

---

# Programmeermethoden

Arrays

week 7: 17–21 oktober 2011 (Den Haag: 24–28)

`http://www.liacs.nl/home/kosters/pm/`

Een **array** is een geordend rijtje variabelen van hetzelfde type, bijvoorbeeld een vector met 10 “reële” getallen: na

```
double A[10];
```

heb je 10 `double`'s, namelijk

```
A[0], A[1], A[2], A[3], A[4],  
A[5], A[6], A[7], A[8] en A[9].
```

Er zijn ook meer-dimensionale arrays: *matrices*, zie *Life*.

Naamgeving: `A[4]` is een **array-element** (het vierde, of eigenlijk het vijfde), 4 de bijbehorende **array-index**.

Maak eerst een constante:

```
const int MAX = 100;
```

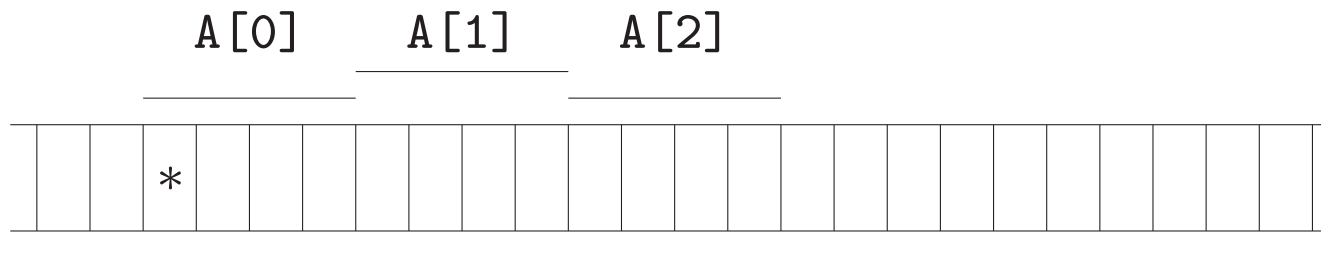
Daarna definiëren (voorlopig hetzelfde als declareren) we een array rij met 100 (of preciezer MAX) int's als volgt:

```
int rij[MAX];
```

Je mag een array **meteen bij definitie** initialiseren (en anders alleen element voor element):

```
double B[5] = {42, 3.14, 1e6, 0, 37};  
char str[10] = "feestje"; // str[7] wordt '\0'  
  
rij[8] = 37;  
rij[2] = rij[5] + rij[9];
```

Met `int A[3];` maken we een array A met 3 integers: `A[0]`, `A[1]` en `A[2]`, achter elkaar in het geheugen. Stel dat een `int` 4 bytes beslaat, dan benutten we in totaal dus  $3 \times 4 = 12$  bytes:



Hier kom je terecht als je `A[4]` gebruikt!

Als je `cout << A << endl;` doet krijg je de waarde van A te zien, en dat is het **geheugenadres** van de eerste byte van het eerste array-element, `A[0]`, oftewel het adres van `*`.

We gebruiken een array `rij` met `MAX` elementen bijvoorbeeld als volgt:

```
int i; // array-index
for ( i = 0; i < MAX; i++ ) rij[i] = 5 * i;
for ( i = 0; i < MAX - 1; i++ ) rij[i] = rij[i+1];
for ( i = MAX - 1; i > 0; i-- ) rij[i-1] = rij[i];
```

Met `MAX` gelijk aan 10 wordt `rij` achtereenvolgens:

0	5	10	15	20	25	30	35	40	45
5	10	15	20	25	30	35	40	45	45
45	45	45	45	45	45	45	45	45	45

Let er op niet het array uit te lopen!

Gebruik dus nooit, ook niet indirect, `rij[MAX]` of `rij[-42]`!

Hoe druk je de inhoud van een array af?

```
void drukaf (int A[ ], int n) {  
    int i;  
    for ( i = 0; i < n; i++ )  
        cout << A[i]; // (*)  
} //drukaf
```

Of (grapje) bij (\*):

```
    cout << A[i] << ( i % 10 == 9 ? '\n' : ' ');
```

met de **ternaire operator** ...?...:..., een voorwaardelijke expressie.

Sommigen zetten de declaratie van *i* in de for-loop:

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

(pas dan op met geldigheid = scope van de variabele *i*).

En het minimum van een array:

```
int minimum (const int A[ ], int n) {  
    int klein = A[0], i;  
    for ( i = 1; i < n; i++ )  
        if ( A[i] < klein ) // kleinere gevonden  
            klein = A[i];  
    return klein;  
} //minimum
```

Die `const` verbiedt toekenningen aan array-elementen. In de heading mag ook `const int * A` staan, of `const int A[123]`. Die 123 wordt genegeerd: het gaat erom dat je doorgeeft dat het een integer-array is (de eerste parameter), met `n` elementen (de tweede parameter).

```
// Zoek getal in array A (n elementen). Lineair zoeken.  
// Geeft index met A[index] = getal, als getal tenminste  
// voorkomt; zo niet: resultaat wordt -1.  
int lineairzoeken (int A[ ], int n, int getal) {  
    int index = 0;  
    bool gevonden = false;  
    while ( ! gevonden && ( index < n ) ) {  
        if ( getal == A[index] )  
            gevonden = true; // of meteen: return index;  
        else  
            index++;  
    }//while  
    if ( gevonden ) // en dan hier: return -1;  
        return index;  
    else  
        return -1;  
}//lineairzoeken
```

Hoe sorteer je een array oplopend? Een eerste idee is: zet herhaald de “kleinste” vooraan.

```
void simpelsort (int inhoud[ ], int n) {
    int voorste, kleinste, plaatskleinste, k;
    for ( voorste = 0; voorste < n; voorste++ ) {
        plaatskleinste = voorste;
        kleinste = inhoud[voorste];
        for ( k = voorste + 1; k < n; k++ )
            if ( inhoud[k] < kleinste ) {
                kleinste = inhoud[k];
                plaatskleinste = k;
            }//if
        if ( plaatskleinste > voorste )
            wissel (inhoud[plaatskleinste], inhoud[voorste]);
    }//for
}//simpelsort
```

Een voorbeeld van de werking van simpelsort:

0 1 2 3 4 5 6 (n=7)

3 8 7 5 2 4 9

2| 8 7 5 3 4 9

2 3| 7 5 8 4 9

2 3 4| 5 8 7 9

2 3 4 5| 8 7 9

2 3 4 5 7| 8 9

2 3 4 5 7 8| 9

2 3 4 5 7 8 9|

En nog een sorteermethode:

```
void bubblesort (int A[ ], int n) {
    int i, j;
    for ( i = 1; i < n; i++ )
        for ( j = 0; j < n - i; j++ )
            if ( A[j] > A[j+1] )
                wissel (A[j],A[j+1]); // (*)
} //bubblesort
```

Bij (\*):

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

of (zonder functie wissel):

```
{ int temp = A[j]; A[j] = A[j+1]; A[j+1] = temp; }
```

Een voorbeeld van de werking van simpelsort (links) en bubblesort (rechts):

0	1	2	3	4	5	6	(n=7)	0	1	2	3	4	5	6
3	8	7	5	2	4	9		3	8	7	5	2	4	9
2	8	7	5	3	4	9		3	7	5	2	4	8	9
2	3	7	5	8	4	9		3	5	2	4	7	8	9
2	3	4	5	8	7	9		3	2	4	5	7	8	9
2	3	4	5	8	7	9		2	3	4	5	7	8	9
2	3	4	5	7	8	9		2	3	4	5	7	8	9
2	3	4	5	7	8	9		2	3	4	5	7	8	9
2	3	4	5	7	8	9								

**Bubblesort** doet bij een rij met  $n$  elementen

$$(n - 1) + (n - 2) + \dots + 3 + 2 + 1 = n(n - 1)/2$$

vergelijkingen tussen array-elementen. Het is een  $O(n^2)$  (“orde  $n^2$ ”) algoritme — en dat is niet zo fijn.

Dezelfde analyse geldt voor “simpelsort” = **Selection sort**.

Later meer over zoeken en sorteren ... het kan namelijk beter = sneller!

Hoe roep je functies met arrays als parameter aan?  
Enkele voorbeelden, waarbij het array `rij` gedefinieerd is via `int rij[MAX];`:

```
drukaf (rij,8); (eerste 8 elementen afdrukken)
```

```
cout << minimum (rij,10) << endl;  
    (druk kleinste van eerste 10 elementen af)
```

```
bubblesort (rij,MAX); (sorteer hele array)
```

```
wissel (rij[5],x); (wissel wat)
```

Dus *nooit* `drukaf (rij[ ],8);!`

De **zeef van Erathosthenes** vindt priemgetallen:

```
bool zeef[MAX]; // getal i priem <=> zeef[i] true
int getal, veelvoud;
double wortel = sqrt (MAX);
zeef[0] = false;
zeef[1] = false;
for ( getal = 2; getal < MAX; getal++ )
    zeef[getal] = true; // ... tot het tegendeel bewezen is
for ( getal = 2; getal < wortel; getal++ )
    if ( zeef[getal] ) { // streep veelvouden door
        veelvoud = 2 * getal; // getal + getal als * "te duur" is
        while ( veelvoud < MAX ) {
            zeef[veelvoud] = false;
            veelvoud = veelvoud + getal;
        } //while
    } //if
for ( getal = 2; getal < MAX; getal++ ) {
    if ( zeef[getal] )
        cout << getal << " ";
} //for
```

Komt het  $m$ -letter woord `woord` voor in het  $n$ -letter verhaal `verhaal`? Retourneer “begin” van de eerste match, of  $-1$ .

```
int komtvoor (char woord[ ], char verhaal[ ], int m, int n) {
    int i = 0, // om door verhaal heen te lopen
        j;    // om door woord heen te lopen
    bool gevonden = false;
    while ( ! gevonden && ( i + m <= n ) ) {
        gevonden = true;           // optimist
        for ( j = 0; j < m; j++ )  // bot
            if ( woord[j] != verhaal[i+j] ) // pech
                gevonden = false;
        i++; // en eventueel if ( gevonden ) return i(-1); ...
    }//while
    if ( gevonden )
        return i-1; // of cout << "Ja, index = " << i-1 << endl;
    else
        return -1; // of cout << "Nee" << endl;
}//komtvoor
```

Er zijn talloze patroonherkennings-algoritmen die sneller een (korte) string in een (lange) string opsporen.

Voorbeelden zijn het **Boyer-Moore** algoritme en het **Knuth-Morris-Pratt** algoritme, zie het college Datastructuren.

Stel je zoekt BABBM, en ziet de mismatch  $A \leftrightarrow M$ :

```
U V W B A B B A T L K ...
      B A B B M
```

Je kunt dan doorschuiven naar

```
U V W B A B B A T L K ...
          B A B B M
```

en T met B gaan vergelijken.

Hoe werk je in de praktijk met rijtjes char's, oftewel **strings**?  
Dat kan in C++ op twee manieren:

- Met “ouderwetse” **C-strings**:

```
char woord[7] = "Het.";
```

Nu wordt woord[4] gelijk aan '\0', en zit er troep in woord[5] en woord[6].

- Beter: met nieuwe strings uit de **string-klasse**, via:

```
#include <string>
```

```
string woord = "De.";
```

Nu heb je woord.length ( ) (in dit geval 3) array-elementen, zoals woord[1], waar 'e' in zit. En woord.c\_str ( ) levert de overeenkomende C-string. Maar hoe die ingewikkelde objecten opgeslagen zitten?

Je kunt beide types strings inlezen, maar er zijn gevaren:

- C-strings “groeien niet mee”, strings uit de string-klasse wel.
- Met `cin >> regel;` lees je in de (C-)string `regel` in, maar dit stopt bij “whitespace”! Gebruik liever `getline`.
- Als je doet

```
cin >> n; // een int
getline (cin, regel); // een string
```

wordt `regel` de lege string: de vorige Enter!

En tot slot: met strings uit de string-klasse werken `==`, en zelfs `<` en `<=` (lexicografische ordening), met C-strings niet ... want daar vergelijk je met `==` adressen.

Hoe kun je parameters doorgeven aan een programma?  
Dat gaat via C-strings, die je meegeeft aan de functie `main`:

```
int main (int argc, char* argv[ ]) {  
    if ( argc > 1 ) // telt aantal parameters  
        cout << "===Parameter " << argv[1] << endl;  
    cout << "===Executable heet " << argv[0] << endl;  
} //main
```

Als je dit programma (zeg `iets.cc`) compileert met  
`g++ -Wall -o doewat iets.cc` levert `./doewat 1234` op:  
===Parameter 1234  
===Executable heet ./doewat

En wat gebeurt er als je een array kopieert met `A = B;`?  
Dan wijzen ze naar ( “zijn ze” ) dezelfde inhoud, maar is het oude array A “zoek”. Je kopieert namelijk het beginadres.  
Dus beter is:

```
void kopieer (int A[ ], int B[ ], int n) {  
    int i;  
    for ( i = 0; i < n; i++ )  
        A[i] = B[i];  
} //kopieer
```

En let op: geen `&` te zien.

Voor `int B[ ]` zou `const` mogen staan.

Tot nu toe moest de array-grootte steeds een constante zijn, dus

```
int n;  
cin >> n;  
int A[n]; // verboden! (maar compileert meestal wel)  
int B[10]; // in orde
```

Wat goed is, maar wellicht weer andere nadelen heeft is:

```
int n;  
cin >> n;  
A = new int[n]; // dynamisch array (zie pointers ...)  
...  
delete [ ] A; // ook zelf opruimen!
```

Voor **Life** is een 2-dimensionaal array nodig:

```
bool deWereld[MAX][MAX];
```

Er geldt: `deWereld[i][j] = true` precies dan als in rij `i` (van boven) en kolom `j` (van links) leven zit.

Dit kan mooi in een klasse `leven` verpakt worden, met methoden als

```
vulrandom ( )
```

Maak eerst een *menu*. Zie de tips op

[www.liacs.nl/home/kosters/pm/pmwc8.php](http://www.liacs.nl/home/kosters/pm/pmwc8.php)

- werkcollege dinsdag 18 en donderdag 20 oktober: **zalen B1/B2** (Den Haag: 24 oktober, zaal Archipel), zie <http://www.liacs.nl/home/kosters/pm/pmwc7.php>
- werk aan de derde programmeeropgave — de deadline is op vrijdag 18 november 2011
- lees Savitch Hoofdstuk 5, en 9 voor strings
- lees dictaat Hoofdstuk 3.8
- maak opgaven 31/36 uit het opgavendictaat
- <http://www.liacs.nl/home/kosters/pm/>