

# HOOFDSTUK 2

## POINTERS

**Helga Naessens**

# Inhoud

- **Pointers: algemeen**
- Call by reference
- Pointers en arrays
- Pointer naar const
- Bewerkingen op pointers
- Pointer als resultaat
- Constante pointer
- Functie-pointers
- C-strings

# Pointers

- **Wijzers** naar gegeven van een welbepaald type

- Voorbeeld:

```
int g = 6;
```

```
int *p;
```

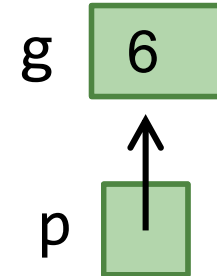
```
p = &g;
```

type p =  
pointer naar int

```
int * p = &g;
```

**&** = adresoperator

**\*** = dereferentieoperator



```
*p = 9; (*p)++; *p *= 2;
```

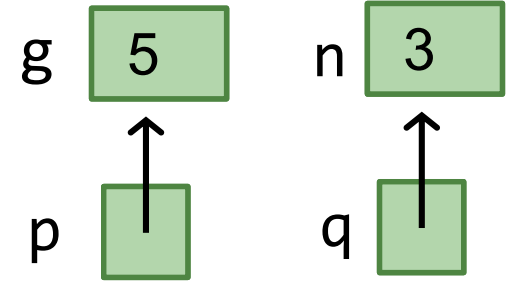
```
printf("*p=%d  p=%p  &p=%p\n", *p, p, &p);
```

```
printf("*g=%d  g=%d  &g=%p\n", *g, g, &g);
```

conversiekarakter **p** = pointer (hexadecimaal adres)

```
int g=5, n=3, *p=&g, *q=&n;
```

Beter: pointers gescheiden houden van basistypes



```
*q = *p;
```

```
q = p;
```

```
*q = p;
```

```
q = *p;
```

compiler geeft warning

- **Nullpointer:**

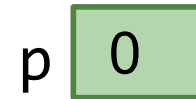
```
double *p;
```

```
p = 0;    of    p = NULL;
```

```
if (p) { ... }
```

```
double d = *p;
```

Nullpointer NIET derefereren!!!



- **void-pointer:**

```
int n; int *pi = &n;
```

```
double d; double *pd = &d;
```

```
pd = &n;  
pd = pi; }
```

compiler geeft warning

```
void *v;
```

```
v = &n;
```

```
v = pd;
```

```
printf("*v=%f\n", *v);
```

```
printf("*v=%f\n", *(double *)v);
```

```
pd = v = pi;
```

void-pointer kan niet  
gederefereerd worden!



Syntactisch ok, maar verwarrend,  
want \*pd bevat nu een int  
(i.p.v. een double)

- voorbeeld: **pointers\_inl.c**

# Inhoud

- Pointers: algemeen
- **Call by reference**
- Pointers en arrays
- Pointer naar const
- Bewerkingen op pointers
- Pointer als resultaat
- Constante pointer
- Functie-pointers
- C-strings

# Call by reference

- **Call by value:** waarde uit oproepende context wijzigt niet.
- **Call by reference:**
  - geef de **adressen** van variabelen door
  - de formele parameters zijn **pointers**
  - gebruik **dereferentie** van parameters om waarde van variabelen uit oproepende context te wijzigen

```
int i = -5;
tegenstelde(i);
...
void tegengestelde(int a) {
    a *= -1;
}
```



```
int i = -5;
tegenstelde(&i);
...
void tegengestelde(int *a) {
    *a *= -1;
}
```

## Oefening 1:

Schrijf een procedure **vwk(a,b,c,aantal,w1,w2)** die de vierkantswortels van de gegeven vergelijking  $ax^2 + bx + c$  bepaalt (a, b en c reële getallen). Het aantal wortels wordt opgeslagen in **aantal**, de eventuele wortel(s) in **w1** en **w2**.

Schrijf daarna een hoofdprogramma dat de vierkantswortels van  $7x^2 - 8x + 16$  bepaalt.



## Oefening 2:

Schrijf een procedure **wissel(a,b)** waarmee de inhoud van twee gehele getallen kan gewisseld worden.

### **Versie 1:**

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

int main() {
    int g1 = 1, g2 = 2;
    wissel(&g1,&g2);
    ...
}
```

### **Versie 2:**

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

int main() {
    int g1 = 1, g2 = 2;
    wissel(&g1,&g2);
    ...
}
```

# Inhoud

- Pointers: algemeen
- Call by reference
- **Pointers en arrays**
- Pointer naar const
- Bewerkingen op pointers
- Pointer als resultaat
- Constante pointer
- Functie-pointers
- C-strings

# Pointers en arrays

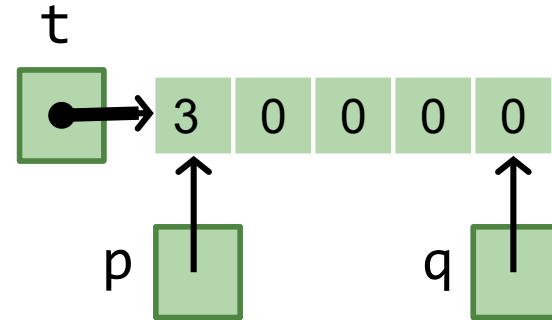
```
double t[5] = {3};
```

```
double *p, *q;
```

```
p = &t[0];    ⇒ p = t;
```

```
q = &t[4];
```

**$\&t[0] \equiv t$**



arraypointer  $t$  = **constante** pointer  
naar 1<sup>e</sup> element van de array

```
printf("*t=%lf\n", *t);
```

**$t[0] \equiv *t$**

⇒

**$*p \equiv p[0]$**

$p[i]$  = inhoud van het element  
 $i$  geheugenplaatsen na/voor  $p$

## Voorbeeld:

```
#include <stdio.h>
#define N 5

int main(void) {
    double t[N];
    double *p=t, *q=&t[N-1];
    int i;
    for (i=0 ; i<N ; i++)
        p[i] = 2*i;
    for (i=0 ; i<N ; i++)
        printf("%lf    ",q[-i]);
    return 0;
}
```

**Wat is de uitvoer van  
dit programma?**

- Oefening 1: Vul onderstaande tabel aan:

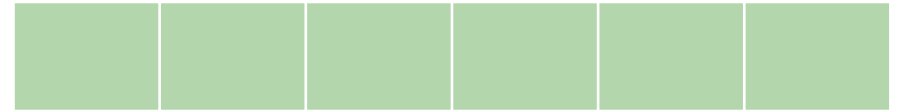
```
int t[6] = {2};
```

```
int *p = &t[4], *q = &p[-3];
```

```
*p = 5; q[0] = 6;
```

```
p[-1] -= *t;
```

t



- Oefening 2: Geef de betekenis van onderstaande declaraties:

```
int *a;    //pointer naar een int
```

```
int **b;   //pointer naar een pointer naar een int
```

```
int *x[5]; //array van pointers naar int
```

```
int (*y)[10]; //pointer naar array van int
```

(zie ook **pointers\_decl.c**)

- Toepassing: arrays als pointerparameters

```
void vulop(int [], int, int);
```

≡ int \*

```
int main() {  
    int tab[100];  
    vulop(tab, 100, 1);  
}
```

≡ int \*t

```
void vulop(int t[], int n, int waarde) {  
    int i;  
    for (i=0 ; i<n ; i++)  
        t[i] = waarde;  
}
```

Oefening: vul de 1<sup>e</sup> helft  
van de array op met -5, de  
2<sup>e</sup> helft met 5

**Let op:**

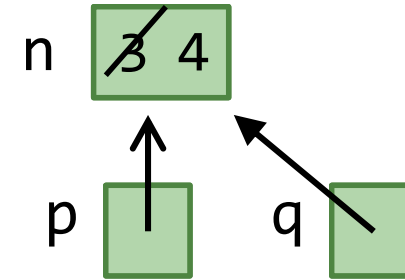
```
int *t = {1,2,3}; //FOUT  
int t[] = {1,2,3} //OK
```

# Inhoud

- Pointers: algemeen
- Call by reference
- Pointers en arrays
- **Pointer naar const**
- Bewerkingen op pointers
- Pointer als resultaat
- Constante pointer
- Functie-pointers
- C-strings

# Pointer naar const

```
int n = 3;  
int *p = &n;  
const int *q = &n;
```



```
(*p)++;
```

```
(*q)++;
```

compiler geeft error

```
*p = *q;
```

```
*q = *p;
```

compiler geeft error

```
p = q;
```

compiler geeft warning: NIET const = const mag NIET

```
q = p;
```

q = pointer naar const  
 $\Rightarrow$  \*q is constant  
 $\equiv$  int const \*q



- Oefening:

Schrijf een logische functie `is_omkeerbaar(t,n)` die nagaat of de gegeven array `t` met `n` gehele getallen omkeerbaar is

Vb: 1 2 3 2 1 : omkeerbaar

1 2 3 2 : niet omkeerbaar

Opl:

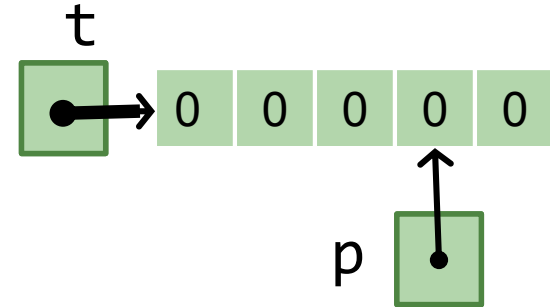
```
bool is_omkeerbaar(const int *t, int n) {  
    const int *p = &t[n-1];  
    int i = 0;  
    while (i < n/2 && t[i] == p[-i])  
        i++;  
    return i == n/2;  
}
```

# Inhoud

- Pointers: algemeen
- Call by reference
- Pointers en arrays
- Pointer naar const
- **Bewerkingen op pointers**
- Pointer als resultaat
- Constante pointer
- Functie-pointers
- C-strings

# Bewerkingen op pointers (= schuivende pointers)

```
int t[5] = {0};  
int *p, *q;  
p = &t[3];  ⇒  p = t+3;
```



**$\&t[i] \equiv t + i$**

```
q = p-2;
```

```
t = p+1;
```

```
p++;
```

```
q--;
```

```
t++;
```

`t` is een arraypointer (= constante pointer)  
en kan dus niet verplaatst worden!!

**Opmerking:** bewerkingen op void-pointers  
zijn niet toegestaan!

- Let op:

`p++    ≠    (*p)++    ≠    *p++`

Eerst `*p` opvragen, dan `p` verzetten  
⇒ is steeds onderdeel van een opdracht  
vb: `printf("*p = %d\n", *p++);`

- Overige bewerkingen

`printf("q-p=%d", q-p);`

afstand tussen `q` en `p`

`if (q < p) printf("q staat voor p");`

- Voorbeeld: **pointers\_bew.c**

- Oefening: herschrijf de logische functie `is_omkeerbaar(t, n)`  
gebruik hierbij enkel schuivende pointers (geen `[ ]`)

- Opmerking:

```
void vulop(int *t, int n, int w) {
```

```
    int i;
```

```
    for (i=0 ; i<n ; i++)
```

```
        *t++ = w;
```

```
}
```

— Ok, t is gewone pointer en  
kan dus verplaatst worden!!

```
int main() {
```

```
    int i;
```

```
    int t[10] = {0};
```

```
    for (i=0 ; i<10 ; i++)
```

```
        *t++ = 5;
```

```
    vulop(t, 10, 5);
```

```
    return 0;
```

```
}
```

— Niet ok, t is arraypointer  
(= constante pointer) en kan dus niet  
verplaatst worden!!

# Inhoud

- Pointers: algemeen
- Call by reference
- Pointers en arrays
- Pointer naar const
- Bewerkingen op pointers
- **Pointer als resultaat**
- Constante pointer
- Functie-pointers
- C-strings

# Pointer als functieresultaat

Voorbeeld: Schrijf een functie `kleinste(t,n)` die in de gegeven array `t` met `n` gehele getallen op zoek gaat naar het kleinste getal en een pointer teruggeeft naar dit kleinste getal.

```
const int * kleinst(const int *t, int n) {  
    const int *kl = t;  
    int i;  
    for (i=1 ; i<n ; i++)  
        if (*kl > *(t+i))  
            kl = t+i;  
    return kl;  
}
```

(zie ook [pointer\\_res.c](#))

# Pointer verplaatsen

Geef een **pointer naar de pointer** mee indien de pointer, die meegegeven wordt als argument, effectief moet gewijzigd/verplaatst worden!!!

## Oefening

Gegeven (in een hoofdprogramma) een array bestaande uit  $n$  gehele getallen en een pointer  $p$  die wijst naar het eerste element van deze array.

Schrijf een procedure `verzet_naar_kleinste(p,n)` die de pointer  $p$  verplaatst naar het kleinste getal in de array.

Maak hierbij verplicht gebruik van schuivende pointers.



## Opmerkingen:

```
const int MAX = 5;
```

```
int *p; const int *pc; const int **ppc; ...
```

```
p = pc; /* warning */
```

```
pc = p; /* ok */
```

```
p = &MAX; /* warning */
```

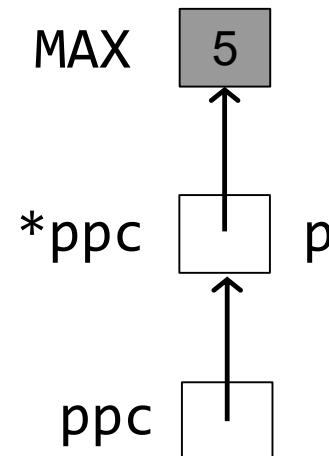
```
(*p)++;
```

```
pc = &MAX; /* ok */
```

```
ppc = &p; /* warning */
```

```
*ppc = &MAX;
```

```
(*p)++;
```



# Inhoud

- Pointers: algemeen
- Call by reference
- Pointers en arrays
- Pointer naar const
- Bewerkingen op pointers
- Pointer als resultaat
- **Constance pointer**
- Functie-pointers
- C-strings

# Constante pointer

```
int n = 7, m = 5;  
const int *p = &n;  
int * const q = &n;  
const int * const l = &n;
```

q = constante pointer  
⇒ q is constant

~~(~~\*p~~)++;~~ /\* error \*/

p = &m;

(\*q)++;

~~q = &m;~~ /\* error \*/

~~(~~\*l~~)++;~~ /\* error \*/

~~l = &m;~~ /\* error \*/

Opmerking: bij constante pointer is initialisatie bij declaratie verplicht!!

# Opmerking const

- const slaat op wat er onmiddellijk voor staat, tenzij er niets voor staat, dan slaat het op wat er na komt.

- Voorbeeld

const int \* p      => p = pointer naar int die constant is      →

int const \* p      => p = pointer naar int die constant is      →

int \* const p      => p = constante pointer naar int      ●→

const int \* const p   => p = constante pointer naar int die  
constant is      ●→

- Oefening

Gegeven de declaratie    const int \* \* const \* a;

Welke van volgende opdrachten zijn syntactisch correct?

a++; (\*a)++; (\*\*a)++; (\*\*\*)a)++;

# Inhoud

- Pointers: algemeen
- Call by reference
- Pointers en arrays
- Pointer naar const
- Bewerkingen op pointers
- Pointer als resultaat
- Constante pointer
- **Functie-pointers**
- C-strings

# Functie-pointer

- Functie-pointer = pointer naar een functie
- Declaratie: `return-type (*fname)([prototype-lijst]);`

Voorbeeld: `double (*fp)(double *, int);`

Interpretatie:

*fname* → pointer naar functie

*\*fname* → functie zelf

*(\*fname)(actuele\_lijst)* → functie-oproep



equivalent met: *fname(actuele\_lijst)*

- Voorbeeld:

```
double som(double x, double y) { return x + y; }
double product(double x, double y) { return x * y;}

int main() {
    double a = 10, b = 5;
    double (*pfun)(double, double) = &som;
    printf("som = %f\n", (*pfun)(a,b));
    pfun = product; //& mag weggelaten worden!
    printf("product = %f\n", pfun(a,b));
    pfun = fmin; //uit math.h
    printf("min = %f\n", pfun(a,b));
    return 0;
}
```

- Array van functie-pointers:

```
double som(double x, double y) { return x + y; }
double product(double x, double y) { return x * y;}

int main() {
    double a = 10, b = 5;
    double (*t[])(double, double) = {som,product,fmin};
    int i;
    for (i=0 ; i<3 ; i++)
        printf("result = %f\n", t[i](a,b));
    return 0;
}
```



- Functie-pointer als parameter:

```
void change(double (*)(double), double *, int);
```

```
int main() {
```

```
    double tab[] = {1.0,2.0,4.0,8.0};
```

```
    change(sqrt,tab,4);
```

```
}
```

```
void change(double (*fp)(double), double *t, int n) {
```

```
    int i=0;
```

```
    for (i=0 ; i<n ; i++)
```

```
        t[i] = fp(t[i]);
```

```
}
```

- Voorbeeld: **fpointer\_vb.c**

- Toepassing: methode qsort uit stdlib.h

```
void qsort(void *base, size_t nitems, size_t size,  
          int (*compar)(const void *, const void*))  
  
//base = pointer naar het eerste element van de array  
//size_t = unsigned type, gedefinieerd in stddef.h  
//nitems = grootte array  
//size = grootte van 1 element uit de array  
//compar = functie die 2 elementen uit de array  
//          met elkaar vergelijkt
```

**vb\_qsort.c**

Hoe bekom je een pointer naar het i-de  
element van de array base?

# Inhoud

- Pointers: algemeen
- Call by reference
- Pointers en arrays
- Pointer naar const
- Bewerkingen op pointers
- Pointer als resultaat
- Constante pointer
- Functie-pointers
- **C-strings**

# C-string

- Pointer naar karakter van een karaktersliert, afgesloten met het **nullkarakter**
- nullkarakter: `char c = '\0';`      `/* of    char c = 0; */`
- Declaratie en initialisatie C-string:

```
char *s1 = "str1";
```

```
char s2[10] = "vb2";
```

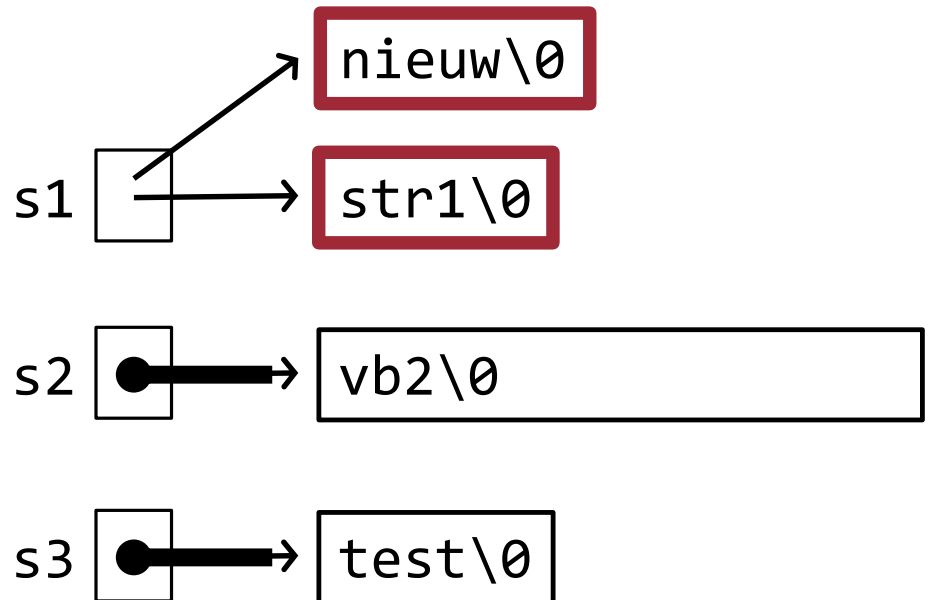
```
char s3[] = "test";
```

```
s1 = "nieuw";
```

```
s2 = "vb"; s3 = "vb";
```

```
*s1 = 'N';
```

```
*s2 = 'V'; *s3 = 'T';
```



- Inlezen van een C-string m.b.v. scanf:

```
char *s1 = "str1"; char s2[80]; char s3[] = "test";
```

```
scanf("%s", s1); Programma crasht!
```

```
scanf("%s", s2); //max. 1 woord inlezen
```

```
scanf("%s", s3); //max. 1 woord inlezen
```

```
scanf("%79s", s2); //max. 1 woord (79 karakters) inlezen
```

```
scanf("%4s", s3);
```

**LET OP:** dit mag **géén variabele** zijn!

Beter en veiliger, zodat enkel gereserveerd geheugen gebruikt wordt!!

**LET OP:** plaats geen & voor de C-string-variabele!

Dit argument is reeds een adres.

- Inlezen van een c-string m.b.v. (f)gets (uit <stdio.h>):

`char* gets(char *s)`

`char* fgets(char *s, int n, FILE *stream)`

=> resultaat van (f)gets = NULL als inlezen mislukt is

`char *s1;`


`char s2[80]; char s3[] = "string";`

~~`gets(s1);`~~      **Programma crasht!**

`gets(s2); gets(s3);`    //telkens (rest van) **1 lijn** inlezen

`fgets(s2,80,stdin);`    //max. **79 karakters** inlezen

`fgets(s3,7,stdin);`



Beter en veiliger, enkel gereserveerd geheugen wordt gebruikt!!

- Ingebouwde stringfuncties (<string.h>):

```
size_t strlen(const char *s); //zonder nullchar  
char* strcpy(char *dest, const char *src);  
int strcmp(const char *s1, const char *s2);  
char* strcat(char *dest, const char *src);
```



**LET OP: strcpy en strcat alloceren GEEN geheugen!!!  
De gebruiker moet zorgen voor voldoende geheugenruimte.**

Veiliger alternatief:

```
char* strncpy(char *dest, const char *src, size_t n);  
char* strncat(char *dest, const char *src, size_t n);
```

- voorbeeld: **strings.c**

- Oefeningen:

1. Schrijf een functie **int lengte(s)** die de lengte van de gegeven C-string s bepaalt (gebruik strlen niet!)

```
int lengte(const char *s) {  
    const char *p =s;  
    while (*p) p++;  
    return p-s;  
}
```

2. Schrijf een procedure **void kopieer(d,s)** die het gedrag van de functie strcpy nabootst (en dus s kopieert naar d).  
Gebruik strcpy niet!



3. Schrijf een functie `int my_atoi(s)` die de gegeven string `s` omzet naar een geheel getal. Gebruik de methode van Horner.
4. Schrijf een functie `eerste_hoofdletter(s)` die een pointer teruggeeft naar de 1<sup>e</sup> hoofdletter in de gegeven string `s` of de nullpointer indien `s` geen hoofdletter bevat.
5. Zet voorgaande functie om naar een procedure, zodat de pointer verzet wordt naar de eerste hoofdletter
6. Schrijf een procedure `void zetom(s)` die alle hoofdletters uit de gegeven string `s` omzet naar kleine letters.
7. Schrijf een procedure `void verwijder(s)` die alle niet-letters uit de gegeven string `s` verwijdert.

- argc en argv:

```
/* argc_argv.c */  
#include <stdio.h>
```

aantal strings op de command  
line (inclusief programmanaam)

```
int main(int argc, char** argv) {
```

array met strings op de command line  
(inclusief programmanaam)

```
    int i;
```

```
    if (argc == 1)
```

```
        printf("geen extra strings meegegeven");
```

```
    else {
```

```
        for (i=1 ; i<argc ; i++)
```

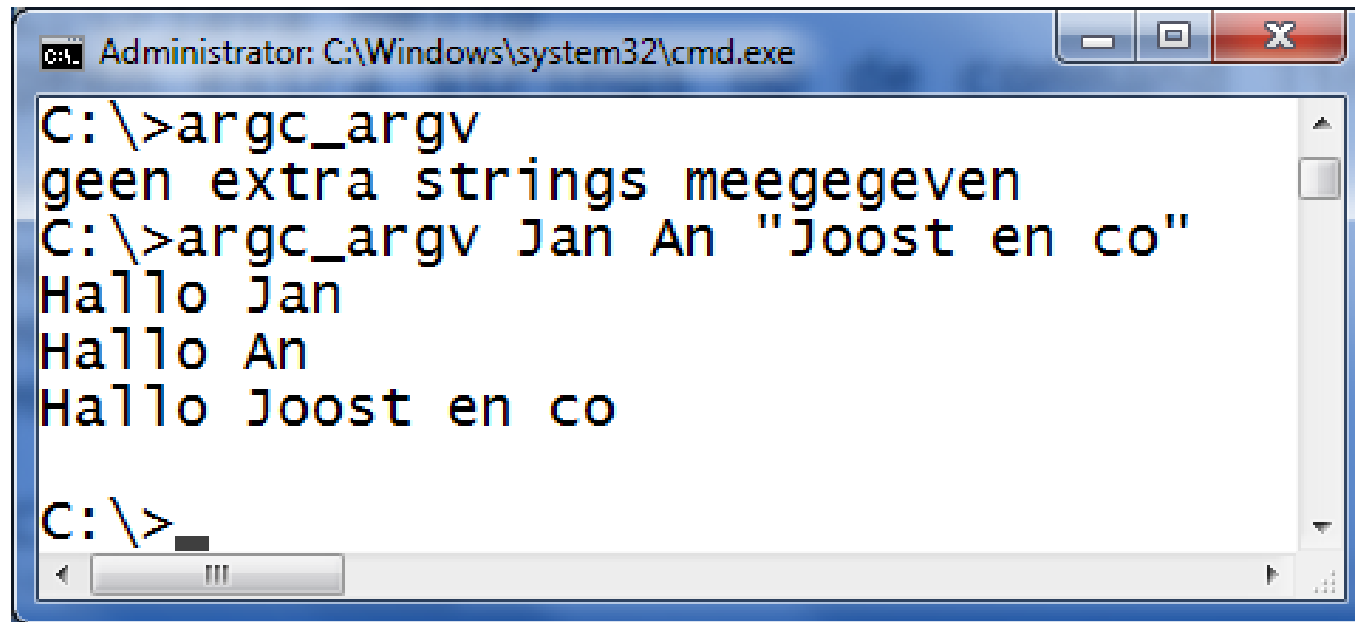
```
            printf("Hallo %s\n", argv[i]);
```

```
    }
```

```
    return 0;
```

```
}
```

Uitvoer programma:



```
C:\>argc_argv
geen extra strings meegegeven
C:\>argc_argv Jan An "Joost en co"
Hallo Jan
Hallo An
Hallo Joost en co

C:\>
```

Command line arguments ingeven in DevC++:

Execute ➡ Parameters... ➡ extra strings invullen bij  
“Parameters to pass...” ➡ klikken op knop OK ➡  
programma laten lopen

# Inhoud

- Pointers: algemeen
- Call by reference
- Pointers en arrays
- Pointer naar const
- Bewerkingen op pointers
- Pointer als resultaat
- Constante pointer
- Functie-pointers
- C-strings