Opakovanie

- Špeciality
  - `++i, i--, ...`
- Skrátené vyhodnotenie
- Príkazy
- elementárne príkazy
- podmienené príkazy (alternatíva)
- Iteračné príkazy (cykly)

Príklad: Výpočet odmocniny

Výpočet odmocniny $\sqrt{x}$ pomocou delenia intervalu!

Idea:

- Zvolíme Intervall [d, h], v ktorom sa $\sqrt{x}$ určite nachádza.
- Rozdelíme interval na dva intervaly [d,s] und [s,h];
- Zvolíme interval, v ktorom sa $\sqrt{x}$ určite nachádza a opakujeme procedúru, kým nedosiahneme požadovanú presnosť, t.j. kým interval nie je dostatočne malý.

Príklad: Výpočet odmocniny

```
// Vstup:  double x;  x >= 0
double d = 0;          // Spodná hranica počiatočného intervalu
double h = x + 1;      // Horná hranica počiatočného intervalu
double eps = 1.0e-10; // želaná relatívna presnosť
double s = (d + h) / 2;

while ( (h - d) / s > eps)
{if (s * s >= x)
    h = s;
  else
    d = s;
  s = (d + h) / 2;
}
// Výstup: s
```

Pozorovanie:

Iteračný príkaz tohto programu
(pre x >= 0) sa vykoná aspoň raz.

Bolo by lepšie podmienku testovať na konci cyklu.

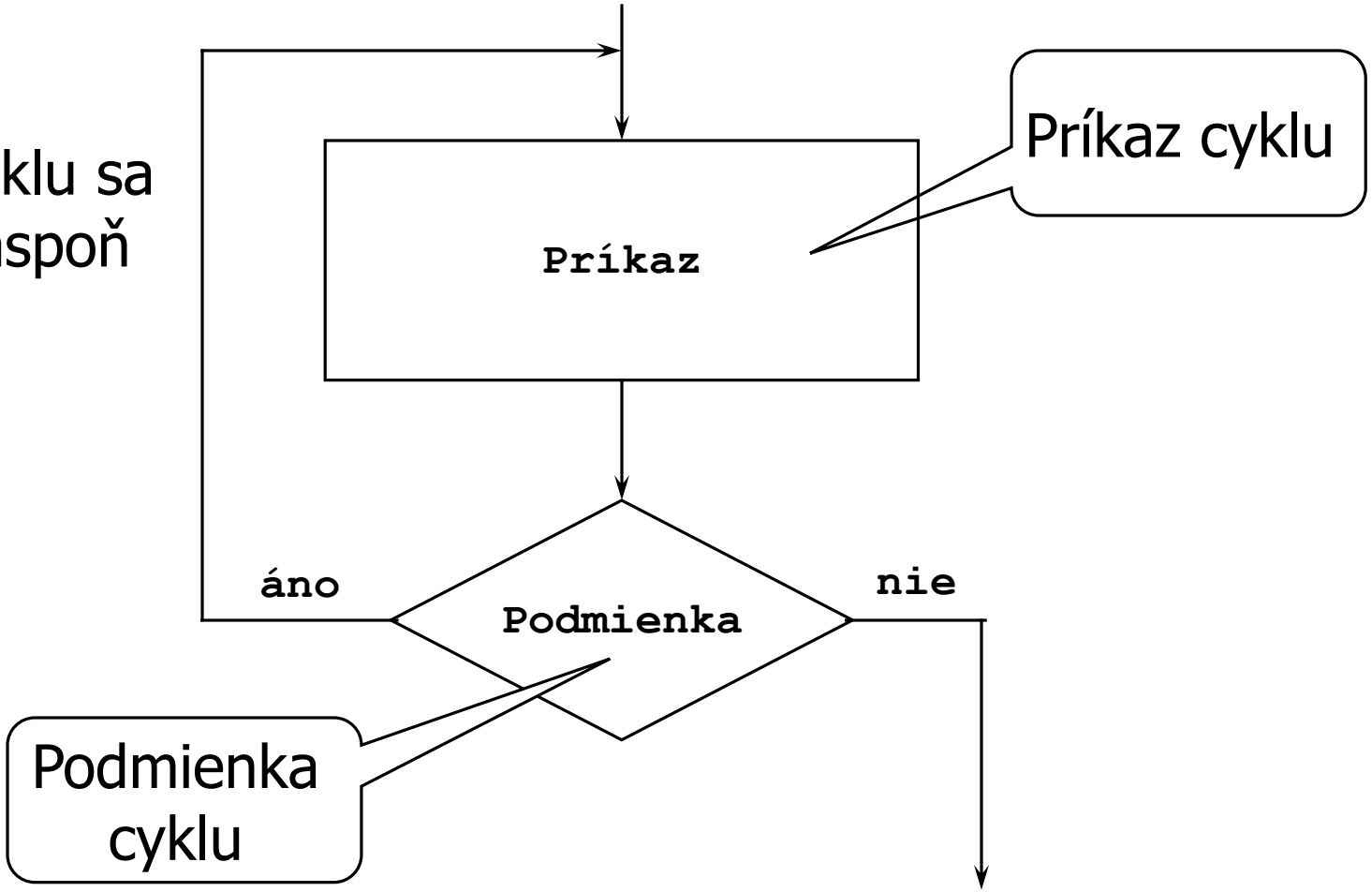
do-while-cyklus

```
// Vstup:  double x;  x >= 0
double d = 0;          // Spodná hranica počiatočného intervalu
double h = x + 1;      // Horná hranica počiatočného intervalu
double eps = 1.0e-10; // želaná relatívna presnosť
double s = (d + h) / 2;

do
{if (s * s >= x)
    h = s;
  else
    d = s;
  s = (d + h) / 2;
} while ( (h - d) / s > eps);
// Výstup: s
```

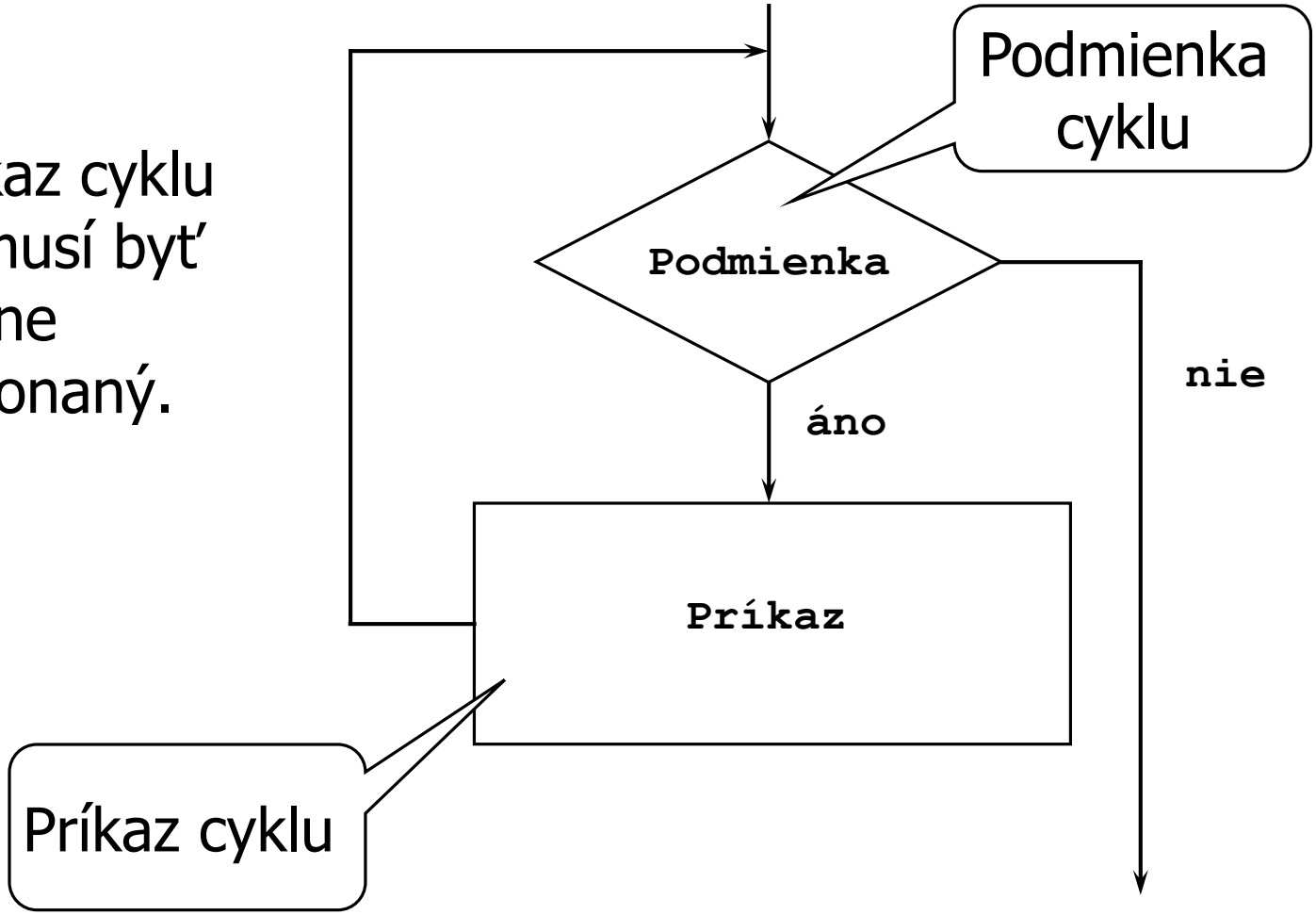
Vývojový diagram do-while-cyklu

Príkaz cyklu sa vykoná aspoň raz.



Vývojový diagram (pre porovnanie) while-(do)-cyklu

Príkaz cyklu nemusí byť nutne vykonaný.

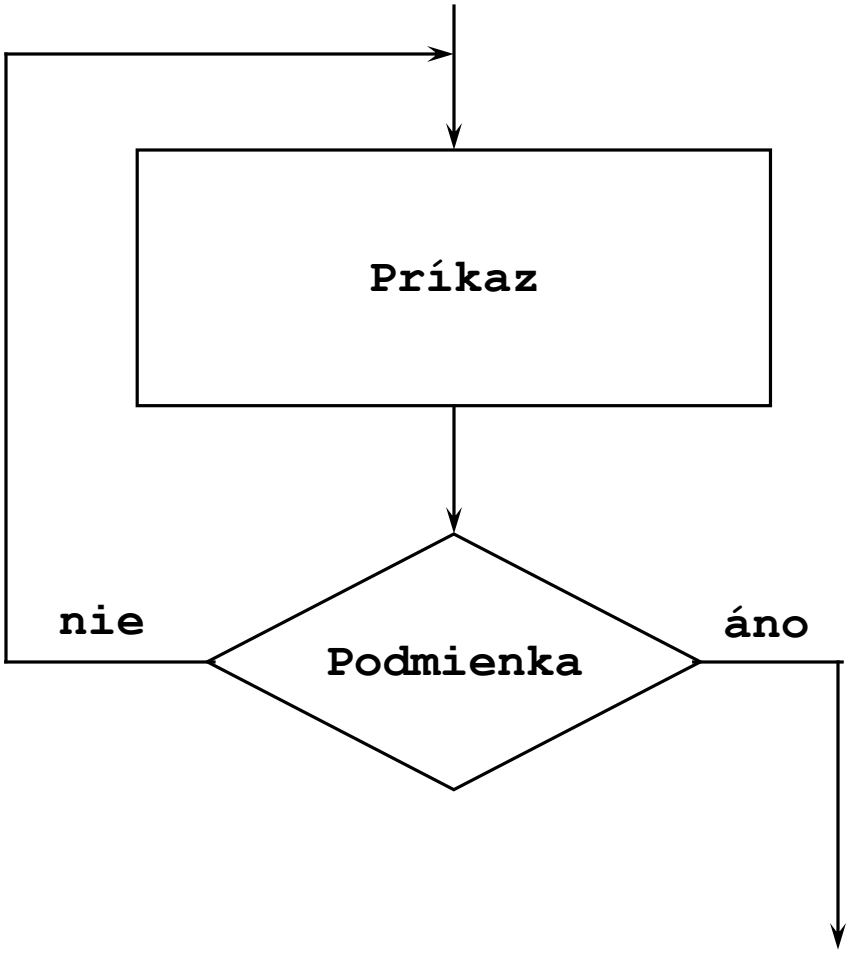


Ďalšie možnosti cyklov (nie v Jave a C)

```
repeat
   Prikaz
until ( Podmienka );
```

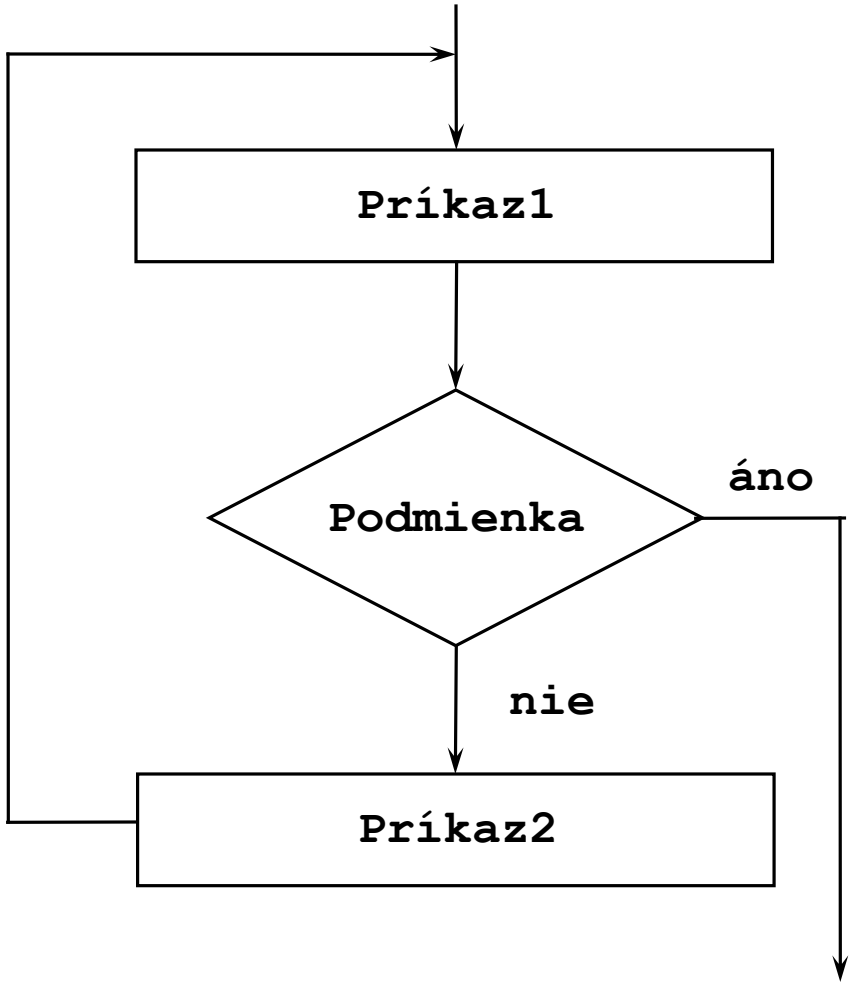
Zodpovedá v Jave a v C:

```
do
   Prikaz
while ( !Podmeinka );
```



Ďalšie možnosti cyklov (nie v Jave a C)

```
repeat
{
   Prikaz1
until ( Podmienka );
   Prikaz2
}
```



Predčasné ukončenie cyklu

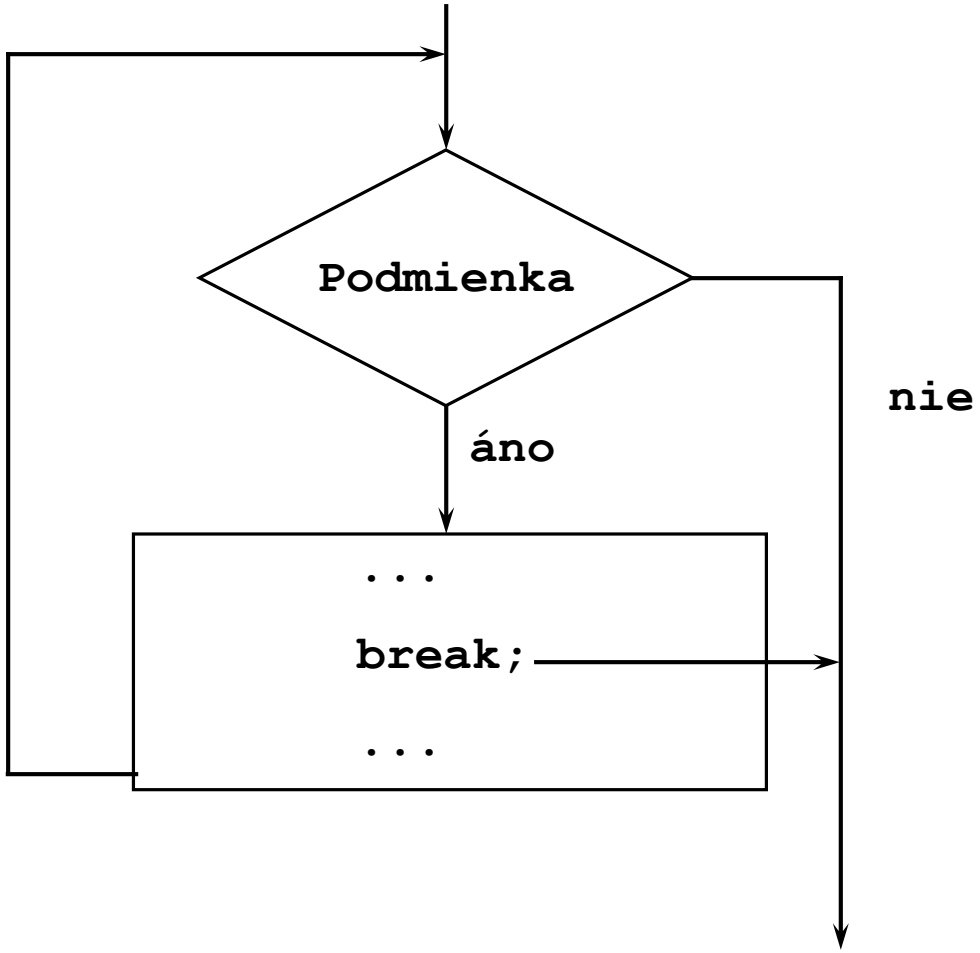
- Aby bolo možné využiť všetky možnosti cyklov, je v Jave a v C k dispozícii príkaz **`break;`**
- príkaz **`break;`** zabezpečí okamžité ukončenie cyklu

Príklad v Jave: break

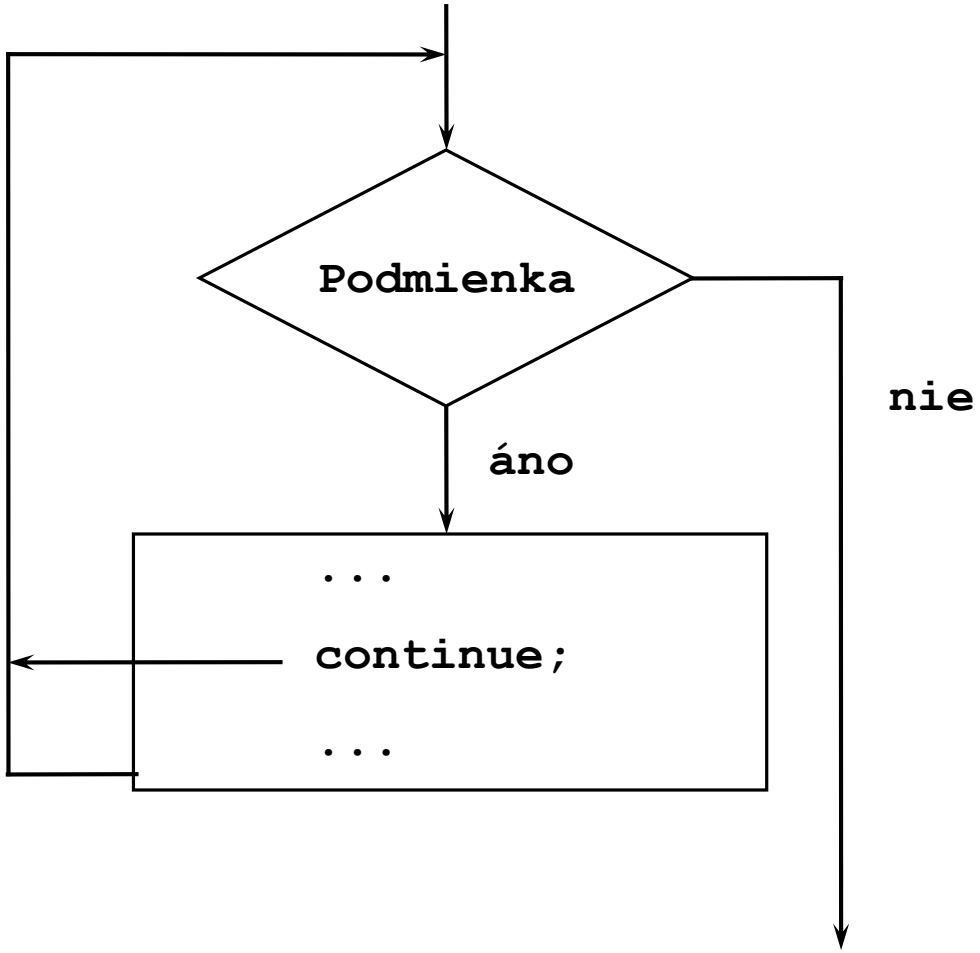
```
public class CheckNegative
{ public static void main (String[] args)
  { int n = args.length;  // Počet parametrov

    int i = 0;
    while ( i < n )
    { double x = Double.parseDouble(args[i]);
      if ( x < 0.0 )
      { System.out.println("Cislo "+ x + " je
              zaporne: Stop");
        break;}
      System.out.println( i + ": " + x );
      i = i + 1;
    }
  }
}
```

Vývojový diagram while-cyklu s príkazom `break;`



Predčasné ukončenie JEDNEJ slučky cyklu príkazom `continue;`



Iteračné príkazy nad intervalom

- V mnohých prípadoch prechádza nejaká premenná (počítadlo) v cykle hodnoty vopred známeho rozsahu (napr. 1 ... n)
  Príklady: **`Suma`**, **`Faktorial`**
- Pre takéto cykly existuje špeciálna konštrukcia: príkaz **`for`**

Príkaz: `for`-cyklus

```
double product = 1.0;

   for (int i = 1; i <= n; i++)
   { product = product * i; }
```

Pre porovnanie:

```
int i = 1;
while (i <= n)
{ product = product * i;
  i = i + 1;}
```

Výhody cyklu `for`

- Jasné oddelenie jednotlivých častí cyklu (Initializácia, Podmienka, Iterácia) a príkazu cyklu
- Keď v príkaze cyklu iteračnej premennej-počítadlu a premenným použitým v podmienke nie je priradená žiadna hodnota, je možné priamo z hlavičky usúdiť, či cyklus terminuje.

Syntax cyklu `for`

Zoznam deklarácií alebo zoznam priradení

Zoznam priradení

```
for (Init; Pod; Iter)
    Príkaz
```

Ľubovoľný príkaz

Odporúčania

- V cykle for použite iba jednu iteračnú premennú
- V časti „iterácia“ cyklu for použite jednoduchú iteráciu iteračnej premennej (napr. `i++` alebo `i--`)
- V časti „podmienka“ použite iba porovnanie (napr. `i < n` alebo `i >= 0`)
- Neprirad'ujte iteračnej premennej a premennej vystupujúcej v podmienke žiadnu hodnotu v príkaze cyklu

Odporúčania (pokračovanie)

- Používajte podmienku `i < ...` resp. `i <= ...` iba v spojení s `i++` alebo `i= i + c`
- Používajte podmienku `i > ...` resp. `i >= ...` iba v spojení s `i--` alebo `i= i - c`
- Nepoužívajte `i != ...`
- Pre iteračné premenné používajte celočíselné premenné

Odporúčania (pokračovanie)

- Používajte cyklus **`for`**, keď jeho použitie koncepčne zodpovedá cyklu nad intervalom (cyklu s póčítadlom) – inak použite radšej cyklus **`while (do while)`**

Vnorené cykly

- V súlad s EBNF definíciou príkazov môžu byť všetky konštrukcie pre príkazy (if-else príkaz, cykly) vnorené (zložené)
- Táto skutočnosť sa hojne využíva!

Príklad (Java):

```
public class Trojuholnik
{
  public static void main (String[] args)
  { int vyska = Integer.parseInt(args[0]);

    for (int riadok = 1; riadok <= vyska; riadok++)
    { for (int stlpec = 1; stlpec <= riadok; stlpec++)
      {
        System.out.print("*");
      }
      System.out.println();
    }
  }
}
```

Príklad (výstup na konsolu):

```
>javac Trojuholnik.java
>java Trojuholnik 7
*
**
***
****
*****
******
*******
```

6.4 Verifikácia

Ako je možné dokázať, že program skutočne robí to, čo sme chceli?

- Kontrolné podmienky
- Invarianty (cyklov)
- Varianty (cyklov)

Príklad: Faktoriál

```
// int n >= 0
int i = 1;
int prod = 1;
// prod == (i-1)!   &&   (i-1) <= n
while (i <= n)
{ // prod == (i-1)!   &&   i <= n
  prod = prod * i;
  // prod == i!   &&   i <= n
  i = i + 1;
  // prod == (i-1)!   &&   (i-1) <= n
} // prod == (i-1)!   &&   (i-1) <= n &&   !(i <= n )
// prod == n!
```

Počiatočná podmienka

Kontrolné podmienky

Koncová podmienka

Kontrolné podmienky

- **Kontrolná podmnienka** je výraz typu boolean (ako komentár v programe), ktorý je **vždy pravdivý**, keď sa vykonanie programu nachádza na danom mieste
- **Počiatočná a koncová podmienka** (časti) programu sú špeciálne kontrolné podmienky

Kontrolné podmienky

- Platnosť kontrolných podmienok je možné overiť príkaz po príkaze!
  (Pri cykloch je to trochu zložitejšie, viď ďalej)
- Formulácia správnych kontrolných podmienok vyžaduje skúsenosti
- Teoretické pozadie verifikácie (žiaľ) nebudeme detailne rozoberať

Invarianty (cyklov)

```
// prod == (i-1)!   &&   (i-1) <= n
while (i <= n)
{ // prod == (i-1)!   &&   i <= n
  prod = prod * i;
  // prod == i!   &&   i <= n
  i = i + 1;
  // prod == (i-1)!   &&   (i-1) <= n
}
// prod == (i-1)!   &&   (i-1) <= n &&   !(i <= n )
```

Invariant

Po cykle platí invariant a negovaná podmienka cyklu!

Invarianty (cyklov)

- **Invariant** je kontrolná podmienka, ktorá platí na začiatku aj na konci cyklu.

Príklad: Faktoriál

```
// int n >= 0
int i = 1;
int prod = 1;

while (i <= n)
{
  prod = prod * i;

  i = i + 1;

}
// prod == n!
```

Pozor:

S pomocou kontrolných podmienok sme dokázali, že program bude mať korektný výsledok, keď k nejakému výsledku dospeje.

Nedokázali sme však, že k nejakému výsledku dospeje (t.j. že cyklus bude terminovať (sa ukončí:-))!

Varianty (cyklov)?!

Ako dokážeme, že cyklus terminuje?

- Pri každom vykonaní príkazu cyklu musíme byť „bližšie“ k cieľu, napr.
  - `int`-výraz, ktorého hodnota je po každom uskutočnení príkazu cyklu menšia a
  - Táto hodnota nemôže byť ľubovoľne malá
- Takýto výraz nazývame **Variant cyklu**

Príklad: Faktoriál

Variant:

(n - i)

```
while (i <= n)
{ //  ...                                   (n - i) == m
  prod = prod * i;
  //  ...                                   (n - i) == m
  i = i + 1
  // prod == (i-1)!  &&  (i-1) <= n && (n - i) < m
}
```

Z invariantu cyklu
vieme, že `(n - i) >= -1`