Programmeermethoden

Object-geOriënteerd Programmeren & arrays

Walter Kosters en Jonathan Vis

week 7: 17–21 oktober 2022

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. Sinds C++11 mag hier ook `invoer.open (filenaam);` staan.

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

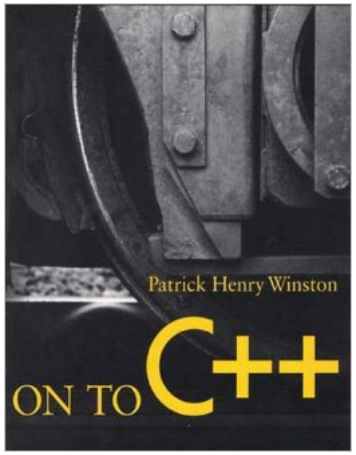
In een OO-programma hebt je objecten (zoals Adam, Eva; Bonzo) van verschillende klassen (Mens; Hond). Klassen hebben hun eigen methoden (praten, slapen, blaffen).

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]` en `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:

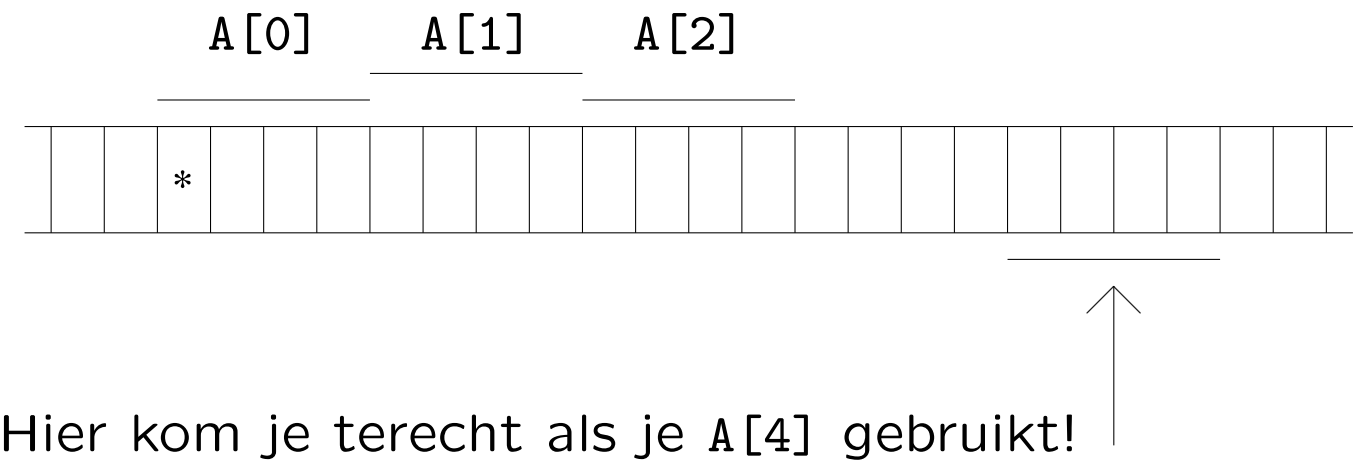
```
int rij[MAX];
```

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 2022
Derde programmeeropgave: LightsOut

De *derde* programmeeropgave van het vak Programmeermethoden in het najaar van 2022 heet *LightsOut*; zie ook het zevende 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 *LightsOut* te spelen via een menu-systeem. Dat betekent dat de gebruiker van het programma kan kiezen uit een aantal mogelijkheden, de zogeheten *opties*. Er zijn drie submenus, waarin ook weer enkele opties zijn. De bedoeling is dat het hele menu steeds op één regel 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 `X` 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 42 is, Het programma moet dan echter wel bestand zijn tegen pogingen grotere getallen in te voeren. Ook het invoeren van letters in plaats van cijfers moet geen problemen opleveren. Houd het simpel!
Maak ook een functie `leesoptie` die de eerste niet-Enter ophaalt. De functie geeft een `char` terug.

Het spel *LightsOut* heeft simpele spelregels, zie bijvoorbeeld Wikipedia, en de referenties daar. In een 2-dimensionaal rechthoekig rooster, (zeg) 4 bij 6, de *puzzel*, bevindt zich in ieder vakje een lamp die aan of uit is. Doel is ze allemaal uit krijgen. Als je een vakje kiest, klapt het lampje daar om tussen aan en uit, evenals de lampjes in de direct horizontaal en verticaal aangrenzende buren. Maximale grootte is 20 bij 20.

In het hoofdmenu kun je kiezen tussen de volgende opties [P]arameters, p[U]zzelmenu en [T]ekenen, die alle drie naar een submenu leiden. En uiteraard [S]toppen.
Het pUzzelmenu heeft de volgende opties:

1. Terug naar het hoofdmenu.
2. Volg. De lampen in de puzzel (geen torus) worden rij voor rij van links naar rechts en van boven af uitgedaan (behalve de onderste), door automatisch "onder" de lampen die aan zijn te selecteren. De tussenstappen komen alle in beeld.
3. Los een 5 bij 5 puzzel (geen torus) op — als dat kan. Na de optie Volg gebruikt te hebben, moet alleen de onderste rij nog worden opgelost. Zie bijvoorbeeld Wikipedia voor aanwijzingen op een 5 bij 5 bord. Onderscheid verschillende gevallen, en doe daarna opnieuw Volg.
4. Speel de "oplossing" af.
5. Doe een zet, na eerst een vakje geselecteerd te hebben. Gebruik hierbij de schaakbord-notatie: hoofdletters A, B, ... voor kolommen en getallen 1,2 ... voor rijen. Linksonder zit A1. Zetten worden geteld, de teller staat steeds naast het bord afgedrukt. Doe deze zet ook in de "oplossing".

Het Tekenmenu heeft de volgende opties:

1. Terug naar het hoofdmenu.
2. Schoon. Maak de puzzel leeg (alle lampen gaan uit). rest ook de oplossing.
3. Random. Maak de puzzel eerst schoon. Zet daarna random lampen aan. Gebruik een zelfgemaakte random-generator (nou ja, random), zie Hoofdstuk 3.9.3 uit het dictaat, gedeelte "aantekeningen bij de hoorcolleges".
4. Toggle. Klapt de lamp op de "cursorpositie" om: van aan naar uit, of juist omgekeerd. (Je doet dus niet de zet, zoals in het pUzzelmenu.) Deze positie kan met vier toetsen omhoog/omlaag/naar links/rechts (bijvoorbeeld W/A/S/Z) gewijzigd worden.
5. Genereer. Maak een puzzel met een gegeven "moeilijkheidsgraad", zeg *g*. Doe dit door random *g* verschillende lampen te selecteren (met de optie uit het pUzzelmenu), te beginnen met een situatie waar alle lampen uit zijn. Onthoud de gekozen lampen in de "oplossing" (in een apart Booleaans 2-dimensionaal array), zodat deze later gespeeld kan worden.

Er zijn verschillende parameters, in te stellen via het gelijknamige submenu:

1. De *hoogte* en *breedte* van het bord, beide maximaal 20.
2. Het *percentage* lampen dat aan moet gaan bij de optie Random (bij benadering).
3. De *twee verschillende karakters* aan/uit die op het scherm gebruikt worden voor de lampen.
4. Is het (*j*/*n*) een *torus* = fietsband of een gewone rechthoek met randen?
5. Een *pen* (0/1/2) die aangeeft of je bij het lopen ook nog meteen lampen aan doet (1) of juist uit (2).

Kies zelf verstandige grenswaarden voor deze parameters, en gebruik `leesgetal` of `leesoptie`.

De bedoeling is een klasse (`class`) `puzzel` te maken, met daarin onder meer functies die ieder voor zich een menuoptie afhandelen. 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! Vooruit, bij maximaal drie functie (zoals afdrukken of 5 bij 5 oplossen) mogen het er wat meer zijn. 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 14 november 2022, 17:00 uur.

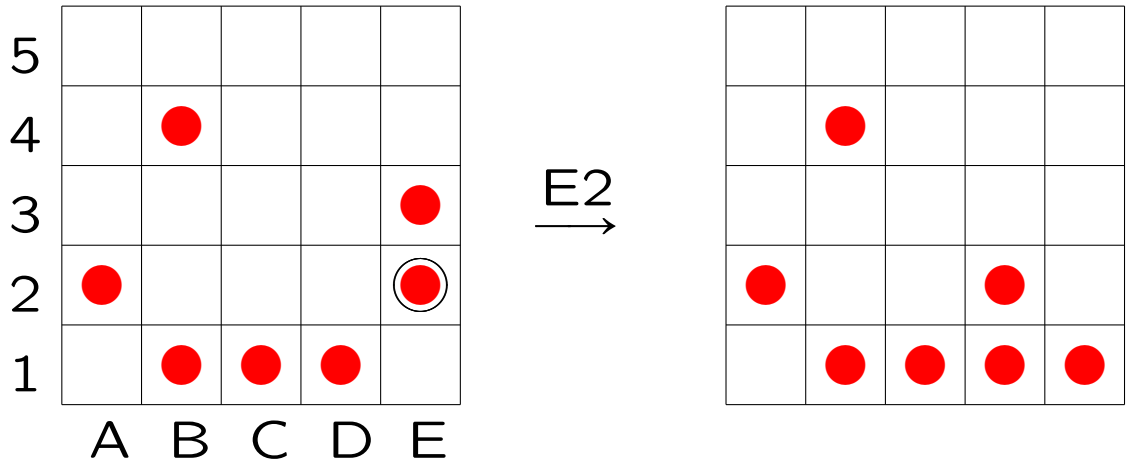
Manier van inleveren:

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 beschrijving van het programma (waarin LightsOut kort maar duidelijk wordt uitgelegd), een plaatje van een niet al te zwart screenshot (in Linux: Shift-PrintScreen) van het eigen programma, een beschrijving van punten waarop het programma faalt (indien van toepassing), en een tabel met gewerkte uren, uitgesplitst per week en per persoon. Geef ook minstens twee citaties = referenties (in LaTeX: "`\cite`"), één bij de uitleg over het spel, en bijvoorbeeld één naar een wiskunde-artikel over het spel. Zie hier hoe je een plaatje verwerkt, en hoe een citatie = referentie gemaakt wordt (en zo ziet dat eruit).

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met g++) als onder Code::Blocks draaien. Test dus in principe op beide systemen! Het programma wordt doorgaans nagekeken met behulp van de compiler die (uiteraard) in het commentaar bovenin het programma vermeld staat. Normering: layout 1; commentaar (inclusief verslag) 2; modulariteit (OOP, functies) 3; werking

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

Bij LightsOut moet je alle lampjes uit doen:



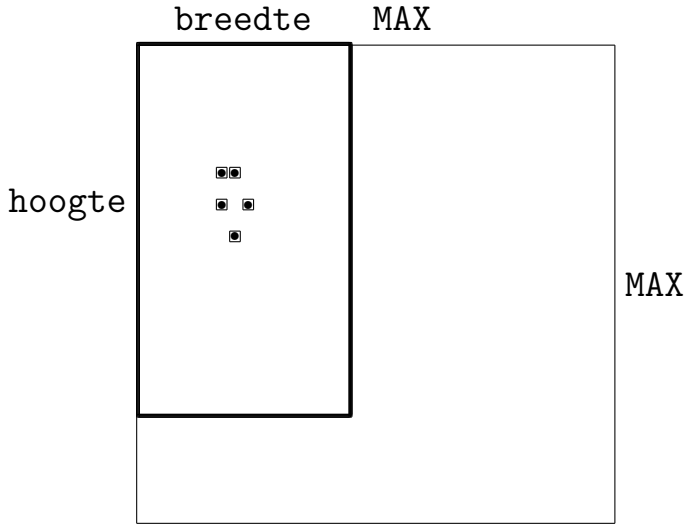
Als je een lamp selecteert, klappen die en de direct aangrenzende horizontale en verticale buren om (aan ↔ uit).

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

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

Een klasse `puzzel` voor LightsOut ziet er $\pm$ zo uit:

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



klasse object

↓ ↓

```
puzzel P;

P.drukaf ( );
```

Maak member-functies als (zie verderop):

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

en

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

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

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

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

Er geldt: `dewereld[i][j]` is `true` precies dan als de lamp in rij `i` (van boven) en kolom `j` (van links) aan is.
En dit allemaal in een klasse `puzzel`, met methoden als `void puzzel::drukaf ( )`, zie eerder.

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

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

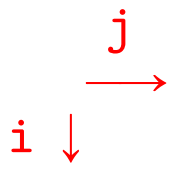
invoer moet $<$ 10000 zijn (bijvoorbeeld):

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

↖ Enter

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

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



En (0,0) is A8?

- werk aan de derde programmeeropgave (menu!) — de deadline is op maandag 14 november 2022, 17:00 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