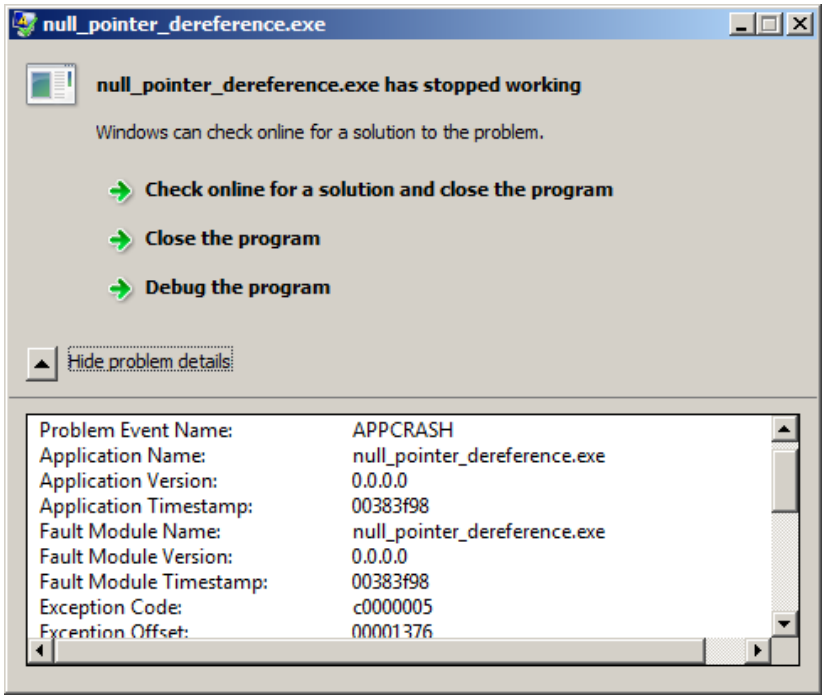
*) Dat hoeft niet zo te zijn op hardware-niveau, maar testen op NULL werkt gewoon. Zie https://stackoverflow.com/questions/4955198/what-does-dereferencing-a-pointer-mean

DE HAAGSE HOGESCHOOL

# NULL-pointer dereference

- Een NULL-pointer mag niet gebruikt worden om een “variabele” te benaderen.

```
char *p = NULL;

*p = *p + 1; /* Oops, NULL-pointer dereference */
```




DE HAAGSE HOGESCHOOL

# Wijzen naar een specifiek adres

- Je kan in C naar een specifiek adres laten wijzen. Dit komt veel voor bij het ontwikkelen van programma's op eenvoudige microprocessoren waar geen Operating System op draait.

```
unsigned char *p = (unsigned char *) 0x1234; /* adres 0x1234 */

*p = *p + 1;
```

- Dit is heel gevaarlijk! Goed opletten wat je doet!

DE HAAGSE HOGESCHOOL

# Huiswerk

- Bestudeer boek/dictaat:
  - Paragrafen 7.2, 7.3, 7.4 (opmerking over arrays **in eerste instantie** overslaan)

- Maak opdrachten:
  - 3,4,5 van https://www.w3resource.com/c-programming-exercises/pointer/index.php

DE HAAGSE HOGESCHOOL

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

# Gestructureerd programmeren in C

**GESPRG: Array's**

DE HAAGSE HOGESCHOOL

# Array

- Meerdere variabelen van hetzelfde type kun je samennemen in één array variabele.
- Stel in een systeem wordt ieder uur de temperatuur gemeten en opgeslagen.
  - Aparte variabele voor elk uur:

```
double temp0, temp1, temp2, temp3, temp4, temp5, temp6,
temp7, temp8, temp9, temp10, temp11, temp12, temp13, temp14,
temp15, temp16, temp17, temp18, temp19, temp20, temp21,
temp22, temp23;
```

  - Array variabele:

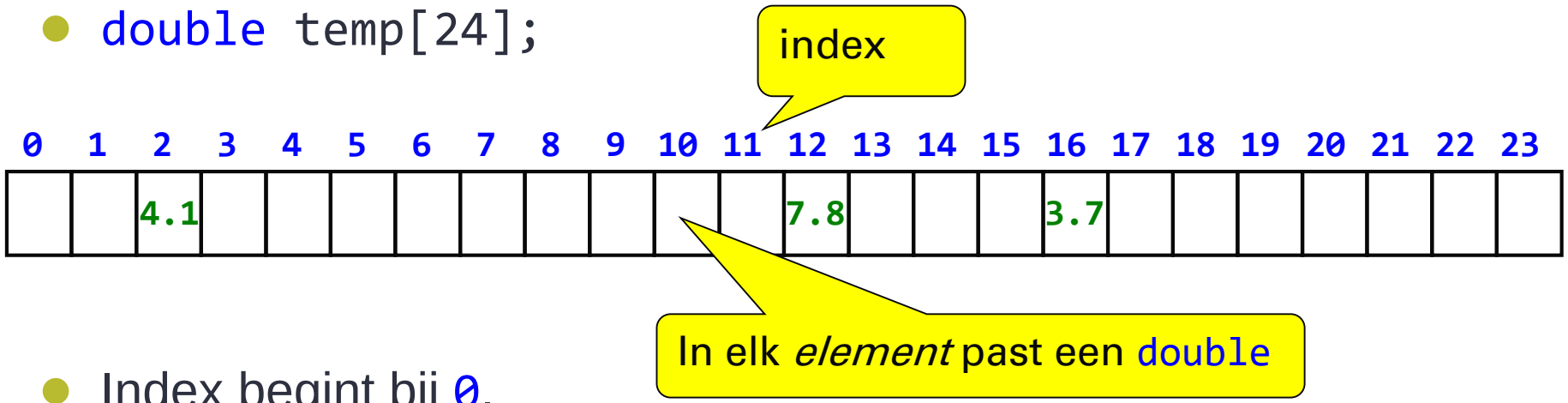
```
double temp[24];
```

Voordeel?

DE HAAGSE HOGESCHOOL

# Array

- `double temp[24];`




- Index begint bij `0`.
- Element kan geselecteerd worden met index operator `[]`
  - `temp[2] = 4.1;`
  - `temp[16] = 3.7;`
  - `temp[12] = temp[2] + temp[16];`

DE HAAGSE HOGESCHOOL

# Relatie tussen pointers en array's

- Er is een sterke relatie tussen pointers en array's.

- Het adres van een element van een array kan opgevraagd worden met de &-operator:

```
double *p = &temp[4]; /* p wijst naar vijfde element */
```

- De *naam* van een array is een *synoniem* voor het adres van het eerste element:

```
double *p = temp;      /* p wijst naar eerste element */
double *p = &temp[0];  /* p wijst naar eerste element */
```

DE HAAGSE HOGESCHOOL

# Voorbeeld array

```c
#include <stdio.h>

int main(void) {
    double temp[24], totaal;
    int i;
    for (i = 0; i < 24; i = i + 1) {
        printf("Geef temperatuur om %d uur: ", i);
        scanf("%lf", &temp[i]);
    }
    totaal = 0.0;
    for (i = 0; i < 24; i = i + 1) {
        totaal = totaal + temp[i];
    }
    printf("Gem. temp. is %f graden.", totaal / 24);
    fflush(stdin);
    getchar();
    return 0;
}
```

Adres van element i

DE HAAGSE HOGESCHOOL

# Voorbeeld array

*Uitvoer:*

```
Geef temperatuur om 0 uur: 1
Geef temperatuur om 1 uur: 2
Geef temperatuur om 2 uur: 3
Geef temperatuur om 3 uur: 4
Geef temperatuur om 4 uur: 5
Geef temperatuur om 5 uur: 1
Geef temperatuur om 6 uur: 2
Geef temperatuur om 7 uur: 3
Geef temperatuur om 8 uur: 4
Geef temperatuur om 9 uur: 5
Geef temperatuur om 10 uur: 1
Geef temperatuur om 11 uur: 2
Geef temperatuur om 12 uur: 3
Geef temperatuur om 13 uur: 4
Geef temperatuur om 14 uur: 5
Geef temperatuur om 15 uur: 1
Geef temperatuur om 16 uur: 2
Geef temperatuur om 17 uur: 3
Geef temperatuur om 18 uur: 4
Geef temperatuur om 19 uur: 5
Geef temperatuur om 20 uur: 1
Geef temperatuur om 21 uur: 2
Geef temperatuur om 22 uur: 3
Geef temperatuur om 23 uur: 4
Gem. temp. is 2.916667 graden.
```

DE HAAGSE HOGESCHOOL

# Initialiseren

```
int i = 8;

int rij[] = { 23, 4, 2, 5 };
```

Aantal elementen hoeft niet opgegeven te worden (wordt in dit geval 4).

```
int rij[10] = { 23, 4, 2, 5 };
```

Aantal elementen mag wel opgegeven worden
Elementen 4 t/m 9 worden in dit geval met 0 geïnitialiseerd.

DE HAAGSE HOGESCHOOL

# Initialiseren

*Code:*

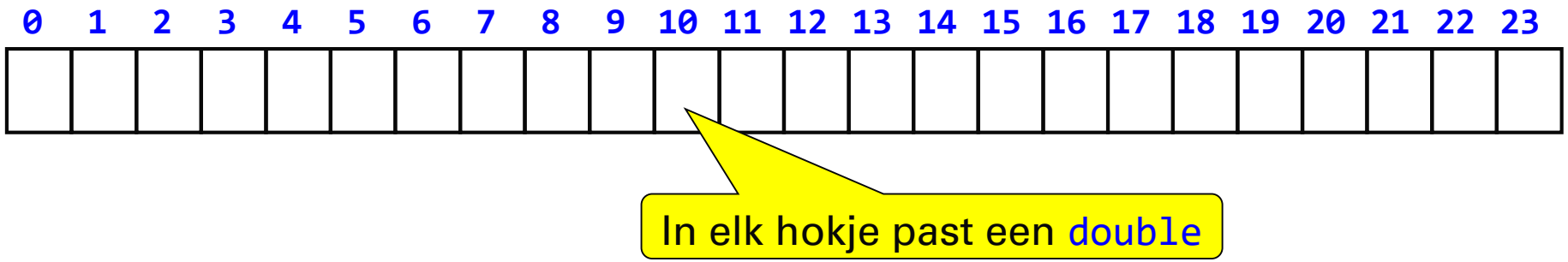
```c
#include <stdio.h>

int main(void) {
    double temp[10] = { 23, 4, 2, 5 };
    int i;
    int s = sizeof temp / sizeof temp[0]; //bepalen grootte array

    //zet hier een breakpoint
    for (i = 0; i < s; i = i + 1) {
        printf("Geef inhoud element %d van arry: %f\n", i,temp[i]);
    }

    getchar();
    return 0;
}
```


*Uitvoer:*

```
Geef inhoud element 0 van arry: 23.000000
Geef inhoud element 1 van arry: 4.000000
Geef inhoud element 2 van arry: 2.000000
Geef inhoud element 3 van arry: 5.000000
Geef inhoud element 4 van arry: 0.000000
Geef inhoud element 5 van arry: 0.000000
Geef inhoud element 6 van arry: 0.000000
Geef inhoud element 7 van arry: 0.000000
Geef inhoud element 8 van arry: 0.000000
Geef inhoud element 9 van arry: 0.000000
```

# Random access

- Elk array element kan even snel bereikt worden.




- Executietijd van `temp[i]` is onafhankelijk van de waarde van `i`.
- Hoe kan dat?
  - adres van element `i` ←→
    beginadres van de array + `i` * aantal bytes per hokje
  - `&temp[i]` ←→ `&temp[0] + i * sizeof temp[0]`

DE HAAGSE HOGESCHOOL

# Array en functies

- Array als returnwaarde. Niet mogelijk in C!

- Array als parameter. Altijd call by reference.
  - Beginadres van de array wordt doorgegeven.

DE HAAGSE HOGESCHOOL

# Array functie parameter

```
#include <stdio.h>


void leesin(double a[], int n) {
    int i;
    for (i = 0; i < n; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", &a[i]);
    }
}


int main(void) {
    double d[5];
    leesin(d,5);
…
```

Beginadres van de array wordt doorgegeven!

Aantal elementen moet als aparte parameter doorgegeven worden!

DE HAAGSE HOGESCHOOL

# Array functie parameter

```
#include <stdio.h>


void leesin(double a[], int n) {
    int i;
    for (i = 0; i < n; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", &a[i]);
    }
}


int main(void) {
    double d[5];
    leesin(&d[0], 5);
…
```

Alternatieve manier om argument door te geven

DE HAAGSE HOGESCHOOL

# Array functie parameter

```
#include <stdio.h>


void leesin(double *a, int n) {
    int i;
    for (i = 0; i < n; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", &a[i]);
    }
}


int main(void) {
    double d[5];
    leesin(&d[0], 5);
…
```

Alternatieve manier om parameter te declareren

Alternatieve manier om argument door te geven

DE HAAGSE HOGESCHOOL

# Array functie parameter

```
#include <stdio.h>


void leesin(double *a, int n) {
    int i;
    for (i = 0; i < n; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", &a[i]);
    }
}


int main(void) {
    double d[5];
    leesin(d, 5);
…
```

Alternatieve manier om parameter te declareren

DE HAAGSE HOGESCHOOL

# Array functie parameter

```
#include <stdio.h>


void leesin(double *a, int n) {
    int i;
    for (i = 0; i < n; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", a + i);
    }
}


int main(void) {
    double d[5];
    leesin(d, 5);
…
```

Alternatieve manier om adres van element te specificeren
`&a[i]` ←→ `a + i`
`a[i]` ←→ `*(a + i)`

DE HAAGSE HOGESCHOOL

# Array functie parameter

```
#include <stdio.h>


void leesin(double a[], int n) {
    int i;
    for (i = 0; i < n; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", &a[i]);
    }
}

int main(void) {
    double d[5];
    leesin(d, sizeof d / sizeof d[0]);
…
```

Alternatieve manier om aantal elementen door te geven

Voordeel?

DE HAAGSE HOGESCHOOL

# Idee! (WERKT NIET)

```
#include <stdio.h>



void leesin(double a[]) {
    int i;
    for (i = 0; i < sizeof a / sizeof a[0]; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", &a[i]);
    }
}

int main(void) {
    double d[5];
    leesin(d);
…
```

`sizeof a` = size of adres!

Er wordt niets ingelezen!

DE HAAGSE HOGESCHOOL

# Operator sizeof in een functie

*Code:*

```c
#include <stdio.h>

int geefAantalElementenArray(double a[]) {

  printf("IN EEN FUNCTIE -> size array: %d\n",sizeof a);
  printf("IN EEN FUNCTIE -> size element array: %d\n\n",sizeof a[0]);

  return sizeof a / sizeof a[0];
}

int main(void) {
  double d[5] = {0};

  printf("size array: %d\n",sizeof d);
  printf("size element array: %d\n\n",sizeof d[0]);

  printf("aantal elementen array: %d\n",geefAantalElementenArray(d));
  printf("aantal elementen array: %d",sizeof d / sizeof d[0]);

  getchar();
  return 0;
}
```

*Uitvoer:*

```
size array: 40
size element array: 8

IN EEN FUNCTIE -> size array: 4
IN EEN FUNCTIE -> size element array: 8

aantal elementen array: 0
aantal elementen array: 5
```

# Array functie parameter

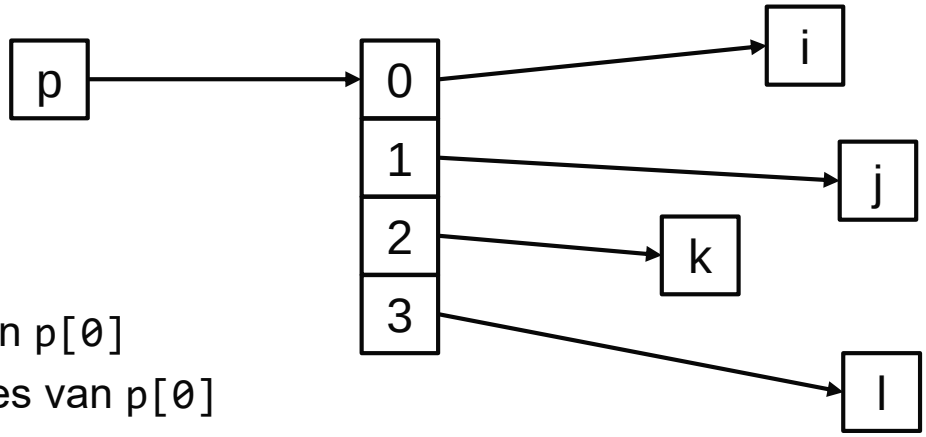
```
…
double gem(double a[], int n) {
     int i;
     double totaal = 0.0;
     for (i = 0; i < n; i = i + 1) {
          totaal = totaal + a[i];
     }
     if (n > 0)
        return totaal / n;
     else
        return 0;
}

int main(void) {
     double d[5];
     size_t size = sizeof d / sizeof d[0];
     leesin(d, size);
     printf("Het gemiddelde is %f", gem(d, size));
}
```

`sizeof` operator geeft getal van type `size_t`

# Het kan erger: array van pointers naar ints

- Je kan een array declareren met pointers als inhoud:

```
int i=3, j=5, k=2, l=4;
int *p[] = {&i, &j, &k, &l};      /* p = array of pointers to ints */
```



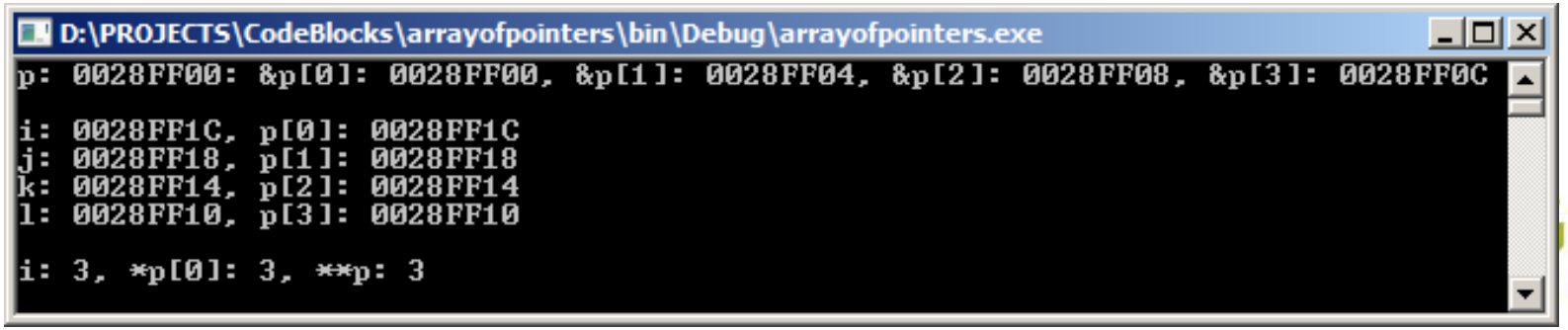

- `p` → adres van `p[0]`
  `&p[0]` → adres van `p[0]`
  `p[0]` → adres van `i`
  `*p[0]` → inhoud van `i`
  `*p` → inhoud van `p[0]`
  `**p` → inhoud van `i`

DE HAAGSE HOGESCHOOL

# Het kan erger: array van pointers naar ints

- Je kan een array declareren met pointers als inhoud:

```
int i=3, j=5, k=2, l=4;
int *p[] = {&i, &j, &k, &l};      /* p = array of pointers to ints */

printf("i: %p, p[0]: %p\n", &i, p[0]); /* print pointer details */
printf("j: %p, p[1]: %p\n", &j, p[1]);
printf("k: %p, p[2]: %p\n", &k, p[2]);
printf("l: %p, p[3]: %p\n", &l, p[3]);

printf("\ni: %d, *p[0]: %d, **p: %d\n", i, *p[0], **p);
                                    /* print i and *p[0] and **p */
```



# Huiswerk

- Schrijf een functie met de naam `reverse` waarmee de inhoud van een rij gehele getallen omgedraaid kan worden. Als de rij de getallen 0, 1, 2 en 3 bevat dan moet de rij na afloop van de functie de getallen 3, 2, 1 en 0 bevatten.

- Bestudeer boek/dictaat:
  - Paragrafen 6.1, 6.2, 6.3, 6.4.1, 6.4.2, 6.4.3, 6.7
  - Paragrafen 7.6, 7.7, 7.9, 7.9.1

- Maak opdrachten:
  - 3 van https://www.w3resource.com/c-programming-exercises/array/index.php

DE HAAGSE HOGESCHOOL