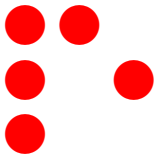
Programmeermethoden



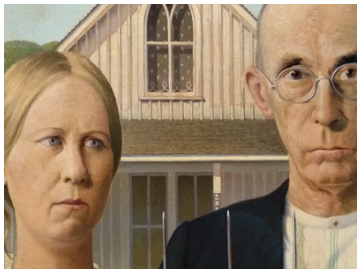
Functies & Life

Walter Kosters en Jonathan Vis

week 6: 9–13 oktober 2023

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

Voor de tweede programmeeropgave moet je een C++-programma schrijven dat een gegeven file codeert of decodeert met run-length encoding.



```
Eet␣meer␣␣␣␣␣␣␣␣␣␣zeeegels
ABC11123ddd\efG\\\1
```

moet worden:

```
Eet␣me2r␣10ze3gels
ABC\13\2\3d3\\efG\\3\1
```

Hier is ␣ een spatie.

Klopt Collatz voor 6171?

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

Tweede programmeeropgave

1. coderen
   test dat goed met voorbeeldfiles
   gebruik zo weinig mogelijk `put`'s en `get`'s

2. decoderen ($\approx$ coderen)

3. daarna, of juist eerder, de Collatz-controle (`INT_MAX`!)

4. en tot slot details, tellers, . . . , en het verslag

$\swarrow \leq \approx$ 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

- voorbeeldfile om werking te illustreren:

  `ABC11123ddd\efG\\\1` ⟶ `ABC\13\2\3d3\\efG\\3\1`

- iets over het Collatz-vermoeden, 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

Bereken $A(n) = 1/2 + 2/4 + 3/8 + \ldots + n/2^n$:

```
double sommetje (int n) {
  int teller;       // teller van teller-de term
  int noemer = 1;   // en de noemer daarvan
  double som = 0;   // de (deel)som
  for ( teller = 1; teller <= n; teller++ ) {
    noemer *= 2;   // noemer = noemer * 2;
    som += static_cast<double>(teller) / noemer;
  }//for
  return som;
}//sommetje
```

(Eigenlijk kun je beter met $n/2^n$ beginnen . . .)

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

```
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`

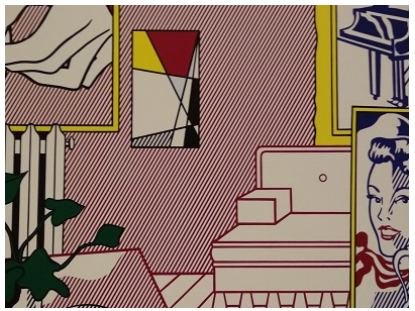
c. Als **a**, met vier &'s
Als eerst `f (a,b)` wordt geëvalueerd: 76, en `a` (dus `x`) is nu 4. Dan `f (a,a)`, geeft 9, en `a` (dus `x`) is nu 3. Dat levert: 3, 85, 19, 86, 85, 19. Met eerst `f (a,a)`: 3, 73, 19, 74, 73, 19. De volgorde is onduidelijk in C++.

```
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

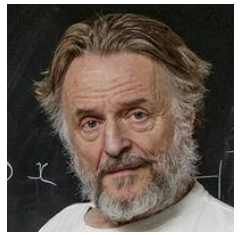
```
int peter (int & r, int & s) { s--; return r+s+2; }//peter
int ellen (int & p, int & q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (q,p); // <-- volgorde anders
  cout << a << p << q << endl; return a+p+q; }//ellen
```

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

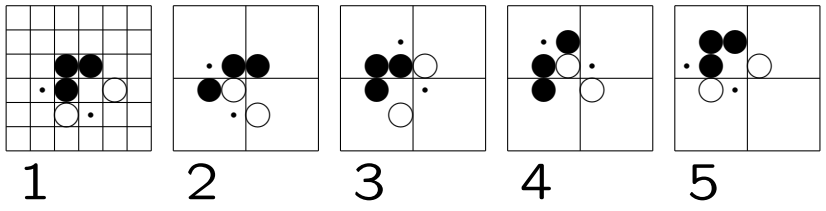
$a = p_{ellen}$	$b = q_{ellen}$	a_{ellen}
2	6	
		7
3		
	4	
		2
10 (*)		
		3
24 (*)		
		4

(*) 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).



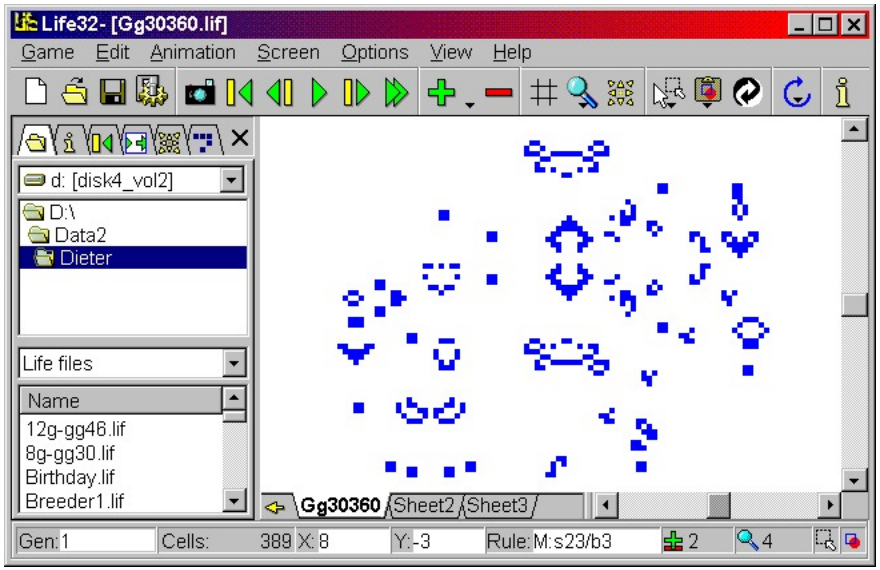
In een 2-dimensionaal oneindig groot rooster beginnen we met een eindig aantal levende vakjes oftewel cellen. Een levend vakje met minder dan 2 of meer dan 3 buren (van de 8) gaat dood, met precies 2 of 3 levende buren overleeft het. In een dood vakje met precies 3 levende buren ontstaat leven. Dit leidt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



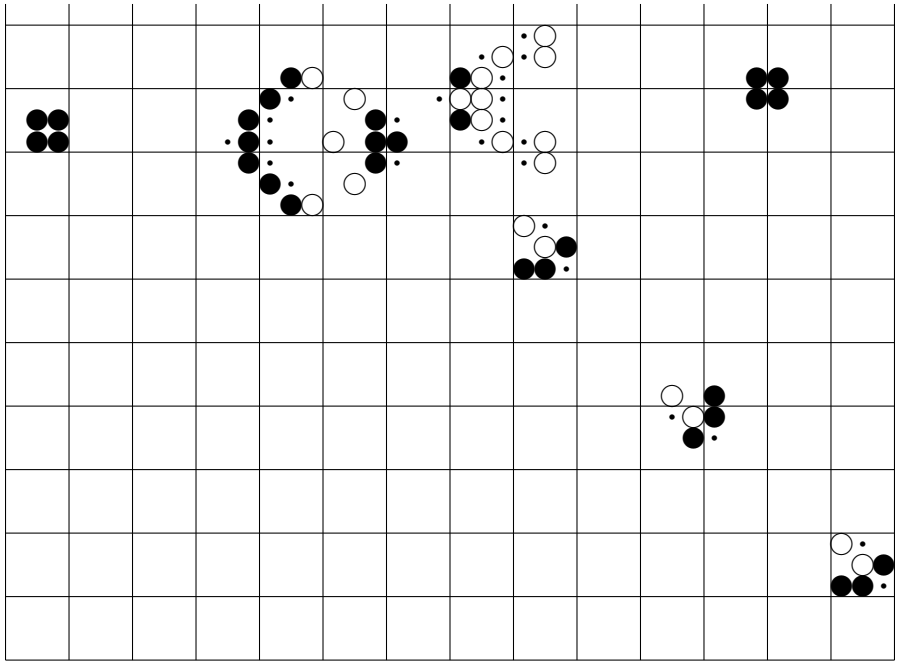
- ● levend
- ○ gaat dood
- · (komt tot leven)

Dit patroon heet glider.

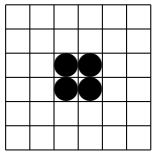
- Wiki: http://www.conwaylife.com/wiki/
- Programma (Windows):
  https://github.com/JBontes/Life32 (Binary)

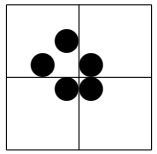


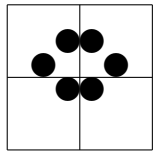
In 1970 wonnen onderzoekers van het M.I.T. in Boston $50 met een beginconfiguratie waarbij het aantal levende cellen groter en groter wordt: Gosper's glider gun, die elke dertigste generatie een nieuwe glider afvuurt:



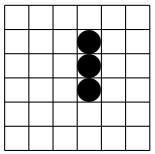
Een stilleven is een Life-configuratie die niet verandert:

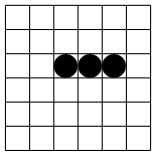

blok

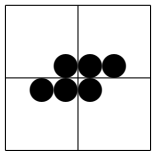

boot

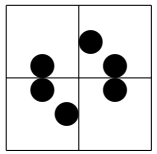

bijenkorf

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

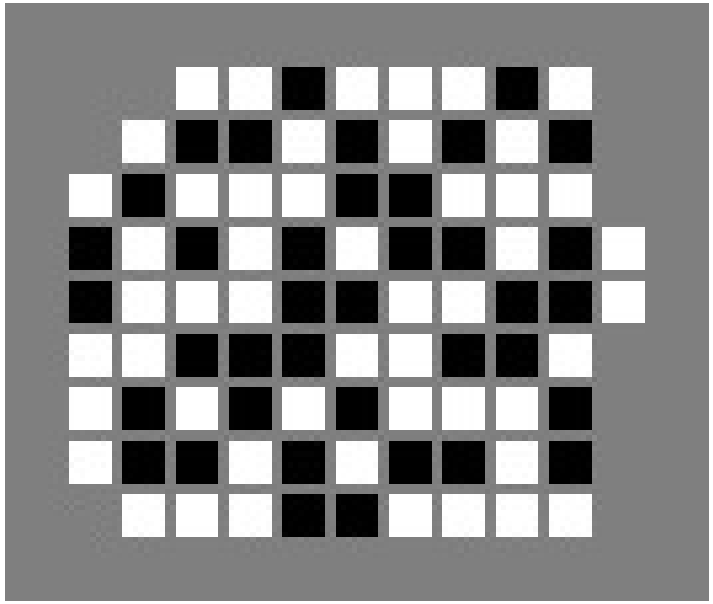

1


blinker 2


1 pad


2

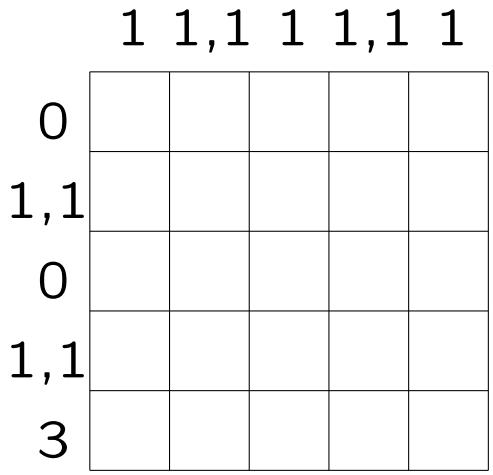
Een wees = orphan is een life-(deel)patroon dat nooit kan ontstaan tijdens de ontwikkeling vanuit een beginpatroon. Minder algemeen, een Hof van Eden heeft geen "ouder".

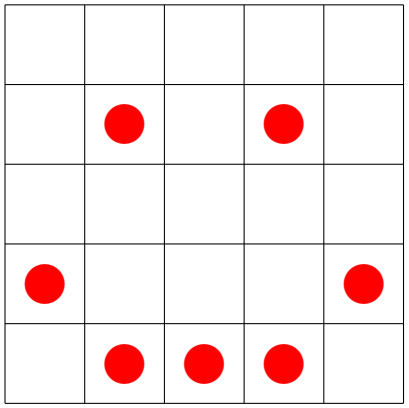


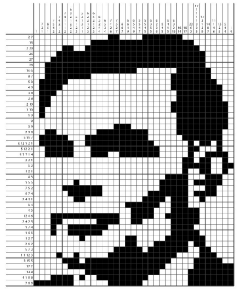
Steven Eker, 2017

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

Japanse puzzels (Nonogrammen) zien er zo uit:



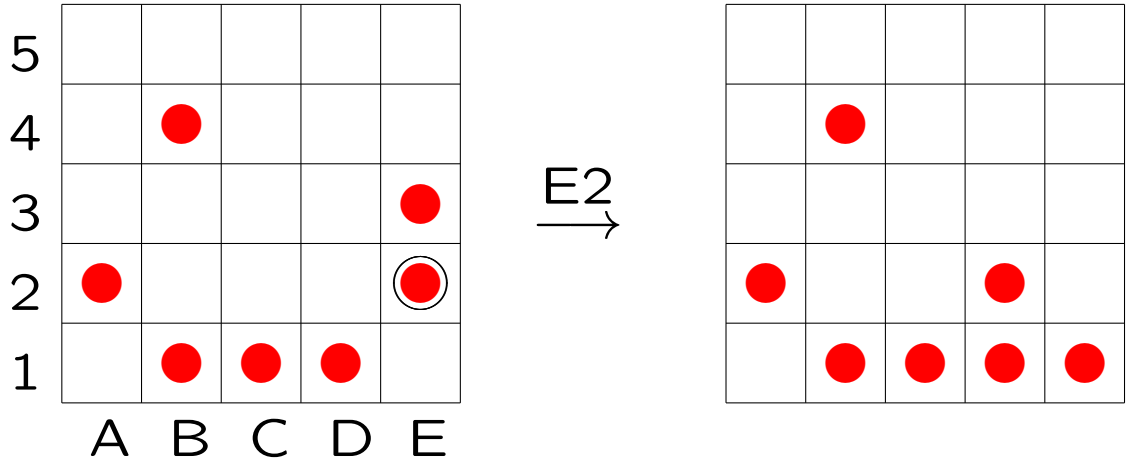




Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes.

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 $\leftrightarrow$ uit).

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

- werk aan de tweede programmeeropgave — de deadline is op maandag 16 oktober 2023, 18:00 uur
  denk aan het vragenuur

- lees daarna de derde programmeeropgave

- maak opgaven tot en met 25 uit het opgavendictaat

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