Programmeermethoden

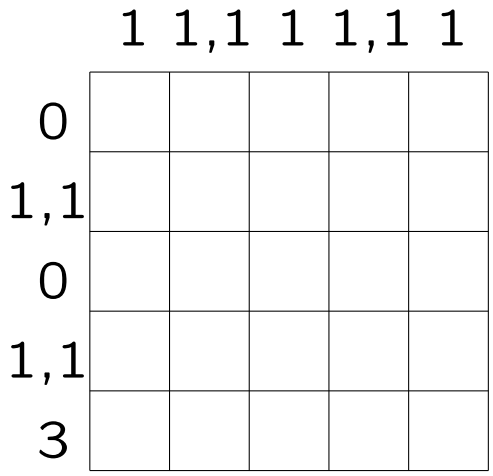
Arrays

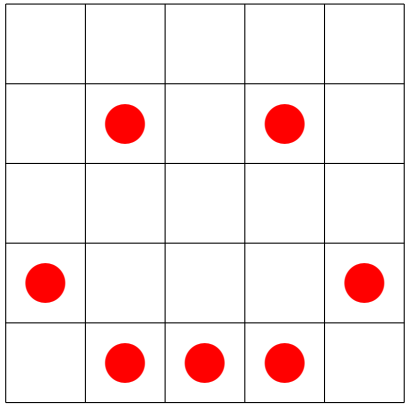
Walter Kosters

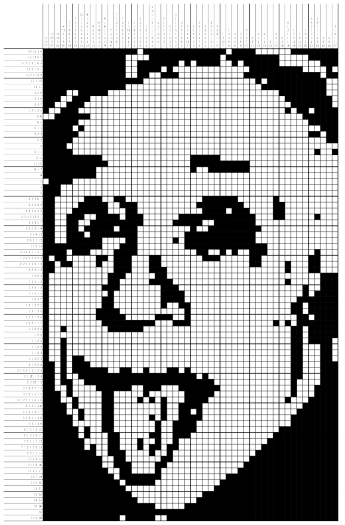
week 7: 15–19 oktober 2018

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

Japanse puzzels (Nonogrammen) zien er zo uit:







Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes = pixels.

Programmeren . . . 2-dimensionale arrays!

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

Derde programmeeropgave — 1

Programmeermethoden

Home Algemeen Materiaal Opdrachten en Cijfers

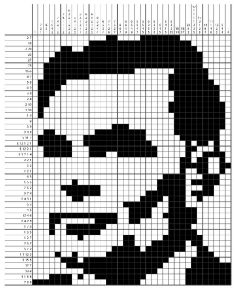
Opdrachten en Cijfers

Programmeermethoden 2018
Derde programmeeropgave: Nonogram

De *derde* programmeeropgave van het vak Programmeermethoden in het najaar van 2018 heet *Nonogram*; zie ook het zevende werkcollege, en lees geregeld deze pagina op WWW.

Spreek/Vragenuur in zalen 302 ... 309: donderdag 18 oktober, maandag 29 oktober, donderdag 1 november, maandag 5 november, donderdag 8 november en maandag 12 november 2018, 15:15 tot 17:00 uur.

De opgave



Het is de bedoeling om een C++-programma te maken dat de gebruiker in staat stelt een Nonogram te maken en op te lossen via een menu-systeem. Dat betekent dat de gebruiker van het programma kan kiezen uit een aantal mogelijkheden, de zogeheten *opties*. Er is één submenu, waarin ook weer opties zijn. De bedoeling is dat het hele menu op één of twee regels staat, onder de puzzel (zie verderop). De opties worden gekozen door de eerste letter van de betreffende optie in te toetsen (gevolgd door Enter), bijvoorbeeld een `s` of `S` om te stoppen. Uiteraard wordt een en ander duidelijk en ondubbelzinnig aan de gebruiker meegedeeld. Gebruik *geen* recursie!
Alle door de gebruiker ingetoetste symbolen moeten gecontroleerd worden, dat wil zeggen dat er binnen redelijke grenzen geen foute invoer geaccepteerd wordt. Zo zal het intoetsen van bijvoorbeeld `q` of `&` in het hoofdmenu genegeerd worden. Verder moet bij getalleninvoer karakter voor karakter ingelezen worden (met `cin.get ( );` als je elders ook nog `cin >> ...` gebruikt krijg je overigens soms problemen met "hangende Enter's"; gebruik dus overal `cin.get ( )`). Er moet ook op gelet worden dat er geen te grote getallen worden ingevoerd. Schrijf dus een geschikte functie `leesgetal` die de gelezen karakters (cijfers) omzet in een getal (tip: negeer alle "voorloop-Enter's"; verwerk alles tot en met de eerstvolgende enter, en maak hiervan zo goed mogelijk een getal, van een maximale grootte; zo kan `abc123defg999h`, als je een getal kleiner dan 10000 wilt, bijvoorbeeld verwerkt worden tot `1239`), en een functie `leesoptie` die netjes één karakter inleest en Enter's afhandelt! Aan de gebruiker mogen "redelijke" beperkingen worden gevraagd, bijvoorbeeld dat de in te voeren getallen maximaal vier cijfers hebben. Het programma moet dan echter wel bestand zijn tegen pogingen meer dan vier cijfers in te voeren. Ook het invoeren van letters in plaats van cijfers moet geen problemen opleveren. Houd het simpel!

Een *Nonogram*, ook bekend als een Japanse puzzel, en niet te verwarren met Sudoku, is een puzzel waarbij een zwart-wit plaatje moet worden gereconstrueerd uit zogeheten beschrijvingen. Het gaat om een rechthoekig *m* bij *n* rooster. Elk vakje = pixel moet zwart (1) of wit (0) worden. Naast alle rijen en boven alle kolommen staat een rijtje gehele getallen: de lijn-beschrijving. Zo'n beschrijving geeft, in volgorde, de lengtes van de aaneengesloten rijtjes zwarte pixels aan. Zo betekent 3 1 dat er eerst nul of meer witte pixels komen, dan 3 zwarte, dan een of meer witte, dan 1 zwarte en tot slot nul of meer witte. De puzzel bestaat uit het vinden van een plaatje dat aan alle beschrijvingen voldoet. Een goede puzzel heeft precies één oplossing. Zie Wikipedia voor meer informatie, of maak zelf een puzzel. We nemen aan dat de hoogte en breedte maximaal 50 zijn. In ons programma hebben we steeds de zogeheten huidige totaal-beschrijving en het huidige beeld. In het begin zijn hier alle lijnbeschrijvingen 0 en alle pixels wit. Verder hebben we steeds het huidige pixel (de "cursor"), in het begin ongeveer in het midden van het rooster. De beschrijvingen staan rechts naast de rijen en onder de kolommen (dit is makkelijker te doen dan er voor en er boven, zoals meer gebruikelijk). Als je wilt kun je nog extra karakters definiëren (een punt bijvoorbeeld) voor pixels die, tijdens het puzzelen, zeker wit moeten zijn.

De gebruiker kan nu een aantal zaken doen:

1. Stoppen.
2. Een klein submenu ingaan, waarin:
   - de grootte van de puzzel kan worden gewijzigd, waarbij beschrijvingen en pixels weer 0 worden;
   - de gebruiker kan instellen of bij verplaatsen van de "cursor" het nieuwe punt wit of zwart wordt, of niet verandert;
   - het random-percentage (zie verderop) kan worden gewijzigd, tussen 0 en 100 procent.
3. Maak het huidige beeld leeg = schoon.
4. Random vullen van het huidige beeld, met (ongeveer) het door de gebruiker gekozen random-percentage zwarte pixels. Gebruik de random-generator van sheet 10.
5. Maak alle beschrijvingen 0.

   ==========
6. De gebruiker kan de "cursor" één positie omhoog, omlaag, naar links of naar rechts bewegen, uiteraard binnen het rooster. Hierna wordt er opnieuw afgebeeld; het plaatje scrollt dus steeds omhoog.
7. Toggelen: het huidige pixel wordt omgeklapt: 0 wordt 1, 1 wordt 0 (of preciezer: `false` wordt `true`, `true` wordt `false`).
8. De beschrijvingen worden de beschrijvingen van het huidige beeld. Het beeld klopt daarna dus precies met de beschrijvingen.

   ==========
9. Inlezen van de huidige beschrijving uit een file. Formaat: de eerste regel bevat hoogte *m* en breedte *n*, gescheiden door een spatie. Daarna komen *m* regels met rij-beschrijvingen en *n* regels met kolom-beschrijvingen. Elke beschrijving wordt afgesloten met een 0; zo wordt 3 1 in de file `3 1 0` (met een spatie tussen de getallen). De beschrijving 0 wordt als 0 gerepresenteerd. Aangenomen mag worden dat de files wel het goede formaat hebben. Zie hier een (lastig) voorbeeld en nog een en nog een.
   Tip: lees getallen gewoon in met `invoer >> getal;`.
10. Wegschrijven van de huidige beschrijving naar een file.
11. [OPTIONEEL] Inlezen van het huidige beeld uit een file. Formaat: de eerste regel bevat hoogte *m* en breedte *n*, gescheiden door een spatie. Daarna *m* regels met een rij van het plaatje, met een 1 voor zwart en een 0 voor wit. Zie hier een voorbeeld.
    Wil je van een willekeurig plaatje `plaatje.jpg` een bestand in (bijna) dit formaat maken, doe dan het volgende (in Linux):
    ```
    convert plaatje.jpg plaatje.pbm
    pnmtoplainpnm plaatje.pbm > invoer.pbm
    ```
    Uiteraard kan de gebruiker de filenaam kiezen. Als deze niet bestaat: een foutmelding, maar het programma komt weer in het menu terecht. Aangenomen mag worden dat de files wel het goede formaat hebben.
12. [OPTIONEEL] Wegschrijven van het huidige beeld.

Steeds staat bij correcte lijnen een "V": het gaat dus om rijen en kolommen waarbij het huidige beeld precies aan de huidige beschrijving voldoet; dit verandert wellicht als de gebruiker iets aan het huidige beeld wijzigt. Gebruik voor zwarte pixels in het huidige beeld een X, voor witte een spatie, en speciale symbolen voor de "cursor", die op een wit of zwart pixel staat.
Als er in de beschrijvingen getallen met twee of meer cijfers staan, is dit lastig bij de kolommen. Druk dan bijvoorbeeld 10 als A af, 11 als B, enzovoorts. Dat geeft ook wel problemen, overigens.

De bedoeling is een klasse (`class`) `nonogram` te maken, met daarin onder meer functies die ieder voor zich een menuoptie afhandelen, zoals `vulrandom`. De parameters zijn typisch membervariabelen. Gebruik nog geen eigen headerfiles, alles moet deze keer in één file staan.

Opmerkingen
Gebruik geschikte (member)functies. Bij deze opgave mogen bij elke functie (zelfs `main`) tussen begin-{ en eind-} *hooguit circa 30* niet al te volle regels staan! Elke functie dient van commentaar voorzien te zijn, bij voorkeur één regel boven de functie. Let op goed parametergebruik: alle parameters, met uitzondering van membervariabelen, in de heading doorgeven, en de variabele-declaraties zowel bij `main` als bij de andere functies aan het begin. De enige te gebruiken headerfiles zijn in principe `iostream`, `fstream`, `cstdlib` en `string`. Zeer ruwe indicatie voor de lengte van het C++-programma: 500 regels. Denk aan het infoblokje.

Uiterste inleverdatum: maandag 12 november 2018, 17:00 uur.

Manier van inleveren:

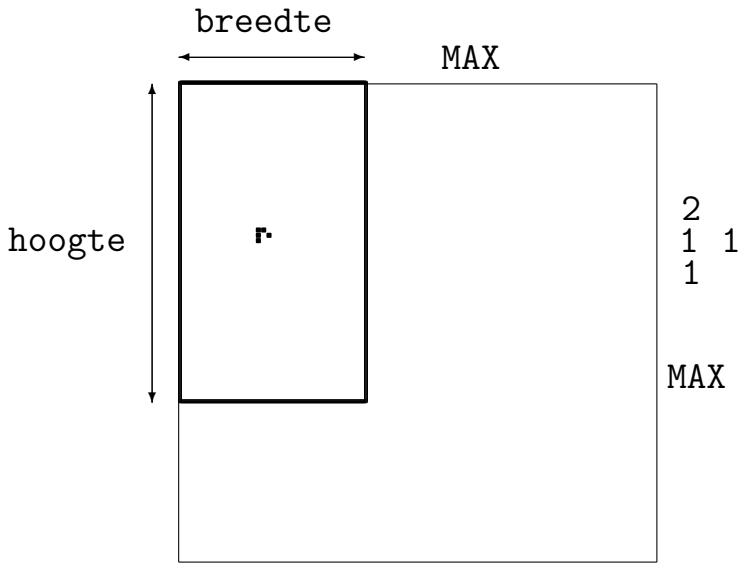
1. Digitaal de C++-code inleveren: stuur een email naar `pm@liacs.leidenuniv.nl`.
   Stuur geen executable's, lever alleen de C++-file digitaal in! Noem deze bij voorkeur zoiets als `jansentilanus3.cc`, dit voor de derde opdracht van het duo Jansen-Tilanus. De laatst voor de deadline ingeleverde versie wordt nagekeken.
2. En *ook* een papieren versie van het verslag (inclusief de C++-code) deponeren in de speciaal daarvoor bestemde doos "Programmeermethoden" bij kamer 159 van het Snellius-gebouw.
   Overal duidelijk datum en namen van de (maximaal twee) makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code.
   Het *verslag* (uiteraard weer in LaTeX, zie de eerdere opgaven) moet het volgende bevatten: een korte beschrijving van het programma (inclusief definitie van Nonogram), een plaatje van het programma in werking, een beschrijving van punten waarop het programma faalt (indien van toepassing), en een tabel met gewerkte uren, uitgesplitst per week en per persoon. En iets over mogelijke oplossingsmethoden.

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

Derde programmeeropgave — 2

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

```
class nonogram {
  public:
    nonogram ( );  // constructor
    void drukaf ( );
    void vulrandom ( );
    void maakschoon ( );
    void zetpercentage ( );
    // ...
  private:
    bool nono[MAX][MAX];
    int beshor[MAX][MAX];
    int besvert[MAX][MAX];
    int hoogte, breedte;
    int percentage;
    // ...
};//nonogram
```




```
nonogram N;
N.drukaf ( );
```

Maak member-functies als:

```
//laat nonogram zien
void nonogram::drukaf ( );
  ...
}//nonogram::drukaf
```

en

```
//stel percentage in tussen 0 en 100
void nonogram::zetpercentage ( ) {
  percentage = leesGetal (100);
}//nonogram::zetpercentage
```

waarbij de zelfgemaakte functie `int leesgetal (int maxi)` een geheel getal, maximaal `maxi`, van toetsenbord inleest.

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 2-dimensionale arrays: *matrices* (Life, Nonogram).

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:

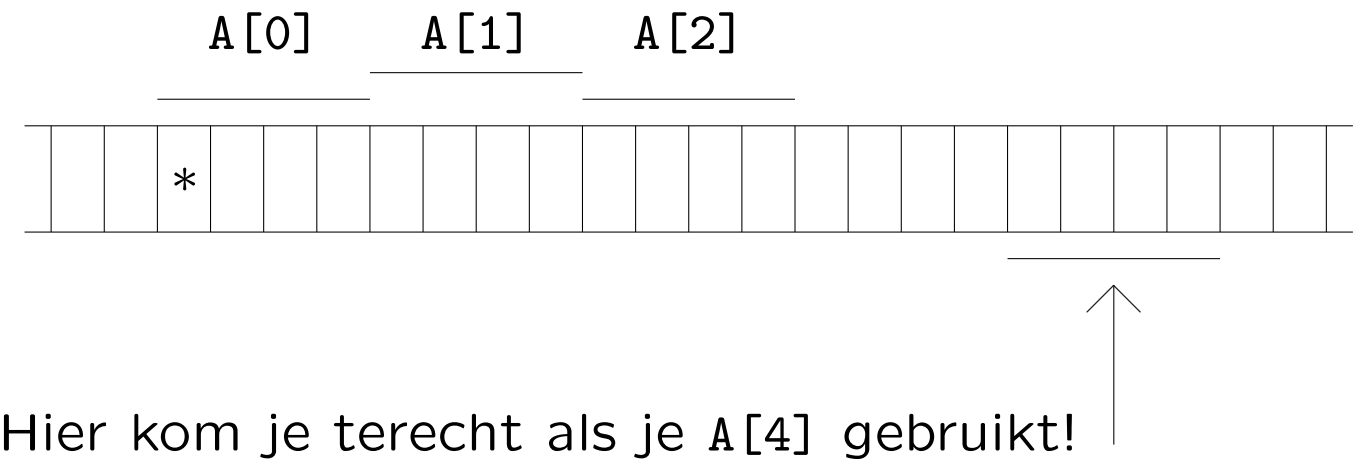
```
int rij[MAX];
```

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 $*$.

Met `int rij[MAX];` maken we een array `rij` met `MAX` elementen dat we bijvoorbeeld als volgt gebruiken:

```
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   1   2   3   4   5   6   7   8   9   <--- array-index
 0   5  10  15  20  25  30  35  40  45   <--- array-inhoud
 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:

YouTube

```
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

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

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!

http://www.sorting-algorithms.com/

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);`!

Opgave 42

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

Voor een Nonogram is een 2-dimensionaal array nodig:

```
bool nono[MAX][MAX];
```

Er geldt: `nono[i][j]` is `true` precies dan als in rij `i` (van boven) en kolom `j` (van links) een zwart pixel zit.
En dit allemaal in een klasse `nonogram`, met methoden als `void nonogram::drukaf ( )` (zie eerder).

Maak eerst een *menu* en de functie `leesgetal`. Zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc7.php

- werk aan de derde programmeeropgave Nonogram (menu!); deadline: maandag 12 november 2018
  www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

- lees Savitch Hoofdstuk 5, en 9 voor strings

- lees dictaat Hoofdstuk 3.8

- maak opgaven 31/36 uit het opgavendictaat

- volgende week geen (werk)college; eerstvolgend college donderdag 1 november, in Gorlaeus zaal 2; maandag-mensen: begin snel aan de opgave!