Programmeermethoden

Object-geOriënteerd Programmeren & arrays

Walter Kosters en Jonathan Vis

week 7: 16–20 oktober 2023

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

```
string filenaam;  // gebruik <string>
ifstream invoer;  // gebruik <fstream>
...
cin >> filenaam;
invoer.open (filenaam.c_str ( )); // (*)
if ( invoer.fail ( ) ) {
  cout << filenaam << " niet te openen" << endl;
  return 1;  // of exit (1);
}//if
```

In bovenstaand programma maken we een object `filenaam` van klasse `string` (voor de naam van de file) en een object `invoer` van klasse `ifstream` (voor de file). In regel `(*)` koppelen we ze, door de methode `open` te gebruiken. Vanaf C++11 mag hier ook `invoer.open (filenaam);` staan.

C++ is — in tegenstelling tot C — een object-georiënteerde (OO) programmeertaal, net als Java.

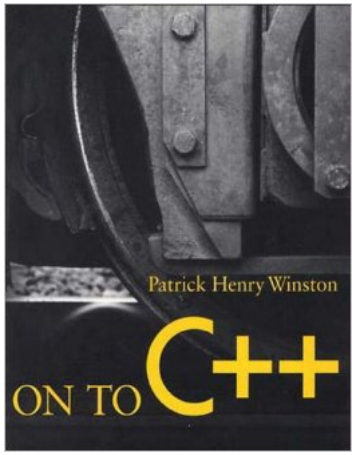
In een OO-programma hebt je objecten (zoals Adam, Eva; Bonzo) van verschillende klassen (Mens; Hond). Klassen hebben hun eigen methoden (praten, slapen, blaffen).

Een voorbeeld: de klasse `double`. Met `double x, y;` maak je twee objecten (variabelen) van deze klasse. En `cout << x;` vraagt `x` zich af te laten drukken. En `x = x - y;` vraagt `x` zich met de waarde van `y` af te lagen.

Denk ook aan files met methodes (member-functies) als `get`, `put`, `eof`, `open` en `close`.

Een eerste klasse — 1

```
class wagon {
  public:
    double hoogte, breedte, lengte;
    double inhoud ( ) {  // member-functie
      return hoogte * breedte * lengte;
    }//inhoud
};//wagon; let op de ;
...
wagon bert;
bert.hoogte = 3.5;
bert.breedte = 4.0;
bert.lengte = 20.5;
cout << "Inhoud: " << bert.inhoud ( ) << endl;
```



Hier is `bert` een object van klasse (= type) `wagon`.

Een eerste klasse — 2

Een object `ernie` van klasse `wagon` bestaat uit drie double's, die zijn hoogte, breedte en lengte aanduiden.

En je kunt (via de methode `inhoud`) om zijn inhoud vragen. Deze functies worden eenmalig opgeslagen, niet in ieder object opnieuw.

Let op de punt-notatie: `ernie.breedte`. En voor functies (methoden) `ernie.inhoud ( )`.

Tevens kan bestaan (overloading van `inhoud` en `lengte`):

```
class tanker {
  public:
    double straal, lengte;  // zelfde namen
    double inhoud ( ) {     // als zo-even!
      return straal * straal * lengte;
    }//inhoud
};//tanker
```

```
class wagon {
  public:
    double hoogte, breedte, lengte;
    double belasting (double);
      // functie-prototype van deze methode
};//wagon
double wagon::belasting (double percentage) {
  return percentage * breedte * lengte;
}//wagon::belasting
```

Hierbij is `::` de binary scope resolution operator.
Met `ernie` een object van klasse `wagon` (dus `wagon ernie;`):

```
cout << "Belasting: "
     << ernie.belasting (0.5) << endl;
```

Het benutten van de (member-)variabelen van een object gaat meestal met speciaal geschreven functies: reader (*getter*, *accessor*) en writer (*setter*, *mutator*) methodes.

```
class tanker { ... als vroeger ...
  double geefstraal ( ) {  // reader
    return straal;
  }//geefstraal
};//tanker
```

Gebruik nu `zeppo.geefstraal ( )` in plaats van `zeppo.straal` (met `zeppo` van type `tanker`). Analoog writer's.
Een uitbreiding voor `tanker`:

```
double tanker::geefdiameter ( ) {  // reader
  return 2.0 * straal;
}//geefdiameter
```

Met behulp van reader's en writer's kun je (member-)variabelen van een object afschermen/verbergen:

```
class tanker {
  private:
    double straal, lengte;
  public:
    double geefstraal ( ) {  // reader
      return straal;
    }//geefstraal
    void maaklang (double t) {  // writer
      lengte = t;
    }//maaklang
};//tanker
```

Nu mag `chico.straal` zelfs niet meer gebruikt worden; het *moet* via `chico.geefstraal ( )` (met `chico` van type `tanker`). En je *moet* nu `chico.maaklang (42.1);` doen in plaats van `chico.lengte = 42.1;`.

```
class tanker {
  public:
    double straal, lengte;
    tanker ( ) {
      straal = 1.0; lengte = 37.0;
    }//default constructor
    tanker (double s, double t) {
      straal = s; lengte = t;
    }//constructor
};//tanker
```

Als je nu een nieuwe variabele maakt van klasse `tanker` kun je die meteen initialiseren:

```
tanker harpo;                  // met default constructor
tanker groucho (7.0,12.12); // met andere constructor
```

Een constructor wordt "nooit" direct aangeroepen, maar automatisch gebruikt bij het ontstaan van objecten.

De klasse `personenwagon` wordt afgeleid (= derived) van de ouder (= superklasse) `wagon`:

```
class personenwagon : public wagon {
  // we erven "alles" van wagon
  private:
    int passagiers;
  public:
    // default constructor:
    personenwagon ( ) { passagiers = 0; }
    personenwagon (int aantal) {
      passagiers = aantal;
    }//constructor
    int hoeveel ( ) { // reader
      return passagiers;
    }//hoeveel
};//personenwagon
```

Ook multiple inheritance/overerving is mogelijk:

```
class gehakt : public dier, eten { ... };
```

Stel we hebben een klasse `voertuig`, met variabelen `gewicht` en `maxsnelheid`, en een methode `belasting ( )`. Er zijn afgeleide klassen `fiets` (met eigen methode `belasting ( )`) en `auto` (met een extra variabele `soort`).

Met `rijwiel` van type `fiets` mag je gebruik maken van `rijwiel.belasting ( )`. Je krijgt dan de belasting speciaal voor een fiets. Als je toch de belasting als voor een voertuig wilt laten berekenen: `rijwiel.voertuig::belasting ( )`.

Als je de constructor voor `fiets` "aanroept", wordt automatisch eerst die voor `voertuig` uitgevoerd.

Stel we willen met gehele getallen van "willekeurige" lengte werken, zoals 1234567891011121314151617181920. Grote getallen dus. We maken daartoe een klasse `gg` met methoden als `drukaf ( )`, `maak (int m)`, `kopie (gg & getal)` en `telop (gg & getal)`.

Je kunt dan een programma schrijven als

```
gg x; gg y;                                  // int x; int y;
x.maak (1); y.kopie (x);                     // x = 1; y = x;
for ( int i = 1; i <= 1000; i++ ) {
  x.telop (y);                               // x = x + y;
  y.kopie (x);                               // y = x;
  x.drukaf ( );                              // cout << x;
}//for
```

Dit berekent uiteindelijk 2^{1000} (het kan anders en beter).

- polymorfisme en late binding
- kopiëren van objecten (“diepe kopie”)
- destructoren
- private, protected, public
- operatoren bijdefiniëren
- this-pointer: `wagon* p = this;`

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] A[9]
```

Er zijn ook 2-dimensionale arrays: *matrices* (Life! Nonogram! LightsOut!).

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:

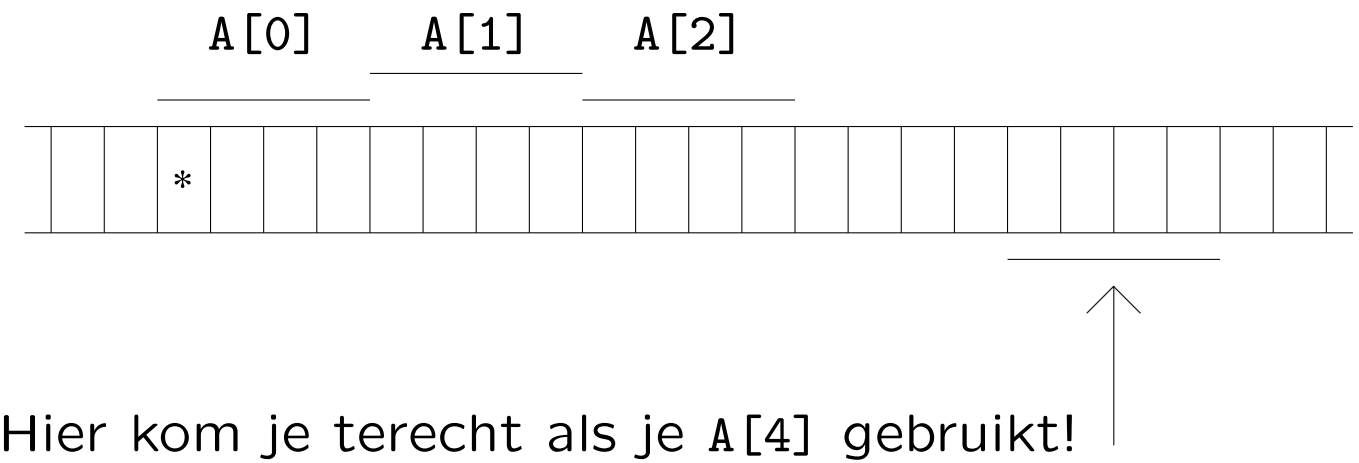
```
int rij[MAX];
```

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 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)

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

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

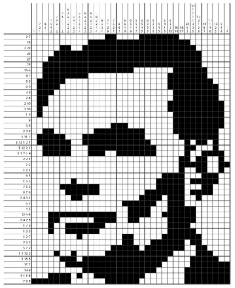
Dus *nooit* `drukaf (rij[ ],8);`!

Derde programmeeropgave — 1

Programmeermethoden 2023
Derde programmeeropgave: Nonogram

De *derde* programmeeropgave van het vak Programmeermethoden in het najaar van 2023 heet *Nonogram*; zie ook het zevende werkcollege, achtste werkcollege en negende werkcollege, en lees geregeld deze pagina op WWW.

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 de menukeuzes op één of twee regels staan, 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 (tenzij er een optie Q is ...).
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* (rijen) bij *n* (kolommen) rooster. Elk vakje = pixel moet zwart (1/true) of wit (0/false) 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 één 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. 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 zien we steeds het "huidige pixel", ook wel de "cursor" genoemd, in het begin ongeveer in het midden van het rooster. (Zorg ervoor dat op de een of andere manier deze locatie duidelijk gemarkeerd is.) 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 de symbolen die voor witte en zwarte pixels gebruikt worden kan kiezen (uiteraard twee verschillende);
   - 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 van het achtste college.
   ==========
5. 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. Gebruik speciale symbolen voor de plek van de cursor, op zowel een wit als een zwart pixel.
6. Toggelen: het huidige pixel wordt omgeklapt: 0 wordt 1, 1 wordt 0 (of preciezer: `false` wordt `true`, `true` wordt `false`).
7. De beschrijvingen worden de beschrijvingen van het huidige beeld. Het beeld klopt dan dus precies met de beschrijvingen.
   ==========
8. 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;`. Controleer ook of de file bestaat.
9. Wegschrijven van de huidige beschrijving naar een file.
10. [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 -resize 40x40 plaatje.pbm
    pnmtoplainpnm plaatje.pbm > invoer.txt
    ```
    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.
11. [OPTIONEEL] Wegschrijven van het huidige beeld.

Steeds staat bij correcte lijnen een "V": het gaat dus om rijen (ernaast) en kolommen (eronder) waarbij het huidige beeld precies aan de huidige beschrijving voldoet; dit verandert wellicht als de gebruiker iets aan het huidige beeld wijzigt. Als dit onderdeel geheel ontbreekt, kost dat 1 punt.

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.

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.

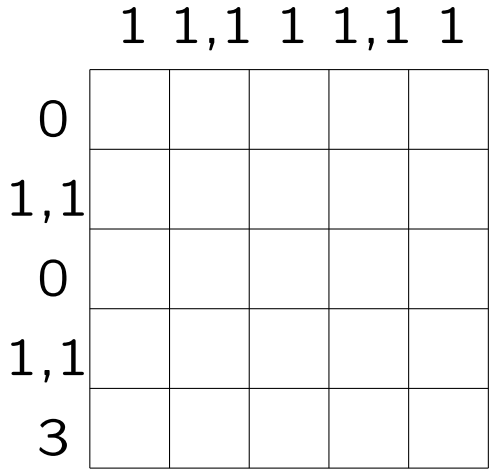
Opmerkingen
Gebruik geschikte (member)functies. Bij deze opgave mogen bij elke functie (zelfs `main`) tussen begin-{ en eind-} *hooguit circa 30* niet al te volle regels staan! Elke functie dient van commentaar voorzien te zijn, bij voorkeur één regel boven de functie. Let op goed parametergebruik: alle 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.

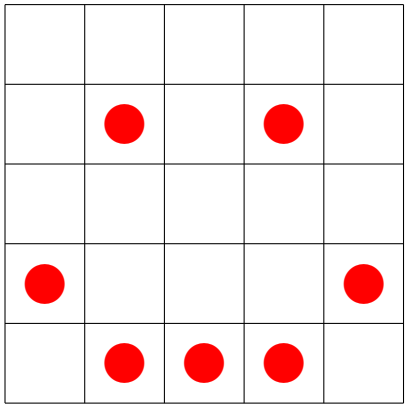
Uiterste inleverdatum: maandag 13 november 2023, 18:00 uur.

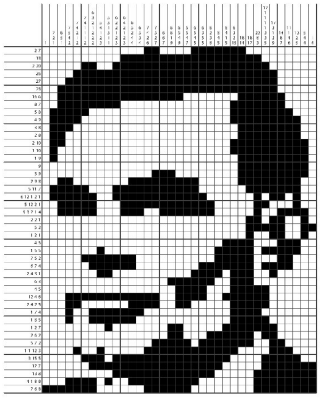
Manier van inleveren (één exemplaar per koppel, dat — ter herinnering — uit maximaal twee personen bestaat) is als volgt:

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

Japanse puzzels (Nonogrammen) zien er zo uit:







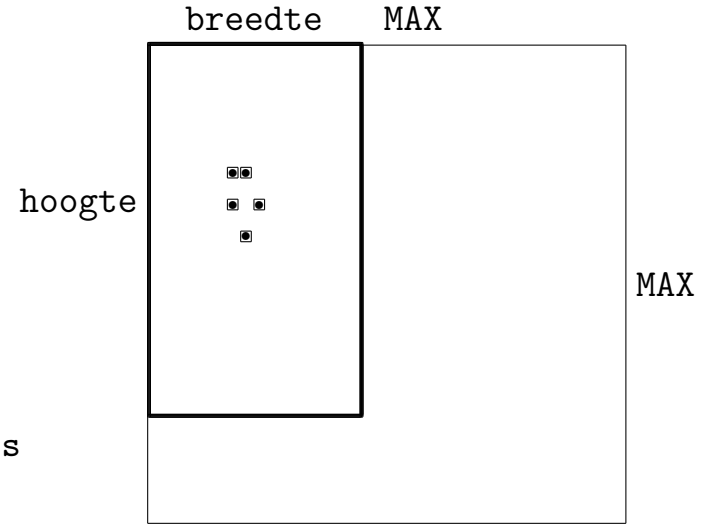
Naast iedere rij en boven (→ onder) iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes.

Voor Life/Nonogram/LightsOut: 2-dimensionale arrays (matrices)!

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

Een klasse `nonogram` voor Nonogram ziet er $\pm$ 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)
```



klasse object

↓ ↓

```
nonogram N;

N.drukaf ( );
```

Maak member-functies als (zie verderop):

```
// laat het 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.

Voor een Nonogram-wereld is een 2-dimensionaal array nodig:

```
bool dewereld[MAX][MAX];
```

Er geldt: `dewereld[i][j]` is `true` precies dan als rij `i` (van boven) en kolom `j` (van links) "zwart" is.
En voor de rij-beschrijvingen:

```
int rijen[MAX][MAX];
```

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

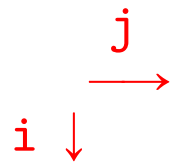
invoer moet $<$ 10000 zijn (bijvoorbeeld):

`\n\nabc123$%@@rr45xx\n` $\longrightarrow$ 1234 OF ...

Enter

Een basisfunctie is dus het afdrukken van een Nonogram-wereld:

```
// laat de nonogram-wereld zien; eerste poging
void nonogram::drukaf ( );
  int i, j; // voor rijen en kolommen
  for ( i = 0; i < hoogte; i++ ) {
    for ( j = 0; j < breedte; j++ ) {
      if ( dewereld[i][j] )
        cout << " X"; // <== later "zwartkarakter"
      else
        cout << " .";
    }//for j
    cout << endl;
  }//for i
}//nonogram::drukaf
```



En is linksboven $(0, 0)$?

- werk aan de derde programmeeropgave (menu!) — de deadline is op maandag 13 november 2023, 18:00 uur
  www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

- lees Savitch Hoofdstuk 5, 6 en 7.1

- lees dictaat Hoofdstuk 3.8 en 3.11

- maak opgaven 26/30 uit het opgavendictaat

- volgende week geen "reguliere" actviteiten bij Programmeermethoden