# Programmeermethoden

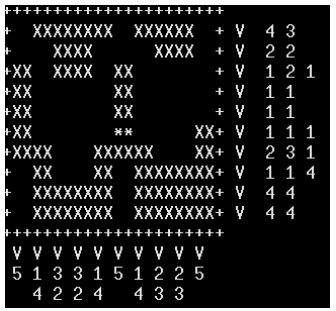
Arrays (vervolg 2)

Walter Kosters en Jonathan Vis

week 9: 6–10 november 2023

www.liacs.leidenuniv.nl/~kosterswa/pm/

Voor het verslag:




- Nonogram: . . . uitleg . . .

- citatie/referentie: `Tja~\cite{abc}` levert "Tja [1]", met

```
\begin{thebibliography}{XX}
\bibitem{abc} P.~Puk, Kabouters in de Tweede
Kamer, Ons Tijdschrift 42 (2023) 12--34.
\end{thebibliography}
```

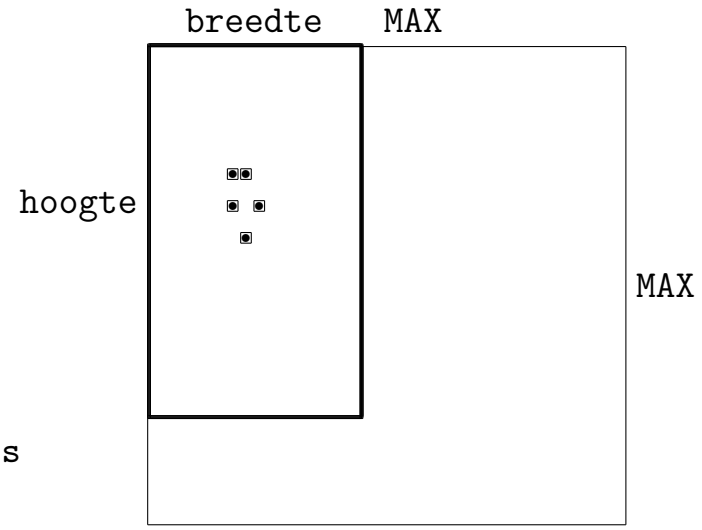
- plaatje: `\includegraphics[height=4cm]{iets}`; let op het type (`.png`, `.jpg`, `.pdf`; gebruik zo nodig `convert`); bovenaan moet `\usepackage{graphicx}` staan; en gebruik Shift-PrtScn; zie mooier.tex

www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

Een klasse `nonogram` voor Nonogram ziet er ± zo uit:

```
class nonogram {
  public:
    nonogram ( );  // constructor
    void drukaf ( );
    void vulrandom ( );
    void maakschoon ( );
    void zetpercentage ( );
    // ...
  private:
    bool dewereld[MAX][MAX]; // array!!!
    int rijen[MAX][MAX];     // en nog zoiets
    int hoogte, breedte;
    int percentage;
    // ...
};//nonogram (let op de punt-komma hier)


// default constructor
nonogram::nonogram ( ) }
  hoogte = 10;
  // ...
}//nonogram::nonogram
```




klasse object

↓ ↓

```
nonogram N;

N.drukaf ( );
```

# Driehoek van Pascal

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
```

↓

```
1   0
1   1   0                       A[i-1][j-1] + A[i-1][j]
1   2   1   0                               = A[i][j]
1   3   3   1   0
1   4   6   4   1   0
1   5   10  10  5   1   0
```

De getallen uit de driehoek van Pascal zijn de zogeheten binomiaalcoëfficiënten: $A[i][j] = \binom{i}{j}$.

```
void pascaldriehoek ( ) {
  int i, j;
  int Pascal[n][n];  // n een constante
  for ( i = 0; i < n; i++ )
    Pascal[i][0] = 1;  // de nulde kolom bevat enen
  Pascal[0][1] = 0;
  for ( i = 1; i < n; i++ ) {
    cout << endl << Pascal[i][0] << "  ";
    for ( j = 1; j <= i ; j++ ) {
       Pascal[i][j] = Pascal[i-1][j-1] + Pascal[i-1][j];
       cout << Pascal[i][j] << "  ";
    }//for j
    if ( i != n - 1 )
      Pascal[i][i+1] = 0;
  }//for i
}//pascaldriehoek
```

In `pascaldriehoek ( )` werd de *hele* driehoek (tijdelijk) opgeslagen in het 2-dimensionale array `Pascal`: dit kan efficiënter.

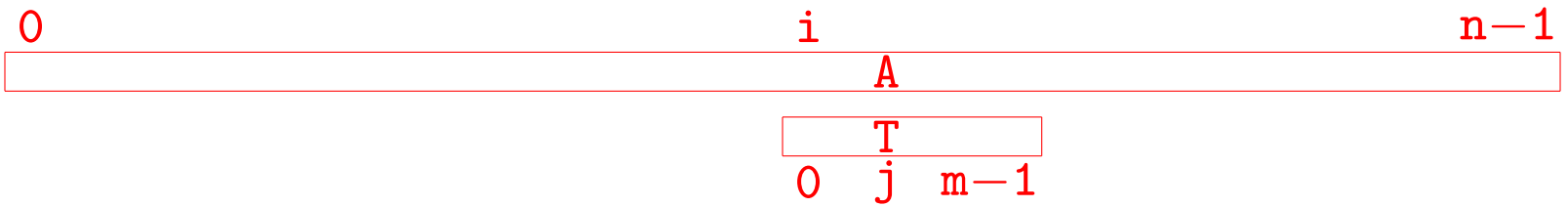
We kunnen volstaan met een 1-dimensionaal array `rij`, dat telkens de `i`-de rij uit de driehoek van Pascal bevat. Om de `(i+1)`-de rij te vinden gebruiken we immers alleen de `i`-de rij, dus alle vorige rijen zijn niet meer nodig.

Merk op dat een volgende rij nu altijd *van rechts naar links* gevuld moet worden, omdat je anders waarden overschrijft die je nog nodig hebt.

```
void pascaldriehoekbeter ( ) {
  int rij[n];   // 1-dimensionaal array !!
  int i, j;
  for ( j = 1; j < n; j++ )  // initialisatie
    rij[j] = 0;
  rij[0] = 1;   // 0-de kolom altijd 1
  for ( i = 1; i < n; i++ ) {   // i-de rij
    for ( j = i; j > 0; j-- ) {
      rij[j] = rij[j-1] + rij[j];   // j-de element
      cout << rij[j] << "  ";
    }//for j
    cout << rij[0] << endl;  // een 1 erachter
  }//for i
}//pascaldriehoekbeter
```

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 ( i + m <= n ) {
    gevonden = true;                      // optimist
    for ( j = 0; j < m; j++ )             // bot
      if ( woord[j] != verhaal[i+j] ) // pech
        gevonden = false;
    if ( gevonden )                       // bingo
      return i;
    i++;
  }//while
  return -1;
}//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`↔`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.

Donald "TEX" Knuth

Gevraagd: rij uit `bord` die geheel uit klinkers bestaat

2D array

```
int rijklinkers (const char bord[8][8]) {
  int i = 0, j = 0, rij = -1;
  bool okee;
  for ( i = 0; i < 8; i++ ) {
    okee = true;   // allemaal klinkers in rij i?
    for ( j = 0; j < 8; j++ ) {
      if ( ! ( ( bord[i][j] == 'A' ) || ( bord[i][j] == 'E' )
               || ( bord[i][j] == 'I' ) || ( bord[i][j] == 'O' )
               || ( bord[i][j] == 'U' ) ) )
        okee = false;   // bord[i][j] is geen klinker
    }//for j
    if ( okee )
      rij = i;
  }//for i
  return rij;   // -1 als zo'n rij niet bestaat
}//rijklinkers
```

Alle 64 array-elementen worden bekeken, 't kan beter ...

Het kan zelfs *efficiënter* en mooier met *één while-loop*:

2D array

```
int klinkerrij (const char bord[ ][8]) {
  int i = 0, j = 0, rij = -1;
  bool stoppen = false;
  while ( ! stoppen && ( i < 8 ) )
    if ( ( bord[i][j] == 'A' ) || ( bord[i][j] == 'E' )
         || ( bord[i][j] == 'I' ) || ( bord[i][j] == 'O' )
         || ( bord[i][j] == 'U' ) )
      if ( j == 7 ) {  // bingo
        stoppen = true;
        rij = i;  // of: return i;
      }//if
      else
        j++;
    else {  // volgende rij vooraan
      i++;
      j = 0;
    }//else
  return rij;   // -1 als zo'n rij niet bestaat
}//klinkerrij
```

## Opgave 43d: plaatje

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | E | E | O | T | | | | |
| 1 | I | X | | | | | | |
| 2 | A | I | O | A | A | A | Z | |
| 3 | U | X | | | | | | |
| 4 | R | | | | | | | |
| 5 | O | I | O | I | A | A | U | A |
| 6 | | | | | | | | |
| 7 | | | | | | | | |

Vraag: begint op positie (i,j) in `puzzel` (met `#` als zwart vakje) een nieuw horizontaal woord?

```
bool bestaatHoriWoord (char puzzel[ ][n], int i, int j) {
  if ( ( puzzel[i][j] != '#' ) &&
       ( ( j == 0 ) || ( puzzel[i][j-1] == '#' ) ) &&
       ( ( j != n-1 ) && ( puzzel[i][j+1] != '#' ) ) )
    return true;
  else                              //   H E T #
    return false;                   //   A # O M
}//bestaatHoriWoord                //   # S P A
```

Er wordt rijkelijk gebruik gemaakt van C++-short-circuiting evaluatie.

Eén statement `return ( ( puzzel[i][j] != ...` kan ook.

Opdracht: vul het array `nummers` (rij voor rij en per rij van links naar rechts): vakjes waar een woord begint (horizontaal of verticaal) krijgen een rangnummer, andere 0:

```
void nummeren (char puzzel[ ][n], int m,
               int nummers[ ][n]) {
  int i, j; int teller = 1;   // rangnummer
  for ( i = 0; i < m; i++ )
    for ( j = 0; j < n; j++ )
      if ( bestaatHoriWoord ( puzzel, i, j ) ||
           bestaatVertiWoord ( puzzel, i, j ) ) {
        nummers[i][j] = teller;
        teller++;            //   H E T #   1 0 2 0
      }//if                   //   A # O M   0 0 3 4
      else                   //   # S P A   0 5 0 0
        nummers[i][j] = 0;
}//nummeren
```

Voorbeeld:

```
const int l = 500;       // 500 bladzijden
const int m = 45;        // 45 regels per bladzijde
const int n = 75;        // 75 karakters per regel
char boek[l][m][n];      // of char boek[500][45][75];
// declareert een driedimensionaal array van karakters
// het heeft 500 * 45 * 75 = 1687500 bytes
```

`boek[342][27][55]` betekent: van pagina 342, regel 27 het 55-e karakter; denk er aan dat ze allemaal bij `0` beginnen met indiceren.
(En gebruik nooit `l` als variabele! (LaTeX: `\ell` $\rightarrow \ell$))

Een priemgetal is een geheel getal $> 1$ dat alleen door 1 en zichzelf deelbaar is:

```
bool priem (int getal) {
  int deler = 2;
  double wortel = sqrt (getal); // uit <cmath>
  bool geendelers = true; // optimist!
  while ( ( deler <= wortel ) && geendelers ) {
    if ( getal % deler == 0 ) // (*)
      geendelers = false;     // (*)
    deler++;
  }//while
  return geendelers;
}//priem
```

Bij (*) mag ook staan: `geendelers = ( getal % deler != 0 );`

Dat kan ook als volgt, waarbij we gebruiken dat `return` de functie meteen stopt:

```
bool priem2 (int getal) {
  int deler = 2;
  double wortel = sqrt (getal); // uit <cmath>

  while ( deler <= wortel ) {
    if ( getal % deler == 0 )
      return false;
    deler++;
  }//while
  return true;
}//priem2
```

Er zijn veel snellere programma's!

Ontbind een getal in factoren, zoals $84 = 2^2 \times 3^1 \times 7^1$:

```
void ontbinden (int getal) {
  int teller = 0;  // hoe vaak past deler in getal?
  int deler = 2;   // kandidaat deler
  while ( getal != 1 ) {
    if ( getal % deler == 0 ) {
      teller = 0;
      while ( getal % deler == 0 ) {
        getal = getal / deler;
        teller++;
      }//while
      cout << deler << " tot de " << teller << "-de macht ";
    }//if
    deler++;
  }//while
}//ontbinden
```

De zeef van Eratosthenes 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
```

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 ( )`, hier 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, mee te geven 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
```

⋈ geen tentamenstof

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.

De array-grootte moet eigenlijk 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 ook wat nadelen heeft, is:

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

⋈ geen tentamenstof

OK, en arrays met rijen van variabele lengte? En dynamische arrays?

```
int** A = new int*[m];
for ( int i = 0; i < m; i++ ) A[i] = new int[n];

... gebruik A[i][j] ...

for ( int i = 0; i < m; i++ ) delete [ ] A[i];
delete [ ] A;
```

⋈ geen tentamenstof

## Opgave 3 van het tentamen van 3 januari 2018:

Gegeven is het 2-dimensionale array `int stemmen[m][n];`, met zekere `const int m` $\geq 2$ en `const int n` $\geq 2$. Er geldt dat `stemmen[i][j]` $\geq 0$ het aantal stemmen van jurylid `i` op liedje `j` is. Een voorbeeld met `m = 4` en `n = 5`:

| | | | | |
|---|---|---|---|---|
| 2 | 5 | 0 | 3 | 2 |
| 1 | 5 | 1 | 4 | 1 |
| 2 | 1 | 3 | 3 | 3 |
| 8 | 1 | 0 | 0 | 3 |

**a**. Schrijf een Booleaanse C++-functie `eerlijk (stemmen)` die controleert of alle juryleden exact evenveel stemmen hebben uitgebracht. Oftewel: hebben alle rijen dezelfde totaalsom? In het voorbeeld: `true`.

**b**. Twee juryleden stemmen *ongeveer hetzelfde* als voor ieder liedje geldt dat hun aantal stemmen hooguit één verschilt, en tevens dat bij minstens één liedje er evenveel stemmen zijn. Schrijf een Booleaanse C++-functie `idem (stemmen,i1,i2)` die dit controleert voor de juryleden `i1` en `i2`. Neem aan dat $0 \leq$ `i1` $<$ `m` en $0 <$ `i2` $<$ `m`. In het voorbeeld `true` bij `0` en `1`.

**c**. Begin bij een zeker jurylid `i1`, met $0 \leq$ `i1` $<$ `m`. Zoek het nieuwe jurylid `i2` dat ongeveer hetzelfde stemt als `i1` (gebruik **b**), waarbij `i2` zo klein mogelijk is. Zoek vanuit `i2` het nieuwe (verschillend van `i1` en `i2`) jurylid `i3` (`i3` zo klein mogelijk) dat ongeveer hetzelfde stemt als `i2`, enzovoorts: `i4` verschilt van `i1`, `i2` en `i3`, en stemt ongeveer hetzelfde als `i3`, ... Je stopt indien je een dergelijk nieuw jurylid niet meer kunt vinden. Schrijf een C++-functie `int vrienden (stemmen,i1)` die dit doet, en teruggeeft hoeveel juryleden je zo gevonden hebt, inclusief `i1`. Tip: gebruik een Booleaans hulparray.

## Opgave 3 van het tentamen van 16 maart 2017:

Gegeven is het twee-dimensionale array `int afstand[n][n];`, met zekere `const int n` $\geq 2$. Er geldt dat `afstand[i][j]` $> 0$ de afstand is tussen de plaatsen `i` en `j` met `i` $\neq$ `j`, waarbij `afstand[i][i]` $0$ is, en de afstand tussen `i` en `j` even groot is als die tussen `j` en `i`. Een voorbeeld met `n = 4` staat hiernaast.

| | | | |
|---|---|---|---|
| 0 | 3 | 7 | 9 |
| 3 | 0 | 4 | 14 |
| 7 | 4 | 0 | 8 |
| 9 | 14 | 8 | 0 |

**a**. Schrijf een C++-functie `bool reis (afstand,km)` die kijkt of er een rondreis van `i` naar `j` naar `k` naar `i` is (voor willekeurige onderling verschillende plaatsen `i`, `j` en `k`) die in totaal precies afstand `km` heeft. In het voorbeeld zou voor 26 het antwoord `true` zijn: van 0 naar 1 naar 3 naar 0 (dat is 3 + 14 + 9 = 26).

**b**. Schrijf een C++-functie `int verste (afstand,i)` die het nummer van de verst van `i` afgelegen plaats oplevert. In het voorbeeld, met `i` = 3, is dat 1 (wegens afstand 14). Als er meerdere plaatsen voldoen: die met de grootste index. Neem aan dat $0 \leq$ `i` $<$ `n`.

**c**. Schrijf een C++-functie `int hoever (afstand,i)` die bepaalt hoeveel je in totaal reist als je begint in `i`, dan steeds naar de verst gelegen plaats gaat, en stopt zodra je ergens komt waar je al eerder geweest was. In het voorbeeld, met `i` = 0, is het antwoord 9 + 14 + 14 = 37 (van 0 naar 3 naar 1 naar 3). Neem aan dat $0 \leq$ `i` $<$ `n`. Hint: gebruik een Booleaans hulparray.

Uitwerking Opgave 3c van tentamen 16 maart 2017:

```
int hoever (int afstand[ ][n], int i) {
  int afst = 0, j;
  bool geweest[n];
  for ( j = 0; j < n; j++ )
    geweest[j] = false;
  while ( ! geweest[i] ) {
    geweest[i] = true;
    j = verste (afstand,i);
    afst += afstand[i][j];
    i = j;
  }//while
  return afst;
}//hoever
```

- werk aan de derde programmeeropgave — de deadline is op maandag 13 november 2023
  bezoek daarom de werkcolleges en vragenuren
- volgende week: pointers!
- lees Savitch Hoofdstuk 5
- lees collegedictaat Hoofdstuk 3.8, 4.1 en 4.2
- maak opgaven 37/43 uit opgavendictaat
- www.liacs.leidenuniv.nl/~kosterswa/pm/