Pole v jazyce C++

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava

Prezentace ke dni 12. února 2020





Osnova přednášky

Pole v jazyce C++

- Výpočty lineární algebry
  - Vektorové výpočty
  - Výpočty se čtvercovými maticemi
- Game of Life
  - Popis hry
  - Zadání úlohy
  - Analýza řešení
  - Implementace
  - Refaktorování

Zadání úlohy

Implementujte funkce pro základní vektorové a maticové výpočty:

- vektory a metice reprezentujte poli s prvky typu `double`,
- dimenze vektorů bude dána konstantou N,
- implementujte nulování, součet, skalární součin, délku a odchylku vektorů,
- implementujte součet a násobení matic, vytvoření nulové a jednotkové matice,
- implementujte výstup vektorů a matic na standardní výstup.

Obecné deklarace

```
1 #include <math.h>
2 #include <iostream>
3
4 using namespace std;
5
6 const int N = 5;
```

Zobrazení vektoru na standardní výstup

```
1 void Print(double v[], const int n)
2 {
3    for(int i = 0; i < n; i++)
4    {
5       cout << v[i] << "\t";
6    }
7    cout << endl;
8 }
```

Řešení: pomocí cyklu postupně vypisujeme všechny prvky daného vektoru.

Nulování vektoru

```
1 void Clear(double v[], const int n)
2 {
3    for(int i = 0; i < n; i++)
4    {
5       v[i] = 0;
6    }
7 }
```

Řešení: pomocí cyklu postupně přiřazujeme všem prvkům daného vektoru hodnotu 0.

Součet vektorů

Součet vektorů je definován jako

$$w_i = u_i + v_i, \forall i = 0, \ldots, n-1$$

```
1 void Add(double u[], double v[], double w[], const int n)
2 {
3    for(int i = 0; i < n; i++)
4    {
5       w[i] = u[i] + v[i];
6    }
7 }
```

Řešení:

- pomocí cyklu postupně sčítáme odpovídající si prvky vektoru $\vec{u}$ a $\vec{v}$,
- součet přiřazujeme do odpovídajícího prvku vektoru $\vec{w}$.

Délka vektoru

Délku vektoru $\vec{u}$ definujeme jako

$$|\vec{u}| = \sqrt{\sum_{i=0}^{n-1} u_i^2} = u_0^2 + u_1^2 + \cdots + u_{n-1}^2$$

Řešení:

- pomocí cyklu postupně umocňujeme prvky vektoru $\vec{u}$ na druhou – tady stačí použít obyčejné násobení,
- výpočet součtu – jednotlivé druhé mocniny musíme průběžně přičítat do pomocné proměnné,
- součet (suma) tedy vzniká postupně, průběžně jsou do pomocné proměnné „přihazovány“ další a další mezivýsledky,
- nesmíme zapomenout pomocnou proměnnou vynulovat!
- nakonec pomocnou proměnnou odmocníme a vrátíme ji jako výsledek výpočtu.

Délka vektoru (pokrač.)

```
1 double Length(double u[], const int n)
2 {
3    double sum = 0;
4    for(int i = 0; i < n; i++)
5    {
6       sum += u[i] * u[i];
7    }
8    return sqrt(sum);
9 }
```

Skalární součin dvou vektorů

Skalární součin vektorů $\vec{u}$ a $\vec{v}$ definujeme jako

$$\vec{u} \cdot \vec{v} = \sum_{i=0}^{n-1} u_i v_i$$

Řešení:

- pomocí cyklu postupně násobíme odpovídající si prvky vektoru $\vec{u}$ a $\vec{v}$,
- výpočet součtu – jednotlivé součiny musíme průběžně přičítat do pomocné proměnné,
- nesmíme zapomenout pomocnou proměnnou vynulovat!
- nakonec pomocnou proměnnou vrátíme jako výsledek výpočtu.

Skalární součin dvou vektorů (pokrač.)

```
1 double DotProduct(double u[], double v[], const int n)
2 {
3    double sum = 0;
4    for(int i = 0; i < n; i++)
5    {
6       sum += u[i] * v[i];
7    }
8 }
```

Odchylka dvou vektorů

Odchylku dvou vektorů $\vec{u}$ a $\vec{v}$ (úhel sevřený temito vektory) definujeme jako

$$\cos\varphi = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}$$

Řešení:

- budeme počítat jen $\cos\varphi$, jde tedy o výpočet zlomku,
- v čitateli je skalární součin, ve jmenovateli je součin délek vektorů,
- dvě možnosti implementace:
  1. implementace funkce na „zelené louce“, vše znovu nebo
  2. využijeme již hotové funkce – preferované řešení.

```
1 double VectorsAngle(double u[], double v[], const int n)
2 {
3    return DotProduct(u, v, n) / (Length(u, n) * Length (v, n));
4 }
```

Zobrazení matice na standardní výstup

Řešení:

- pomocí dvou cyklů postupně vypisujeme všechny prvky dané matice,
- první cyklus iteruje přes řádky matice, druhý přes sloupce matice.

```
1 void Print(double A[][N], const int n)
2 {
3    for(int i = 0; i < n; i++)
4    {
5       for(int j = 0; j < n; j++)
6       {
7          cout << A[i][j] << "\t";
8       }
9       cout << endl;
10   }
11   cout << endl;
12 }
```

Zobrazení matice na standardní výstup (pokrač.)

Doporučení

Nikdy nepoužívejte pro proměnné v cyklu při iterování přes řádky a sloupce jména `x` a `y`! Vždy tradiční `i` a `j`, či `r` a `c`. Jména promměných `x` a `y` budí dojem souřadnic – tyto proměnné mají ale přesně opačný význam!

Nulová matice

```
1 void Clear(double A[][N], const int n)
2 {
3    for(int i = 0; i < n; i++)
4    {
5       for(int j = 0; j < n; j++)
6       {
7          A[i][j] = 0;
8       }
9    }
10 }
```

Řešení: pomocí dvou cyklů postupně přiřazujeme všem prvkům dané matice hodnotu 0.

Jednotková matice

```
1 void Unity(double A[][N], const int n)
2 {
3    for(int i = 0; i < n; i++)
4    {
5       for(int j = 0; j < n; j++)
6       {
7          A[i][j] = i == j ? 1 : 0;
8       }
9    }
10 }
```

Řešení: pomocí dvou cyklů postupně přiřazujeme všem prvkům na diagonále dané matice hodnotu 1, jinak 0.

Součet dvou matic

Součet matic je definován jako

$$C = A + B$$

v zápisu pomocí prvků matice bude součet definován jako

$$c_{i,j} = a_{i,j} + b_{i,j}$$

Řešení:

- pomocí dvou cyklů postupně sčítáme odpovídající si prvky matic A a B,
- součet přiřazujeme do odpovídajícího prvku matice C.

Součet dvou matic (pokrač.)

```
1  void Add(double A[][N], double B[][N], double C[][N], const int n)
2  {
3     for(int i = 0; i < n; i++)
4     {
5        for(int j = 0; j < n; j++)
6        {
7           C[i][j] = A[i][j] + B[i][j];
8        }
9     }
10 }
```

Násobení dvou matic

Násobení dvou matic je definováno po prvcích takto

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

pro všechny indexy i a j. Řešení:

- podíváme-li se na vzorec, je zřejmé, že se jedná o skalární součin řádku a_i matice A se sloupcem bj matice B,
- implementace bude vyžadovat cyklus a pomocnou proměnnou,
- dále tento skalární součin musíme provést pro všechny řádky a všechny sloupce,
- budeme tedy potřebovat další dva cykly.

Násobení dvou matic (pokrač.)

```
1 void Multiply(double A[][N], double B[][N], double C[][N], const int n)
2 {
3    for(int i = 0; i < n; i++)
4    {
5       for(int j = 0; j < n; j++)
6       {
7          double sum = 0;
8          for(int k = 0; k < n; k++)
9          {
10            sum += A[i][k] * B[k][j];
11         }
12         C[i][j] = sum;
13      }
14   }
15 }
```

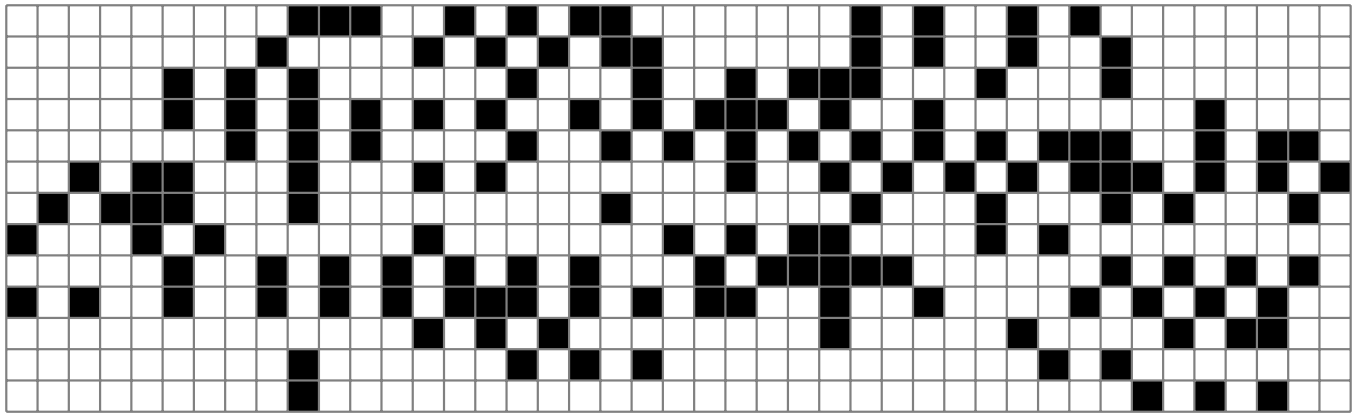
Obecný poznatek o sumách

Z předchozích ukázek vzorců a odpovídajících funkcí je patrný obecný závěr, který můžeme neformálně vyjádřit následovně:
Napíše-li matematik ve vzorci sumu, informatik ji čte jako cyklus.

Game of Life

- autorem je britský matematik John Horton Conway
- publikováno v říjnu 1970
- dvoustavový, dvourozměrný celulární automat, který svým chováním připomíná vývoj společenství živých organismů
- ukázka systému, kde z jednoduchých pravidel vzniká komplexní chování - objevují se zde tzv. emergentní struktury

Popis hry



- hrací plochou je 2D čtvercová mřížka
- políčka mřížky se nazývají buňky, každá buňka je buď živá (zobrazena černě) nebo mrtvá (zobrazena bíle), buňky mimo mřížku jsou mrtvé
- každá buňka sousedí s dalšími buňkami, se kterými se dotýká hranou (ortogonálně) či vrcholem (diagonálně) – Moorovo okolí
- kolonie buněk v mřížce přežívá z generace na generaci
- stav buňky v nové generaci závisí na jejím současném stavu a počtu živých sousedů

Pravidla hry

Hra má následující pravidla:

1. Každá živá buňka s méně než dvěma živými sousedy nebo s více než třemi živými sousedy zemře. V příští generaci bude označena jako mrtvá.
2. Každá živá buňka se dvěma nebo třemi živými sousedy zůstává v příští generaci žít.
3. Každá mrtvá buňka s právě třemi živými sousedy v příští generaci obživne.

Poznámky

- Stav všech buněk se mění naráz, paralelně.
- Řešení na sekvenčním počítači – ze současné generace vypočteme generaci novou, kterou poté nahradíme starou generaci.

Ukázka výpočtu nové generace buněk

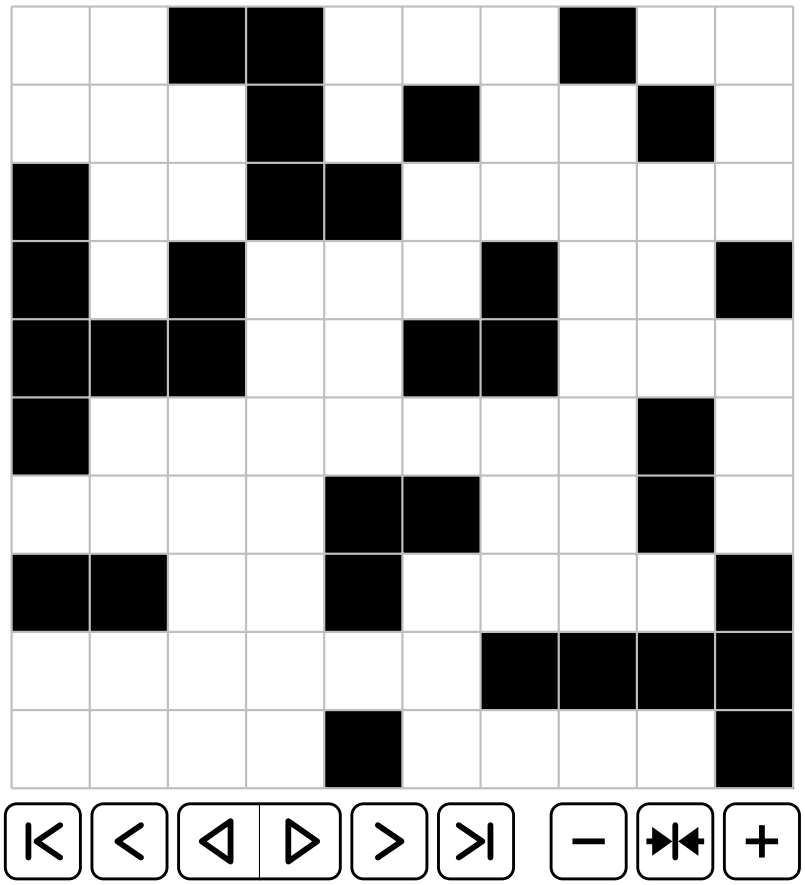
Stávající generace buněk

0	0	1	1	1	0
1	1	2	**1**	2	1
2	**1**	3	3	**2**	2
2	**2**	3	3	**1**	2
1	2	**1**	2	1	1
0	1	1	1	0	0

Nová generace buňek

0	0	0	0	0	0
0	1	2	3	2	1
1	3	**4**	**4**	**2**	1
1	**2**	**4**	**4**	3	1
1	2	3	2	1	0
0	0	0	0	0	0

Animace života kolonie



Vzory vznikající v Game of Life

Přestože pravidla hry jsou velice jednoduchá, vznikají ve hře složité vzory:

Zátiší (Still life) – stabilní konfigurace buněk.

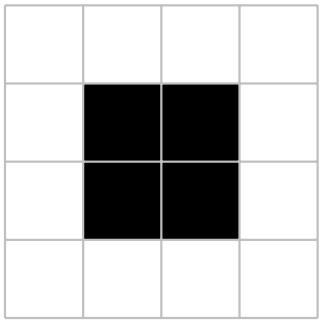
Oscilátory (Oscillators) – nestabilní konečný vzor, který periodicky přechází mezi několika konfiguracemi.

Děla (Guns) – vzor rostoucí do nekonečna

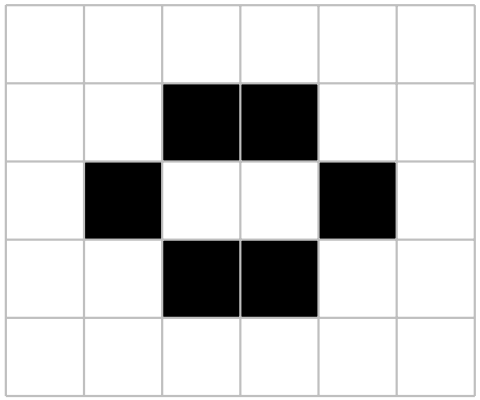
Metuzalémové (Methuselahs) – relativně malý vzor, jehož stabilizace trvá naopak velmi velký počet generací.

Výčet není konečný...

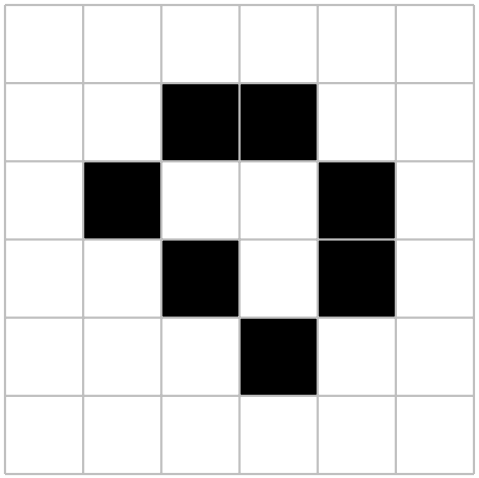
Zátiší (Still life)



Blok
(Block



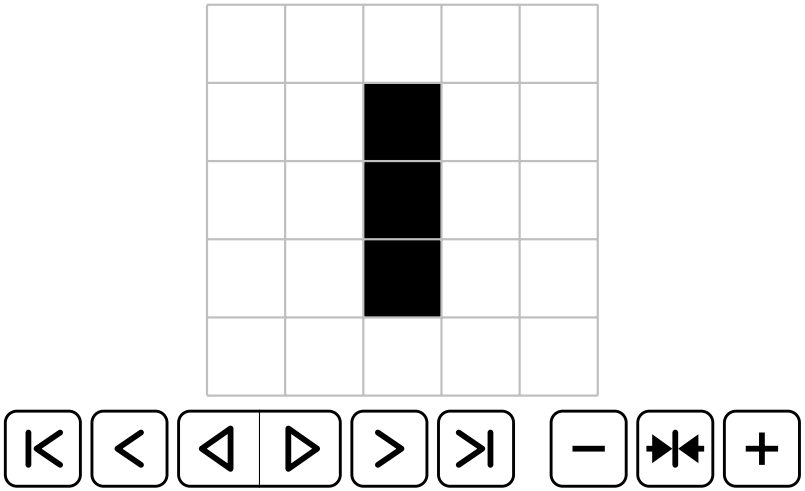
Včelín
(Beehive)



Bochník
(Loaf)

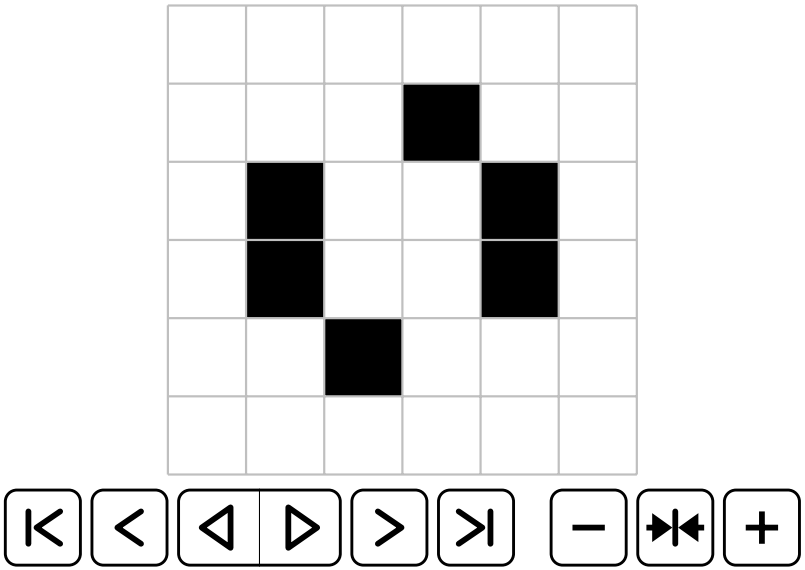
Oscilátory (Oscillators)

Blikač (Blinker), perioda 2



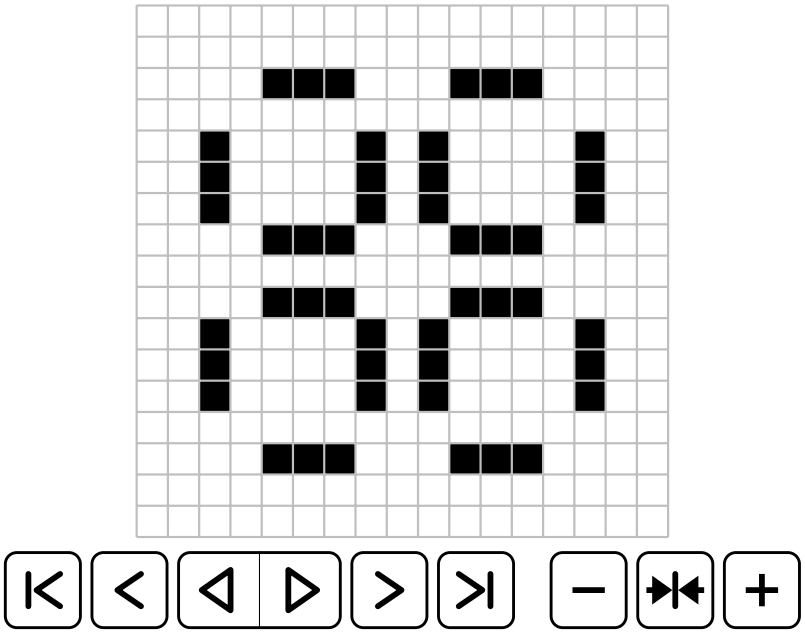
Oscilátory (Oscillators) (pokrač.)

Ropucha (Toad), perioda 2



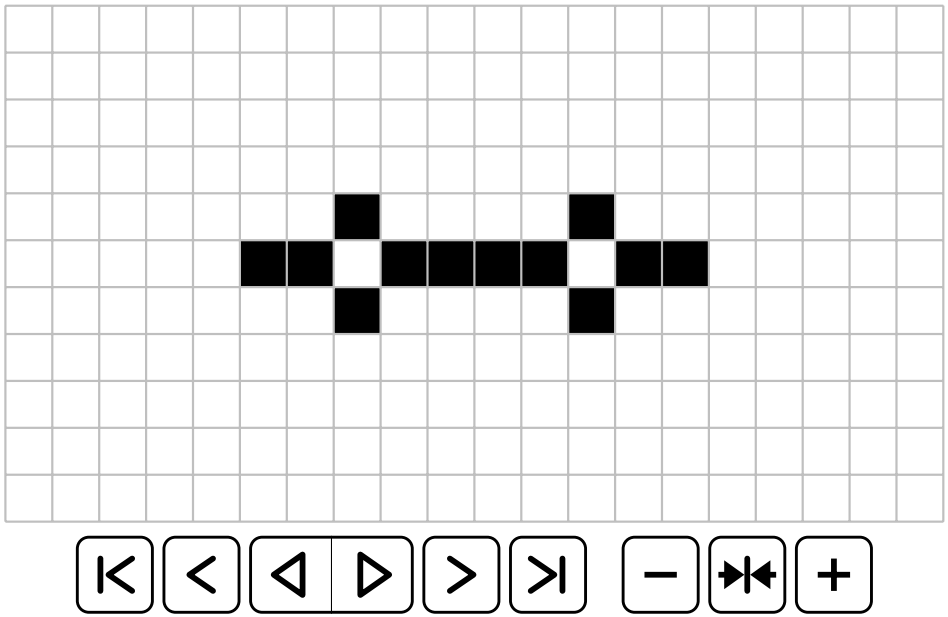
Oscilátory (Oscillators) (pokrač.)

Pulsar, perioda 3

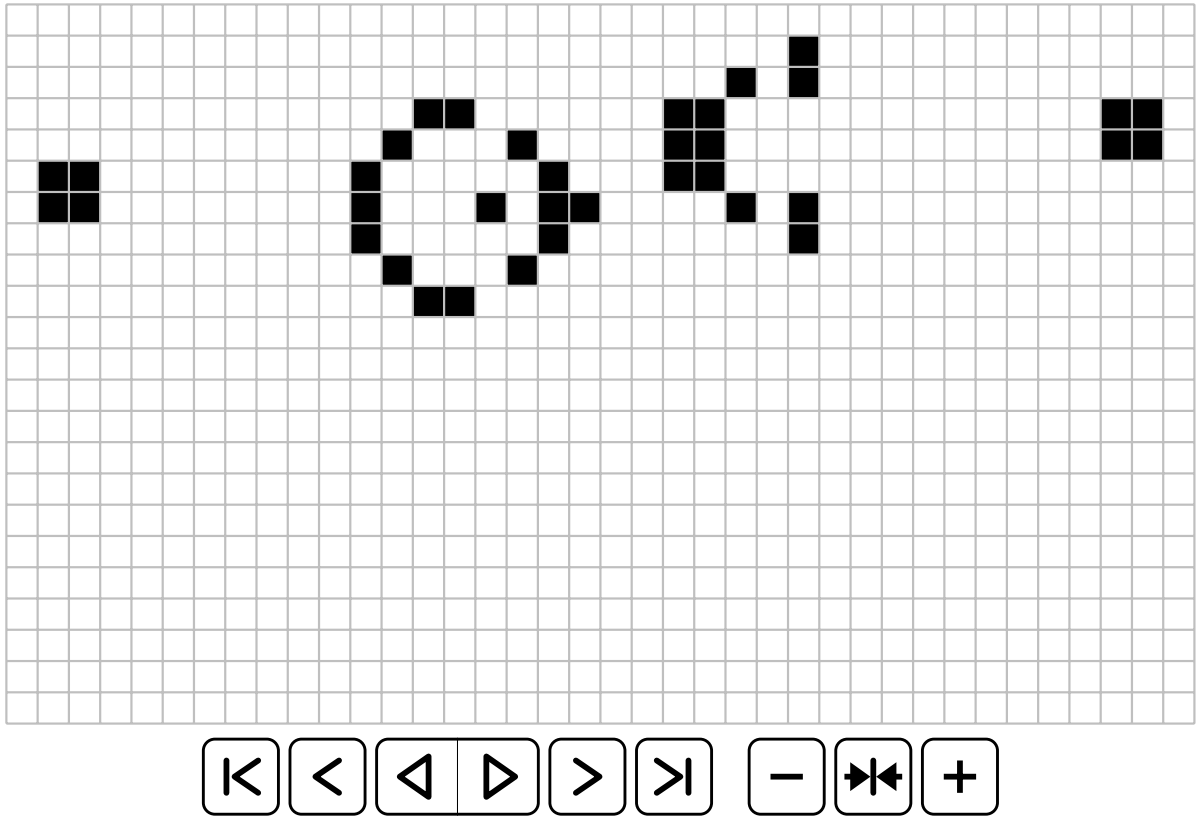


Oscilátory (Oscillators) (pokrač.)

Pentadecathlon, perioda 15



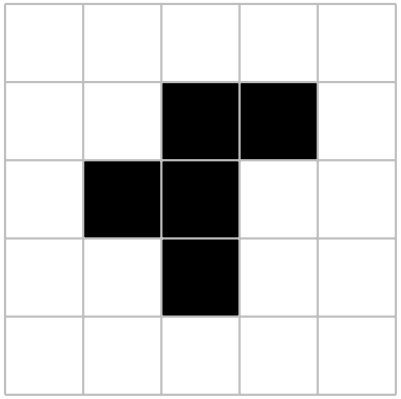
Děla – Gosper glider gun



- Dělo každých 30 generací vypustí kluzák.
- Kluzák se skládá ze 5 buněk, má 4 konfigurace, pohybuje se rychlostí $\frac{1}{4}c$, kde c je rychlost světla v tomto „vesmíru“ – jedna buňka za jednu generaci.

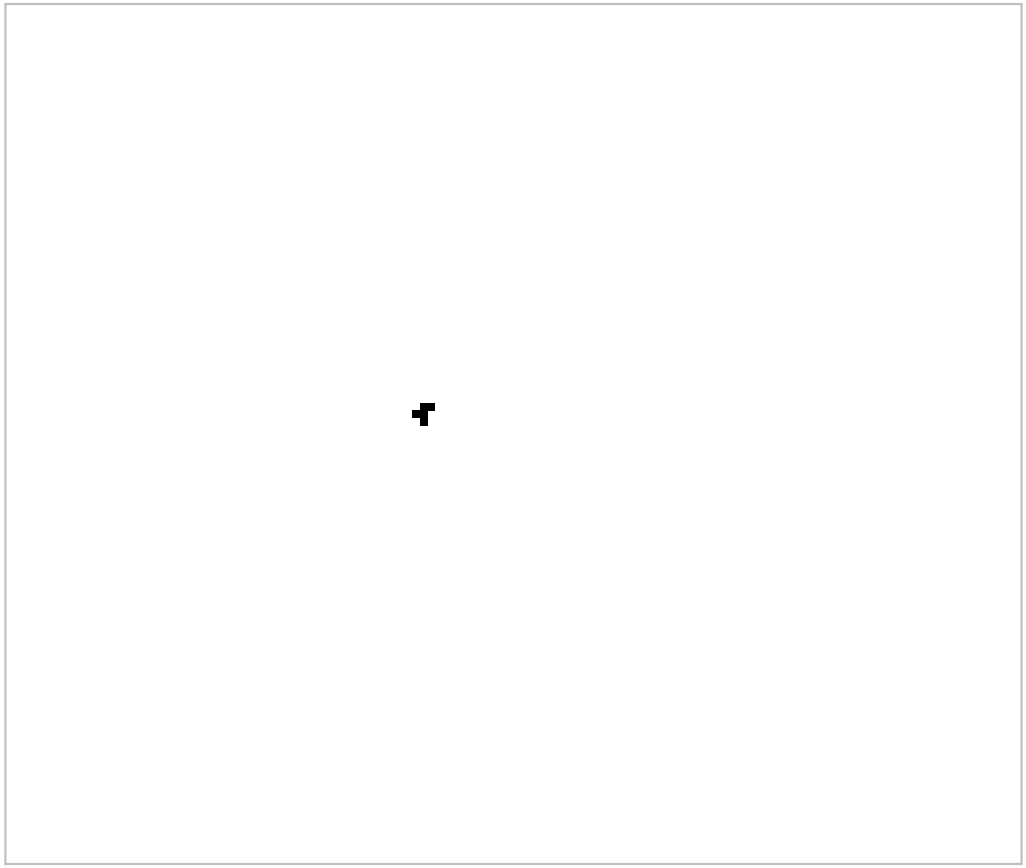
Metuzalémové – R-pentomino

Počáteční konfigurace

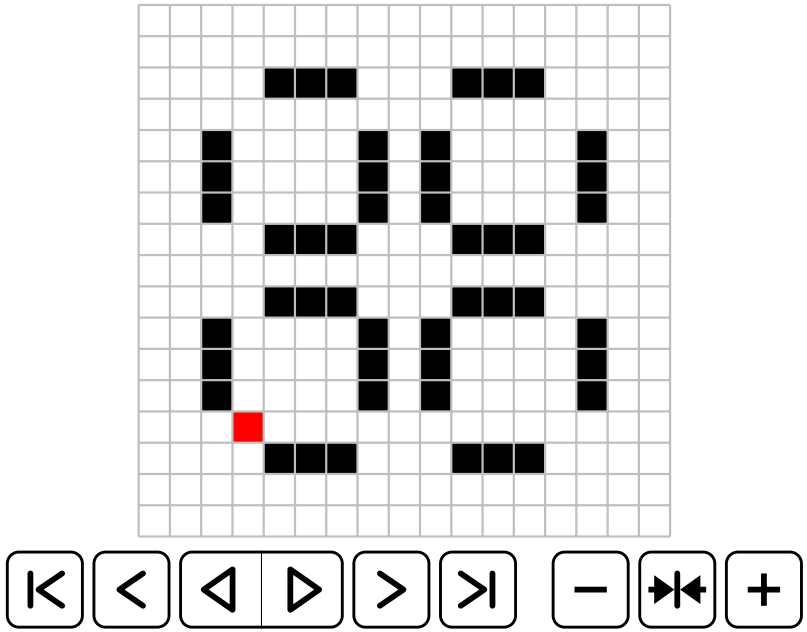


Stabilizace kolonie

po 1103 generacích



Chaotické chování



- Infikujeme oscilátor Pulsar jednou buňkou navíc, označena červeně.
- Drobná změna na počátku způsobila diametrálně odlišné chování jinak periodického vzoru.
- Některé „infekce“, ale odlišné chování nezpůsobí. Například buňka uprostřed vzoru by bez následků hned v druhé generaci zanikla.

Zadání úlohy

Zadání úlohy

Vytvořte program, který ve čtvercové mřížce dané velikosti pro náhodně vygenerovanou kolonii s daným počtem buněk vypočítá a vypíše na standardní výstup daný počet následujících generací.

Parametry úlohy

- N – velikost mřížky, mřížka bude mít rozměry $N \times N$,
- p – počet živých buněk v počáteční kolonii a
- g – požadovaný počet generací.

Pro demonstraci můžeme například použít tyto konkrétní hodnoty: $N = 10$, $p = 50$ a $g = 10$.

Nástin řešení

1. Vygeneruj počáteční náhodnou kolonii buněk.
2. Zobraz původní kolonii buněk na standardní výstup.
3. Opakuj pro daný počet generací:
   - 3.1 Vypočti novou generaci kolonie buněk, tj. pro každou buňku v současné kolonii proveď:
     - 3.1.1 zjisti počet živých buněk v jejím okolí,
     - 3.1.2 aplikuj pravidlo pro přežití, zánik nebo zrození buňky a
     - 3.1.3 nový stav buňky zaznamenej do nové kolonie.
   - 3.2 Přepiš současnou kolonii buněk nově vypočtenou.
   - 3.3 Zobraz současnou kolonii buněk na standardní výstup.
4. Konec

Nástin řešení – otázky a odpovědi

Otázky:

1. Jak reprezentovat kolonii buněk?
2. Jak vygenerovat náhodnou kolonii buněk?
3. Zobrazení kolonie na std. výstup, výpočet nové generace a přepis staré kolonie vyžaduje provést jistou akci pro každou buňku v kolonii. Jak toto provést?
4. Jak implementovat pravidla pravidla hry (přežití, zánik nebo zrod buňky)?

Odpovědi:

1. Nejlépe dvourozměrným polem logických hodnot.
2. Použijeme funkci `rand()` generující náhodná čísla plus úprava rozsahu vygenerovaných hodnot na $0, \ldots, N-1$.
3. Dvěma vnořenými cykly: jeden přes všechny řádky, druhý přes všechny sloupce.
4. Logickým výrazem, který bude pravdivý v okamžiku, kdy má být buňka v další generaci živá.

Reprezentace kolonie buněk

- Kolonii buněk reprezentujeme polem `bool a[N][N]`, kde hodnota `true` bude reprezentovat **živou** buňku a hodnota `false` **mrtvou** buňku.
- Okolí buňky `a[j][k]` definujeme následujícím schématem:

`a[j-1][k-1]`	`a[j-1][k]`	`a[j-1][k+1]`
`a[j][k-1]`	`a[j][k]`	`a[j][k+1]`
`a[j+1][k-1]`	`a[j+1][k]`	`a[j+1][k+1]`

- Tento druh okolí se někdy také nazývá Moorovo okolí.
- Při implementaci musíme dávat pozor, abychom nepřistupovali k neexistujícím buňkám.

Průchod všemi buňkami v kolonii

```
1 for(int i = 0; i < N; i++)
2 {
3    for(int j = 0; j < N; j++)
4    {
5       // some action for a[i][j]
6    }
7 }
```

- **Vnořené cykly** – cyklus je „vnořen“, „zanořen“, či „vložen“ do těla jiného cyklu.
- Cyklus s řídící proměnnou `i` je tzv. **vnější cyklus**, cyklus s řídící proměnnou `j` je tzv. **vnitřní cyklus**.
- Počet vnořených cyklů není omezen, s rostoucím vnořením ale klesá čitelnost kódu.
- Vnější cyklus probíhá přes všechny řádky kolonie, vnitřní přes všechny sloupce kolonie buněk.

Generování náhodné kolonie buněk

Náhodnou kolonii vygenerujeme ve dvou krocích:

1. Projdeme všechny buňky v kolonii a nastavíme je na hodnotu `false`.
2. Pomocí cyklu `for` budeme p krát opakovat:
   - 2.1 Vygeneruj náhodné číslo řádku `r = rand()% N`
   - 2.2 Vygeneruj náhodné číslo sloupce `c = rand()% N`
   - 2.3 Nastav buňku `a[r][c] = true`

Poznámky

- První krok je nezbytný, prvky v poli, buňky, nemají definovanou hodnotu!
- Funkce `rand()` vrací hodnoty v rozsahu `0` až `RAND_MAX` včetně. Protože $N <$`RAND_MAX` upravíme vygenerované náhodné číslo pomocí operace zbytku po dělení číslem N.
- V tomto postupu je však jedna malá logická chyba :-).

Implementace pravidel hry

V nové generaci bude buňka a_{ij} živá, pokud v současné generaci je:

- živá a zároveň má dva nebo tři živé sousedy (přežití) nebo
- mrtvá a zároveň má tři živé sousedy (zrození).

Označme L počet živých sousedů buňky a_{ij}, pak přežití můžeme zapsat jako

$$a_{ij} \wedge (L = 2 \vee L = 3)$$

a zrození jako

$$\neg a_{ij} \wedge L = 3$$

Oba logické výrazy spojíme spojkou $\vee$ do jednoho výrazu

$$[a_{ij} \wedge (L = 2 \vee L = 3)] \vee [\neg a_{ij} \wedge L = 3]$$

V jazyce C++ bude tento logický výraz vypadat takto

```
1 a[i][j] && (L == 2 || L == 3) || !a[i][j] && L == 3
```

Implementace pravidel hry (pokrač.)

Nnyní máme dvě možnosti:

1. použít tento výraz v implementaci tak jak je nebo
2. pokusit se jej zjednodušit.

Zjednodušení

Na výraz v první hranaté závorce můžeme aplikovat distributivní zákon a dostáváme

$$[a_{ij} \wedge L = 2 \vee a_{ij} \wedge L = 3] \vee [\neg a_{ij} \wedge L = 3]$$

Přeskupíme závorky

$$(a_{ij} \wedge L = 2) \vee (a_{ij} \wedge L = 3) \vee (\neg a_{ij} \wedge L = 3)$$

Nyní předpokládejme, že $L = 3$. Pokud se podíváme na druhou a třetí závorku, zjistíme že:

- pokud a_{ij} má hodnotu 1, `true`, platí druhá „závorka“ nebo

Implementace pravidel hry (pokrač.)

- pokud a_{ij} má hodnotu 0, `false`, platí třetí „závorka“.

Je jasné, že na hodnotě a_{ij} nezáleží, záleží jen na tom, zda je $L = 3$. Formálně bychom mohli postupovat tak, že nejprve aplikujeme na druhou a třetí závorku distributivní zákon

$$(a_{ij} \wedge L = 2) \vee [(a_{ij} \vee \neg a_{ij}) \wedge (L = 3)]$$

Podle zákona o vyloučení třetího má výraz $a_{ij} \vee \neg a_{ij}$ vždy hodnotu `true`, 1. Dosazením dostáváme

$$(a_{ij} \wedge L = 2) \vee [1 \wedge (L = 3)]$$

Dále podle zákona o neutralitě 1 v konjunkci dostáváme

$$(a_{ij} \wedge L = 2) \vee [L = 3]$$

Implementace pravidel hry (pokrač.)

Závorky v tomto výrazu můžeme zcela vynechat a dostaneme konečnou podobu výrazu

$$a_{ij} \wedge L = 2 \vee L = 3$$

V jazyce C++ bude tento logický výraz vypadat takto

```
1 a[i][j] && L == 2 || L == 3
```

Řešení verze 1 – na formátování přece nezáleží!

```
1 #include <iostream>
2
3    using namespace std;
4
5          void main()
6          {
7          bool a[10][10], b[10][10];
8 int i, j, k, l;
9             for (i = 0; i < 10; i++)
10         for (j = 0; j < 10; j++)
11 a[i][j] = false;
12
13            for (i = 0; i < 50; i++)
14      {
15                j = rand() % 10;
16      k = rand() % 10;
17    a[j][k] = true;
18         }
19    cout << "Puvodni kolonie" << endl;
20               for (i = 0; i < 10; i++)
21         {
22               for (j = 0; j < 10; j++)
23      {
24            if (a[i][j])
25 cout << "X";
26      else
27            cout << ".";
28
29    }
30    cout << endl;
```

Řešení verze 1 – na formátování přece nezáleží! (pokrač.)

```
31                }
32                cout << endl;
33    for (i = 1; i <= 10; i++)
34
35
36
37       {
38    for (j = 0; j < 10; j++)
39 for (k = 0; k < 10; k++)
40                {
41             l = 0;
42    if (0 <= (j - 1) && (j - 1) < 10 && 0 <= (k - 1) && (k - 1) < 10 && a[j - 1][k - 1])
43          l += 1;
44             if (0 <= (j - 1) && (j - 1) < 10 && 0 <= k && k < 10 && a[j - 1][k])
45          l += 1;
46    if (0 <= (j - 1) && (j - 1) < 10 && 0 <= (k + 1) && (k + 1) < 10 && a[j - 1][k + 1])
47          l += 1;
48    if (0 <= j && j < 10 && 0 <= (k - 1) && (k - 1) < 10 && a[j][k - 1])
49             l += 1;
50       if (0 <= j && j < 10 && 0 <= (k + 1) && (k + 1) < 10 && a[j][k + 1])
51    l += 1;
52             if (0 <= (j + 1) && (j + 1) < 10 && 0 <= (k - 1) && (k - 1) < 10 && a[j + 1][k - 1])
53
54
55
56                l += 1;
57          if (0 <= (j + 1) && (j + 1) < 10 && 0 <= k && k < 10 && a[j + 1][k])
58 l += 1;
59             if (0 <= (j + 1) && (j + 1) < 10 && 0 <= (k + 1) && (k + 1) < 10 && a[j + 1][k + 1])
60       l += 1;
```

Řešení verze 1 – na formátování přece nezáleží! (pokrač.)

```
61            if (a[j][k] && l == 2 || l == 3)
62
63
64    b[j][k] = true;
65
66          else
67                b[j][k] = false;
68       }
69 for (j = 0; j < 10; j++)
70                for (k = 0; k < 10; k++)
71 a[j][k] = b[j][k];
72       cout << "Generace #" << i << endl;
73            for (j = 0; j < 10; j++)
74 {
75                for (k = 0; k < 10; k++)
76          {
77       if (a[j][k])
78    cout << "X";
79       else
80
81 cout << ".";
82    }
83
84    cout << endl;
85 }
86
87    cout << endl;
88          }
89       }
```

Řešení verze 2 – zdrojový kód je přece pro počítač!

```
 1 #include <iostream>
 2 using namespace std;
 3 void main(){bool a[10][10],b[10][10];int i,j,k,l;for(i=0;
 4 i<10;i++)for(j=0;j<10;j++)a[i][j]=false;for(i=0;i<50;
 5 i++){j=rand()%10;k=rand()%10;a[j][k]=true;}cout<<
 6 "Puvodni kolonie"<<endl;for(i=0;i<10;i++){for(j=0;j<10;
 7 j++){if(a[i][j])cout<<"X";else cout<<".";}cout<<endl
 8 ;}cout<<endl;for(i=1;i<= 10;i++){for(j=0;j<10;j++
 9 )for(k=0;k<10;k++){l=0;if(0<=(j-1)&&(j-1)<10&&0<=
10 (k-1)&&(k-1)<10&&a[j-1][k-1])l+=1;if(0<=(j-1)&&(j-1)
11 <10&&0<= k&&k<10&&a[j-1][k])l+=1;if(0<=(j-1)&&(j-1)
12 <10&&0<=(k+1)&&(k+1)<10&&a[j-1][k+1])l+=1;if(0<=j
13 &&j<10&&0<=(k-1)&&(k-1)<10&&a[j][k-1])l+=1;if(0<=
14 j&&j<10&&0<=(k+1)&&(k+1)<10&&a[j][k+1])l+=1;if(0<=(
15 j+1)&&(j+1)<10&&0<=(k-1)&&(k-1)<10&&a[j+1][k-1])l
16 +=1;if(0<=(j+1)&&(j+1)<10&&0<= k&&k<10&&a[j+1][k
17 ])l+=1;if(0<=(j+1)&&(j+1)<10&&0<=(k+1)&&(k+1)<10
18 &&a[j+1][k+1])l+=1;if(a[j][k]&&l==2||l == 3)b[j]
19 [k]=true;else b[j][k]=false;}for(j=0;j<10;j++)for
20 (k=0;k<10;k++)a[j][k]=b[j][k];cout<<"Generace #"
21 <<i<<endl;for(j=0;j<10;j++){for(k=0;k<10;k++){if(a[j]
22 [k])cout<<"X";else cout<<".";}cout<<endl;}cout<<endl;}}
```

Řešení verze 3 – tohle je přece jen čitelnější, ne?

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 void main()
 6 {
 7    bool a[10][10], b[10][10];
 8    int i, j, k, l;
 9    for (i = 0; i < 10; i++)
10       for (j = 0; j < 10; j++)
11          a[i][j] = false;
12    for (i = 0; i < 50; i++)
13    {
14       j = rand() % 10;
15       k = rand() % 10;
16       a[j][k] = true;
17    }
18    cout << "Puvodni kolonie" << endl;
19    for (i = 0; i < 10; i++)
20    {
21       for (j = 0; j < 10; j++)
22       {
23          if (a[i][j])
24             cout << "X";
25          else
26             cout << ".";
27       }
28       cout << endl;
29    }
30    cout << endl;
```

Řešení verze 3 – tohle je přece jen čitelnější, ne? (pokrač.)

```
31    for (i = 1; i <= 10; i++)
32    {
33       for (j = 0; j < 10; j++)
34          for (k = 0; k < 10; k++)
35          {
36             l = 0;
37             if (0 <= (j - 1) && (j - 1) < 10 && 0 <= (k - 1) && (k - 1) < 10 && a[j - 1][k - 1])
38                l += 1;
39             if (0 <= (j - 1) && (j - 1) < 10 && 0 <= k && k < 10 && a[j - 1][k])
40                l += 1;
41             if (0 <= (j - 1) && (j - 1) < 10 && 0 <= (k + 1) && (k + 1) < 10 && a[j - 1][k + 1])
42                l += 1;
43             if (0 <= j && j < 10 && 0 <= (k - 1) && (k - 1) < 10 && a[j][k - 1])
44                l += 1;
45             if (0 <= j && j < 10 && 0 <= (k + 1) && (k + 1) < 10 && a[j][k + 1])
46                l += 1;
47             if (0 <= (j + 1) && (j + 1) < 10 && 0 <= (k - 1) && (k - 1) < 10 && a[j + 1][k - 1])
48                l += 1;
49             if (0 <= (j + 1) && (j + 1) < 10 && 0 <= k && k < 10 && a[j + 1][k])
50                l += 1;
51             if (0 <= (j + 1) && (j + 1) < 10 && 0 <= (k + 1) && (k + 1) < 10 && a[j + 1][k + 1])
52                l += 1;
53             if (a[j][k] && l == 2 || l == 3)
54                b[j][k] = true;
55             else
56                b[j][k] = false;
57          }
58       for (j = 0; j < 10; j++)
59          for (k = 0; k < 10; k++)
60             a[j][k] = b[j][k];
```

Řešení verze 3 – tohle je přece jen čitelnější, ne? (pokrač.)

```
61        cout << "Generace #" << i << endl;
62        for (j = 0; j < 10; j++)
63        {
64           for (k = 0; k < 10; k++)
65           {
66              if (a[j][k])
67                 cout << "X";
68              else
69                 cout << ".";
70           }
71           cout << endl;
72        }
73        cout << endl;
74     }
75  }
```

Dosavadní řešení

Verze 1 – velice obtížně čitelné, ale přesto text připomíná vzdáleně zdrojový kód v C++.

Verze 2 – troufne si někdo odhadnout co tento zdrojový kód vlastně implementuje? Nebo v tomto zdrojovém kódu najít chybu?

Verze 3 – dosud nejlepší zdrojový kód

- Kladné rysy
  1. přijatelná čitelnost díky formátování zdrojového kódu
- Záporné rysy
  1. nulové strukturování zdrojového kódu – veškerý kód ve funkci `main`,
  2. přímo zapsané číselné hodnoty
  3. a mnoho dalších ...

Pokusíme se proto zdrojový kód zlepšit.

Přímo zapsané číselné hodnoty

```
1 for (i = 1; i <= 10; i++)
2 {
3    for (j = 0; j < 10; j++)
4       for (k = 0; k < 10; k++)
5       {
6          l = 0;
7          if (0 <= (j - 1) && (j - 1) < 10 && 0 <= (k - 1) && (k - 1) <
               10 && a[j - 1][k - 1] == 1) l += 1;
```

- Jaký je význam jednotlivých čísel 10?
- V zadání je 10 generací, velikost kolonie 10×10 buněk.
- Změníme zadání a budeme chtít vypsat 20 generací. Kterou desítku máme přepsat a kterou ne? Přepsali jsme všechny?

Řešením je použít místo přímo zapsaných číselných hodnot konstanty.

Řešení verze 4 – konstanty

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int NumberOfRows = 10;
6  const int NumberOfColumns = 10;
7  const int NumberOfGenerations = 10;
8  const int NumberOfLivingCells = 50;
9
10 void main()
11 {
12    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
13    int i, j, k, l;
14    for (i = 0; i < NumberOfRows; i++)
15       for (j = 0; j < NumberOfColumns; j++)
16          a[i][j] = false;
17    for (i = 0; i < NumberOfLivingCells; i++)
18    {
19       j = rand() % NumberOfRows;
20       k = rand() % NumberOfColumns;
21       a[j][k] = true;
22    }
23    cout << "Puvodni kolonie" << endl;
24    for (i = 0; i < NumberOfRows; i++)
25    {
26       for (j = 0; j < NumberOfColumns; j++)
27       {
28          if (a[i][j])
29             cout << "X";
30          else
```

Řešení verze 4 – konstanty (pokrač.)

```
31              cout << ".";
32          }
33          cout << endl;
34      }
35      cout << endl;
36      for (i = 1; i <= NumberOfGenerations; i++)
37      {
38          for (j = 0; j < NumberOfRows; j++)
39              for (k = 0; k < NumberOfColumns; k++)
40              {
41                  l = 0;
42                  if (0 <= (j - 1) && (j - 1) < NumberOfRows && 0 <= (k - 1) && (k - 1) < NumberOfColumns && a[j
                            - 1][k - 1])
43                      l += 1;
44                  if (0 <= (j - 1) && (j - 1) < NumberOfRows && 0 <= k && k < NumberOfColumns && a[j - 1][k])
45                      l += 1;
46                  if (0 <= (j - 1) && (j - 1) < NumberOfRows && 0 <= (k + 1) && (k + 1) < NumberOfColumns && a[j
                            - 1][k + 1])
47                      l += 1;
48                  if (0 <= j && j < NumberOfRows && 0 <= (k - 1) && (k - 1) < NumberOfColumns && a[j][(k - 1)])
49                      l += 1;
50                  if (0 <= j && j < NumberOfRows && 0 <= (k + 1) && (k + 1) < NumberOfColumns && a[j][k + 1])
51                      l += 1;
52                  if (0 <= (j + 1) && (j + 1) < NumberOfRows && 0 <= (k - 1) && (k - 1) < NumberOfColumns && a[j
                            + 1][k - 1])
53                      l += 1;
54                  if (0 <= (j + 1) && (j + 1) < NumberOfRows && 0 <= k && k < NumberOfColumns && a[j + 1][k])
55                      l += 1;
56                  if (0 <= (j + 1) && (j + 1) < NumberOfRows && 0 <= (k + 1) && (k + 1) < NumberOfColumns && a[j
                            + 1][k + 1])
```

Řešení verze 4 – konstanty (pokrač.)

```
57                l += 1;
58             if (a[j][k] && l == 2 || l == 3)
59                b[j][k] = true;
60             else
61                b[j][k] = false;
62          }
63       for (j = 0; j < NumberOfRows; j++)
64          for (k = 0; k < NumberOfColumns; k++)
65             a[j][k] = b[j][k];
66       cout << "Generace #" << i << endl;
67       for (j = 0; j < NumberOfRows; j++)
68       {
69          for (k = 0; k < NumberOfColumns; k++)
70          {
71             if (a[j][k])
72                cout << "X";
73             else
74                cout << ".";
75          }
76          cout << endl;
77       }
78       cout << endl;
79    }
80 }
```

Opakující se části kódu

- Prozkoumejme řádky 42 až 57 – celkem 8(!) velice podobných řádků.
- Řádky testují všech 8 buněk v Moorově okolí.

`a[j-1][k-1]`	`a[j-1][k]`	`a[j-1][k+1]`
`a[j][k-1]`	`a[j][k]`	`a[j][k+1]`
`a[j+1][k-1]`	`a[j+1][k]`	`a[j+1][k+1]`

- Opakující se řádky kódu se liší jen ± 1 v indexech.
- Nebylo by vhodnější řešit opakování cyklem či cykly?
- Cykly $p = j - 1, \ldots, j + 1$ a $q = k - 1, \ldots k + 1$?
- Vnější cyklus proběhne $3\times$ a vnitřní také $3\times$, celkem $9\times$. Moorovo okolí má ale jen 8 buněk.
- Musíme totiž vyloučit případ kdy $p = j \vee q = k$!

Opakující se části kódu (pokrač.)

```
1 for (p = j - 1; p <= j + 1; p++)
2    for (q = k - 1; q <= k + 1; q++)
3       if ((p != j || q != k) && 0 <= p && p < NumberOfRows && 0 <=
             q && q < NumberOfColumns && a[p][q])
4          l += 1;
```

Řešení verze 5 – opakující se části kódu

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int NumberOfRows = 10;
6  const int NumberOfColumns = 10;
7  const int NumberOfGenerations = 10;
8  const int NumberOfLivingCells = 50;
9
10 void main()
11 {
12    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
13    int i, j, k, l, p, q;
14    for (i = 0; i < NumberOfRows; i++)
15       for (j = 0; j < NumberOfColumns; j++)
16          a[i][j] = false;
17    for (i = 0; i < NumberOfLivingCells; i++)
18    {
19       j = rand() % NumberOfRows;
20       k = rand() % NumberOfColumns;
21       a[j][k] = true;
22    }
23    cout << "Puvodni kolonie" << endl;
24    for (i = 0; i < NumberOfRows; i++)
25    {
26       for (j = 0; j < NumberOfColumns; j++)
27       {
28          if (a[i][j])
29             cout << "X";
30          else
```

Řešení verze 5 – opakující se části kódu (pokrač.)

```
31            cout << ".";
32         }
33         cout << endl;
34      }
35      cout << endl;
36      for (i = 1; i <= NumberOfGenerations; i++)
37      {
38         for (j = 0; j < NumberOfRows; j++)
39            for (k = 0; k < NumberOfColumns; k++)
40            {
41               l = 0;
42               for (p = j - 1; p <= j + 1; p++)
43                  for (q = k - 1; q <= k + 1; q++)
44                     if ((p != j || q != k) && 0 <= p && p < NumberOfRows && 0 <= q && q < NumberOfColumns &&
                           a[p][q])
45                        l += 1;
46               if (a[j][k] && l == 2 || l == 3)
47                  b[j][k] = true;
48               else
49                  b[j][k] = false;
50            }
51         for (j = 0; j < NumberOfRows; j++)
52            for (k = 0; k < NumberOfColumns; k++)
53               a[j][k] = b[j][k];
54         cout << "Generace #" << i << endl;
55         for (j = 0; j < NumberOfRows; j++)
56         {
57            for (k = 0; k < NumberOfColumns; k++)
58            {
59               if (a[j][k])
```

Řešení verze 5 – opakující se části kódu (pokrač.)

```
60                cout << "X";
61             else
62                cout << ".";
63          }
64          cout << endl;
65       }
66       cout << endl;
67    }
68 }
```

Nepřehledná podmínka

Podívejme se ještě jednou na podmínku uvnitř cyklů

```
1 for (p = j - 1; p <= j + 1; p++)
2   for (q = k - 1; q <= k + 1; q++)
3     if ((p != j || q != k) && 0 <= p && p < NumberOfRows && 0 <= q &&
          q < NumberOfColumns && a[p][q])
4       l += 1;
```

- Je složena z celkem šesti nerovností a jedné rovnosti – což není příliš přehledné.
- Zkoumá se tu najednou několik záležitostí:
  1. vynechání buňky ve středu okolí `(p != j || q != k)`
  2. test existence buňky
     `0 <= p && p < NumberOfRows && 0 <= q && q < NumberOfColumns`
  3. test zda je buňka živá `a[p][q]`
- Pokud toto vše platí zároveň, inkrementuje se počitadlo `l` živých buněk v Moorově okolí.

Nepřehledná podmínka (pokrač.)

Řešením je přesun některých podmínek do samostatné funkce.

```
1 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns],
      const int Row, const int Column)
2 {
3    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column <
        NumberOfColumns)
4    {
5       return Colony[Row][Column];
6    }
7    return false;
8 }
```

Řešení verze 6 – test živosti buňky jako funkce

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 void main()
20 {
21    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
22    int i, j, k, l, p, q;
23    for (i = 0; i < NumberOfRows; i++)
24       for (j = 0; j < NumberOfColumns; j++)
25          a[i][j] = false;
26    for (i = 0; i < NumberOfLivingCells; i++)
27    {
28       j = rand() % NumberOfRows;
29       k = rand() % NumberOfColumns;
30       a[j][k] = true;
```

Řešení verze 6 – test živosti buňky jako funkce (pokrač.)

```
31    }
32    cout << "Puvodni kolonie" << endl;
33    for (i = 0; i < NumberOfRows; i++)
34    {
35       for (j = 0; j < NumberOfColumns; j++)
36       {
37          if (a[i][j])
38             cout << "X";
39          else
40             cout << ".";
41       }
42       cout << endl;
43    }
44    cout << endl;
45    for (i = 1; i <= NumberOfGenerations; i++)
46    {
47       for (j = 0; j < NumberOfRows; j++)
48          for (k = 0; k < NumberOfColumns; k++)
49          {
50             l = 0;
51             for (p = j - 1; p <= j + 1; p++)
52                for (q = k - 1; q <= k + 1; q++)
53                   if ((p != j || q != k) && IsCellAlive(a, p, q))
54                      l += 1;
55             if (a[j][k] && l == 2 || l == 3)
56                b[j][k] = true;
57             else
58                b[j][k] = false;
59          }
60       for (j = 0; j < NumberOfRows; j++)
```

Řešení verze 6 – test živosti buňky jako funkce (pokrač.)

```
61          for (k = 0; k < NumberOfColumns; k++)
62             a[j][k] = b[j][k];
63       cout << "Generace #" << i << endl;
64       for (j = 0; j < NumberOfRows; j++)
65       {
66          for (k = 0; k < NumberOfColumns; k++)
67          {
68             if (a[j][k])
69                cout << "X";
70             else
71                cout << ".";
72          }
73          cout << endl;
74       }
75       cout << endl;
76    }
77 }
```

Počet vnořených cyklů

Podívejme se ještě jednou cykly řešící naši úlohu:

```
1 for (i = 1; i <= NumberOfGenerations; i++)
2 {
3    for (j = 0; j < NumberOfRows; j++)
4       for (k = 0; k < NumberOfColumns; k++)
5       {
6          l = 0;
7          for (p = j - 1; p <= j + 1; p++)
8             for (q = k - 1; q <= k + 1; q++)
9                if ((p != j || q != k) && IsCellAlive(a, p, q))
```

Odstranili jsme sice osminásobné opakování téměř identické podmínky, ale přibyly nám další dva vnořené cykly. **Pět** (!!!) vnořených cyklů není asi zrovna přehledné řešení.

Počet vnořených cyklů (pokrač.)

Úrovně vnoření:

1. iterace přes generace kolonie,
2. iterace přes řádky kolonie,
3. iterace přes sloupce kolonie,
4. iterace přes řádky Moorova okolí a
5. iterace přes sloupce Moorova okolí aktuální buňky.

Řešení?

- Iterace přes všechny buňky Moorova okolí přesunout do samostatné funkce.
- Tyto dva cykly řeší jen lokální, samostatnou úlohu.

Počet vnořených cyklů (pokrač.)

```
1  int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][
       NumberOfColumns], const int Row, const int Column)
2  {
3     int Counter = 0;
4     for (int i = Row - 1; i <= Row + 1; i++)
5     {
6        for (int j = Column - 1; j <= Column + 1; j++)
7        {
8           if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
9           {
10             Counter += 1;
11          }
12       }
13    }
14    return Counter;
15 }
```

Řešení verze 8 – počet živých sousedů implementován jako funkce

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
       Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
27          {
```

Řešení verze 8 – počet živých sousedů implementován jako funkce (pokrač.)

```
28            Counter += 1;
29         }
30      }
31   }
32   return Counter;
33 }
34
35 void main()
36 {
37   bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
38   int i, j, k, l, p, q;
39   for (i = 0; i < NumberOfRows; i++)
40      for (j = 0; j < NumberOfColumns; j++)
41         a[i][j] = false;
42   for (i = 0; i < NumberOfLivingCells; i++)
43   {
44      j = rand() % NumberOfRows;
45      k = rand() % NumberOfColumns;
46      a[j][k] = true;
47   }
48   cout << "Puvodni kolonie" << endl;
49   for (i = 0; i < NumberOfRows; i++)
50   {
51      for (j = 0; j < NumberOfColumns; j++)
52      {
53         if (a[i][j])
54            cout << "X";
```

Řešení verze 8 – počet živých sousedů implementován jako funkce (pokrač.)

```
55          else
56             cout << ".";
57       }
58       cout << endl;
59    }
60    cout << endl;
61    for (i = 1; i <= NumberOfGenerations; i++)
62    {
63       for (j = 0; j < NumberOfRows; j++)
64          for (k = 0; k < NumberOfColumns; k++)
65          {
66             l = NumberOfLivingNeighbors(a, j, k);
67             if (a[j][k] && l == 2 || l == 3)
68                b[j][k] = true;
69             else
70                b[j][k] = false;
71          }
72       for (j = 0; j < NumberOfRows; j++)
73          for (k = 0; k < NumberOfColumns; k++)
74             a[j][k] = b[j][k];
75       cout << "Generace #" << i << endl;
76       for (j = 0; j < NumberOfRows; j++)
77       {
78          for (k = 0; k < NumberOfColumns; k++)
79          {
80             if (a[j][k])
81                cout << "X";
```

Řešení verze 8 – počet živých sousedů implementován jako funkce (pokrač.)

```
82              else
83                  cout << ".";
84          }
85          cout << endl;
86      }
87      cout << endl;
88   }
89 }
```

Zjednodušení výpočtu stavu buňky v nové generaci

Podívejme se na výpočet stavu nové buňky

```
1 if (a[j][k] && l == 2 || l == 3)
2    b[j][k] = true;
3 else
4    b[j][k] = false;
```

Co tato konstrukce znamená?

- Pokud je výraz v podmínce pravdivý, `true`, nastavíme `b[j][k]` na `true`.
- Pokud je nepravdivý, `false`, nastavíme `b[j][k]` na `false`.

Přiřazení tak vlastně jen kopíruje výsledek výrazu v podmínce. Nešlo by to `b[j][k]` přiřadit rovnou výsledek výrazu v podmínce? Ano, šlo.

```
1 b[j][k] = a[j][k] && l == 2 || l == 3;
```

Řešení verze 9 – zjednodušení výpočtu stavu buňky v nové generaci

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
         Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
```

Řešení verze 9 – zjednodušení výpočtu stavu buňky v nové generaci (pokrač.)

```
27          {
28             Counter += 1;
29          }
30       }
31    }
32    return Counter;
33 }
34
35 void main()
36 {
37    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
38    int i, j, k, l, p, q;
39    for (i = 0; i < NumberOfRows; i++)
40       for (j = 0; j < NumberOfColumns; j++)
41          a[i][j] = false;
42    for (i = 0; i < NumberOfLivingCells; i++)
43    {
44       j = rand() % NumberOfRows;
45       k = rand() % NumberOfColumns;
46       a[j][k] = true;
47    }
48    cout << "Puvodni kolonie" << endl;
49    for (i = 0; i < NumberOfRows; i++)
50    {
51       for (j = 0; j < NumberOfColumns; j++)
52       {
53          if (a[i][j])
```

Řešení verze 9 – zjednodušení výpočtu stavu buňky v nové generaci (pokrač.)

```
54            cout << "X";
55         else
56            cout << ".";
57      }
58      cout << endl;
59   }
60   cout << endl;
61   for (i = 1; i <= NumberOfGenerations; i++)
62   {
63      for (j = 0; j < NumberOfRows; j++)
64         for (k = 0; k < NumberOfColumns; k++)
65         {
66            int l = NumberOfLivingNeighbors(a, j, k);
67            b[j][k] = a[j][k] && l == 2 || l == 3;
68         }
69      for (j = 0; j < NumberOfRows; j++)
70         for (k = 0; k < NumberOfColumns; k++)
71            a[j][k] = b[j][k];
72      cout << "Generace #" << i << endl;
73      for (j = 0; j < NumberOfRows; j++)
74      {
75         for (k = 0; k < NumberOfColumns; k++)
76         {
77            if (a[j][k])
78               cout << "X";
79            else
80               cout << ".";
```

Řešení verze 9 – zjednodušení výpočtu stavu buňky v nové generaci (pokrač.)

```
81          }
82          cout << endl;
83       }
84       cout << endl;
85    }
86 }
```

Identifikace významu částí kódu

Pokud se podíváme na současný stav zdrojového kódu, tak zjistíme:

- podařilo se nám odstranit množství duplicitních testů (testování Moorova okolí), ale
- dosud je v cyklu iterujícím přes generace příliš mnoho vnořených cyklů.

Naší snahou nyní bude

1. identifikovat části kódu, které lze významově ohraničit, části které lze pojmenovat,
2. tyto části kódu přesunout do samostatných funkcí a
3. nahradit identifikované části kódu voláním nových funkcí.

V kódu lze identifikovat tyto části:

- řádky 63 až 68 – výpočet nové kolonie buněk,
- řádky 69 až 71 – přepsání současné kolonie buněk novou kolonií,
- řádky 73 až 84 – tisk současné kolonie buněk,

Identifikace významu částí kódu (pokrač.)

- řádky 49 až 60 – tisk současné kolonie buněk, identická kopie řádků 73 až 84 (!!!) a
- řádky 39 až 47 – inicializace počáteční kolonie buněk.

Přesunutím těchto částí do funkcí získáme:

1. vyšší modularitu programu – funkce jako menší samostatné části lze samostatně implementovat, ladit, testovat a znovupoužívat
2. vyšší čitelnost celého kódu i funkce `main` samotné.

V dalších fází zlepšování kódu budeme pracovat s těmito identifikovanými částmi.

Výpočet nové generace

Tato část řeší výpočet nové generace – vyjmeme ji a vytvoříme z ní funkci.

```
1 for (j = 0; j < NumberOfRows; j++)
2    for (k = 0; k < NumberOfColumns; k++)
3    {
4       int l = NumberOfLivingNeighbors(a, j, k);
5       b[j][k] = a[j][k] && l == 2 || l == 3;
6    }
```

```
1 void CreateNewColony(const bool CurrentColony[NumberOfRows][
      NumberOfColumns], bool NewColony[NumberOfRows][NumberOfColumns])
2 {
3    for (int i = 0; i < NumberOfRows; i++)
4    {
5       for (int j = 0; j < NumberOfColumns; j++)
6       {
7          int l = NumberOfLivingNeighbors(CurrentColony, i, j);
8          NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
9       }
```

Výpočet nové generace (pokrač.)

```
10      }
11  }
```

Řešení verze 10 – výpočet nové generace buněk jako funkce

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
       Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
27          {
```

Řešení verze 10 – výpočet nové generace buněk jako funkce (pokrač.)

```
28            Counter += 1;
29         }
30      }
31   }
32   return Counter;
33 }
34
35 void CreateNewColony(const bool CurrentColony[NumberOfRows][NumberOfColumns], bool NewColony[NumberOfRows
       ][NumberOfColumns])
36 {
37   for (int i = 0; i < NumberOfRows; i++)
38   {
39      for (int j = 0; j < NumberOfColumns; j++)
40      {
41         int l = NumberOfLivingNeighbors(CurrentColony, i, j);
42         NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
43      }
44   }
45 }
46
47 void main()
48 {
49   bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
50   int i, j, k, l, p, q;
51   for (i = 0; i < NumberOfRows; i++)
52      for (j = 0; j < NumberOfColumns; j++)
53         a[i][j] = false;
```

Řešení verze 10 – výpočet nové generace buněk jako funkce (pokrač.)

```
54    for (i = 0; i < NumberOfLivingCells; i++)
55    {
56       j = rand() % NumberOfRows;
57       k = rand() % NumberOfColumns;
58       a[j][k] = true;
59    }
60    cout << "Puvodni kolonie" << endl;
61    for (i = 0; i < NumberOfRows; i++)
62    {
63       for (j = 0; j < NumberOfColumns; j++)
64       {
65          if (a[i][j])
66             cout << "X";
67          else
68             cout << ".";
69       }
70       cout << endl;
71    }
72    cout << endl;
73    for (i = 1; i <= NumberOfGenerations; i++)
74    {
75       CreateNewColony(a, b);
76       for (j = 0; j < NumberOfRows; j++)
77          for (k = 0; k < NumberOfColumns; k++)
78             a[j][k] = b[j][k];
79       cout << "Generace #" << i << endl;
80       for (j = 0; j < NumberOfRows; j++)
```

Řešení verze 10 – výpočet nové generace buněk jako funkce (pokrač.)

```
81      {
82         for (k = 0; k < NumberOfColumns; k++)
83         {
84            if (a[j][k])
85               cout << "X";
86            else
87               cout << ".";
88         }
89         cout << endl;
90      }
91      cout << endl;
92   }
93 }
```

Výstup kolonie na standardní výstup

Fragment kódu

```
1 for (i = 0; i < NumberOfRows; i++)
2 {
3    for (j = 0; j < NumberOfColumns; j++)
4    {
5       if (a[i][j])
6          cout << "X";
7       else
8          cout << ".";
9    }
10   cout << endl;
11 }
12 cout << endl;
```

nahradíme funkcí

Výstup kolonie na standardní výstup (pokrač.)

```
1 void PrintColony(const bool Colony[NumberOfRows][NumberOfColumns])
2 {
3    for (int i = 0; i < NumberOfRows; i++)
4    {
5       for (int j = 0; j < NumberOfColumns; j++)
6       {
7          if (Colony[i][j])
8             cout << "X";
9          else
10            cout << ".";
11      }
12      cout << endl;
13   }
14   cout << endl;
15 }
```

Řešení verze 11 – výstup kolonie na standardní výstup jako funkce

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
       Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
27          {
```

Řešení verze 11 – výstup kolonie na standardní výstup jako funkce (pokrač.)

```
28            Counter += 1;
29         }
30      }
31   }
32   return Counter;
33 }
34
35 void CreateNewColony(const bool CurrentColony[NumberOfRows][NumberOfColumns], bool NewColony[NumberOfRows
       ][NumberOfColumns])
36 {
37   for (int i = 0; i < NumberOfRows; i++)
38   {
39      for (int j = 0; j < NumberOfColumns; j++)
40      {
41         int l = NumberOfLivingNeighbors(CurrentColony, i, j);
42         NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
43      }
44   }
45 }
46
47 void PrintColony(const bool Colony[NumberOfRows][NumberOfColumns])
48 {
49   for (int i = 0; i < NumberOfRows; i++)
50   {
51      for (int j = 0; j < NumberOfColumns; j++)
52      {
53         if (Colony[i][j])
```

Řešení verze 11 – výstup kolonie na standardní výstup jako funkce (pokrač.)

```
54            cout << "X";
55         else
56            cout << ".";
57      }
58      cout << endl;
59   }
60   cout << endl;
61 }
62
63 void main()
64 {
65   bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
66   int i, j, k, l, p, q;
67   for (i = 0; i < NumberOfRows; i++)
68      for (j = 0; j < NumberOfColumns; j++)
69         a[i][j] = false;
70   for (i = 0; i < NumberOfLivingCells; i++)
71   {
72      j = rand() % NumberOfRows;
73      k = rand() % NumberOfColumns;
74      a[j][k] = true;
75   }
76   cout << "Puvodni kolonie" << endl;
77   PrintColony(a);
78   cout << endl;
79   for (i = 1; i <= NumberOfGenerations; i++)
80   {
```

Řešení verze 11 – výstup kolonie na standardní výstup jako funkce (pokrač.)

```
81        CreateNewColony(a, b);
82        for (j = 0; j < NumberOfRows; j++)
83           for (k = 0; k < NumberOfColumns; k++)
84              a[j][k] = b[j][k];
85        cout << "Generace #" << i << endl;
86        PrintColony(a);
87     }
88  }
```

Přepsání současné kolonie buněk novou kolonií

Fragment kódu

```
1 for (j = 0; j < NumberOfRows; j++)
2    for (k = 0; k < NumberOfColumns; k++)
3       a[j][k] = b[j][k];
```

nahradíme funkcí

```
1 void CopyColony(const bool SourceColony[NumberOfRows][NumberOfColumns],
        bool DestinationColony[NumberOfRows][NumberOfColumns])
2 {
3    for (int i = 0; i < NumberOfRows; i++)
4    {
5       for (int j = 0; j < NumberOfColumns; j++)
6       {
7          DestinationColony[i][j] = SourceColony[i][j];
8       }
9    }
10 }
```

Řešení verze 12 – přepsání současné kolonie buněk novou kolonií jako funkce

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
         Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
```

Řešení verze 12 – přepsání současné kolonie buněk novou kolonií jako funkce (pokrač.)

```
27          {
28             Counter += 1;
29          }
30       }
31    }
32    return Counter;
33 }
34
35 void CreateNewColony(const bool CurrentColony[NumberOfRows][NumberOfColumns], bool NewColony[NumberOfRows
         ][NumberOfColumns])
36 {
37    for (int i = 0; i < NumberOfRows; i++)
38    {
39       for (int j = 0; j < NumberOfColumns; j++)
40       {
41          int l = NumberOfLivingNeighbors(CurrentColony, i, j);
42          NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
43       }
44    }
45 }
46
47 void PrintColony(const bool Colony[NumberOfRows][NumberOfColumns])
48 {
49    for (int i = 0; i < NumberOfRows; i++)
50    {
51       for (int j = 0; j < NumberOfColumns; j++)
52       {
```

Řešení verze 12 – přepsání současné kolonie buněk novou kolonií jako funkce (pokrač.)

```
53          if (Colony[i][j])
54             cout << "X";
55          else
56             cout << ".";
57       }
58       cout << endl;
59    }
60    cout << endl;
61 }
62
63 void CopyColony(const bool SourceColony[NumberOfRows][NumberOfColumns], bool DestinationColony[
       NumberOfRows][NumberOfColumns])
64 {
65    for (int i = 0; i < NumberOfRows; i++)
66    {
67       for (int j = 0; j < NumberOfColumns; j++)
68       {
69          DestinationColony[i][j] = SourceColony[i][j];
70       }
71    }
72 }
73
74 void main()
75 {
76    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
77    int i, j, k, l, p, q;
78    for (i = 0; i < NumberOfRows; i++)
```

Řešení verze 12 – přepsání současné kolonie buněk novou kolonií jako funkce (pokrač.)

```
79          for (j = 0; j < NumberOfColumns; j++)
80             a[i][j] = false;
81       for (i = 0; i < NumberOfLivingCells; i++)
82       {
83          j = rand() % NumberOfRows;
84          k = rand() % NumberOfColumns;
85          a[j][k] = true;
86       }
87       cout << "Puvodni kolonie" << endl;
88       PrintColony(a);
89       cout << endl;
90       for (i = 1; i <= NumberOfGenerations; i++)
91       {
92          CreateNewColony(a, b);
93          CopyColony(b, a);
94          cout << "Generace #" << i << endl;
95          PrintColony(a);
96       }
97    }
```

Inicializace počáteční kolonie buněk

Fragment kódu

```
1 for (i = 0; i < NumberOfRows; i++)
2    for (j = 0; j < NumberOfColumns; j++)
3       a[i][j] = false;
4 for (i = 0; i < NumberOfLivingCells; i++)
5 {
6    j = rand() % NumberOfRows;
7    k = rand() % NumberOfColumns;
8    a[j][k] = true;
9 }
```

nahradíme funkcí

Inicializace počáteční kolonie buněk (pokrač.)

```
1  void InitColony(bool Colony[NumberOfRows][NumberOfColumns], const int
       NumberOfLivingCells)
2  {
3     for (int i = 0; i < NumberOfRows; i++)
4     {
5        for (int j = 0; j < NumberOfColumns; j++)
6        {
7           Colony[i][j] = false;
8        }
9     }
10    for (int i = 0; i < NumberOfLivingCells; i++)
11    {
12       int r = rand() % NumberOfRows;
13       int c = rand() % NumberOfColumns;
14       Colony[r][c] = true;
15    }
16 }
```

Řešení verze 13 – inicializace počáteční kolonie buněk jako funkce

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
         Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
27          {
```

Řešení verze 13 – inicializace počáteční kolonie buněk jako funkce (pokrač.)

```
28            Counter += 1;
29         }
30      }
31   }
32   return Counter;
33 }
34
35 void CreateNewColony(const bool CurrentColony[NumberOfRows][NumberOfColumns], bool NewColony[NumberOfRows
       ][NumberOfColumns])
36 {
37   for (int i = 0; i < NumberOfRows; i++)
38   {
39      for (int j = 0; j < NumberOfColumns; j++)
40      {
41         int l = NumberOfLivingNeighbors(CurrentColony, i, j);
42         NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
43      }
44   }
45 }
46
47 void PrintColony(const bool Colony[NumberOfRows][NumberOfColumns])
48 {
49   for (int i = 0; i < NumberOfRows; i++)
50   {
51      for (int j = 0; j < NumberOfColumns; j++)
52      {
53         if (Colony[i][j])
```

Řešení verze 13 – inicializace počáteční kolonie buněk jako funkce (pokrač.)

```
54            cout << "X";
55         else
56            cout << ".";
57      }
58      cout << endl;
59   }
60   cout << endl;
61 }
62
63 void CopyColony(const bool SourceColony[NumberOfRows][NumberOfColumns], bool DestinationColony[
       NumberOfRows][NumberOfColumns])
64 {
65   for (int i = 0; i < NumberOfRows; i++)
66   {
67      for (int j = 0; j < NumberOfColumns; j++)
68      {
69         DestinationColony[i][j] = SourceColony[i][j];
70      }
71   }
72 }
73
74 void InitColony(bool Colony[NumberOfRows][NumberOfColumns], const int NumberOfLivingCells)
75 {
76   for (int i = 0; i < NumberOfRows; i++)
77   {
78      for (int j = 0; j < NumberOfColumns; j++)
79      {
```

Řešení verze 13 – inicializace počáteční kolonie buněk jako funkce (pokrač.)

```
80          Colony[i][j] = false;
81       }
82    }
83    for (int i = 0; i < NumberOfLivingCells; i++)
84    {
85       int r = rand() % NumberOfRows;
86       int c = rand() % NumberOfColumns;
87       Colony[r][c] = true;
88    }
89 }
90
91 void main()
92 {
93    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
94    int i, j, k, l, p, q;
95    InitColony(a, NumberOfLivingCells);
96    cout << "Puvodni kolonie" << endl;
97    PrintColony(a);
98    cout << endl;
99    for (i = 1; i <= NumberOfGenerations; i++)
100   {
101      CreateNewColony(a, b);
102      CopyColony(b, a);
103      cout << "Generace #" << i << endl;
104      PrintColony(a);
105   }
106 }
```

Řešení verze 13 – inicializace počáteční kolonie buněk jako funkce (pokrač.)

Je inicializace počáteční kolonie buněk korektní?

Podívejme se na náhodné generování živých buněk v počáteční kolonii

```
1 for (int i = 0; i < NumberOfLivingCells; i++)
2 {
3    int r = rand() % NumberOfRows;
4    int c = rand() % NumberOfColumns;
5    Colony[r][c] = true;
6 }
```

Problém

Při inicializaci kolonie může dojít k situaci, kdy se vygeneruje pozice již živé buňky! Buňka může tedy obživnout několikrát, což je chyba a výsledný počet živých buněk tak může být menší než je požadováno.

Je inicializace počáteční kolonie buněk korektní? (pokrač.)

Řešení

- Počítadlo živých buněk `i` můžeme tedy inkrementovat jen v okamžiku, kdy jsme vygenerovali **novou** živou buňku.
- Cyklus `for` ale proměnnou `i` inkrementuje po každém průchodu cyklem.
- Při neúspěšném generování nové buňky musíme proměnnou `i` zmenšit o 1 abychom korigovali automatickou inkrementaci po průchodu cyklem.

```
1 for (int i = 0; i < NumberOfLivingCells; i++)
2 {
3    int r = rand() % NumberOfRows;
4    int c = rand() % NumberOfColumns;
5    if (!Colony[r][c])
6    {
7       Colony[r][c] = true;
8    }
```

Je inicializace počáteční kolonie buněk korektní? (pokrač.)

```
 9     else
10     {
11         i -= 1;
12     }
13 }
```

Řešení verze 14 – korekce počitadla živých buněk

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
         Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
27          {
28             Counter += 1;
29          }
```

Řešení verze 14 – korekce počitadla živých buněk (pokrač.)

```
30       }
31    }
32    return Counter;
33 }
34
35 void CreateNewColony(const bool CurrentColony[NumberOfRows][NumberOfColumns], bool NewColony[NumberOfRows
         ][NumberOfColumns])
36 {
37    for (int i = 0; i < NumberOfRows; i++)
38    {
39       for (int j = 0; j < NumberOfColumns; j++)
40       {
41          int l = NumberOfLivingNeighbors(CurrentColony, i, j);
42          NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
43       }
44    }
45 }
46
47 void PrintColony(const bool Colony[NumberOfRows][NumberOfColumns])
48 {
49    for (int i = 0; i < NumberOfRows; i++)
50    {
51       for (int j = 0; j < NumberOfColumns; j++)
52       {
53          if (Colony[i][j])
54             cout << "X";
55          else
56             cout << ".";
57       }
58       cout << endl;
```

Řešení verze 14 – korekce počitadla živých buněk (pokrač.)

```
59    }
60    cout << endl;
61 }
62
63 void CopyColony(const bool SourceColony[NumberOfRows][NumberOfColumns], bool DestinationColony[
         NumberOfRows][NumberOfColumns])
64 {
65    for (int i = 0; i < NumberOfRows; i++)
66    {
67       for (int j = 0; j < NumberOfColumns; j++)
68       {
69          DestinationColony[i][j] = SourceColony[i][j];
70       }
71    }
72 }
73
74 void InitColony(bool Colony[NumberOfRows][NumberOfColumns], const int NumberOfLivingCells)
75 {
76    for (int i = 0; i < NumberOfRows; i++)
77    {
78       for (int j = 0; j < NumberOfColumns; j++)
79       {
80          Colony[i][j] = false;
81       }
82    }
83    for (int i = 0; i < NumberOfLivingCells; i++)
84    {
85       int r = rand() % NumberOfRows;
86       int c = rand() % NumberOfColumns;
87       if (!Colony[r][c])
```

Řešení verze 14 – korekce počitadla živých buněk (pokrač.)

```
88       {
89          Colony[r][c] = true;
90       }
91       else
92       {
93          i -= 1;
94       }
95    }
96 }
97
98 void main()
99 {
100    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
101    int i, j, k, l, p, q;
102    InitColony(a, NumberOfLivingCells);
103    cout << "Puvodni kolonie" << endl;
104    PrintColony(a);
105    cout << endl;
106    for (i = 1; i <= NumberOfGenerations; i++)
107    {
108       CreateNewColony(a, b);
109       CopyColony(b, a);
110       cout << "Generace #" << i << endl;
111       PrintColony(a);
112    }
113 }
```

Inicializace počáteční kolonie buněk ještě jednou

- Můžeme být s předchozím řešením spokojeni?
- Určitě ne!
- Současné řešení je ukázkou, jak nikdy s cyklem `for` nepracovat!
- Cyklus `for` existuje proto, aby veškerá manipulace s iterační proměnnou byla soustředěna přehledně na jednom řádku kódu.
- Pokud toto neplatí musíme
  - použít cyklus `while`, kde se předpokládá manipulace s iterační proměnnou na více řádcích nebo
  - cyklus zcela přepsat.

Zvolíme druhou možnost:

- zachováme cyklus `for` a
- tělo cyklu přepíšeme tak, aby se bez vygenerování nové živé buňky vůbec nedalo v cyklu `for` pokračovat.

Inicializace počáteční kolonie buněk ještě jednou (pokrač.)

```
1 for (int i = 0; i < NumberOfLivingCells; i++)
2 {
3    int r, c;
4    do
5    {
6       r = rand() % NumberOfRows;
7       c = rand() % NumberOfColumns;
8    } while (Colony[r][c]);
9    Colony[r][c] = true;
10 }
```

Řešení verze 15 – korekce počitadla živých buněk II

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
         Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
27          {
28             Counter += 1;
29          }
```

Řešení verze 15 – korekce počitadla živých buněk II (pokrač.)

```
30       }
31    }
32    return Counter;
33 }
34
35 void CreateNewColony(const bool CurrentColony[NumberOfRows][NumberOfColumns], bool NewColony[NumberOfRows
         ][NumberOfColumns])
36 {
37    for (int i = 0; i < NumberOfRows; i++)
38    {
39       for (int j = 0; j < NumberOfColumns; j++)
40       {
41          int l = NumberOfLivingNeighbors(CurrentColony, i, j);
42          NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
43       }
44    }
45 }
46
47 void PrintColony(const bool Colony[NumberOfRows][NumberOfColumns])
48 {
49    for (int i = 0; i < NumberOfRows; i++)
50    {
51       for (int j = 0; j < NumberOfColumns; j++)
52       {
53          if (Colony[i][j])
54             cout << "X";
55          else
```

Řešení verze 15 – korekce počitadla živých buněk II (pokrač.)

```
56             cout << ".";
57          }
58          cout << endl;
59       }
60       cout << endl;
61 }
62
63 void CopyColony(const bool SourceColony[NumberOfRows][NumberOfColumns], bool DestinationColony[
       NumberOfRows][NumberOfColumns])
64 {
65    for (int i = 0; i < NumberOfRows; i++)
66    {
67       for (int j = 0; j < NumberOfColumns; j++)
68       {
69          DestinationColony[i][j] = SourceColony[i][j];
70       }
71    }
72 }
73
74 void InitColony(bool Colony[NumberOfRows][NumberOfColumns], const int NumberOfLivingCells)
75 {
76    for (int i = 0; i < NumberOfRows; i++)
77    {
78       for (int j = 0; j < NumberOfColumns; j++)
79       {
80          Colony[i][j] = false;
81       }
```

Řešení verze 15 – korekce počitadla živých buněk II (pokrač.)

```
 82    }
 83    for (int i = 0; i < NumberOfLivingCells; i++)
 84    {
 85       int r, c;
 86       do
 87       {
 88          r = rand() % NumberOfRows;
 89          c = rand() % NumberOfColumns;
 90       } while (Colony[r][c]);
 91       Colony[r][c] = true;
 92    }
 93 }
 94
 95 void main()
 96 {
 97    bool a[NumberOfRows][NumberOfColumns], b[NumberOfRows][NumberOfColumns];
 98    int i, j, k, l, p, q;
 99    InitColony(a, NumberOfLivingCells);
100    cout << "Puvodni kolonie" << endl;
101    PrintColony(a);
102    cout << endl;
103    for (i = 1; i <= NumberOfGenerations; i++)
104    {
105       CreateNewColony(a, b);
106       CopyColony(b, a);
107       cout << "Generace #" << i << endl;
108       PrintColony(a);
```

Řešení verze 15 – korekce počitadla živých buněk II (pokrač.)

```
109     }
110 }
```

Závěrečný úklid

Program jsme kompletně přepsali. Co závěrem?

- odstranění nepoužívaných proměnných,
- výstižnější jména současné a nové kolonie buněk.

Řešení verze 15 – závěrečný úklid

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 const int NumberOfRows = 10;
 6 const int NumberOfColumns = 10;
 7 const int NumberOfGenerations = 10;
 8 const int NumberOfLivingCells = 50;
 9
10 bool IsCellAlive(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int Column)
11 {
12    if (0 <= Row && Row < NumberOfRows && 0 <= Column && Column < NumberOfColumns)
13    {
14       return Colony[Row][Column];
15    }
16    return false;
17 }
18
19 int NumberOfLivingNeighbors(const bool Colony[NumberOfRows][NumberOfColumns], const int Row, const int
         Column)
20 {
21    int Counter = 0;
22    for (int i = Row - 1; i <= Row + 1; i++)
23    {
24       for (int j = Column - 1; j <= Column + 1; j++)
25       {
26          if ((i != Row || j != Column) && IsCellAlive(Colony, i, j))
27          {
28             Counter += 1;
29          }
```

Řešení verze 15 – závěrečný úklid (pokrač.)

```
30       }
31    }
32    return Counter;
33 }
34
35 void CreateNewColony(const bool CurrentColony[NumberOfRows][NumberOfColumns], bool NewColony[NumberOfRows
         ][NumberOfColumns])
36 {
37    for (int i = 0; i < NumberOfRows; i++)
38    {
39       for (int j = 0; j < NumberOfColumns; j++)
40       {
41          int l = NumberOfLivingNeighbors(CurrentColony, i, j);
42          NewColony[i][j] = CurrentColony[i][j] && l == 2 || l == 3;
43       }
44    }
45 }
46
47 void PrintColony(const bool Colony[NumberOfRows][NumberOfColumns])
48 {
49    for (int i = 0; i < NumberOfRows; i++)
50    {
51       for (int j = 0; j < NumberOfColumns; j++)
52       {
53          if (Colony[i][j])
54             cout << "X";
55          else
56             cout << ".";
57       }
58       cout << endl;
```

Řešení verze 15 – závěrečný úklid (pokrač.)

```
59    }
60    cout << endl;
61 }
62
63 void CopyColony(const bool SourceColony[NumberOfRows][NumberOfColumns], bool DestinationColony[
         NumberOfRows][NumberOfColumns])
64 {
65    for (int i = 0; i < NumberOfRows; i++)
66    {
67       for (int j = 0; j < NumberOfColumns; j++)
68       {
69          DestinationColony[i][j] = SourceColony[i][j];
70       }
71    }
72 }
73
74 void InitColony(bool Colony[NumberOfRows][NumberOfColumns], const int NumberOfLivingCells)
75 {
76    for (int i = 0; i < NumberOfRows; i++)
77    {
78       for (int j = 0; j < NumberOfColumns; j++)
79       {
80          Colony[i][j] = false;
81       }
82    }
83    for (int i = 0; i < NumberOfLivingCells; i++)
84    {
85       int r, c;
86       do
87       {
```

Řešení verze 15 – závěrečný úklid (pokrač.)

```
88          r = rand() % NumberOfRows;
89          c = rand() % NumberOfColumns;
90       } while (Colony[r][c]);
91       Colony[r][c] = true;
92    }
93 }
94
95 void main()
96 {
97    bool CurrentColony[NumberOfRows][NumberOfColumns];
98    bool NewColony[NumberOfRows][NumberOfColumns];
99    InitColony(CurrentColony, NumberOfLivingCells);
100   cout << "Puvodni kolonie" << endl;
101   PrintColony(CurrentColony);
102   cout << endl;
103   for (int i = 1; i <= NumberOfGenerations; i++)
104   {
105      CreateNewColony(CurrentColony, NewColony);
106      CopyColony(NewColony, CurrentColony);
107      cout << "Generace #" << i << endl;
108      PrintColony(CurrentColony);
109   }
110 }
```

Refaktorování

- Refaktorování je disciplinovaný proces provádění změn v softwarovém systému takovým způsobem, že nemají vliv na vnější chování kódu, ale vylepšují jeho vnitřní strukturu s minimálním rizikem vnášení chyb.
- Při refaktoringu provádíme malé až primitivní změny, ale celkový efekt je velký a to v podobě čistšího, průhlednějšího a čitelnějšího kódu, kód se také lépe udržuje a rozšiřuje.
- Zlepšuje se také celková kvalita kódu a architektura, snižuje se počet chyb a tím i zvyšuje rychlost vývoje programu.
- Refaktoring nám pomáhá pochopit a více si ujasnit kód, což je vhodné zejména, pokud upravujeme zdrojový kód po někom jiném.

Kdy refaktorovat?

Martin Fowler uvádí 22 takzvaných „pachů v kódu“, které jsou zvěstí toho, že bychom kód měli začít refaktorovat. Nejzákladnější a většinou i nejčastější pachy, které potkáte:

Refaktorování (pokrač.)

Duplicitní kód – pokud najdeme v kódu na více místech stejnou strukturu

Dlouhá funkce – většinou se v takové funkci řeší více problémů najednou a měli bychom ji rozdělit menší funkce, které řeší právě jeden problém.

Komentáře – pokud máme tendenci komentovat uvnitř metody náš kód, je většinou lepší tento kód vyčlenit do samostatné funkce, a pomocí vhodného názvu této funkce nám odpadne nutnost komentáře

Složité struktury podmínek – vyčleněním do samostatné funkce se zlepší čitelnost kódu,

Doporučená literatura

- FOWLER, Martin. Refaktoring : Zlepšení existujícího kódu (původním názvem: Refactoring: Improving the Design of Existing Code). Překlad Vladimír Lahoda. Praha : Grada, 2003. ISBN 80-247-0299-1

Refaktorování (pokrač.)

- Refaktorování. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-10-15]. Dostupné z: `https://cs.wikipedia.org/wiki/Refaktorov%C3%A1n%C3%AD`

Poznámka na závěr aneb proč „pach kódu“?

- Označení pach kódu vymyslel Martin Fowler v době, kdy se stal otcem a měl přebalovat mimino.
- Ideální stav mimina – čisté mimino s novou plenou. Pach, v tomto případě vůně, mimina je ideální.
- Mimino se počurá – pach je už nějaký jiný.
- Mimino se pokaká – opět jiný pach.
- Mimino má průjem – děsivý pach.
- Pach tedy není fatální, život ohrožující problém, ale signalizuje nám cosi podezřelého. . .

Závěr – ucítíme-li jiný než správný pach je nutné refaktorovat – vyměnit plenu.

Děkuji za pozornost