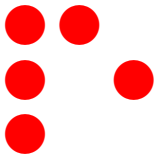
Programmeermethoden



Functies & Life

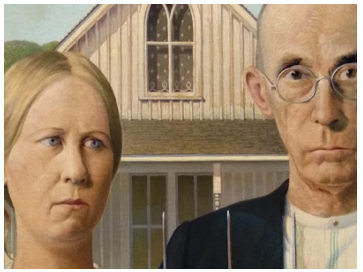
Walter Kosters en Jonathan Vis

week 6: 7–11 oktober 2024

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

Voor de tweede programmeeropgave moet je een C++-programma schrijven dat een gegeven file netjes ingesprongen afdrukt, en daarbij //-commentaar verwijdert:

```
␣␣if␣(␣leeftijd␣␣>␣12560/196␣)␣␣{{␣//␣commentaar!
cout␣<<␣"Oud";//␣>64␣of␣niet?
```



moet worden:

```
if␣(␣leeftijd␣␣>␣12560/196␣)␣␣{{␣
␣␣␣␣␣␣cout␣<<␣"Oud";
```

Hier is ␣ een spatie. En "tab = 3".

Is $196 \to 196 + 691 = 887 \to \ldots$ een Lychrel-getal?
Hoe vaak komt `mag` voor?

www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php

Tweede programmeeropgave

1. commentaar verwijderen; gebruik zo weinig mogelijk `put`'s en `get`'s (en geen `peek`)
2. inspringen
3. en het woordje `het` herkennen?
4. daarna, of juist eerder, de Lychrel-controle (`INT_MAX`!)
5. en tot slot details, . . . , en het verslag

video

↙ $\leq \approx$ 250 regels

Houd het kort! Gebruik geschikte functies; zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc4.php
www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc5.php
www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc6.php

Stel dat iemand karakters (`char`'s, waaronder cijfers) op je afstuurt, en je daar een getal van moet maken. Hoe doe je dat?

Gebruik `int getal = 0;`, en herhaal:

```
if ( '0' <= kar && kar <= '9' )
  getal = 10 * getal + ( kar - '0' );
else
  ...
```

`qwerty7392abc  de---12fghijklmnopq`

↑

`getal` is 73 en `kar` is '9'

`getal` wordt 739

Deze `int`-functie telt het aantal keer dat een 'd' direct door een 'e' gevolgd wordt in een al geopende file `invoer`:

```
int telDE (ifstream & invoer) {
  char prevkar = '\n', kar = invoer.get ( );
  int tel = 0; // locaal tellertje
  while ( ! invoer.eof ( ) ) {
    if ( prevkar == 'd' && kar == 'e' )
      tel++;
    prevkar = kar;
    kar = invoer.get ( );
  }//while
  return tel;                    // <======= int functie!
}//telDE
```

Deze `void`-functie vertaalt file `invoer` naar file `uitvoer`:

```
void manipuleer (ifstream & invoer, ofstream & uitvoer) {
  char kar = invoer.get ( );
  while ( ! invoer.eof ( ) ) {
    if ( ... )                  // GEEN get's,
      ...                       // GEEN while's
    if ( ... )                  // GEEN strings
      ...
    uitvoer.put (kar);          // EEN MAAL put
    kar = invoer.get ( );       // "EEN" MAAL get
  }//while
}//manipuleer
```

- weer in LaTeX, gebruik mooi.tex
- kleine voorbeeldfile om werking te illustreren
- iets over Lychrel-getallen (22?), en enkele vragen beantwoorden
- urentabel

week	1	2	3	totaal
Dilan	6	4	6	16
Mona	6	3	5	14
totaal	12	7	11	30

- zie www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc6.php

```
void john (int x, int y) { ... }//john

int paul (double x, bool b) { ... }//paul

void george ( ) { ... }//george

bool ringo (int & getal) { ... }//ringo

int main ( ) { ... }//main
```

De functie `george` mag de functies `george` ("recursie"), `john` en `paul` gebruiken = aanroepen, maar *niet* de functie `ringo`! Wil je dat toch, dan moet je boven `george` een prototype `bool ringo (int & getal);` toevoegen.

Dus bij

```
bool ringo (int & getal); // prototype ringo

void george ( ) { ... }//george

bool ringo (int & getal) { ... }//ringo

int main ( ) { ... }//main
```

mogen `ringo` en `george` elkaar beide(n) aanroepen! Dankzij het prototype van `ringo` (let op de ;) mag `george` de eigenlijk verderop gedefinieerde `ringo` toch gebruiken.

En om misverstanden te vermijden: functies worden binnen functies aangeroepen, maar *na elkaar* en niet binnen elkaar gemaakt.

Bij top-down maak je een functie als je deze nodig hebt, bij bottom-up bedenk je deze daarvoor al.

Voorbeeld: machtverheffen, $y = x^7$. Bij bottom-up gebruik je `pow` uit `<cmath>` of uit `"zelf.h"`, bij top-down maak je:

```
// bereken x tot de n-de voor n >= 0
int machtsverheffen (int x, int n) {
  int i;  // tellertje
  int res = 1;  // om resultaat in op te bouwen
  for ( i = 1; i <= n; i++ ) { res = res * x; }//for
  return res;
}//machtsverheffen
```

Daarna zet je dit misschien alsnog in `"zelf.h"`.

PS En met 5 vermenigvuldigingen x^{15} berekenen?

Een functie mag zichzelf (in)direct aanroepen: recursie.

```
int som (int n) { // berekent 1 + 2 + ... + n   versie 1
  int i, res = 0;
  for ( i = 1; i <= n; i++ ) res += i;
  return res;
}//som

int somrecursief (int n) { // idem, recursief   versie 2
  if ( n == 0 ) return 0;
  else return n + somrecursief (n-1);
}//somrecursief

int somslimGauss (int n) { // en nog eens ...   versie 3
  return ( n * ( n + 1 ) ) / 2;
}//somslim
```

De ggd kan ook recursief berekend worden:

```
int ggdrecursief (int x, int y) {
  if ( y == 0 ) return x;
  else return ggdrecursief (y,x % y);
}//ggdrecursief
```

Je gebruikt eigenlijk:

$$\text{ggd}(x,y) = \begin{cases} x & \text{als } y = 0 \\ \text{ggd}(y, x \bmod y) & \text{als } y \neq 0 \end{cases}$$

Voor meer over recursie, zie later.

```
int a; int b;
void kwadraat (int   a) { // call by value
  a = pow (a,2); // uit <cmath>, oftewel a * a
  b++;
  cout << "0: " << a << " en " << b << endl;
}//kwadraat
```

Nu doen we:

```
(1) a = 5; b = 13; kwadraat (a);
    cout << "1: " << a << " en " << b << endl;
(2) a = 2; b = 7;  kwadraat (b);
    cout << "2: " << a << " en " << b << endl;
```

Dat levert

```
  0: 25 en 14          0: 49 en 8
  1: 5 en 14           2: 2 en 8
```

```
int a; int b;
void kwadraat (int & a) { // call by reference
  a = pow (a,2); // uit <cmath>, oftewel a * a
  b++;
  cout << "0: " << a << " en " << b << endl;
}//kwadraat
```

Nu doen we:

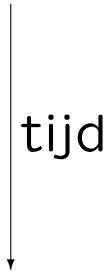
```
(3) a = 5; b = 13; kwadraat (a);
    cout << "3: " << a << " en " << b << endl;
(4) a = 2; b = 7;  kwadraat (b);
    cout << "4: " << a << " en " << b << endl;
```

Dat levert

```
  0: 25 en 14          0: 50 en 50
  3: 25 en 14          4: 2 en 50
```

```
void alias (int r, int & s) {
  int t;
  t = 3;
  r = r + 2;
  s = s + r + t;
  t = t + 1;
  r = r - 3;
  cout << r << " " << s << " " << t << endl;
}//alias
...
  t = 12; alias (t,t); cout << t << endl;
```

t = s	r	t'
12		
	12	?
		3
	14	
29		
		4
	11	



Dit levert: `11 29 4` en 29.

En met een `&` voor `r`: 28 28 4 en 28.

```
void test (int x, int & y) {
  int z = 9; x = 5; y = 6; z = 7;
}//test
```

Steeds eerst `x = 1; y = 2; z = 3;` , en daarna `x`, `y` en `z` afdrukken:

a. `test (x,y);` geeft 1, 6, 3

b. `test (y,x);` geeft 6, 2, 3

c. `test (1,z);` geeft 1, 2, 6

d. `test (z,1);` mag niet, er moet een "variabele" (l-value) op de tweede plek staan!

e. `test (z,x);` geeft 6, 2 ,3

Eigenlijk maakt de functie alleen zijn tweede variabele 6.

```
int f (int x, int y) { x--; return x * y; }//f
int g (int a, int b) {
  int x = 3; b += x; a--; a = f (a,b) + f (a,a);
  cout << x << a << b << endl; return a + x - 2; }//g
```

a. `x = 6; y = 16; cout << g(x,y); cout << x << y << endl;`
levert: 3, 96, 19 97, 6, 16

b. `int G (int a, int b) { return (a-2)*(a+b+2) + 1; }//G`

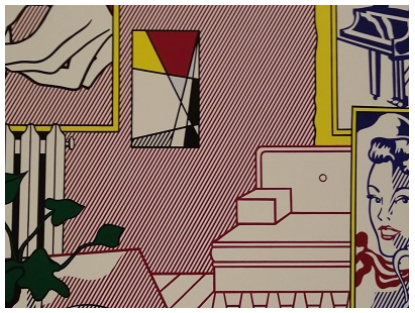
c. Als **a**, met vier &'s
Als eerst `f (a,b)` wordt geëvalueerd: 76, en `a` (dus `x`) is nu 4. Dan `f (a,a)`, geeft 9, en `a` (dus `x`) is nu 3. Dat levert: 3, 85, 19, 86, 85, 19. Met eerst `f (a,a)`: 3, 73, 19, 74, 73, 19. De volgorde is onduidelijk in C++.

```
int peter (int r, int s) { s--; return r+s+2; }//peter
int ellen (int p, int q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (p,q);
  cout << a << p << q << endl; return a+p+q; }//ellen
```

a. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

b. Idem, met vier &'s.

c. Als **b**, nu met `p = p + peter (q,p);` .



```
int peter (int r, int s) { s--; return r+s+2; }//peter
int ellen (int p, int q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (p,q);
  cout << a << p << q << endl; return a+p+q; }//ellen
```

a. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

a	b	p_{ellen}	q_{ellen}	a_{ellen}
2	6			
		2	6	
				7
		3		
			4	
				2
		11 (*)		
				3
		27 (*)		
				4

tijd

(*) `peter (3,4)` geeft 8, en `peter (11,4)` geeft 16.

Afgedrukt wordt: 4, 27, 4 35, 2, 6

```
int peter (int & r, int & s) { s--; return r+s+2; }//peter
int ellen (int & p, int & q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (p,q);
  cout << a << p << q << endl; return a+p+q; }//ellen
```

b. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

$a = p_{ellen}$	$b = q_{ellen}$	a_{ellen}
2	6	
		7
3		
	4	
		2
	3 (*)	
11 (*)		
		3

(*) `peter (p,q)` geeft 8, en laagt `q` met 1 af. Loop stopt!

Afgedrukt wordt: 3, 11, 3 17, 11, 3

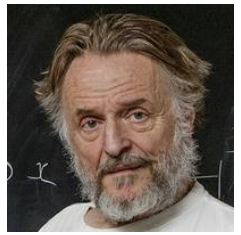
```
int peter (int & r, int & s) { s--; return r+s+2; }//peter
int ellen (int & p, int & q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (q,p); // <-- volgorde anders
  cout << a << p << q << endl; return a+p+q; }//ellen
```

c. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

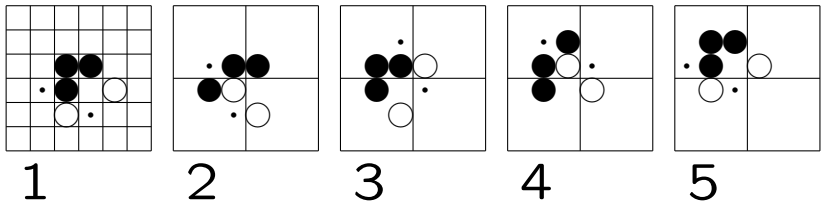
$a = p_{ellen}$	$b = q_{ellen}$	a_{ellen}
2	6	
		7
3		
	4	
		2
10 (*)		
		3
24 (*)		
		4

(*) Stel eerst: `peter (q,p)` geeft 8, en laagt `p` met 1 af naar 2. Analoog, de tweede keer: `p` wordt 24.
Afgedrukt wordt: 4, 24, 4 32, 24, 4
Maar (bij andere volgorde) kan a ook 25/26/27 zijn . . .

Life is een "cellulaire automaat", in 1970 bedacht door John Horton Conway (1937–2020).



In een 2-dimensionaal oneindig groot rooster beginnen we met een eindig aantal levende vakjes oftewel cellen. Een levend vakje met minder dan 2 of meer dan 3 buren (van de 8) gaat dood, met precies 2 of 3 levende buren overleeft het. In een dood vakje met precies 3 levende buren ontstaat leven. Dit leidt tot de volgende generatie. Let erop dat dit voor alle vakjes tegelijk gebeurt.

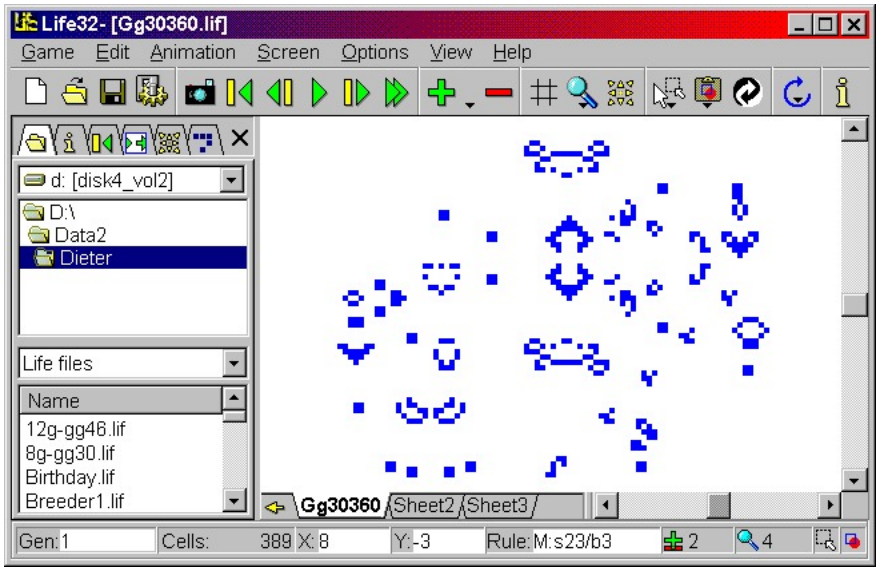


- ● levend
- ○ gaat dood
- · (komt tot leven)

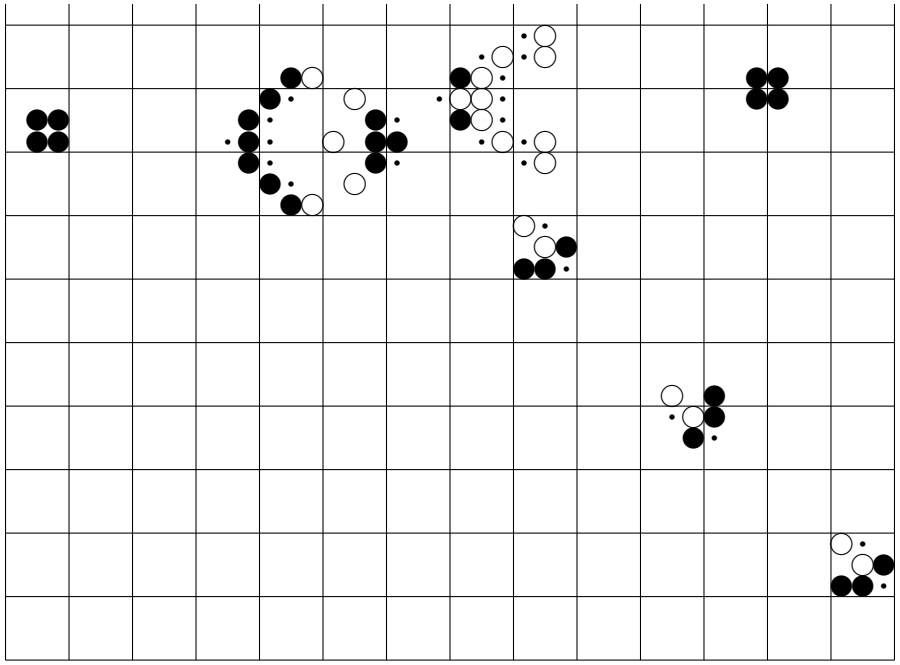
Dit patroon heet glider.

- Wiki: http://www.conwaylife.com/wiki/

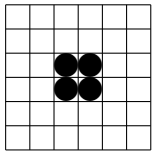
- Programma (Windows):
  https://github.com/JBontes/Life32 (Binary)

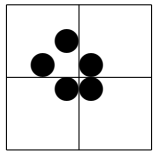


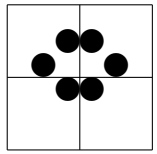
In 1970 wonnen onderzoekers van het M.I.T. in Boston $50 met een beginconfiguratie waarbij het aantal levende cellen groter en groter wordt: Gosper's glider gun, die elke dertigste generatie een nieuwe glider afvuurt:



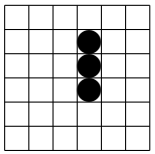
Een stilleven is een Life-configuratie die niet verandert:

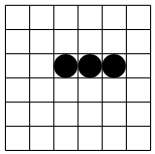

blok

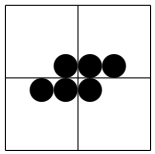

boot

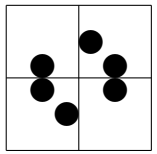

bijenkorf

En een oscillator repeteert met een zekere periode (stilleven is een periode-0 oscillator):

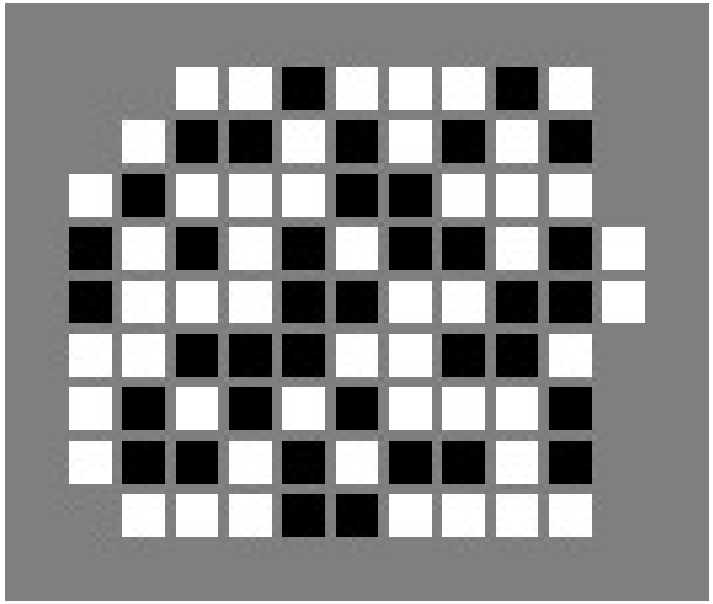

1


blinker 2


1 pad

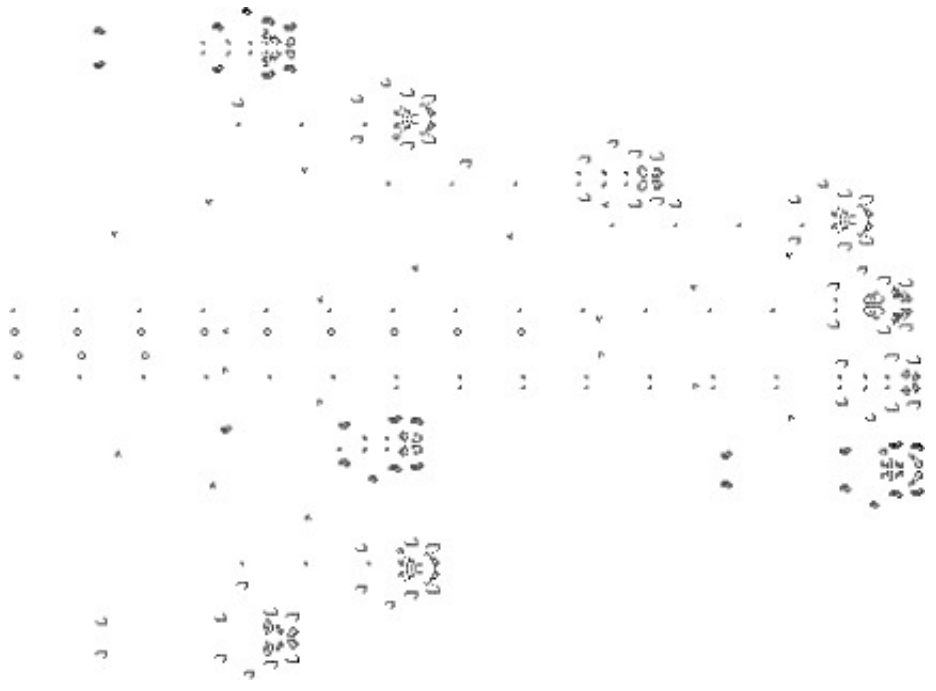

2

Een wees = orphan is een life-(deel)patroon dat nooit kan ontstaan tijdens de ontwikkeling vanuit een beginpatroon. Minder algemeen, een Hof van Eden heeft geen "ouder".



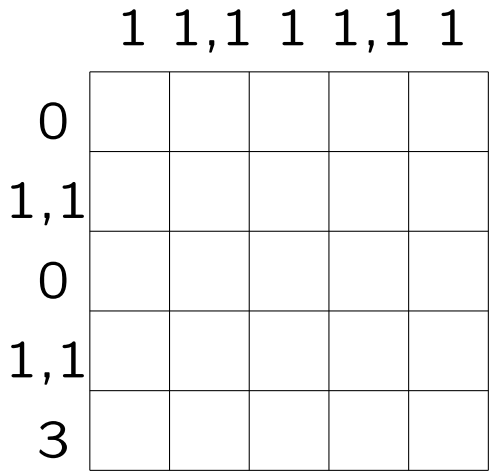
Steven Eker, 2017

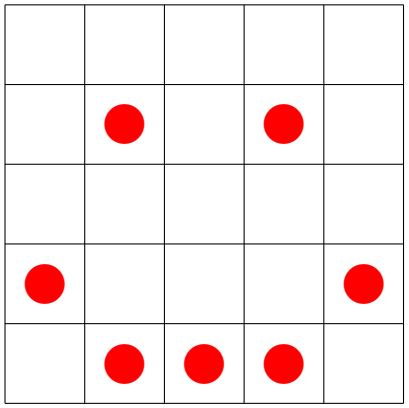
Een breeder is een life-configuratie die glider guns produceert:

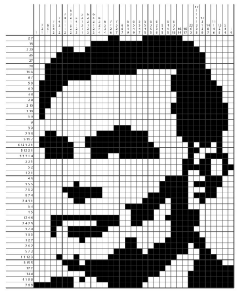


www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

Japanse puzzels (Nonogrammen) zien er zo uit:

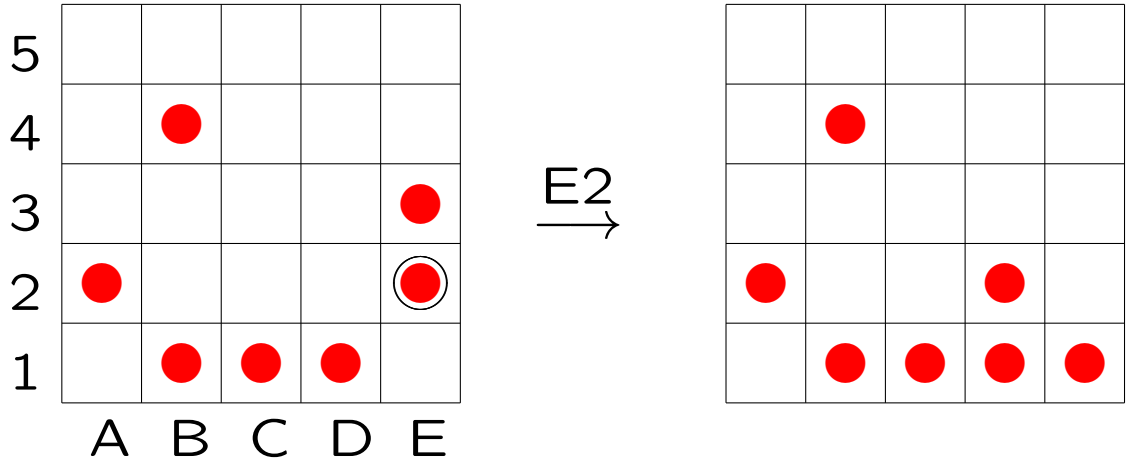






Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes.

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 $\leftrightarrow$ uit).

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

- werk aan de tweede programmeeropgave — de deadline is op maandag 14 oktober 2024, 18:00 uur
  denk aan het vragenuur

video

- lees daarna de derde programmeeropgave
- maak opgaven tot en met 25 uit het opgavendictaat
- www.liacs.leidenuniv.nl/~kosterswa/pm/