F7PMIPAZ

Pokročilá algoritmizace

Blok 1

adaptační, základní pojmy a postupy algoritmizace I.

Kontakty

- doc. Ing. Pavel Smrčka, Ph.D. - smrcka@fbmi.cvut.cz
- Ing. Radim Kliment, Ph.D. - radim.kliment@fbmi.cvut.cz
- Ing. Jan Broulím, Ph.D. - jan.broulim@cvut.cz

Katedra informačních a komunikačních technologií v lékařství

FBMI ČVUT v Praze, José Martího 31 162 52 Praha 6 - Veleslavín

Cíle předmětu

Cíl předmětu je seznámit studenty s problematikou algoritmizace a základů teoretické informatiky. Studenti se seznámí s metodami návrhů algoritmů, určení jejich složitosti, s grafovými a optimalizačními algoritmy. V předmětu budou popsány běžné využívané datové struktury a způsoby jejich implementace. Přednášky budou také věnované formálním jazykům a automatům. Důležitou součástí cvičení je samostatná implementace datových typů a algoritmů přednášky.

Studijní materiály

- Moodle https://moodle-vyuka.cvut.cz

Podrobná osnova jednotlivých lekcí.

Sylaby bloků a zdrojové kódy ukázkových úloh, v prezentacích odkazy na doporučenou studijní literaturu.

- statická fakultní stránka předmětu

https://predmety.fbmi.cvut.cz/cs/f7pmipaz

---

potřeba sledovat školní e-mail !

Hodnocení předmětu

2P+2C Z,ZK, zakončení zkouškou

3 složky hodnocení:

(a) průběžné úlohy realizované samostatně během daného výukového bloku (1/3 bodů). U některých průběžných úloh vyučující může vyhlásit možnost dokončit / odevzdat je jako d.cv.
(b) minitesty / miniprojekty zpravidla na začátku následujícího bloku (1/3 bodů)
(a) zkouška (1/3 bodů)

Povolena 1 absence bez omluvy na libovolném bloku (bez možnosti doplnění bodů z průběžných úloh). V případě omluvené absence (nemoc apod.) bude možné body doplnit po individuální domluvě.

Bodování: **Hodnocení (max. je 100 bodů) – ECTS stupnice :**

50 - 59 bodů … 3 (E)	60 - 60 … 2,5 (D)
70 - 79 bodů … 2 (C)	80 - 89 … 1,5 (B)
90 - 100 bodů … 1 (A)	

Harmonogram ZS 2025/2026

- 1. blok: 1.10.2025,
- 2. blok: 8.10.2025,
- 3. blok: 15.10.2025, děkanské volno, bude nahrazen
- 4. blok: 22.10.2020,
- 5. blok: 5.11.2025,
- 6. blok: 26.11.2025,
- 7. blok: 3.12.2025,
- 8. blok: 10.12.2025.

Podrobná předběžná osnova jednotlivých bloků je na konci této prezentace a rovněž on-line na moodle-vyuka.cvut.cz

Studentská anketa ČVUT

Anketa v předmětu F7PMIPAZ přispěla:

- Zohlednění studentů, kteří se dosud s tématy předmětu setkali okrajově => 2 úvodní vyrovnávací lekce
- Více materiálů pro studium => na Moodle Harmonogram, Podrobná osnova jednotlivých lekcí, Sylaby bloků a zdrojové kódy ukázkových úloh, v prezentacích důsledně odkazy na doporučenou studijní literaturu.

Časté dotazy / připomínky

Příkl. cit. : "…. Řadím se k části studentů, která se předtím s programováním nesetkala, tudíž začátek byl hodně složitý. Zpětně si vážím těch prvních 2 bloků, kde jsme probírali základy. Zezačátku jsem z toho moc neměla, ale vím, že mi to pomohlo se v Javě zorientovat. Musím ocenit, že v tomto předmětu se body sbírají v průběhu celého semestru. Donutilo mě to se učit průběžně. Zkouška je pak také mnohem příjemnější…."

Příkl. cit.: „…Předmět byl dobře koncipován. Díky průběžným úkolům a testům lze jednoduše nastřádat body, které se pak velice hodí ke zkoušce. Všichni vyučující byli velice nápomocní a a myslím si, že byl předmět postaven tak, aby byl zvládnutelný i začátečníky.…"

Program 1. bloku (1.10.2025)

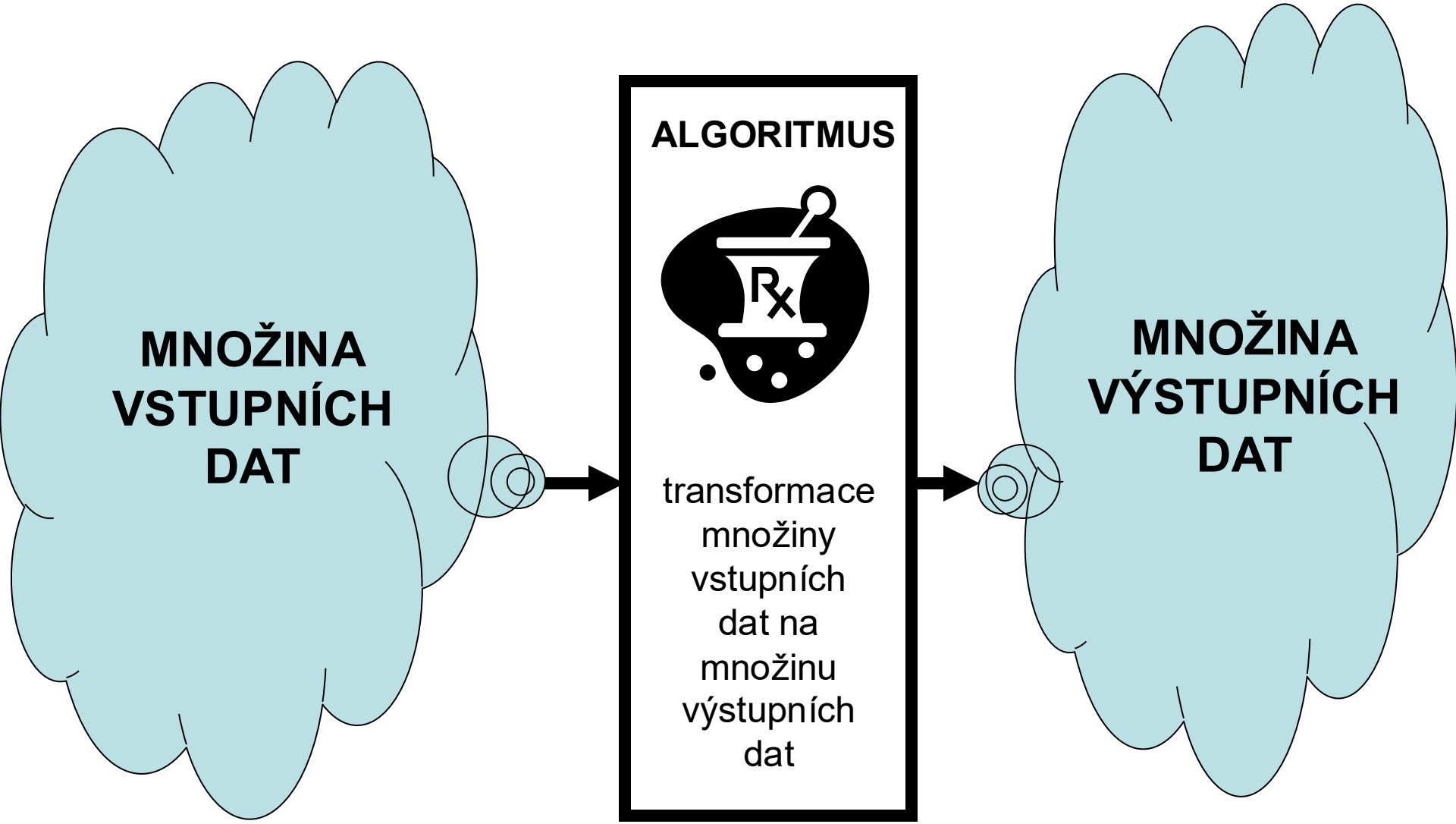
- zopakování pojmu „algoritmus“, způsoby popisu/zápisu algoritmů, metoda top-down
- primitivní datové typy v Javě, deklarace a inicializace proměnných
- přiřazovací příkaz, aritmetické a logické výrazy a operátory v Javě
- vstup dat z konzole a výstup dat na konzoli v Javě
- podmíněný příkaz (if a switch-case) v Javě a logické výrazy
- cykly (while, do-while a for) v Javě
- tvorba uživatelských funkcí v Javě – úvod
- praktické demonstrační a samostatné průběžné úlohy

Zakončení : miniprojekt na začátku 2. bloku

Zopakování pojmu „Algoritmus“

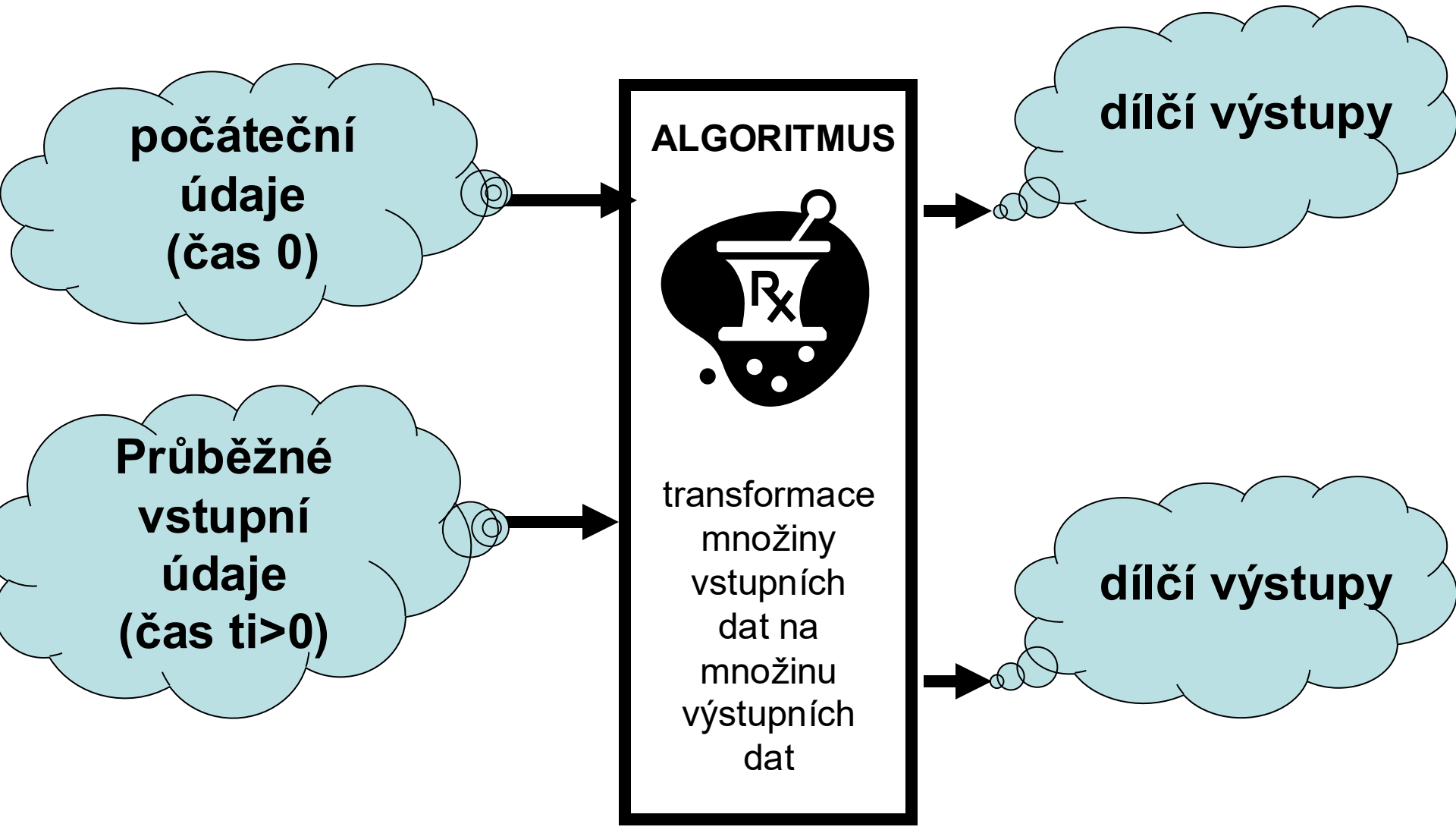
- 800-825 arabský matematik Muhamad ibn Musá al Chwárizmí, lat. překlad jeho knihy o početních postupech „Algoritmi dicit“ – “Tak praví Al Chwárizmí“.
- 30. léta 20. století znovuzavedl Alan Turing + navrhnul formální popis algoritmu (tkzv. Turingův stroj)
- algoritmus = matematický pojem, definice pouze opisem (podobně jako bod, množina)
- **po krocích rozepsaný návod k nějaké činnosti**

Algoritmus


MNOŽINA VSTUPNÍCH DAT
ALGORITMUS
Rx
transformace množiny vstupních dat na množinu výstupních dat
MNOŽINA VÝSTUPNÍCH DAT

Algoritmus

dávkové X dialogové zpracování X událostmi řízené zpracování (stav. automat)



Různé způsoby zápisu algoritmů

- Slovní popis (přirozeným jazykem)
- Grafické znázornění (např. vývojový diagram)
- Programovací jazyk (např. Java, Python, C/C++, C#). Obsahují klíčová slova, operátory, speciální znaky a identifikátory.

Tyto tkzv. lexikální elementy mají formální pravidla pro zápis (syntax).
Srovnej s přirozeným jazykem: syntaxe a sémantika

Pojem „program“

Program: algoritmus zapsaný v nějakém programovacím jazyce.

Procesor: systém (člověk či stroj), který vykonává algoritmem popisovanou činnost.

Druh procesoru ➔ různé formulace kroků algoritmu

Nějak se musíme ale k algoritmu dobrat – vytvořit ho:

Pojem algoritmizace úlohy

Etapy řešení problému

1. Specifikace (vymezení) problému
2. Analýza problému, dekompozice (rozklad na podproblémy, dílčí úlohy). Tkzv. TOP DOWN METODA
3. Sestavení a zápis (kódování) dílčích algoritmů
4. Sestavení a zápis výsledného celkového algoritmu (spojení řešení dílčích úloh) SYNTETICKÁ FÁZE
5. Testování algoritmu (u programů ladění)

+ Nejdříve si uvědomit „co vlastně (příp. proč) řešit ?“ Teprve potom „jak to řešit“ ! ! !

Vlastnosti algoritmu

- **Elementárnost.** Skládá se z konečného počtu jednoduchých (elementárních) činností (kroků).
- **Determinovanost.** V každém kroku lze rozhodnout, jak se má pokračovat (a jestli nenastal konec).
- **Konečnost.** Algoritmus vždy skončí po konečném počtu kroků.
- **Rezultativnost.** Vede ke správnému výsledku (k nějakému…).
- **Hromadnost.** Použitelný pro celou třídu podobných problémů.
- **Opakovatelnost.** Pro stejné vstupy vždy stejný výstup. Souvisí s determinovaností a rezultativností.

DP 1.0

Skoro algoritmus - příklad

Úloha:

Sestavit návod na domácí ruční výrobu pomerančového džusu ve formě algoritmu

Vstup: 1kg šťavnatých pomerančů

Výstup: cca 250ml pomerančového džusu

Prostředky:

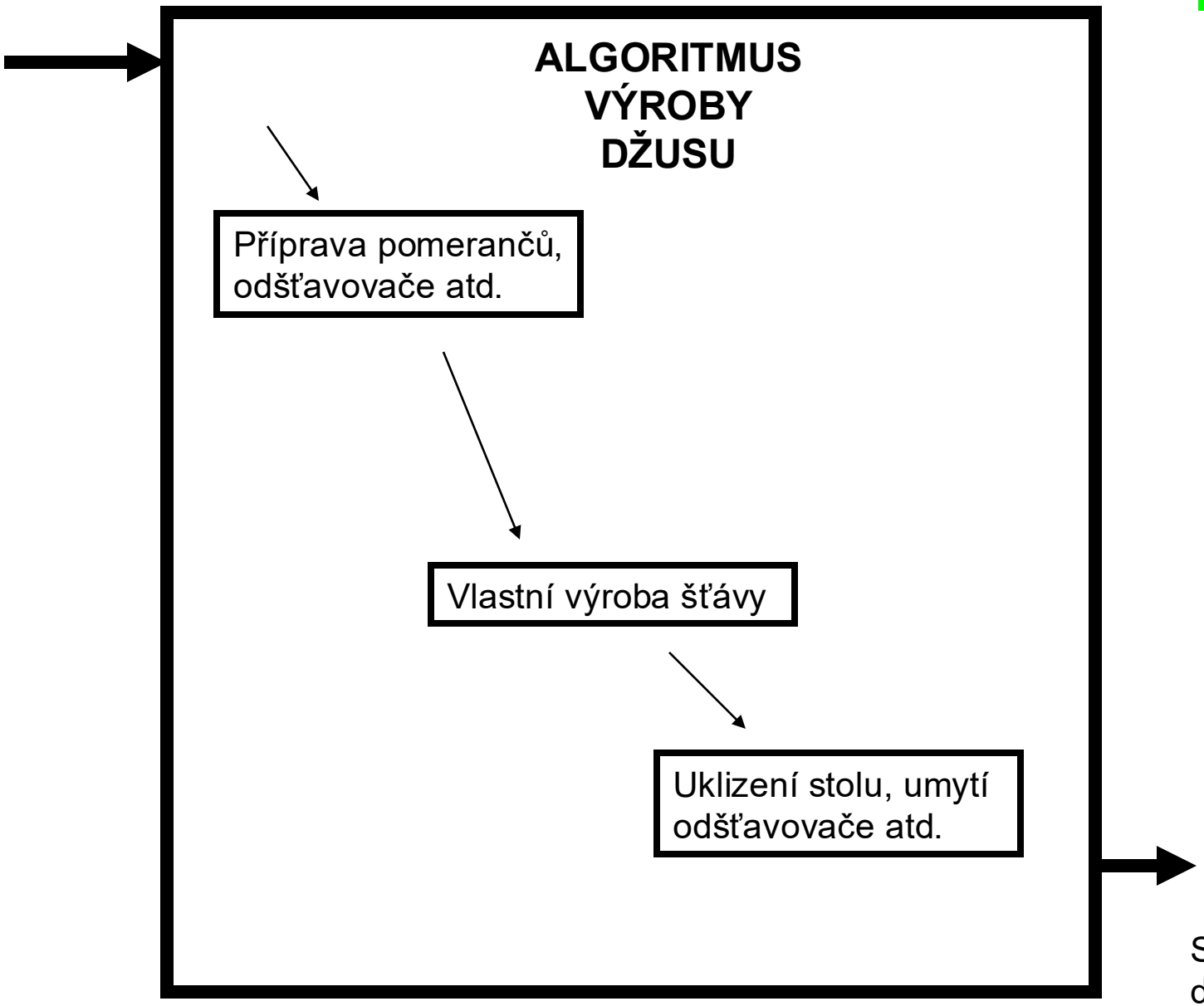
zdravá osoba (pohyblivé ruce, prsty, …)

nůž, ruční odšťavovač, mistička, sklenka …

1kg pomerančů

Algoritmus - příklad

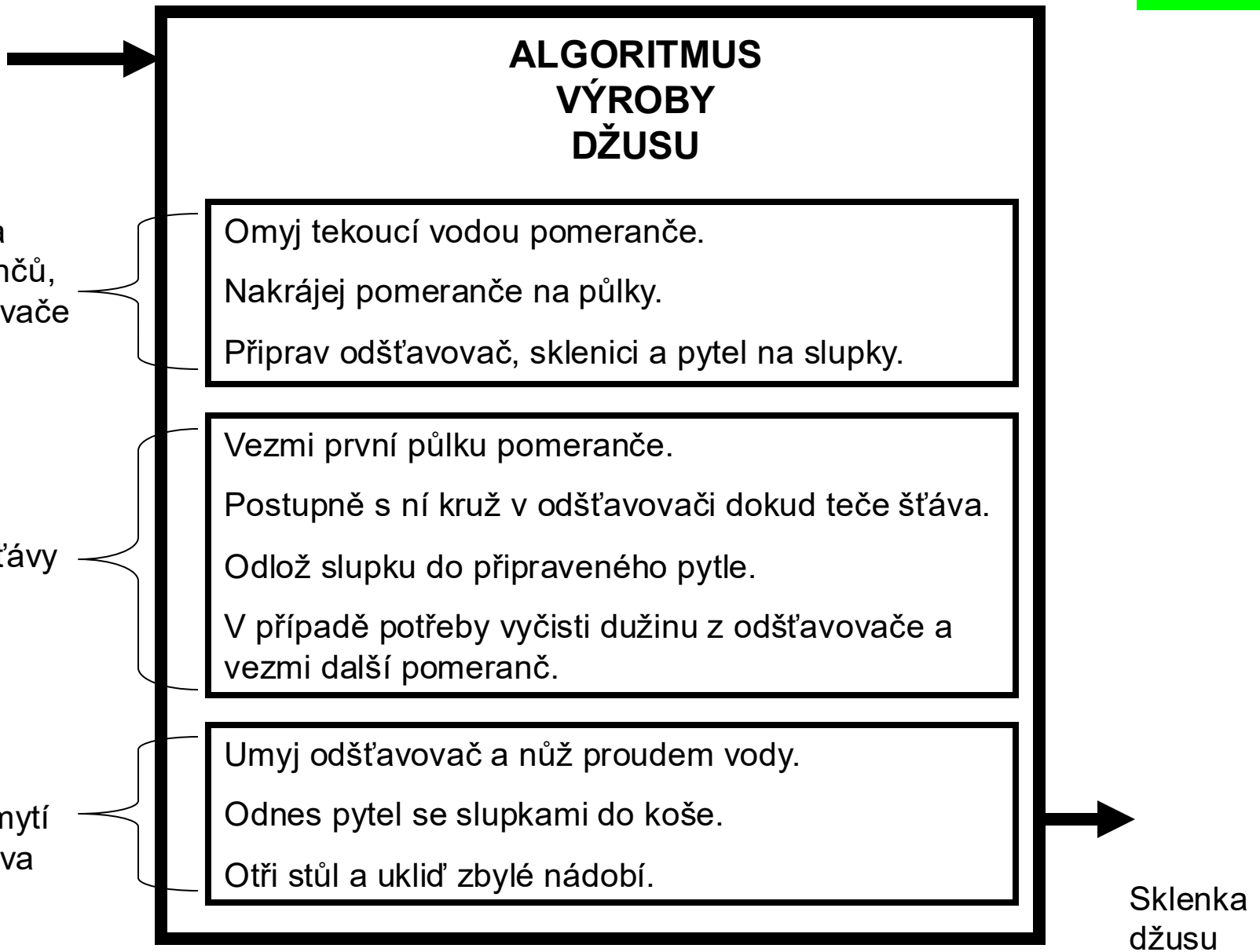
DP 1.0



1kg pomerančů

Algoritmus - příklad

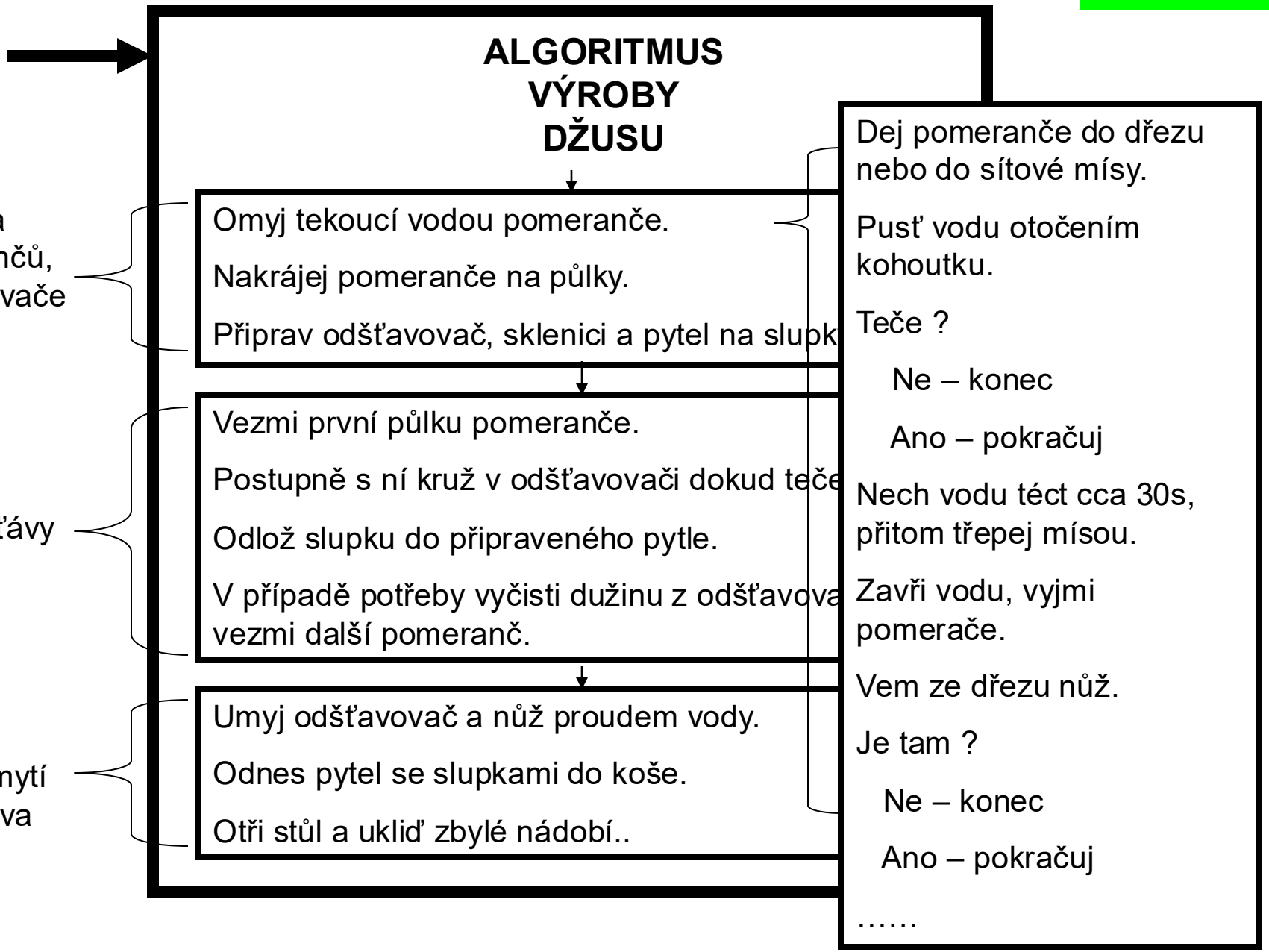
DP 1.0



1kg pomerančů

Algoritmus - příklad

DP 1.0



Uvědomíme si …

- Způsob provedení rozkladu úlohy na posloupnost dílčích podúloh (**dekompozice**)
- D.C.V.: Co je zde „procesor“, tj. co/kdo vykonává zakódovaný algoritmus ?
- D.C.V.: splňuje tento „algoritmus“ všechny vlastnosti skutečného algoritmu ? Jak to zhodnotíme ?

- Algoritmus obsahuje:
  1) pevně dané sekvence povelů,
  2) místa rozhodování na základě vhodně zformulovaných „podmínek“ →
  3) možnost „výhybky“ v sekvenci povelů,
  4) možnost opakovat určitou sekvenci povelů.

V algoritmu vidíme: řídicí povely, možnost zapamatovat si předchozí stav a sledovat stav okolí

Jevy, jejichž základní úroveň technického uchopení je náplní této úvodní lekce

DP 1.1 – začátek

Př. Sestavte algoritmus pro výpočet obvodu kruhu, když předem znáte poloměr tohoto kruhu.

Analýza:

Vstupní množina dat:
-poloměr kruhu – číslo. Fyzikální jednotky ? Bude reálné nebo celé číslo ?
-Přesnost? Rozsah?
- bude v praxi např. zadán z klávesnice

Výstupní množina dat:
-Spočtený obvod kruhu (číslo).
-výsledek bude v reálu např. vypsán na obrazovku

Algoritmus:
Co musí umět ?
- Načíst poloměr (obecně reálné číslo, možno omezit na kladná reálná) z klávesnice a zapamatovat si ho,
- Dosadit toto číslo do známého vzorce a zapamatovat si výsledek,
- Vypsat tento výsledek na obrazovku.

DP 1.1 – začátek

Tip: zkuste úlohu analyzovat pomocí chat GPT.
klíčový bude správný prompt ☺

Sestavte algoritmus pro výpočet obvodu kruhu, když předem znáte poloměr tohoto kruhu.

Pro výpočet obvodu kruhu, pokud znáte poloměr, můžeme použít následující vzorec:

$$O = 2\pi r$$

Kde:

- O je obvod kruhu,
- r je poloměr kruhu,
- π je přibližně rovno 3.14159.

Definuj množinu vstupních dat této úlohy.

Algoritmus:

1. Zadejte hodnotu poloměru r.
2. Vypočítejte obvod pomocí vzorce $O = 2\pi r$.
3. Vraťte vypočítaný obvod O.

DP 1.1 – začátek

Tip: zkuste úlohu analyzovat pomocí chat GPT.

Definuj množinu vstupních dat této úlohy.

Množina vstupních dat pro tuto úlohu, která počítá obvod kruhu na základě zadaného poloměru, může být definována následovně:

Množina vstupních dat:

1. **Poloměr kruhu (r):**
   - **Typ:** Reálné číslo (float).
   - **Rozsah:** $r > 0$ (poloměr musí být kladné číslo, protože délka nemůže být záporná nebo nulová).
   - **Jednotky:** Libovolné jednotky délky (např. metry, centimetry, palce), ale je důležité, aby byly konzistentní, pokud se používají v kontextu.

Specifika vstupních dat:

- **Příklady validních vstupů:**
  - $r = 5.0$
  - $r = 12.3$
  - $r = 0.001$
- **Příklady nevalidních vstupů:**
  - $r = -4$ (negativní číslo není platné pro poloměr)
  - $r = 0$ (poloměr nemůže být nulový)
  - $r = "text"$ (řetězec není platný vstup)

Shrnutí:

Pro každý platný vstup musí být poloměr reálné číslo větší než nula.

Klíčová slova: chatGPT, Github Copilot, . . .

Při užití UI se řídíme Metodickým pokynem ČVUT ze září 2023, viz https://www.cvut.cz/sites/default/files/content/d1dc93cd-5894-4521-b799-c7e715d3c59e/cs/20230922-metodicky-pokyn-c-52023.pdf

Výňatek z odstavce **Programování**: ...Při využití nástrojů UI je třeba řídit se pokyny vyučujících. Nástroje UI lze i při programování využít k samostudiu, konzultacím, či samozkoušení …. Autorem kódu je student, který by měl přesně vědět, co vygenerovaný kód opravdu dělá, měl by být schopen jej upravit podle potřeby**. Stejný program musí být student schopen napsat i bez pomoci těchto nástrojů**…

DP 1.1 – začátek

Př. Sestavte algoritmus pro výpočet obvodu kruhu, když předem znáte poloměr tohoto kruhu.

Slovní formulace:

0. příprava: budeme pracovat se 2 reálnými čísly (POLOMĚR a OBVOD) a jednou konstantou PI rovnou 3.1415926…

1. Načti z klávesnice POLOMĚR
2. Spočítej obvod podle vzorce OBVOD=2.PI.poloměr
3. Vypiš na obrazovku výsledek OBVOD
4. skonči

K úloze se ještě vrátíme…

Všimněte si, že k práci „procesoru“ potřebujeme, aby porozuměl jen několika lexikálním konstrukcím, zde zejména:

- *načtení čísla ze vstupního zařízení (např. klávesnice)*
- *výpis čísla na výstupní zařízení (např. obrazovka)*
- *zapamatování hodnoty v pojmenovaném paměťovém prostoru („proměnné“)*
- *vyhodnocení aritmetického výrazu (zde vzorec pro výpočet obv. kruhu)*

Datové a řídicí struktury

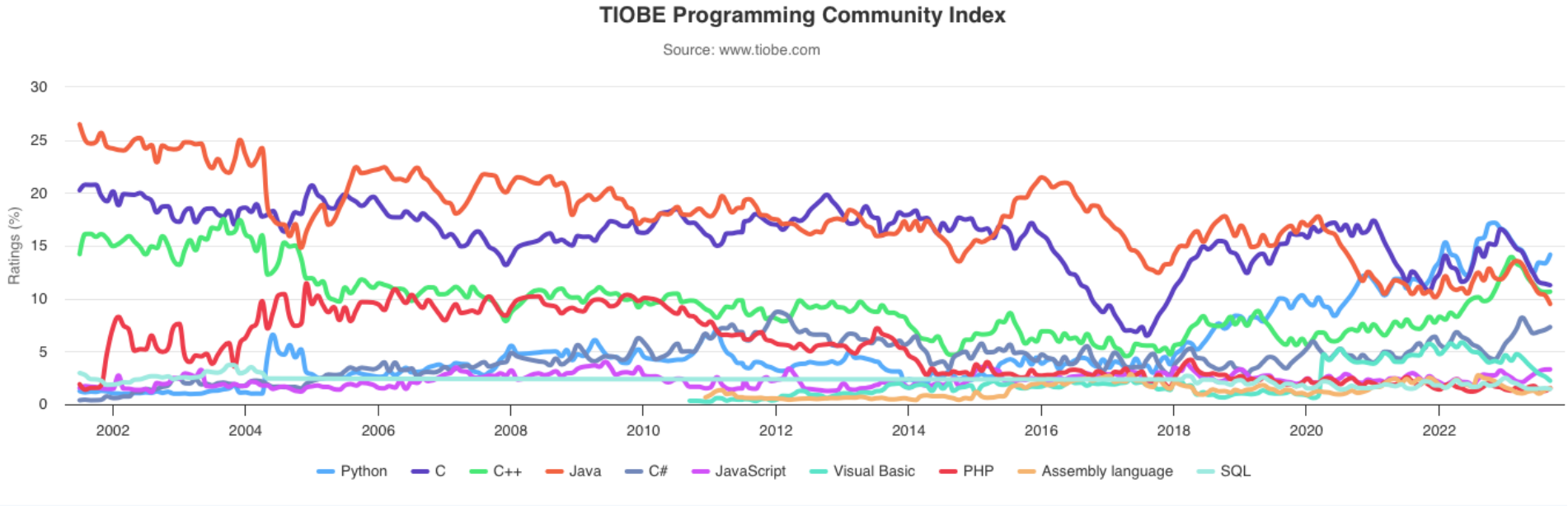
… aneb abychom mohli úlohu 1.1 algoritmicky vyřešit,
je na čase probrat základy formalizovanějšího konceptu.

Dále se současně (na stejném příkladu) začnete učit:

- grafický zápis elementárních algoritmů
  (tkzv. vývojové diagramy), ČSN ISO 5807

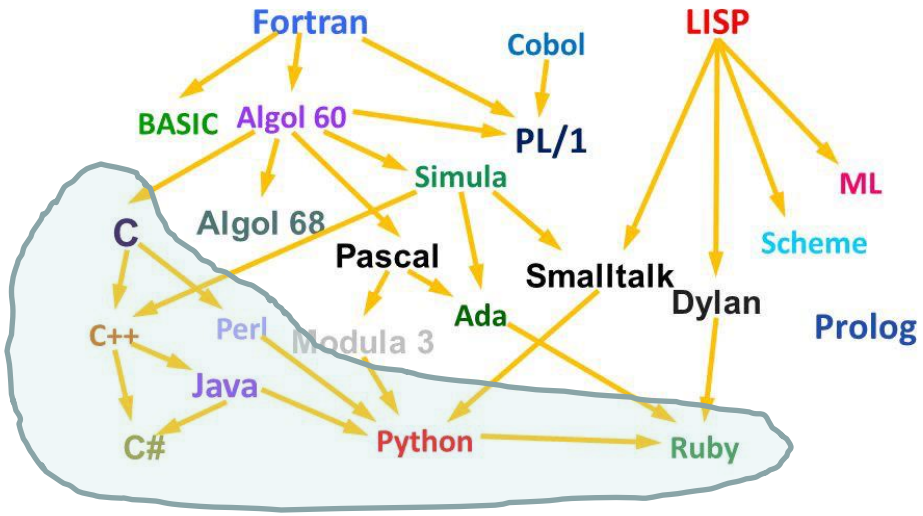
- zápis algoritmů v jazyce Java.

September 2023

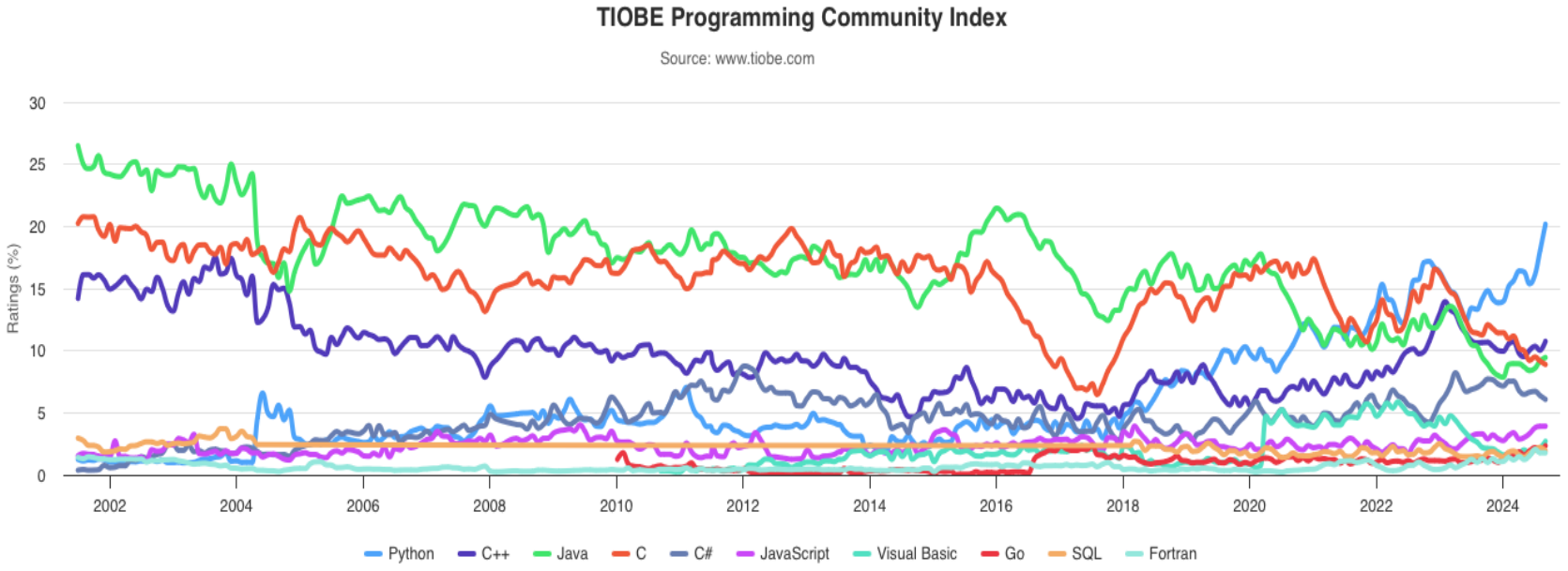


A family tree of languages

Some of the 2400 + programming languages

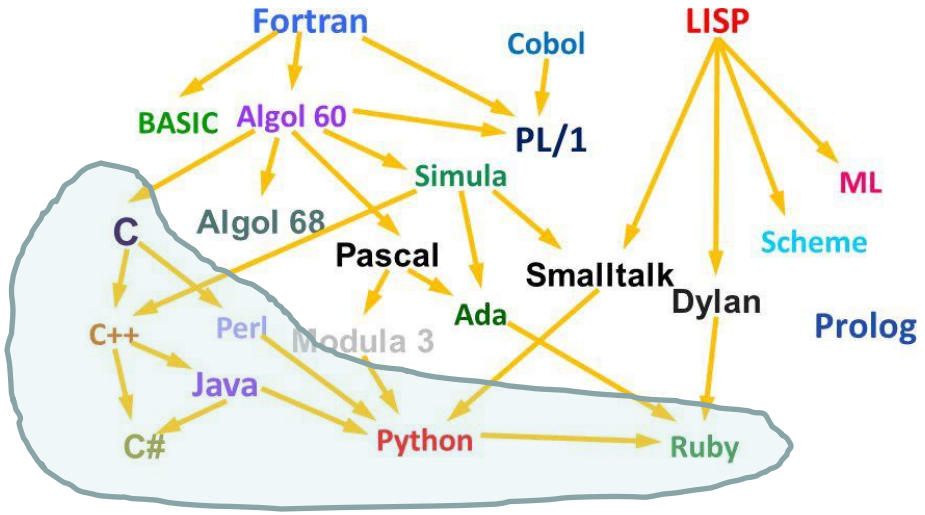


September 2024

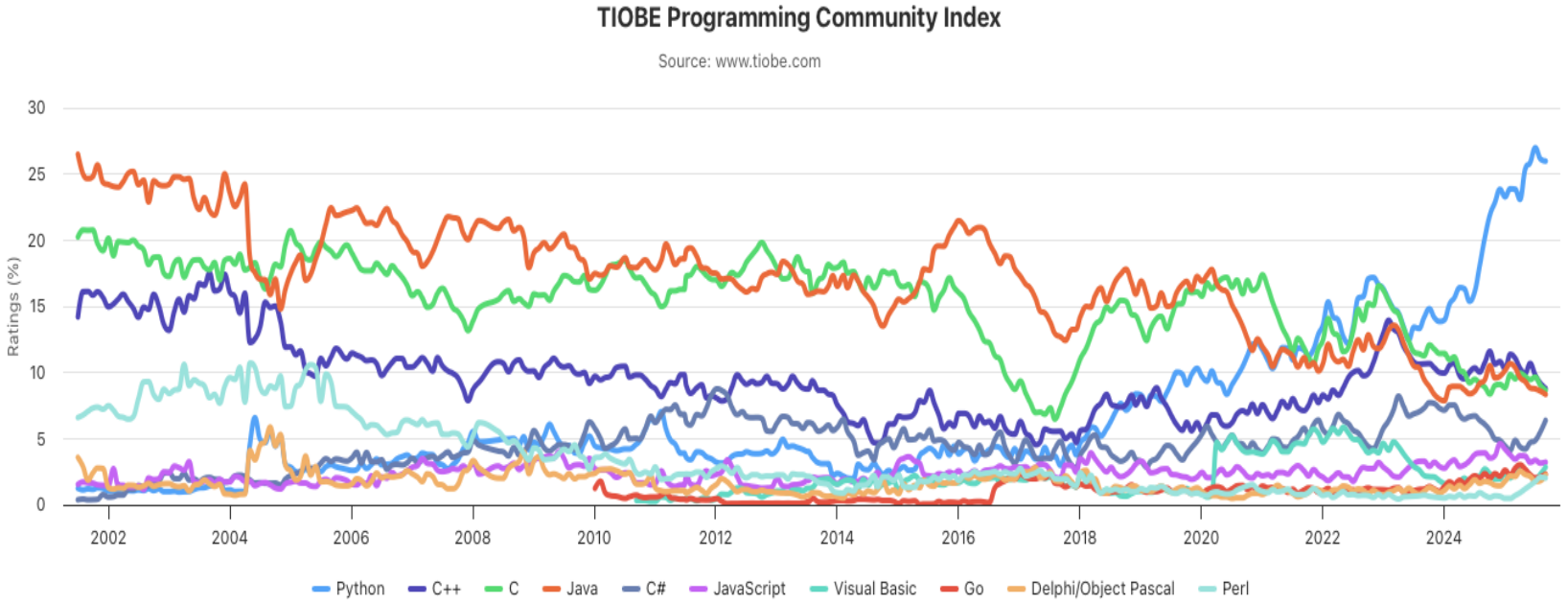


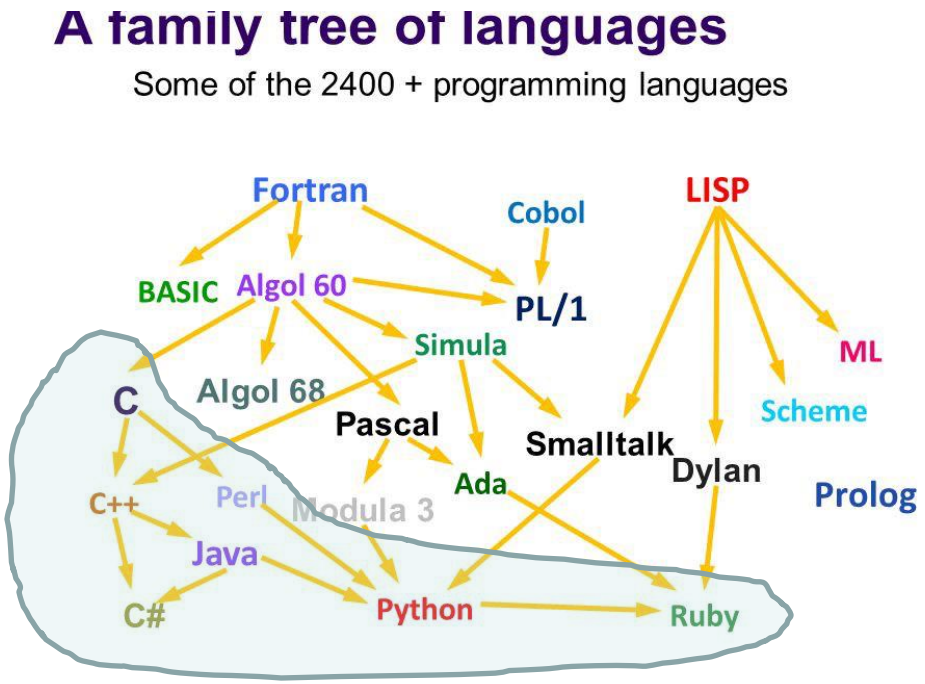
A family tree of languages

Some of the 2400 + programming languages



September 2025





Primitivní datové typy v Javě

deklarace a inicializace proměnných

Datové struktury

- **Proměnná:** veličina, která během řešení problému mění svoji hodnotu.

- **Identifikátor:** námi zvolené pojmenování proměnné

  (! konvence volby jména – později zpřesníme)

- **Datový typ:** určuje množinu hodnot, kterých může proměnná nabýt a množinu operací, které lze s proměnnými daného typu provádět (rovněž určuje, kolik místa proměnná zabere v paměti).

Zápisové konvence

Zápis celých čísel: se znaménkem, bez znaménka

+10 10 -20 0 +2452

uvažujeme typicky rozsah zarovnaný po bajtech – viz dále

Zápis čísel v plovoucí řádové čárce:

desetinný tvar mantisa+exponent „semilogaritmický tvar“

-0.0000345 -3.45E-5 -34.5E-6

Zápis jednotlivých znaků a znakových řetězců

’A’ – jeden znak reprezentující písmeno A latinské abecedy

“Toto je platný řetězec“ - vnitřní reprezentace zpravidla UTF-8, občas ASCII, viz dále

Pozn.: mnohé progr. jazyky (jakož i Java) zavádějí reprezentaci logické hodnoty – ano / ne

Zavedení proměnné

- Odhadnu, jaký datový typ budu pro danou proměnnou potřebovat (odvíjí se od řešené úlohy - rozsah, požadovaná přesnost)

Např.
- mezivýsledek výpočtu obsahu plošného útvaru → volím reálné číslo
- pořadí pokusu → volím celočíselný typ
- příjmení → volím znakový řezězec

- Proměnnou názorně pojmenuji, tj. zvolím takzvaný **identifikátor**

  Př.
  - **polomer** NIKOLIV **x**
  - **CisloPokusu** NIKOLIV **I**

- Identifikátor v Javě může být libovolně dlouhý, musí začínat písmenem nebo znakem _ či $ a nesmí být vyhrazeným klíčovým slovem jazyka (např. názvem datového typu apod.)

Primitivní datové typy v Javě

Pozn.: v Javě pojem typ **primitivní** vs. referenční. (např. String - v některé z příštích lekcí)

celočíselné…

Datový typ	Rozsah	Velikost
byte	-128 až 127	8 bitů
short	-32 768 až 32 767	16 bitů
int	-2 147 483 648 až 2 147 483 647	32 bitů
long	-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807	64 bitů

plovoucí řádová čárka…

Datový typ	Rozsah	Přesnost
float	$+-1.5 * 10^{-45}$ až $+-3.4 * 10^{38}$	7 čísel
double	$+-5.0 * 10^{-324}$ až $+-1.7 * 10^{308}$	15-16 čísel

znak a logický typ…

Datový typ	Rozsah	Velikost/Přesnost
char	U+0000 až U+ffff	16 bitů
boolean	`true` nebo `false`	8 bitů

Deklarace proměnné v Javě

V mnoha programovacích jazycích, třeba v Javě, C#, C/C++ je nutné proměnnou před prvním použitím takzvaně **deklarovat** – k technické realizaci se za chvíli vrátíme. Jedná se o tkzv. STATICKÝ TYPOVÝ SYSTÉM. Na rozdíl od DYNAMICKÉHO (např. v Pythonu).

Obecný tvar:

```
type name ;
```

Příklady:

```
int pokusu;
float vysledek;


int moje_promenna, jina_promenna;
```

Pojem **inicializace proměnné:**

```
int moje_promenna=10;

int jina_promenna;
jina_promenna = 20;
```

Pozn.: s deklarací proměnné souvisí také pojem **rozsah platnosti proměnné**, viz dále.

Konverze aritm. typů v Javě

Rozšiřující, beze ztráty přesnosti….

- byte ---> short, int, long, float, double,
- char, short ---> int, long, float, double,
- int ---> long, float, double,
- long ---> float, double,
- float ---> double

Zužující, s možností ztráty přesnosti….

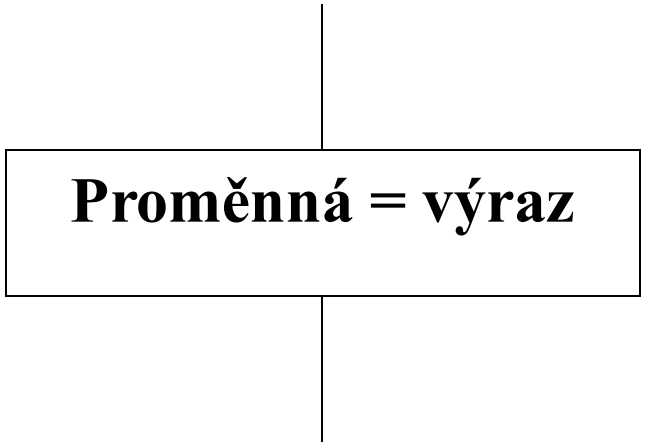
- short ---> byte,
- short ---> byte, char,
- char ---> byte, short,
- int ---> byte, short, char
- long ---> byte, short, char, int
- float ---> byte, short, char, int, long,
- double ---> byte, short, char, int, long, float.

Výchozí hodnoty, na které jsou primitivní typy v Javě inicializovány

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Přiřazovací příkaz

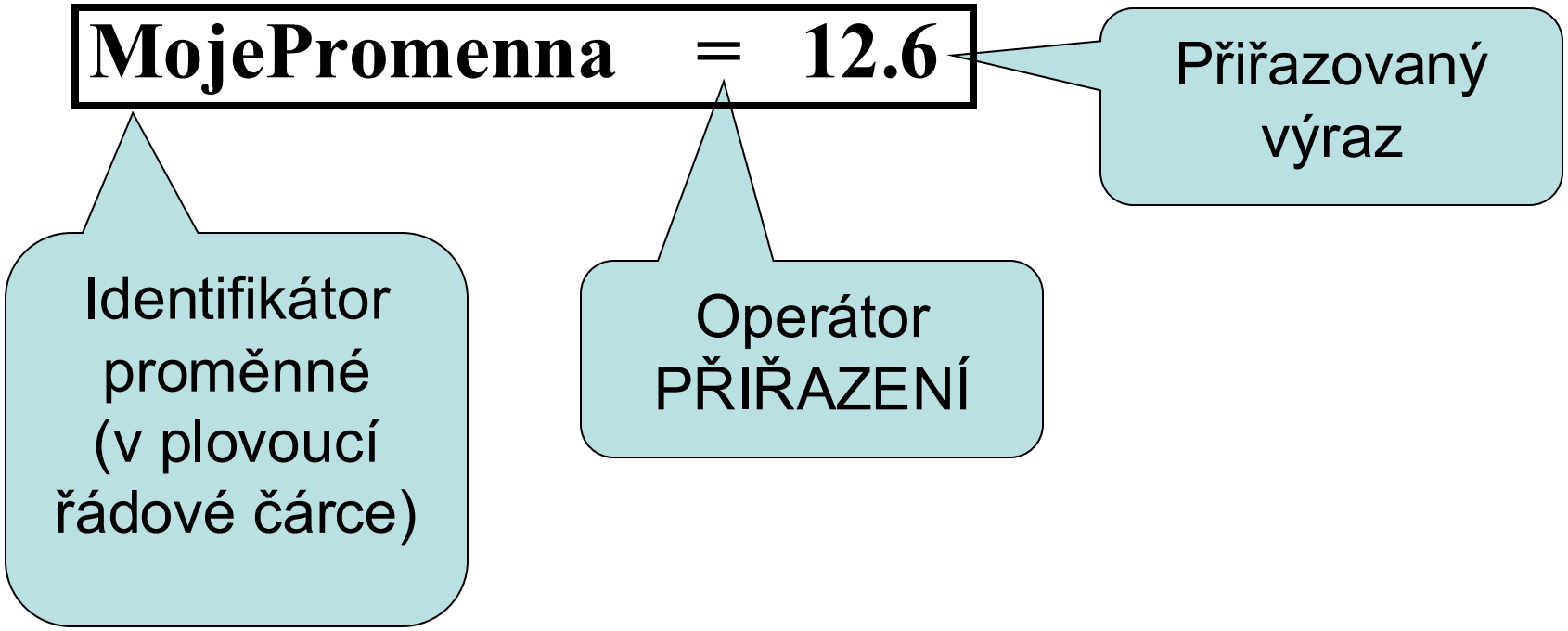
aritmetické a logické výrazy a operátory v Javě



Promenna = *vyraz* ;

První řídicí struktury

Jednoduché příkazy - přiřazovací příkaz



Řídicí struktury

Jednoduché příkazy - přiřazovací příkaz

```
MojePromenna = 12.6 + (10 * 2) / 4
```

Složený výraz může obsahovat
číselné konstanty, závorky ( ), operátory + - * / atd.

Řídicí struktury

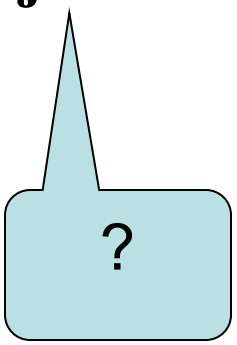
Jednoduché příkazy - přiřazovací příkaz

```
MojePromenna = 12.6 + (10 * obvod) / pocet
```

Složený výraz
může obsahovat další proměnné

Př.

```
float obvod, MojePromenna; int pocet;
. . .
obvod  = 2 ;
pocet = 5 ;
MojePromenna   =   12.6  +  (10 * obvod) / pocet ;
```



Řídicí struktury

Jednoduché příkazy - přiřazovací příkaz

```
MojePromenna = 12.6 + MojePromenna
```

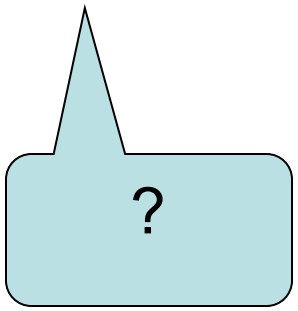
Přiřazovací příkaz může obsahovat ve výrazu napravo tutéž proměnnou jako nalevo

Př.

```
float MojePromenna;
. . .
```

```
MojePromenna = 3 ;
```

```
MojePromenna  =  12.6  +  MojePromenna ;
```



Přiřazovací příkaz - shrnutí

Proměnná = výraz

Promenna = *vyraz* ;

OPERÁTOR PŘIŘAZENÍ

MojeProm = 12.6 + (10 * MojeProm) / pocet

IDENTIFIKÁTOR VÝSLEDNÉ PROMĚNNÉ

SLOŽENÝ VÝRAZ

- číselné konstanty
- operátory **+ - * /**
- závorky **( )**
- další proměnné, včetně výsledné
- volání funkcí apod….

Přehled operátorů v Javě

aritmetické binární….

Oper.	Použití	Popis
+	op1 + op2	součet operandů op1 a op2
-	op1 - op2	rozdíl operandů op1 a op2
*	op1 * op2	součin operandů op1 a op2
/	op1 / op2	podíl operandů op1 a op2
%	op1 % op2	zbytek po dělení op1 operandem op2

aritmetické unární….

Oper.	Použití	Popis
+	+op	indikace kladné hodnoty
-	-op	aritmetická negace operandu
++	op++	inkrementace op o 1 po jeho vyhodnocení
++	++op	inkrementace op o 1 před jeho vyhodnocením
--	op--	dekrementace op o 1 po jeho vyhodnocení
--	--op	dekrementace op o 1 před jeho vyhodnocením

Přehled operátorů v Javě

relační…. (viz další výklad)

Oper.	Použití	Výsledek je true jestliže
>	op1 > op2	op1 je větší než op2
>=	op1 >= op2	op1 je větší než nebo roven op2
<	op1 < op2	op1 je menší než op2
<=	op1 <= op2	op1 je menší než nebo roven op2
==	op1 == op2	op1 a op2 jsou si rovné
!=	op1 != op2	op1 a op2 si nejsou rovné

logické spojky …. (viz další výklad)

Oper.	Použití	Výsledek je true jestliže:
&&	op1 && op2	op1 a op2 nabývají hodnotu true
||	op1 || op2	alespoň jeden z op je true
!	!op	op nabývá hodnotu false (negace)

bitové ….

Oper.	Použití	Operace
>>	op1 >> op2	bitový posuv op1 doprava o op2 bitů
<<	op1 << op2	bitový posuv op1 doleva o op2 bitů
>>>	op1 >>>op2	jako >>, ale neznaménkově
&	op1 & op2	bitový AND
|	op1 | op2	bitový OR
^	op1 ^ op2	bitový XOR
~	~op	bitový doplněk

Přehled operátorů v Javě

Přiřazení, kromě prostého **=**

Oper.	Použití	Ekvivalent
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

Ternární operátor….

výraz1 ? výraz2 : výraz3

Např.
b != 0 ? c = a / b : c = 0; // pokud b je nenulové, provede se dělení, jinak se do c přiřadí 0

operátor přetypování….

(nový_typ) proměnná_původního_typu

Např.
float a = 10.5; int b;

b = (int)a;

Operátor **new** …

Operátor new slouží k vytvoření instance třídy (objektu), např.:

Trida prom=new Trida;

```
Scanner sc = new Scanner(System.in);
```

Priorita operátorů v Javě

P	Typ operátorů	Operátory
1	postfixové [] . (parametry)	op++ op–
2	unární operátory	++op –op +op –op ~ !
3	vytváření a přetyp.	new (typ) výraz
4	multiplikativní	* / %
5	aditivní	+ –
6	posuvy	<< >> >>>
7	relace	< > <= >= instanceof
8	ekvivalence	== !=
9	bitové AND	&
10	bitové XOR	^
11	bitové OR	|
12	logické AND	&&
13	logické OR	||
14	ternární	? :
15	přiřazení	= += -= *= /= %= ^= &= |= <<= >>= >>>=

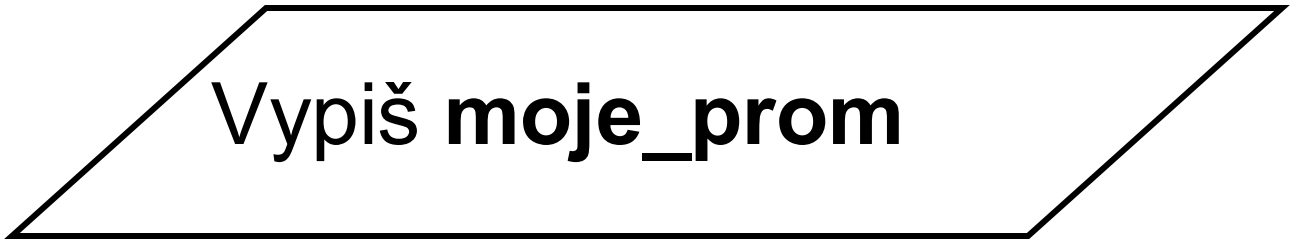
Výstup dat na konzoli

v Javě

První řídicí struktury

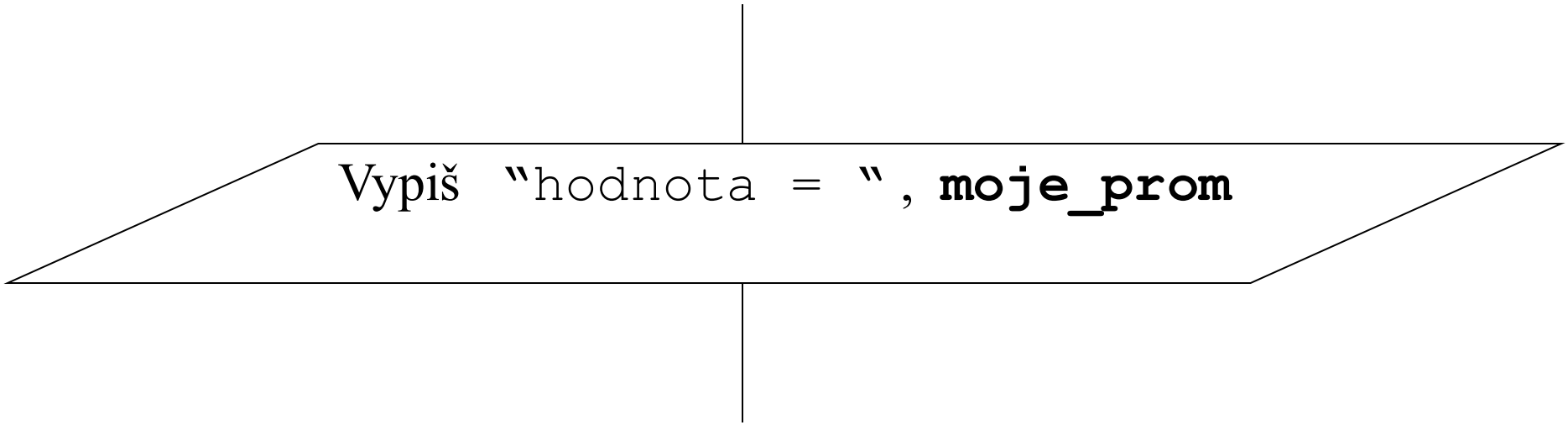
Jednoduché příkazy - operace výstupu:

potřebujeme pouze možnost vypsat na obrazovku obsah proměnné

- př. 

Příkaz „Vypiš“ vypíše obsah specifikované proměnné

Příkaz výstupu v1



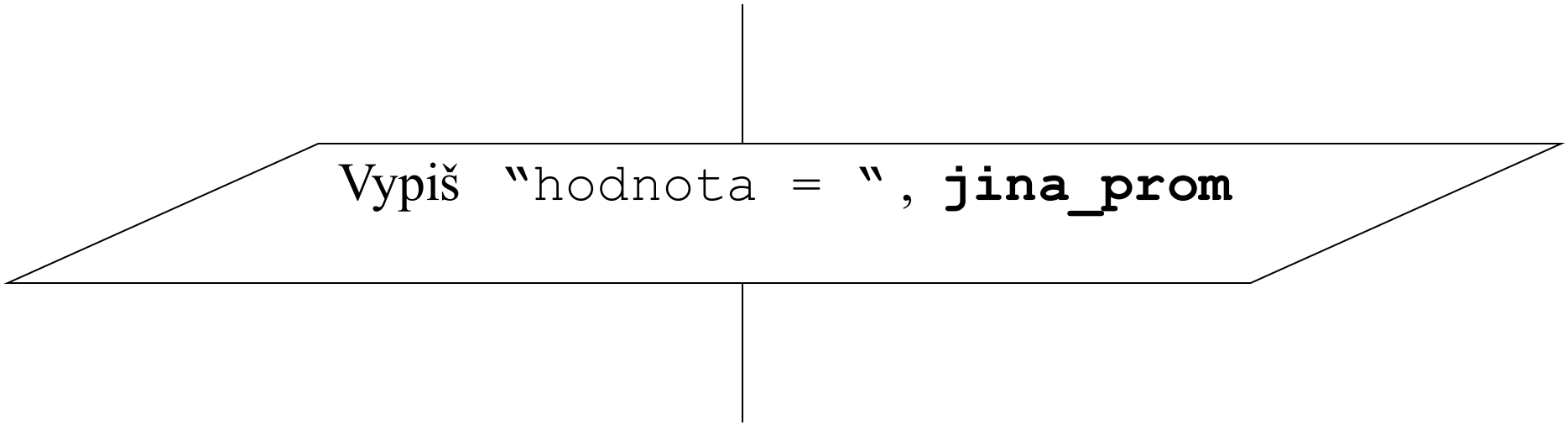
Funkce pro výstup na monitor, jedna z možných variant :

```
System.out.printf("hodnota = %d \n ", moje_prom );
```

<u>Formátovací řetězec</u> "proměnná bude typu celé číslo"
(`int moje_prom;` **)**

<u>Spec. řídicí znak</u> "přejdi na začátek dalšího řádku"

Příkaz výstupu v1



Funkce pro výstup na monitor, jedna z možných variant :

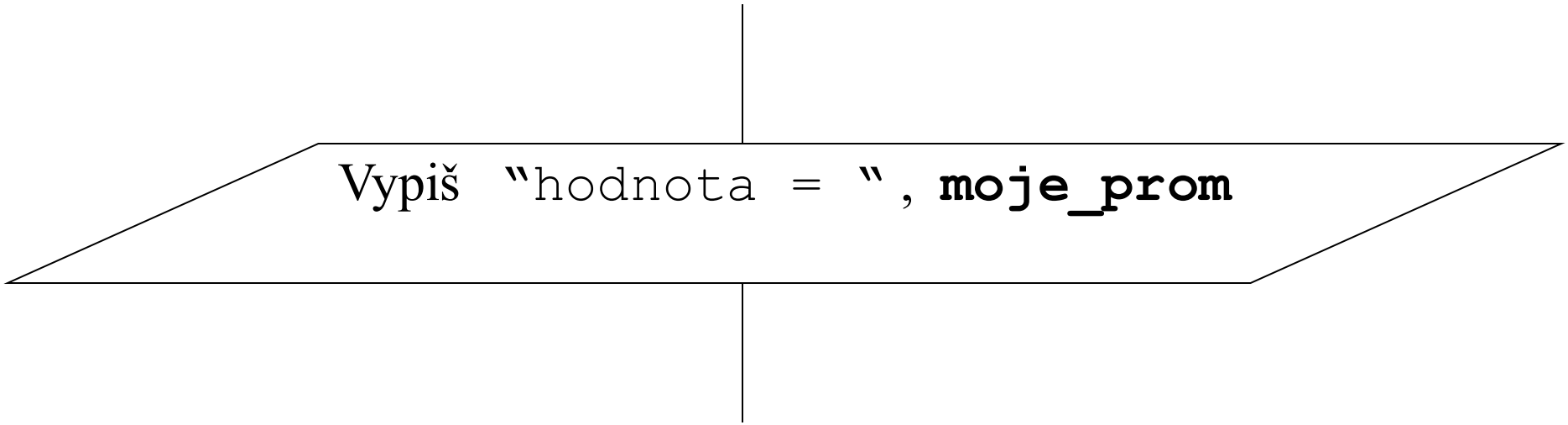
```
System.out.printf("hodnota = %f \n ", jina_prom );
```

Formátovací řetězec "proměnná bude typu číslo v pl. ř. č."

(`float jina_prom;` **)**

Spec. řídicí znak "přejdi na začátek dalšího řádku"

Příkaz výstupu v2



Funkce pro výstup na monitor, další z možných variant :

```
System.out.println("hodnota = " + moje_prom );

System.out.print("hodnota = " + moje_prom + "\n");
```

Vstup dat z konzole

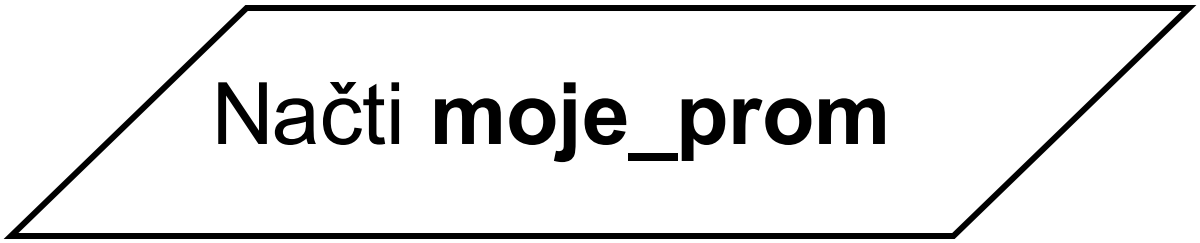
v Javě

První řídicí struktury

Jednoduché příkazy - operace vstupu:

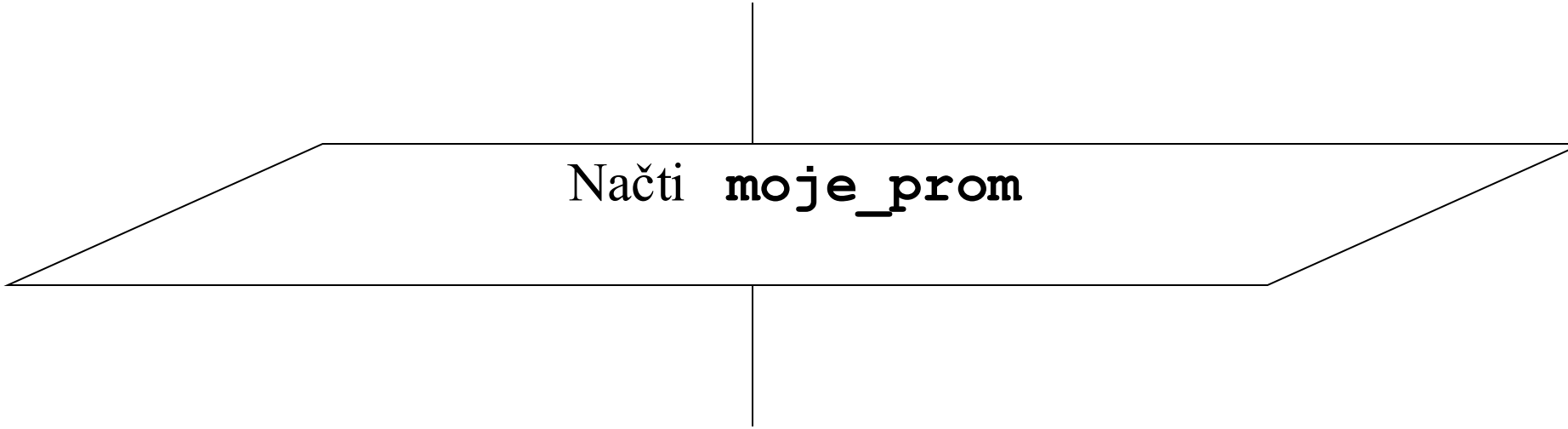
Prozatím potřebujeme pouze možnost vstupu dat z klávesnice

př.



Na příkazu „Načti“ se algoritmus přeruší a očekává vstup hodnoty příslušné proměnné

Vstup celočíselné proměnné

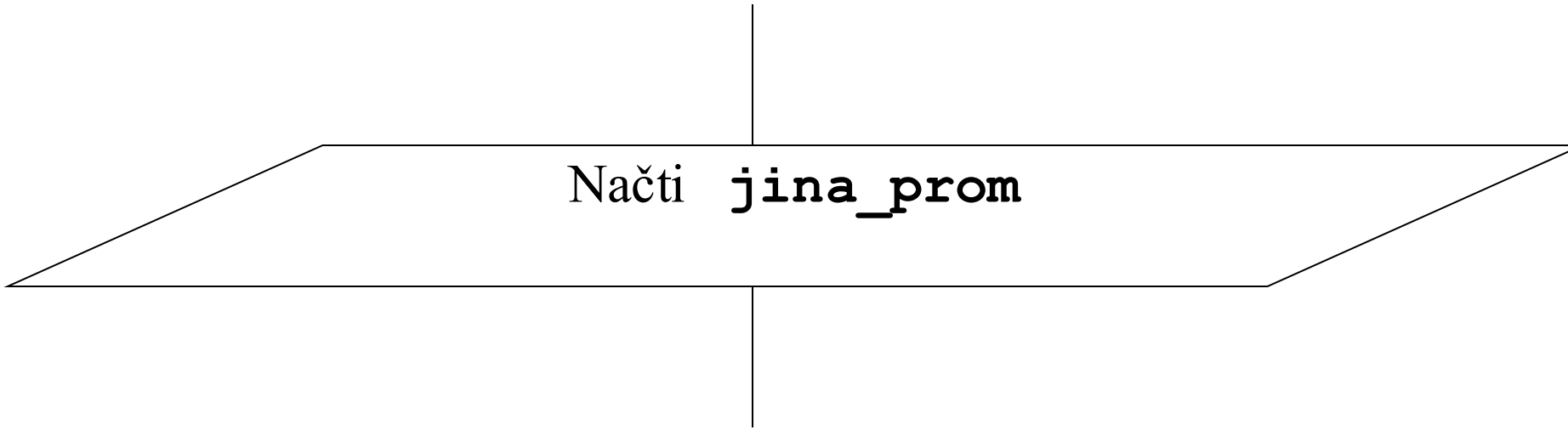


Postup pro vstup dat z klávesnice, jedna z možných variant :

```
int moje_prom;
. . .
Scanner sc = new Scanner(System.in);
. . .

moje_prom = sc.nextInt();
```

Načtení celého čísla

Vstup reálné proměnné

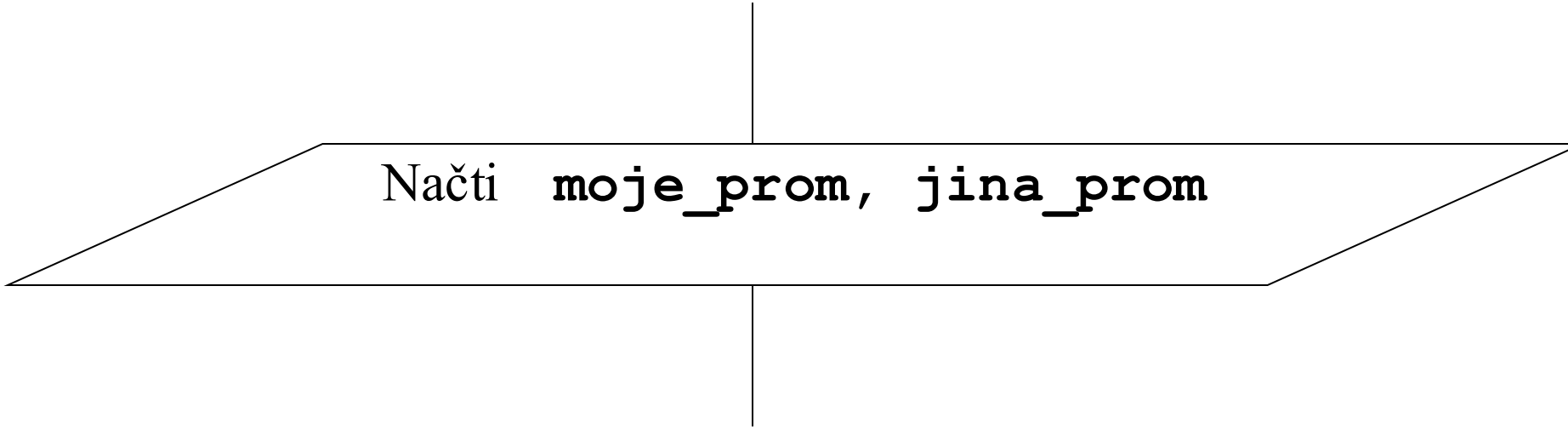


Postup pro vstup dat z klávesnice, jedna z možných variant :

```
float jina_prom;
. . .
Scanner sc = new Scanner(System.in);
. . .

jina_prom = sc.nextFloat();
```

Načtení čísla v plovoucí řádové čárce

Vstup více proměnných …



Postup pro vstup dat z klávesnice, jedna z možných variant :

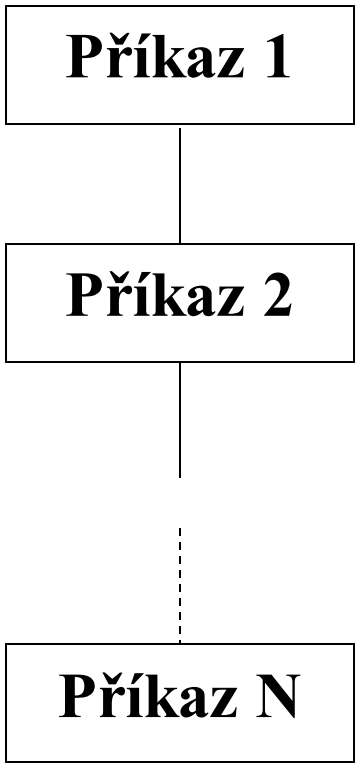
```
float jina_prom; int moje_prom;
. . .
Scanner sc = new Scanner(System.in);
. . .
```

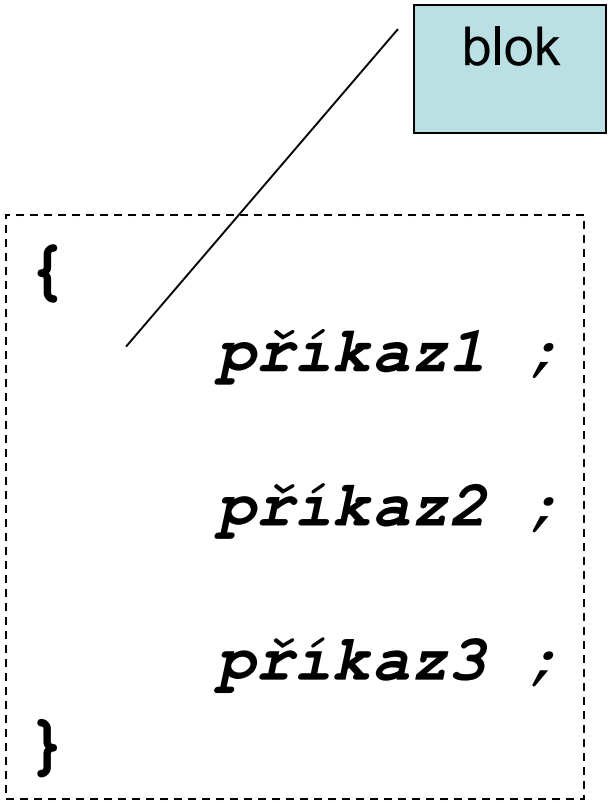
```
moje_prom = sc.nextInt();
jina_prom = sc.nextFloat();
```

První řídicí struktury

Složené příkazy - sekvence:

Sekvence je tvořena posloupností jednoho nebo více příkazů, které se provádějí v pevně daném pořadí. Příkaz se začne provádět až po ukončení předchozího příkazu.





Nyní máme k dispozici vše, abychom mohli „poskládat“ algoritmus pro demonstrační úlohu 1.1

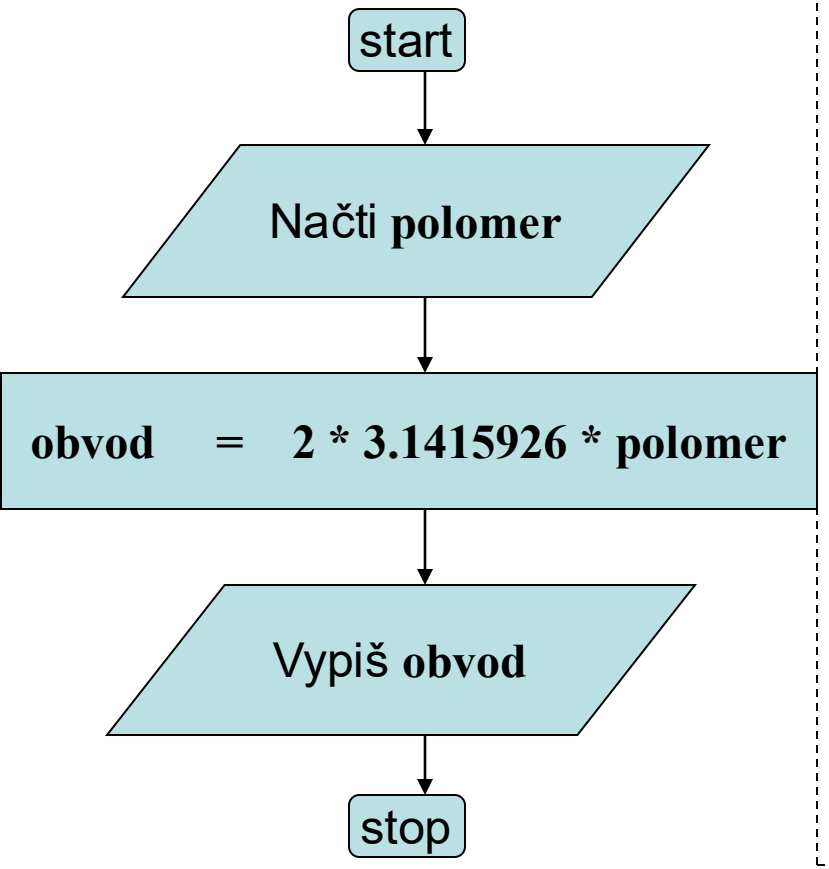
použijeme následující „stavební bloky“ :

- ***příkaz vstupu*** *pro reálnou proměnnou s identifikátorem „polomer“*
- ***přiřazovací příkaz*** *k výpočtu obvodu a zapamatování výsledku v reálné proměnné s identifikátorem „obvod“*
- ***Příkaz výstupu*** *pro výpis výsledku uloženého v proměnné „obvod“*

- *Vše zřetězíme do sekvence příkazů*

DP 1.1 – pokračování

Př. Sestavte algoritmus pro výpočet obvodu kruhu. Poloměr kruhu bude zadán z klávesnice, výsledek vypište na obrazovku




```
{

   float polomer, obvod;

   Scanner sc = new Scanner(System.in);


   polomer = sc.nextFloat();


   obvod = 2 * 3.1415926 * polomer ;


   System.out.printf("obvod je %f \n", obvod );


}
```

Nyní je výkonná část kódu kompletní, ale program ještě v této podobě zkompilovat nepůjde, zbývá doplnit pár technických drobností…..

Závěrečné poznámky k základní struktuře programu v jazyce Java

aneb ještě musíme probrat několik prvních pravidel pro formálně správný zápis programu v jazyce Java.

(zatím co nejvíce “procedurálním“ způsobem)

DP 1.1 – dokončení

Př. Sestavte algoritmus pro výpočet obvodu kruhu. Poloměr kruhu bude zadán z klávesnice, výsledek vypište na obrazovku

```
import java.util.Scanner;

class DP11
{
        public static void main(String args[])
        {
                float polomer, obvod;

                System.out.printf("Vlozte polomer:");

                Scanner sc = new Scanner(System.in);
                polomer = sc.nextFloat();

                obvod = 2 * (float)Math.PI * polomer ;
                System.out.printf("obvod je %f \n", obvod );

        }

}
```

DP 1.1 – dokončení

Jiná varianta …

```
import java.util.Scanner;

class DP11_varianta2
{
        public static void main(String args[])
        {
                double polomer=1.0, obvod;

                try (Scanner sc = new Scanner(System.in)) {
                        polomer = sc.nextFloat();

                } catch (Exception e) {
                        System.out.printf("Chyba vstupu, ponecham
                vychozi hodnotu polomeru %f\n", polomer);
                }

                obvod = 2 * Math.PI * polomer ;
                System.out.printf("obvod je %f \n", obvod );

        }
}
```

DP 1.1 – dokončení

Př. Sestavte algoritmus pro výpočet obvodu kruhu.
Poloměr kruhu bude zadán z klávesnice, výsledek vypište na obrazovku

Příklady testovacích dat

Vstup: polomer	očekávaný výstup: obvod
1	6.28318
0	0
5.6	35.18583

Chybná vstupní data

-1	ch. hláška, místo toho je -6.28318
Abc	ch. hláška, místo toho je pád

První praktické kroky v Javě

Co budeme potřebovat …

- **Editor** – napsání zdrojového textu programu, ideálně programátorský, např. **VS Code**. Výhodou je možnost rozšířit pomocí plugin až téměř do podoby plnohodnotného IDE pro Javu
  Jiné varinaty: IntelliJ IDEA, Eclipse, Netbeans, ….

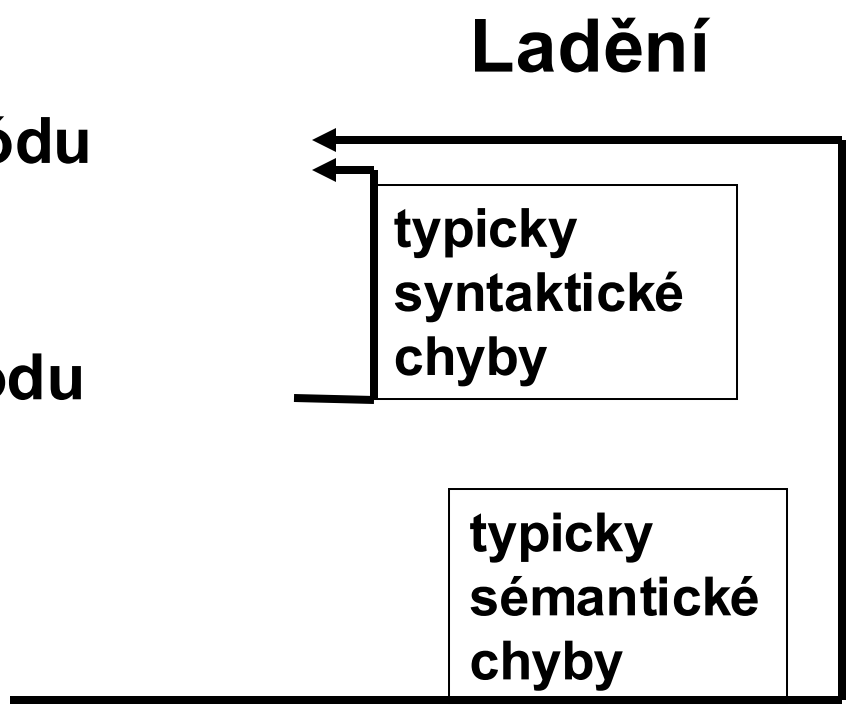
- **Vývojové a běhové prostředí Java** –
  stáhneme některé JDK, např. od Oracle, to obsahuje i potřebné JRE. Java je jazyk kompilovaný do mezikódu, který je vykonáván virtuálním strojem (JVM) - viz dále.

- Provedení a testování a pravděpodobně re-editace, re-kompilace, …

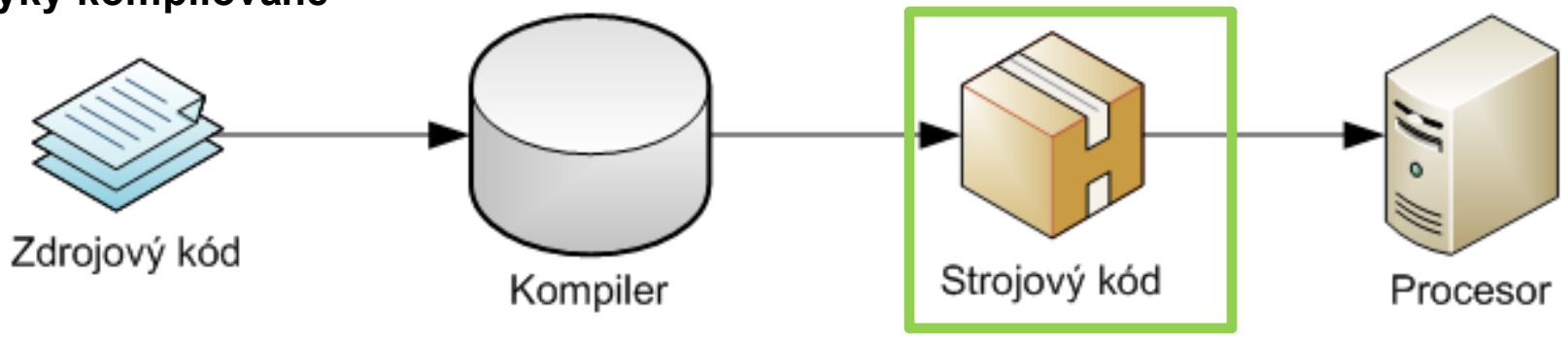
fáze vývoje programu

- editace zdrojového kódu
- překlad zdrojového kódu
- spuštění programu

. . . . výhody použití IDE

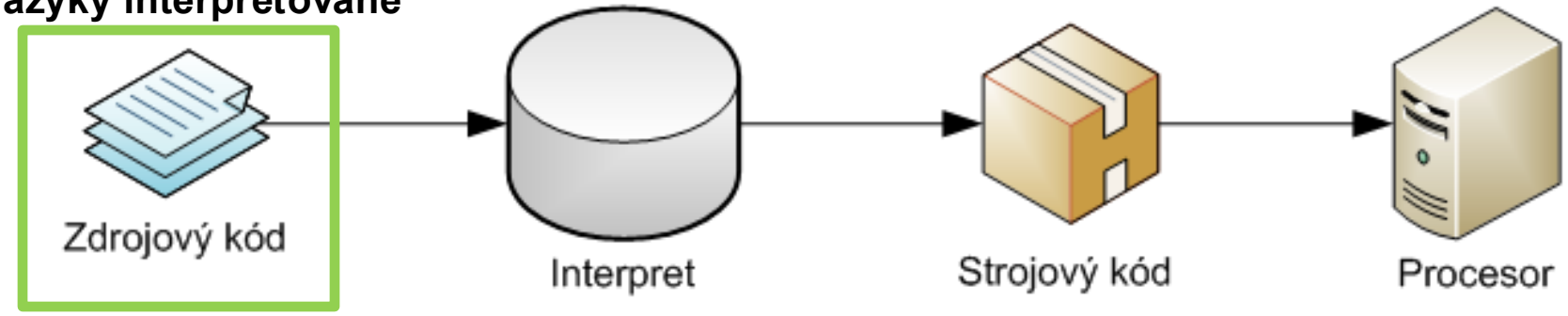


Jazyky kompilované



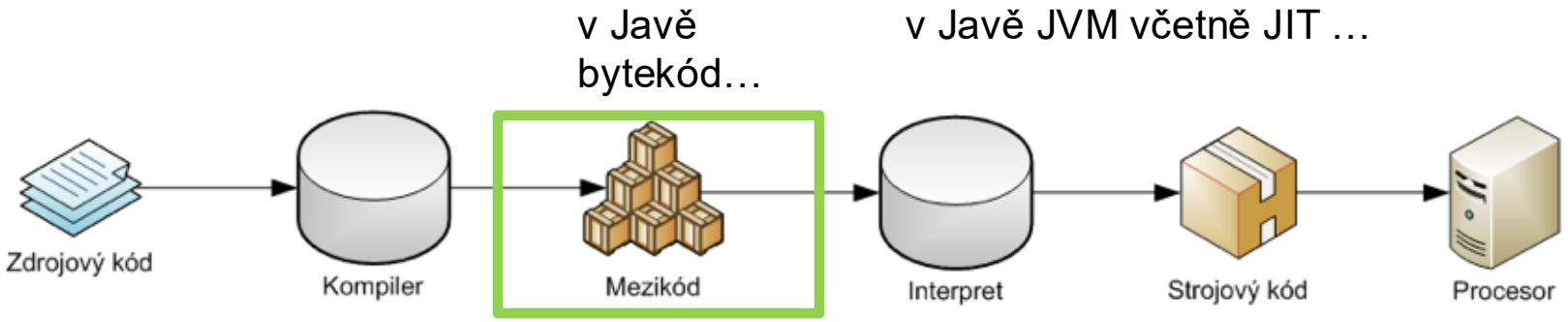
+ maximální rychlost, obtížné reverzní inženýrství
- závislé na platformě, nemožnost změny programu bez rekompilace, u klasických kompilovaných jazyků jako C, C++ obtížná správa paměti (potenciální zdroj chyb buffer overflow, memory leak, porušení ochrany paměti, ukazatel nikam neukazující, …)

Jazyky interpretované



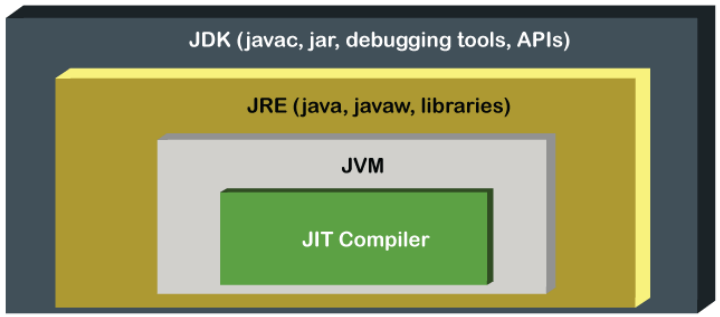
+ přenositelnost, jednoduchý vývoj, možnost úprav bez rekompilace
- pomalost, některé chyby které by zachytil kompiler, se projeví až v run-time, snadné reverzní inženýrství

Jazyky s virtuálním strojem



+ uspokojivá rychlost, středně těžké reverzní inženýrství, dobrá přenositelnost, odhalení mnoha chyb přísným kompilátorem, virtuální stroj má obvyke GC
(takže prevence chyb v alokaci paměti….)

- JRE včetně JVM je gigantické + má to přece jenom nižší výkon jazykům kompilovaným do strojového kódu



Ahoj světe v Javě !

```
class AhojSvete
{
    public static void main(String []args)
    {
        System.out.println(”Ahoj Svete.");
    }

};
```

V mezičase si nainstalujeme VS Code, JDK a potřebné pluginy.

Ukážeme si proces kompilace, hledání syntaktických a chytání sémantických chyb atd…

Zkusíme si rozběhnout úlohu DP 1.1 a navážeme na něj dalšími příklady…

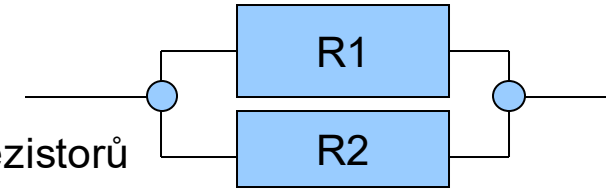
Tip k dalším vlastním experimentům ….
(výpisy hlášek do konzole česky i pod Windows)

```
import java.io.*;
import java.nio.charset.Charset;

class AhojSveteCZ {
        public static void main(String[] args)
        {
                /* kodovani nastavime na UTF-8 */
                OutputStreamWriter osw = new OutputStreamWriter(System.out,
                        Charset.forName("UTF-8").newEncoder() );
                System.out.println("Nastavene kodovani konzole: " +
        osw.getEncoding());
                PrintWriter p = new PrintWriter(osw);

                /* a muzeme psat na konzoli cesky */
                p.print("Příšerně žluťoučký kůň úpěl ďábelské ódy.\n");
                p.print("áčďéěíňóřšťúůýž\n");
                p.print("PŘÍŠERNĚ ŽLUŤOUČKÝ KŮŇ ÚPĚL ĎÁBELSKÉ ÓDY.\n");
                p.printf("ÁČĎÉĚÍŇÓŘŠŤÚŮÝŽ\n");
                p.flush();
        }

}
```

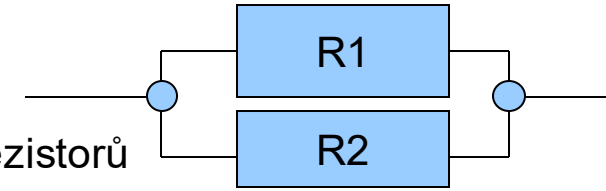
Další demonstrační příklady

DP 1.2

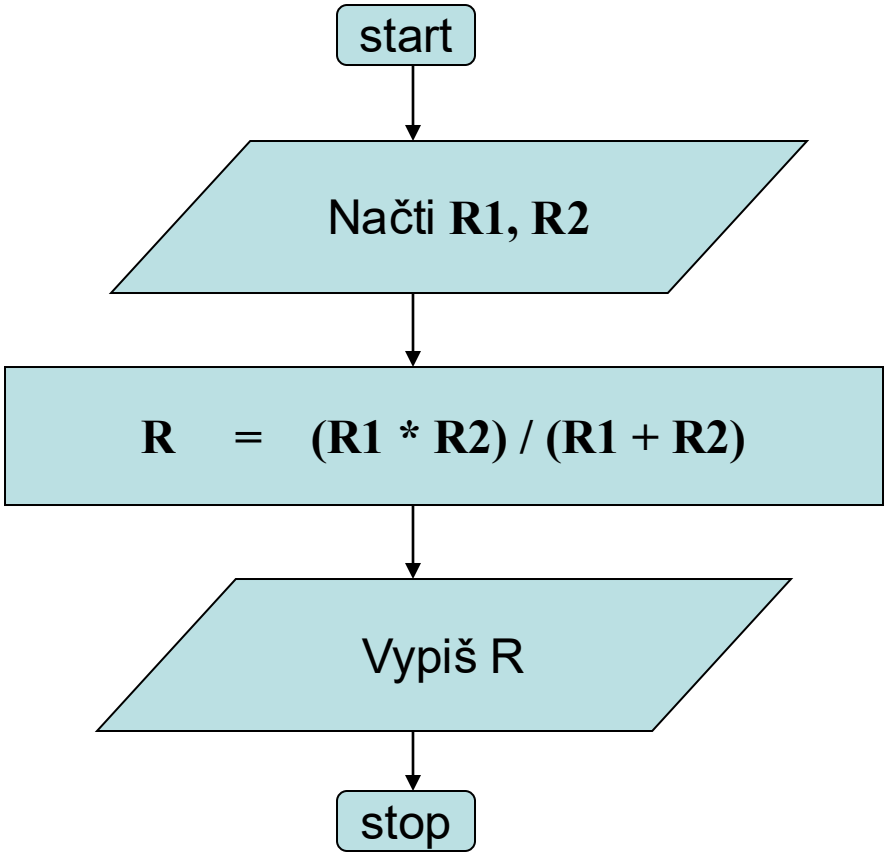
Př. Napište algoritmus pro výpočet celkového odporu dvou rezistorů při paralelním zapojení.



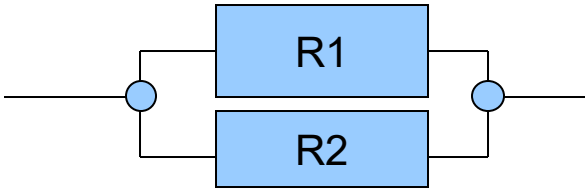
DP 1.2



Př. Napište algoritmus pro výpočet celkového odporu dvou rezistorů při paralelním zapojení.



DP 1.2

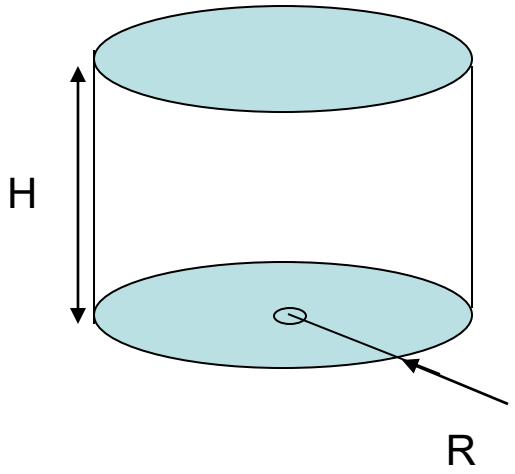


Př. testovacích dat …

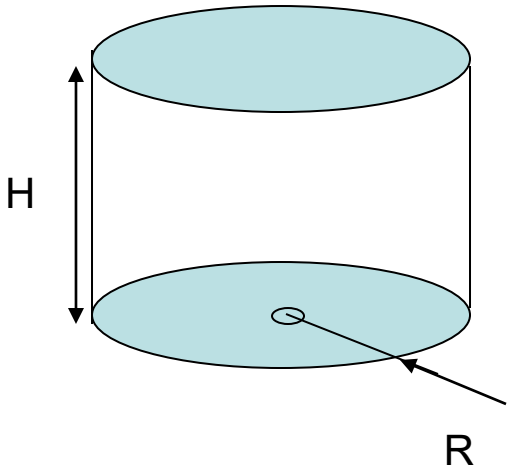
R1	**R2**	**R1 || R2**
10	10	5
1	100	0,990099
-20	30	oček. ch. hláška, místo toho je -60
Abc		oček. ch. hláška, místo toho je … pád

DP 1.3

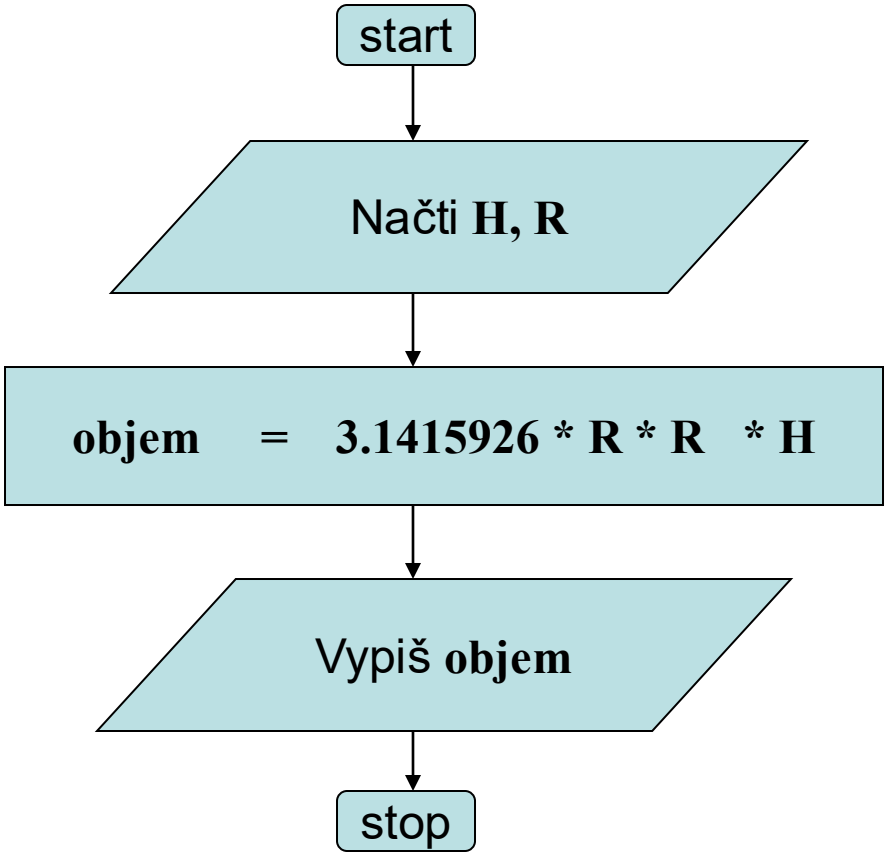
Př. Napište algoritmus pro výpočet objemu válcové nádoby, známe-li předem poloměr její podstavy a výšku.



DP 1.3



Př. Napište algoritmus pro výpočet objemu válcové nádoby, známe-li předem poloměr její podstavy a výšku.



Podmíněný příkaz

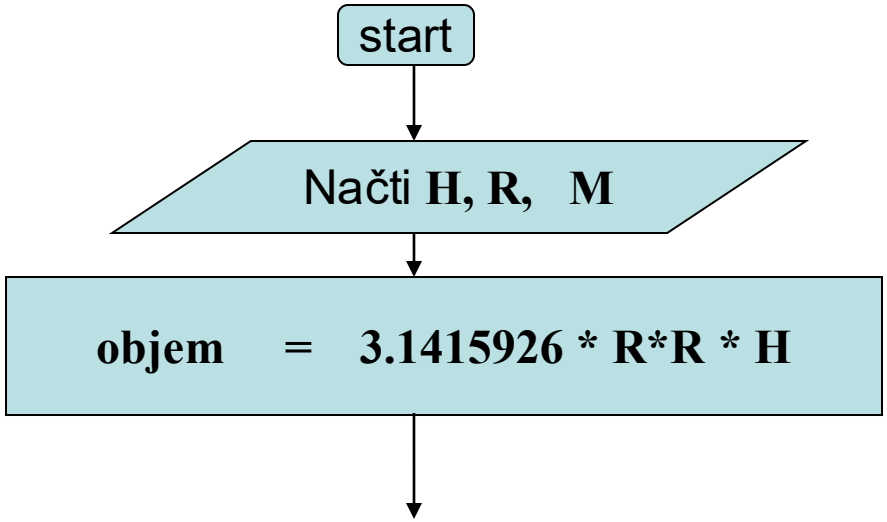
(if a switch-case) v Javě a logické výrazy

DP 1.4

MOTIVAČNÍ PŘÍKLAD …..

- **Př.** Mějme válcovou nádobu určenou poloměrem podstavy R a výškou H (v dm).

Sestavte algoritmus, který určí, zda se do nádoby vejde zadané množství vody M (v litrech).

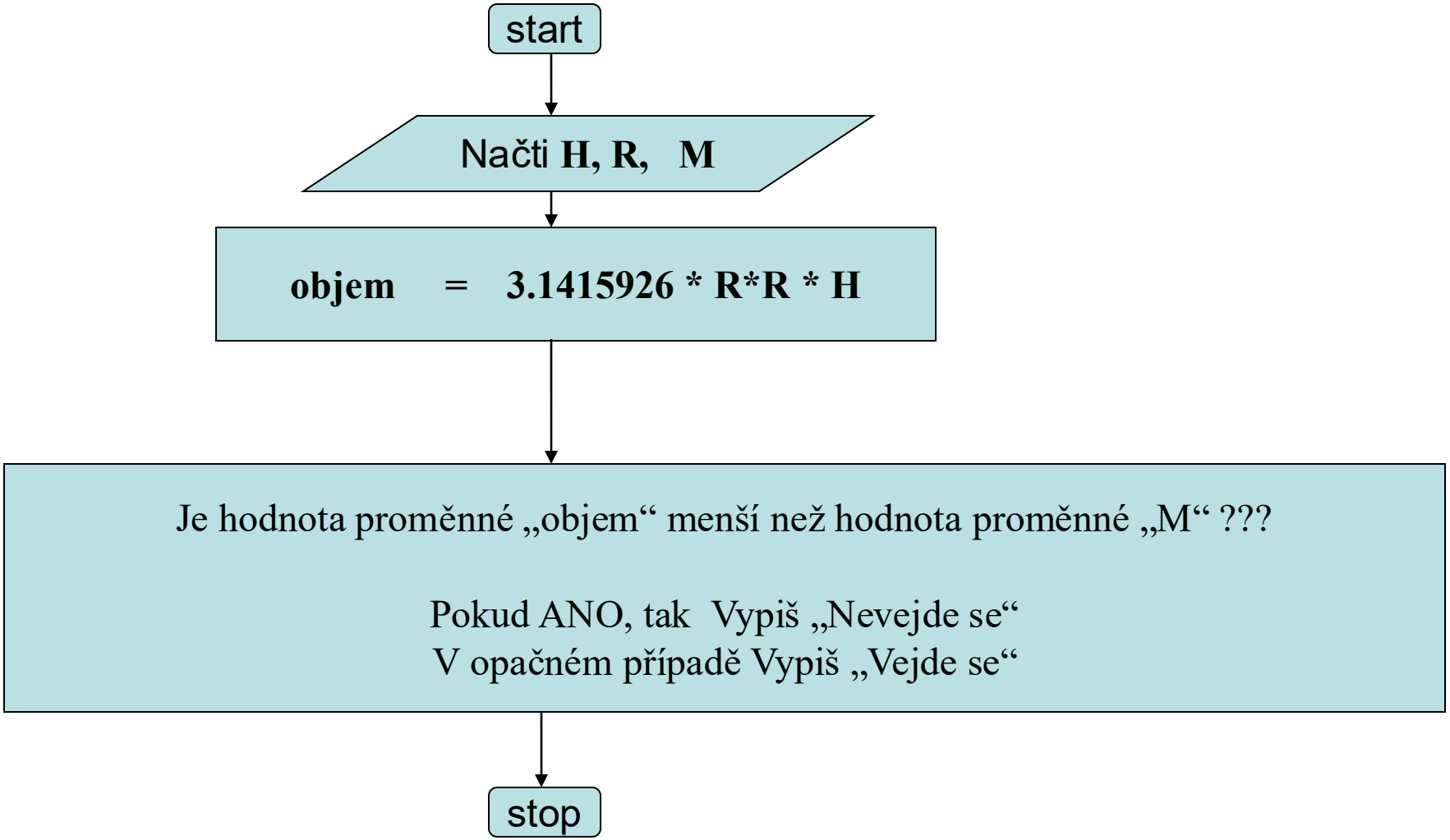


DP 1.4

MOTIVAČNÍ PŘÍKLAD …..

- **Př.** Mějme válcovou nádobu určenou poloměrem podstavy R a výškou H (v dm).

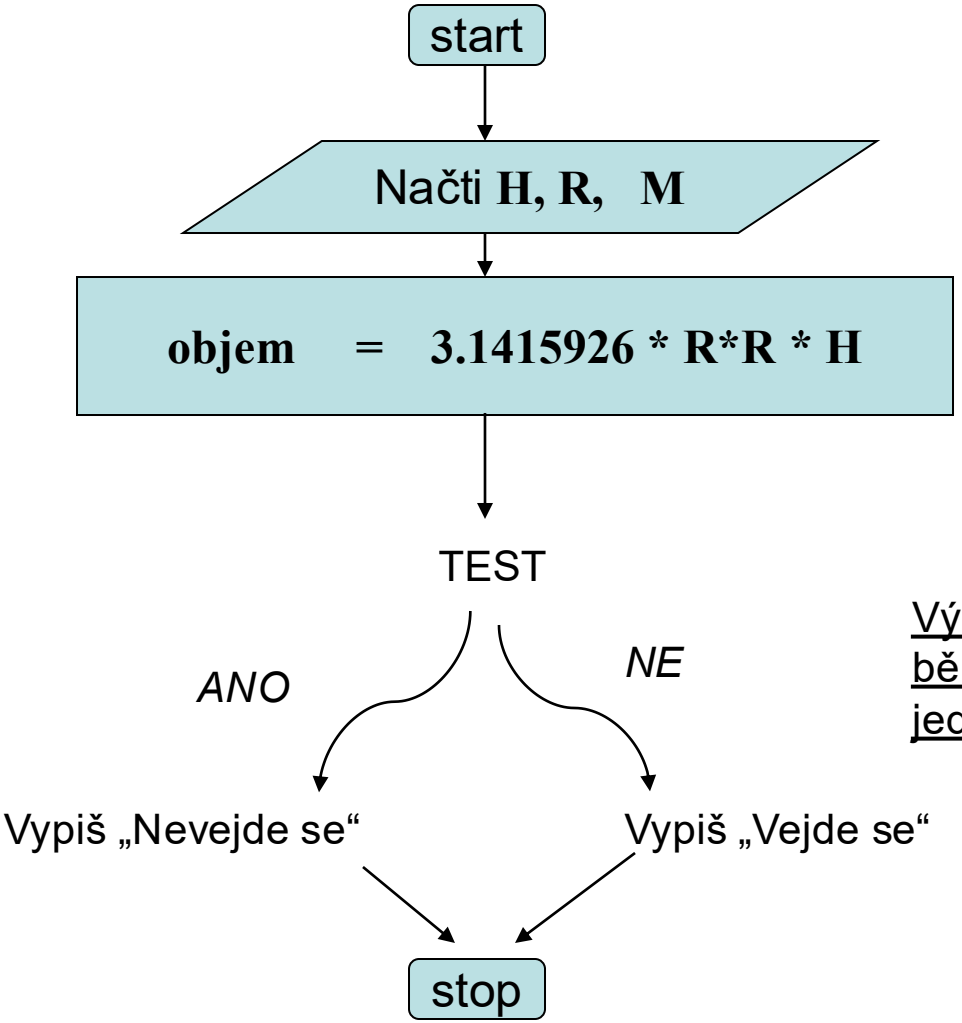
Sestavte algoritmus, který určí, zda se do nádoby vejde zadané množství vody M (v litrech).



DP 1.4

MOTIVAČNÍ PŘÍKLAD …..

- **Př.** Mějme válcovou nádobu určenou poloměrem podstavy R a výškou H (v dm). Sestavte algoritmus, který určí, zda se do nádoby vejde zadané množství vody M (v litrech).

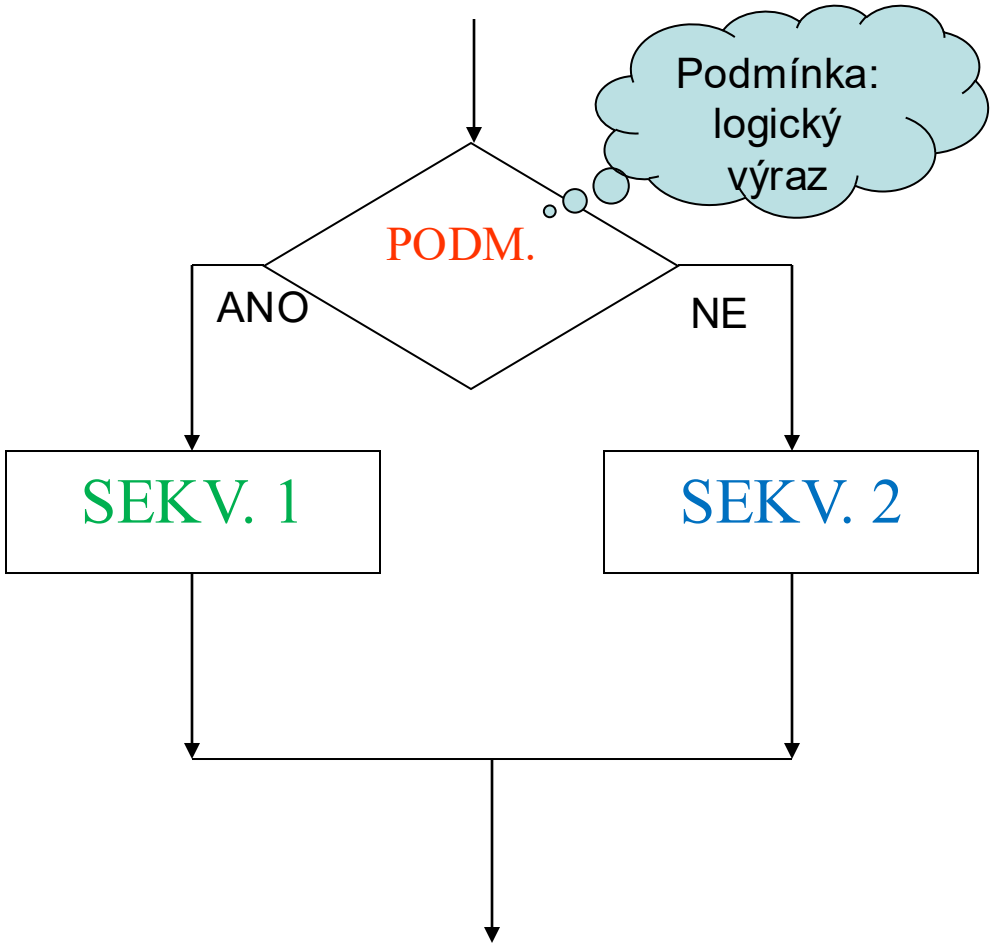


Výhybka,
běh pokračuje
jednou z větví …

Složené příkazy – selekce (podmínka, větvění):

Provedení dalšího kroku

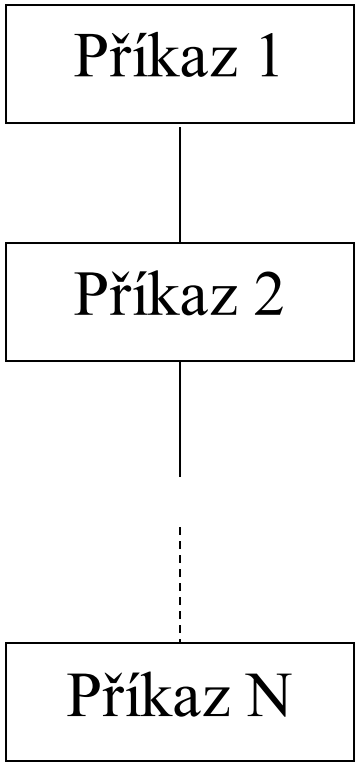
je podmíněno splněním podmínky.

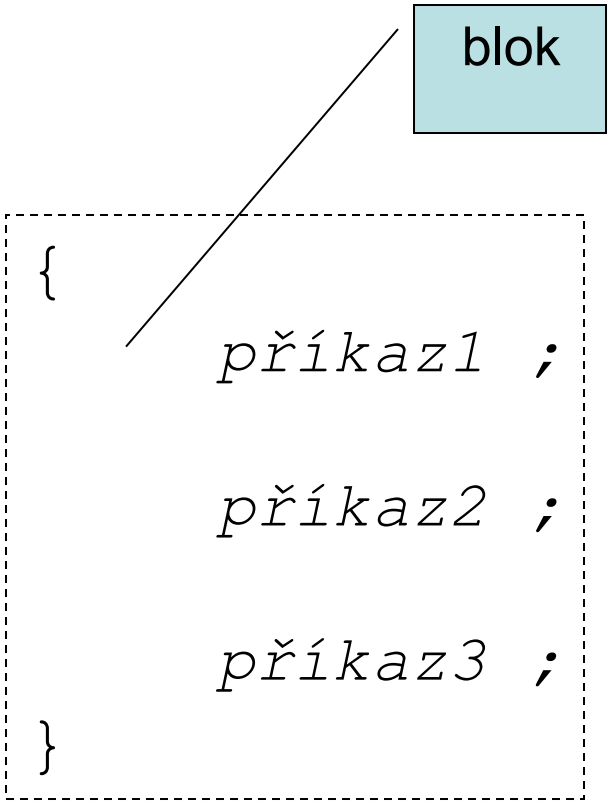



```
if ( PODM. ){

      SEKV.1;

} else {

      SEKV.2;
}
```

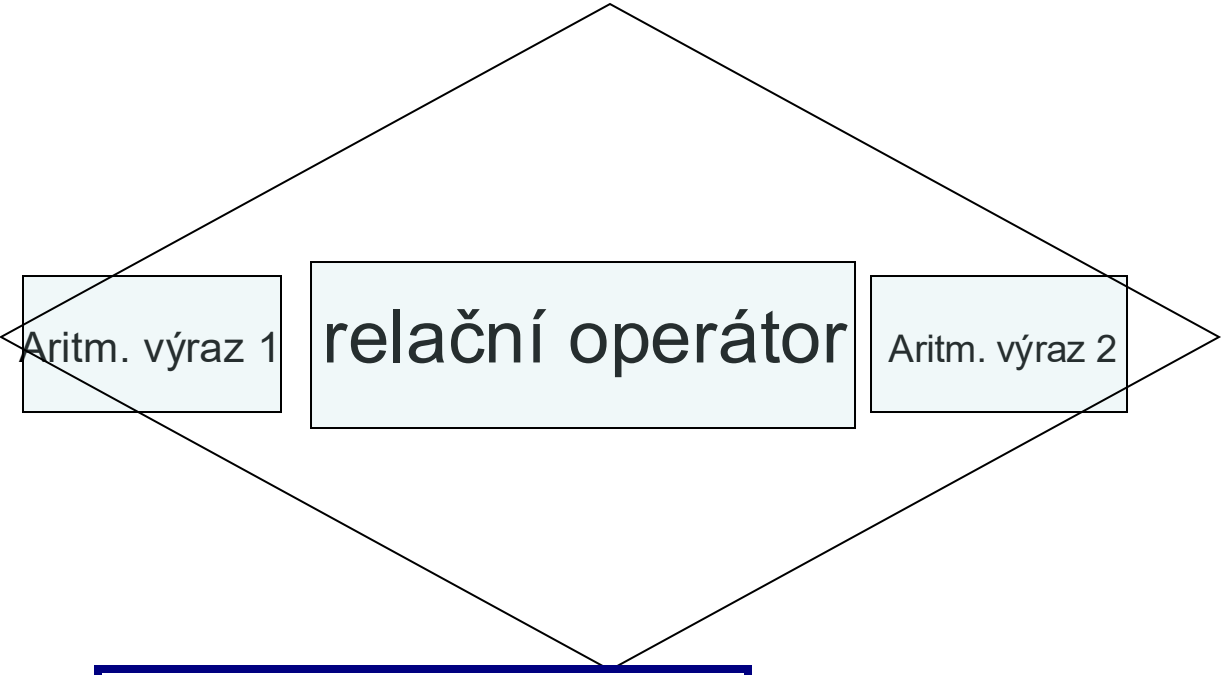
Připomínka co je SEKV. . . .

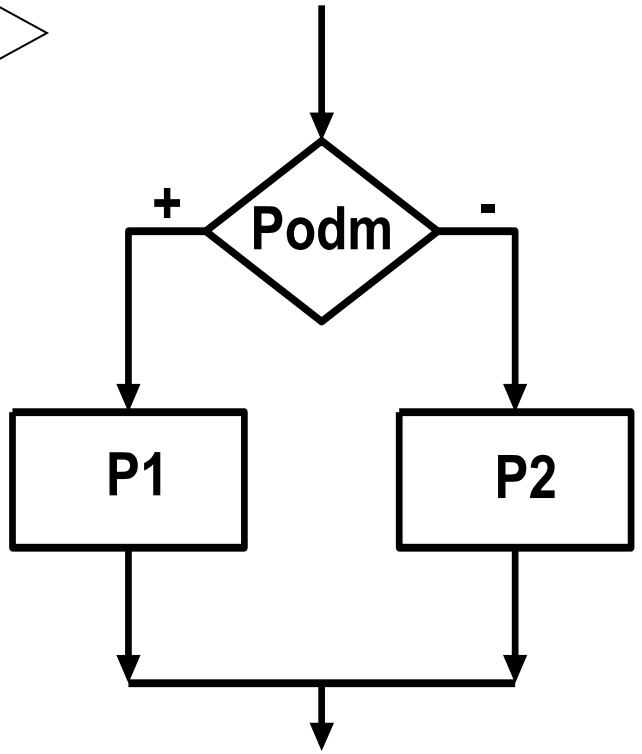
Složený příkaz - sekvence:





Minimální varianta log. výrazu PODM. - tkzv. „jednoduchá podmínka“

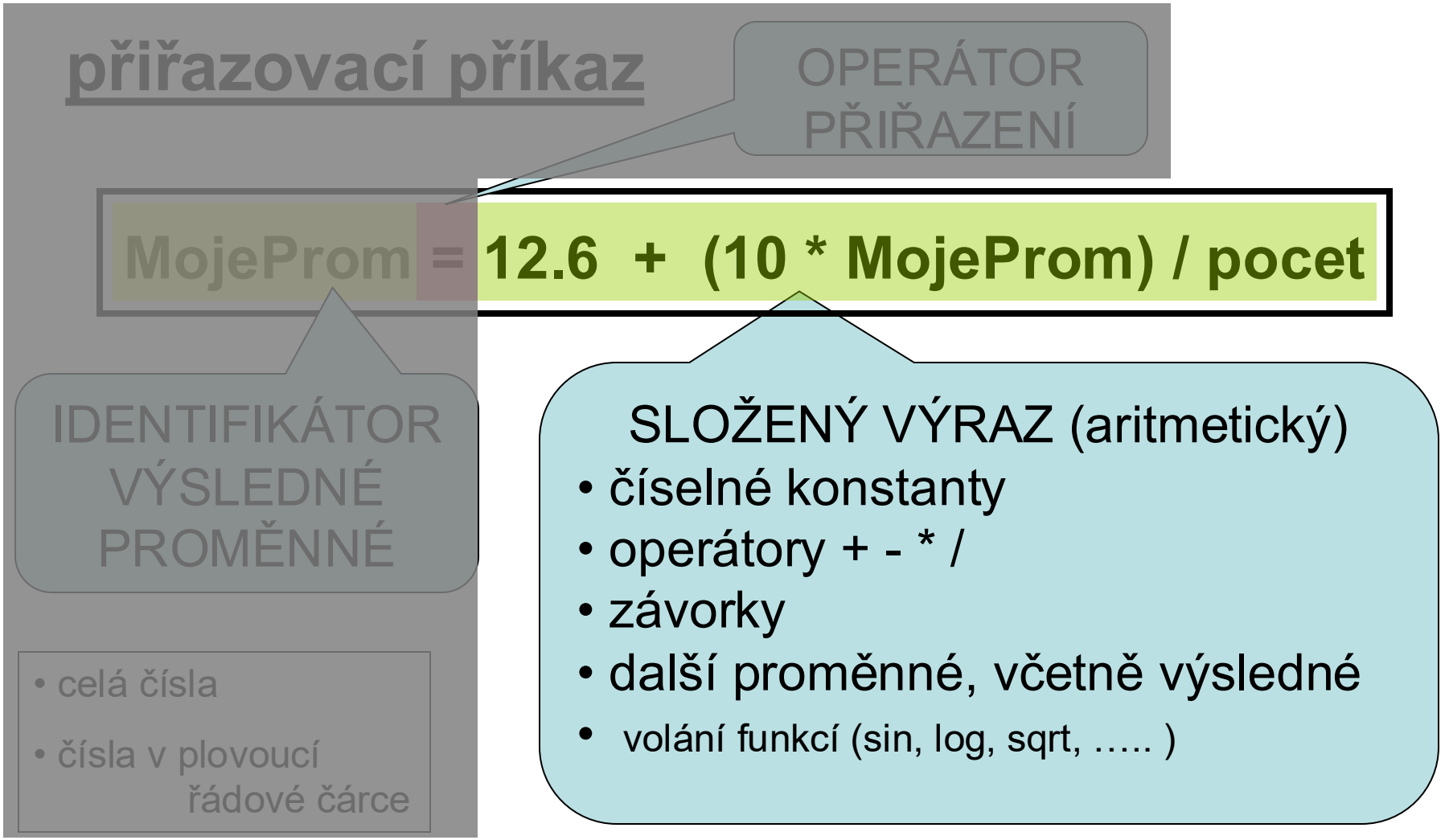




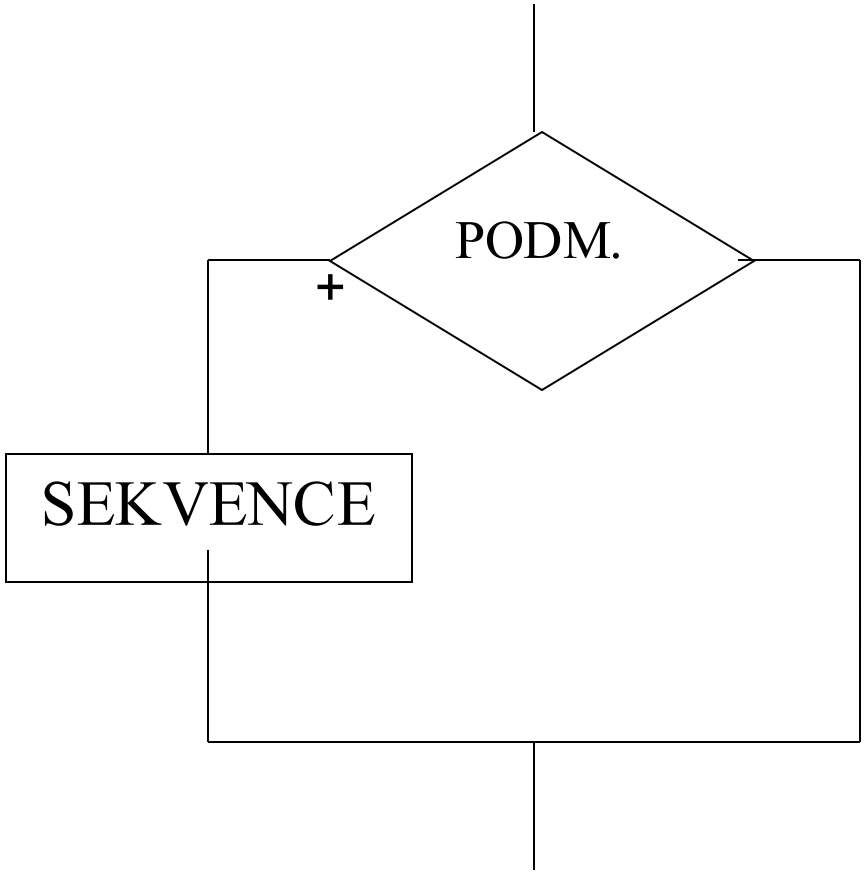
Relační operátory:

==	rovno
<	menší
>	větší
<=	menší nebo rovno
>=	větší nebo rovno
!=	nerovná se

Připomínka co je Aritm. výraz. . . .



Pro úplnost: tkzv. neúplná podmínka

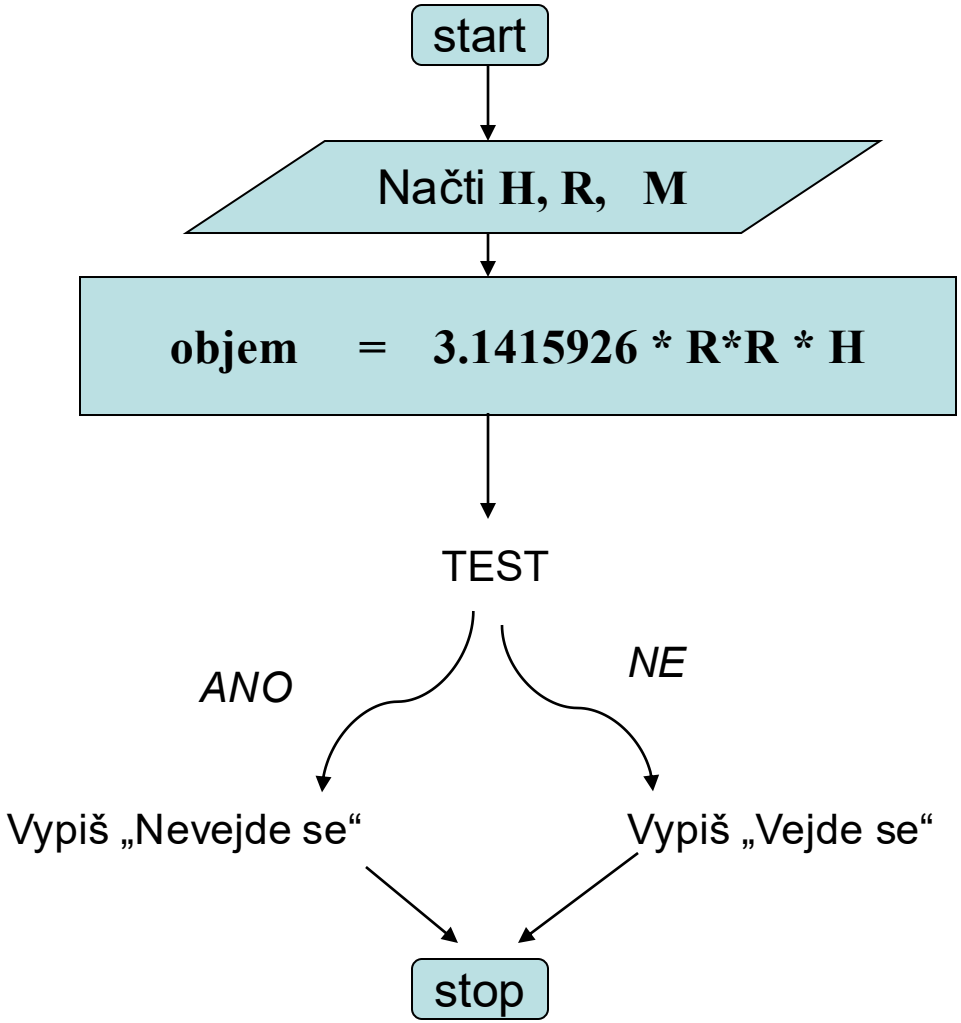



```
if ( PODM. ){

      SEKVENCE;
}
```

MOTIVAČNÍ PŘÍKLAD ….. (dokončení)

DP 1.4

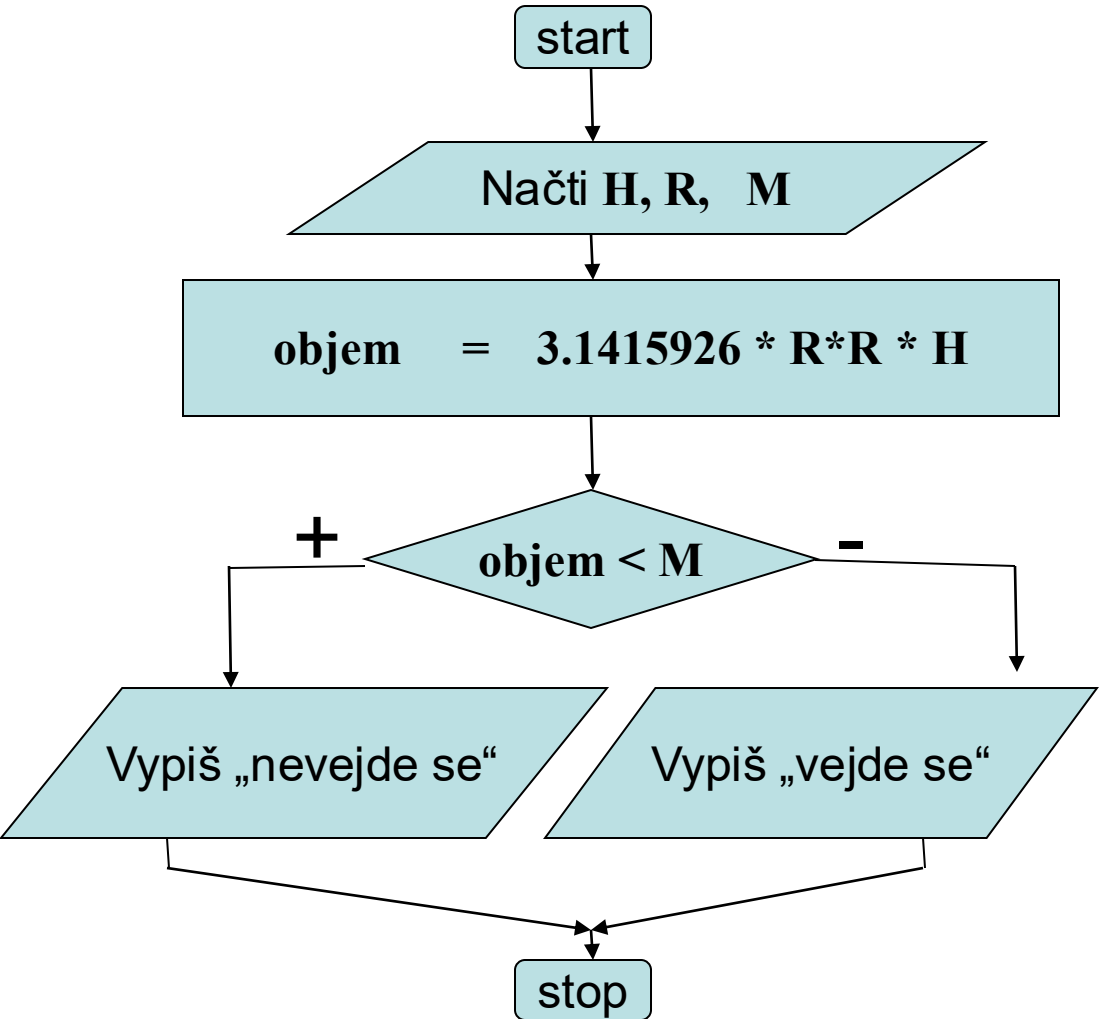
- Př. Mějme válcovou nádobu určenou poloměrem podstavy R a výškou H (v dm).
  Sestavte algoritmus, který určí, zda se do nádoby vejde zadané množství vody M (v litrech).



Výhybka,
běh pokračuje
jednou z větví …

MOTIVAČNÍ PŘÍKLAD ….. (dokončení)

- **Př.** Mějme válcovou nádobu určenou poloměrem podstavy R a výškou H (v dm).
  Sestavte algoritmus, který určí, zda se do nádoby vejde zadané množství vody M (v litrech).



Promyslete alternativní formy podmínky, např.

M<=objem

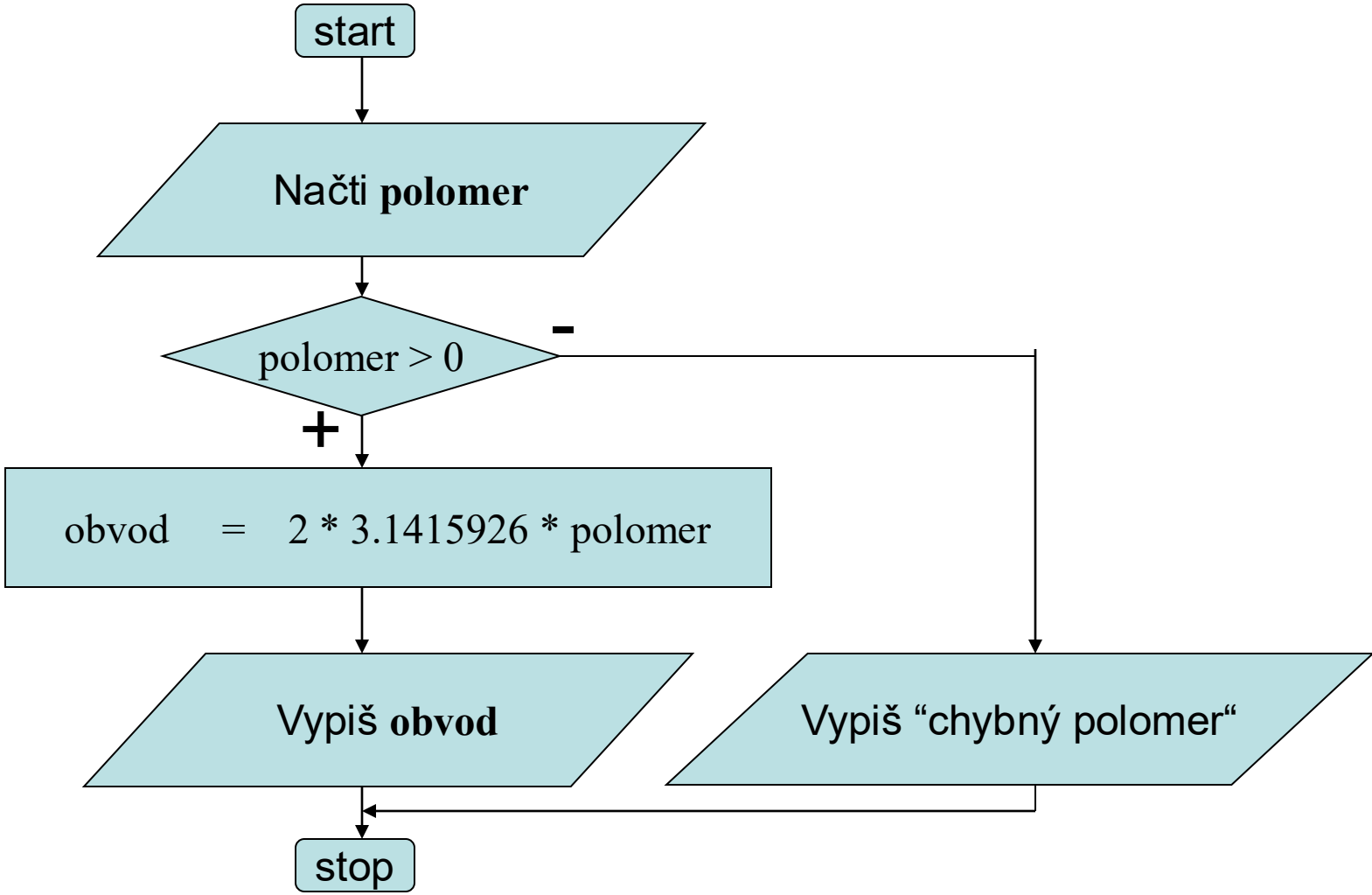
DP 1.4

```
import java.util.Scanner;


class DP21
{
    public static void main(String args[])
    {
        double H, R, M, objem;

        Scanner sc = new Scanner(System.in);
        H = sc.nextFloat();
        R = sc.nextFloat();
        M = sc.nextFloat();

        objem = Math.PI * R*R * H;


        if (objem < M) {
          System.out.printf("Nevejde se.\n" );
        } else {
          System.out.printf("Vejde se.\n" );
        }

    }

}
```

DP 1.5

Podmíněný příkaz

- **Př.** Ošetření smysluplnosti vstupu v příkladu DP1.1



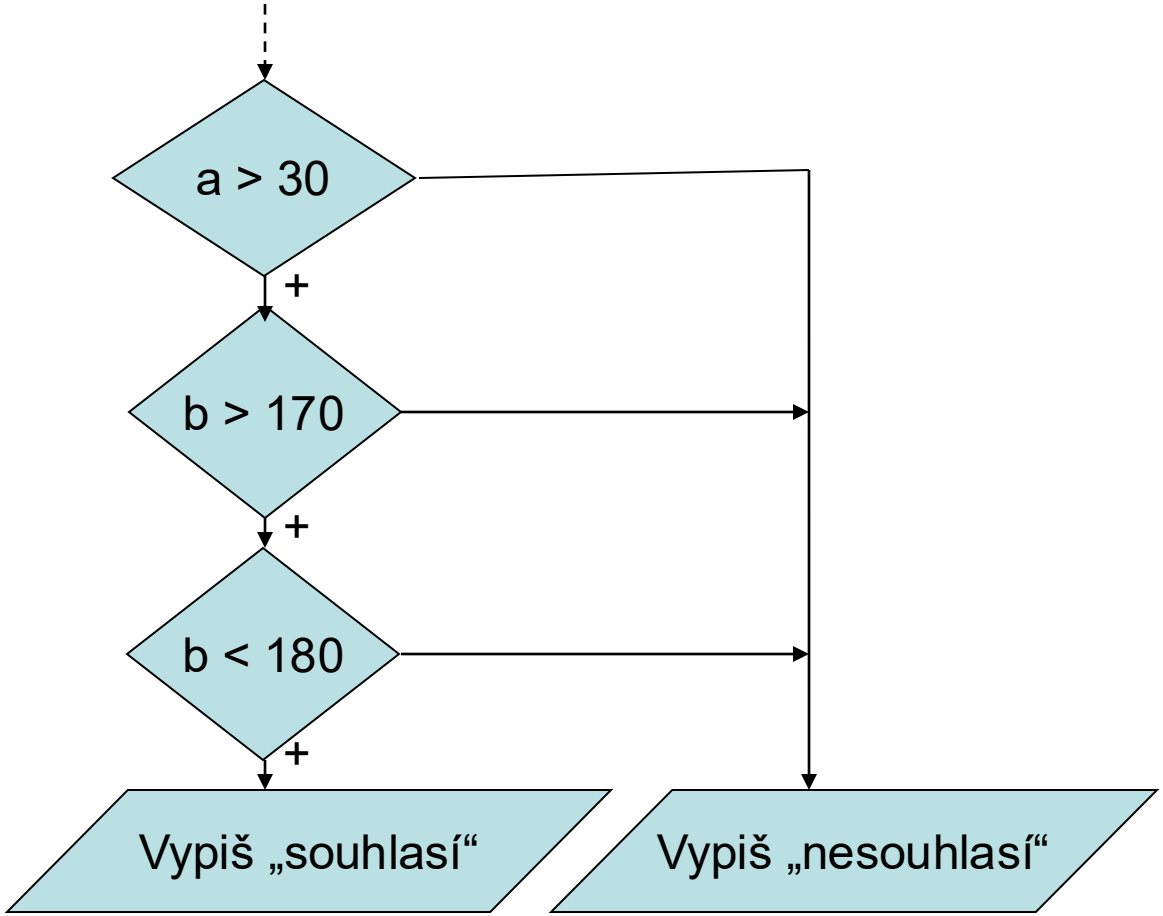
DP 1.6

Podmíněný příkaz

Př.

Známe věk pacienta **a** [roky] a výšku pacienta **b** [cm].

Chceme algoritmus, který určí, zda je pacient starší než 30 let a zároveň velký mezi 170 a 180 cm.



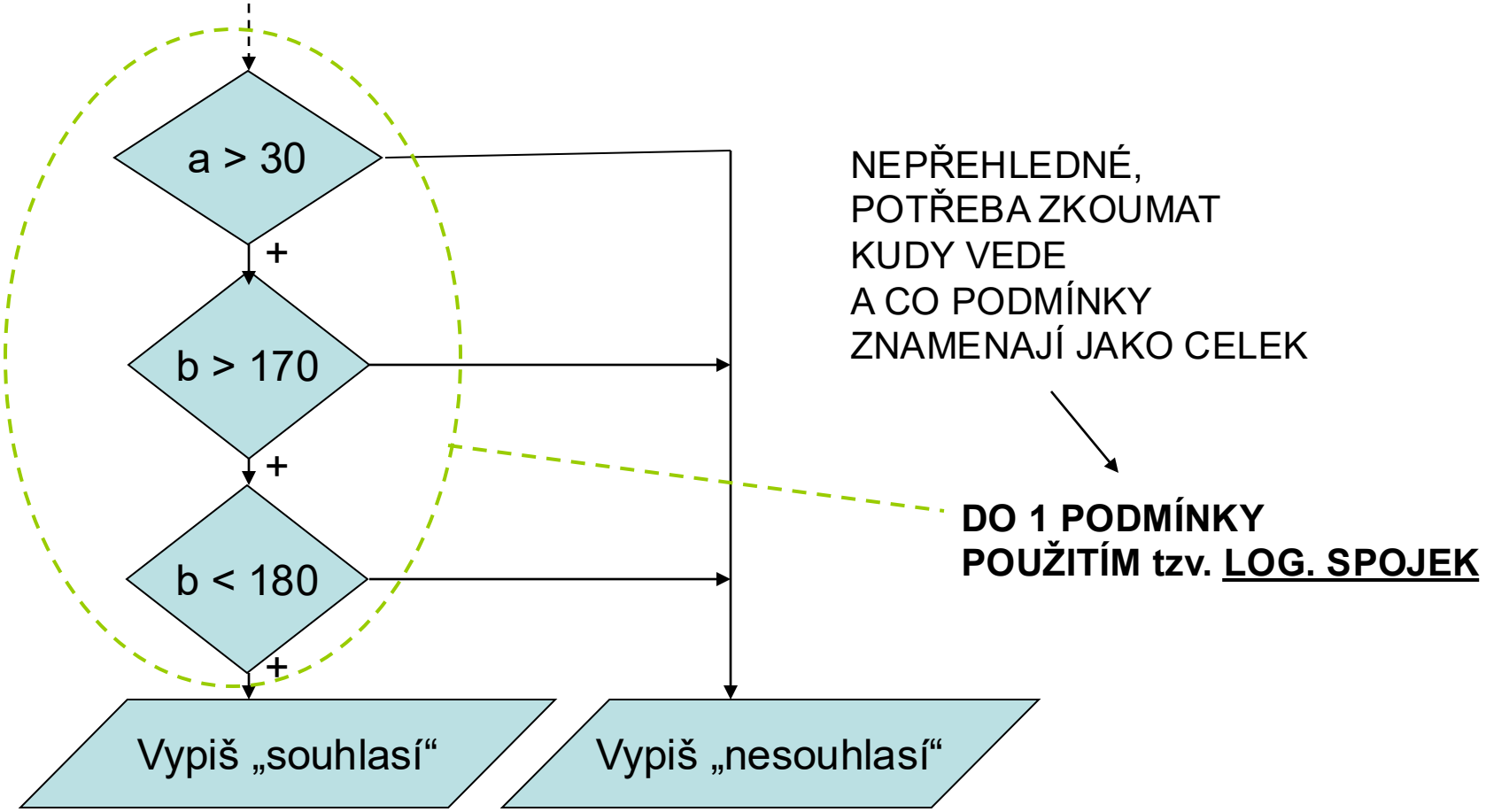
DP 1.6

Podmíněný příkaz

Př.

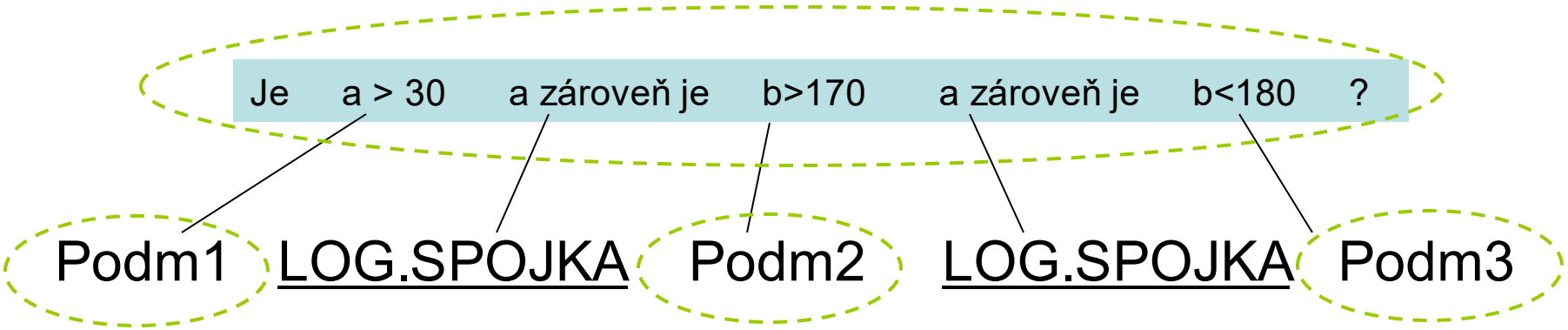
Známe věk pacienta **a** [roky] a výšku pacienta **b** [cm].

Chceme algoritmus, který určí, zda je pacient starší než 30 let a zároveň velký mezi 170 a 180 cm.



DP 1.6 – pokrač.

Skládání log. výrazů pomocí log. spojek



možnost složit v komplikovanější výrazy

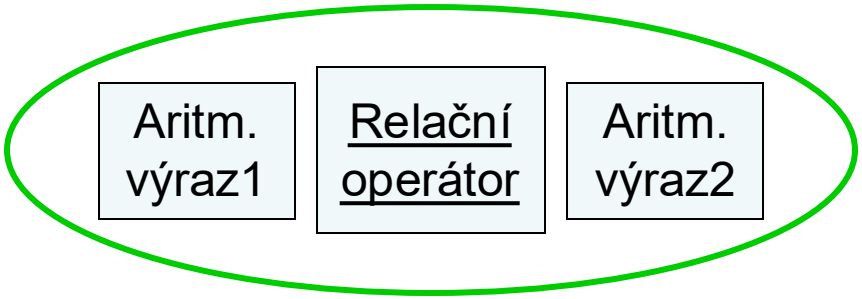
Logické spojky:

a zároveň	konjunkce
nebo	disjunkce
negace	negace

<u>význam:</u> úspornější a někdy přehlednější zápis, než při použití více podm. příkazů

Podmíněný příkaz - použití logických spojek:

PODM. ..... logický výraz

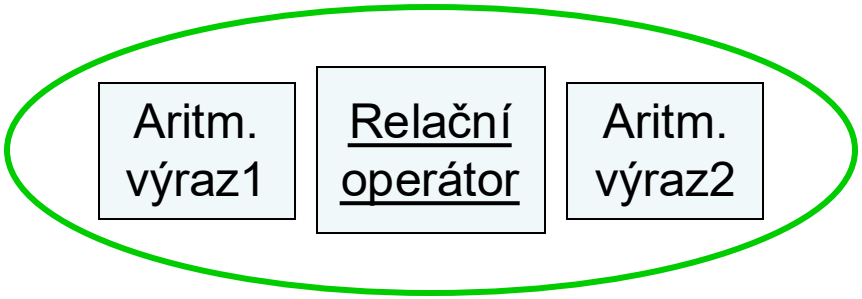


Relační operátory:

==	rovno
<	menší
>	větší
<=	menší nebo rovno
>=	větší nebo rovno
!=	nerovná se

Podmíněný příkaz - použití logických spojek:

PODM. ..... logický výraz



Logické výrazy možno skládat - řetězit a vnořovat :

(log.v. 1) LOG.SPOJKA (log.v 2) LOG.SPOJKA ((log.v 3) LOG.SPOJKA (log.v 4))

Relační operátory:

==	rovno
<	menší
>	větší
<=	menší nebo rovno
>=	větší nebo rovno
!=	nerovná se

Logické operátory :

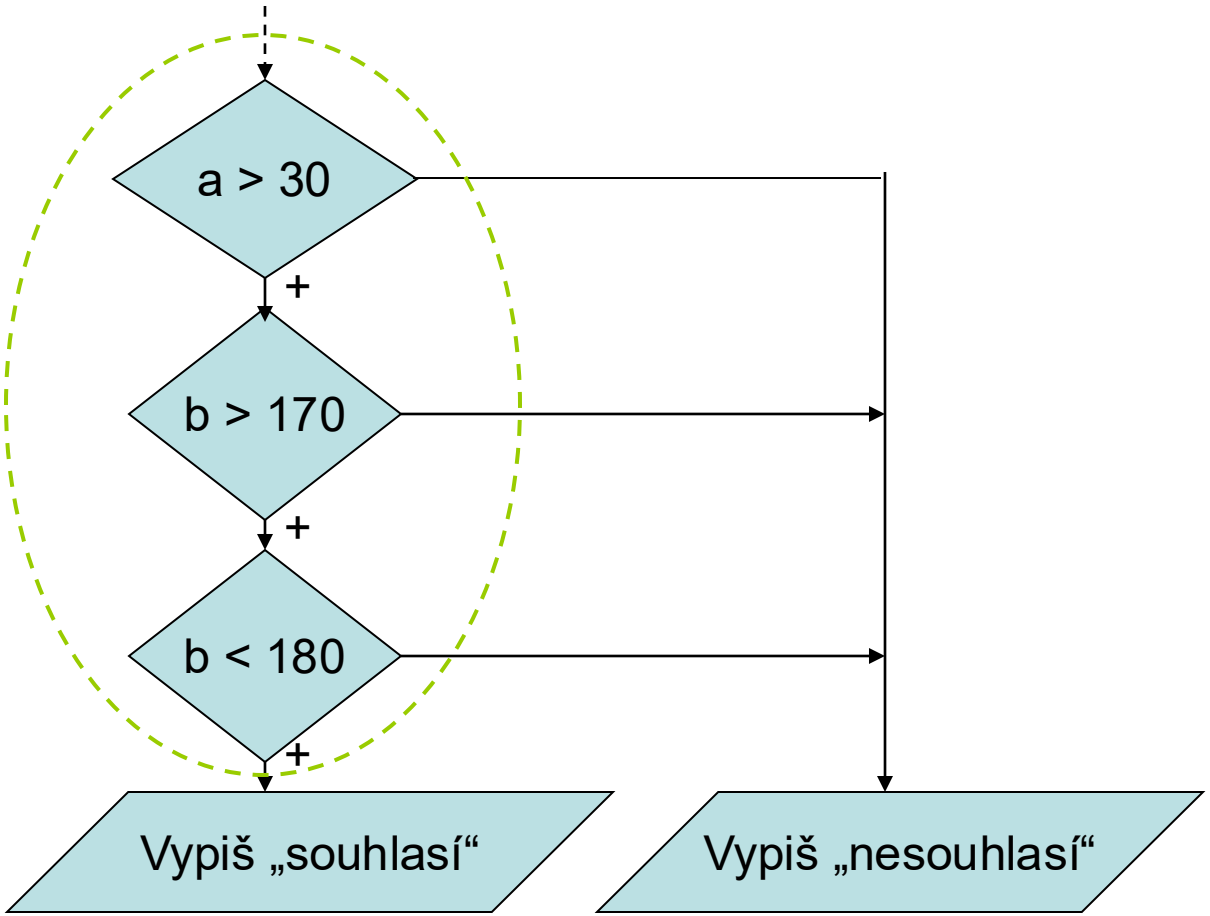
&&	a zároveň,	AND
||	nebo,	OR
!	negace,	NOT

DP 1.6 – pokrač.

Skládání log. výrazů pomocí log. spojek

Př.

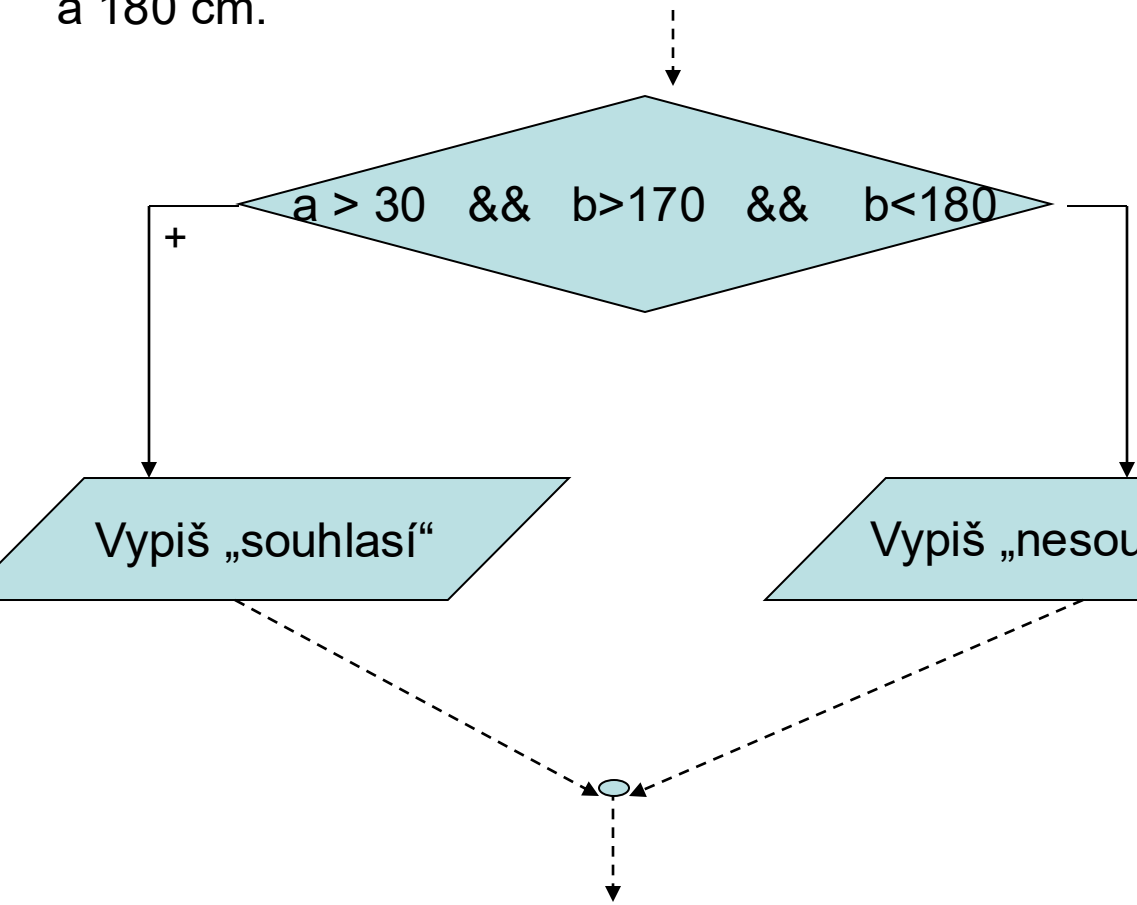
Známe věk pacienta a [roky] a výšku pacienta b [cm].
Chceme algoritmus, který určí, zda je pacient starší než 30 let a zároveň velký mezi 170 a 180 cm.



DP 1.6 – dokonč.

DOKONČENÍ - Skládání log. výrazů pomocí log. spojek

Př.
Známe věk pacienta a [roky] a výšku pacienta b [cm]. Chceme algoritmus, který určí, zda je pacient starší než 30 let a zároveň velký mezi 170 a 180 cm.




```
. . .

if ( a>30 && b>170 && b<180 )
{
   printf ("souhlasi");
}
else
{
   printf("nesouhlasi");
}
. . .
```

DP 1.6 – dokonč.

```
import java.util.Scanner;

class DP23
{
	public static void main(String args[])
	{
		double a, b;

		Scanner sc = new Scanner(System.in);
		a = sc.nextFloat();
		b = sc.nextFloat();

		if (a>30 && b>170 && b<180) {
			System.out.printf("Zaradit.\n" );
		} else {
			System.out.printf("Nezarazovat.\n" );
		}

	}

}
```

logický typ

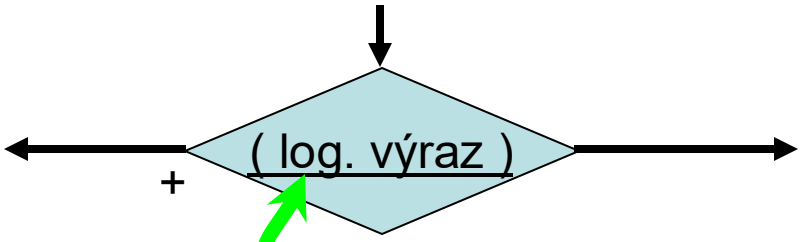
- Logický výraz
- Logické spojky (skládání-řetězení log. výrazů)
- Úpravy log. výrazů, log. doplněk podmínky
- **boolean**

logický typ

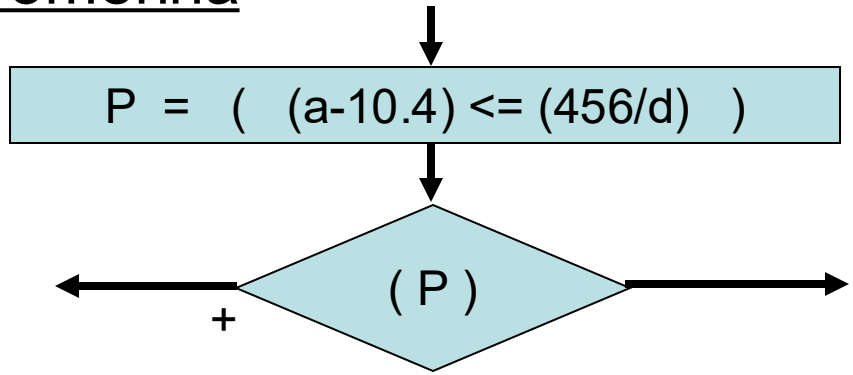
Obor hodnot - pouze 2 možné hodnoty: **1** nebo **0**
Ano nebo **Ne**
true nebo **false**

logický výraz P

Aritm. výraz1	Relační operátor	Aritm. výraz2
(a-10.4)	<=	456/d



logická proměnná



Relační operátory:

==	rovno
<	menší
>	větší
<=	menší nebo rovno
>=	větší nebo rovno
!=	nerovná se

Skládání log. výrazů pomocí log. spojek

Logické výrazy možno skládat - řetězit a vnořovat :

význam: úspornější a někdy přehlednější zápis, než při použití více podm. příkazů

(log. v. P1) LOG.SPOJKA (P2) LOG.SPOJKA (P3 LOG.SPOJKA P4) *atd…*

Logické spojky (operátory) :

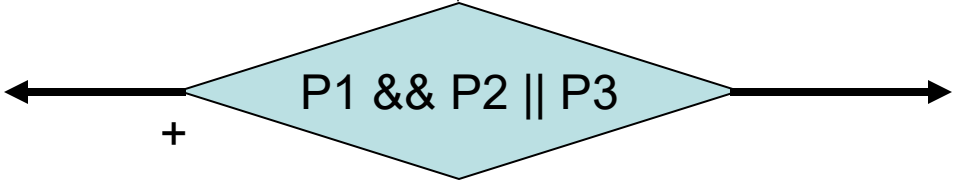
&&	a zároveň,	AND
||	nebo,	OR
!	negace,	NOT

P1	P2	! P1	P1 && P2	P1 || P2
A	A	N	A	A
A	N	N	N	A
N	A	A	N	A
N	N	A	N	N

PŘÍKLAD Log. výraz : (a<10) && (b<123) || (((a+b)==0) && (a<1))

při použití logických proměnných např.:

P1 = a<10
P2 = b<123
P3 = ((a+b)==0) && (a<1)



DP 1.6 – varianta

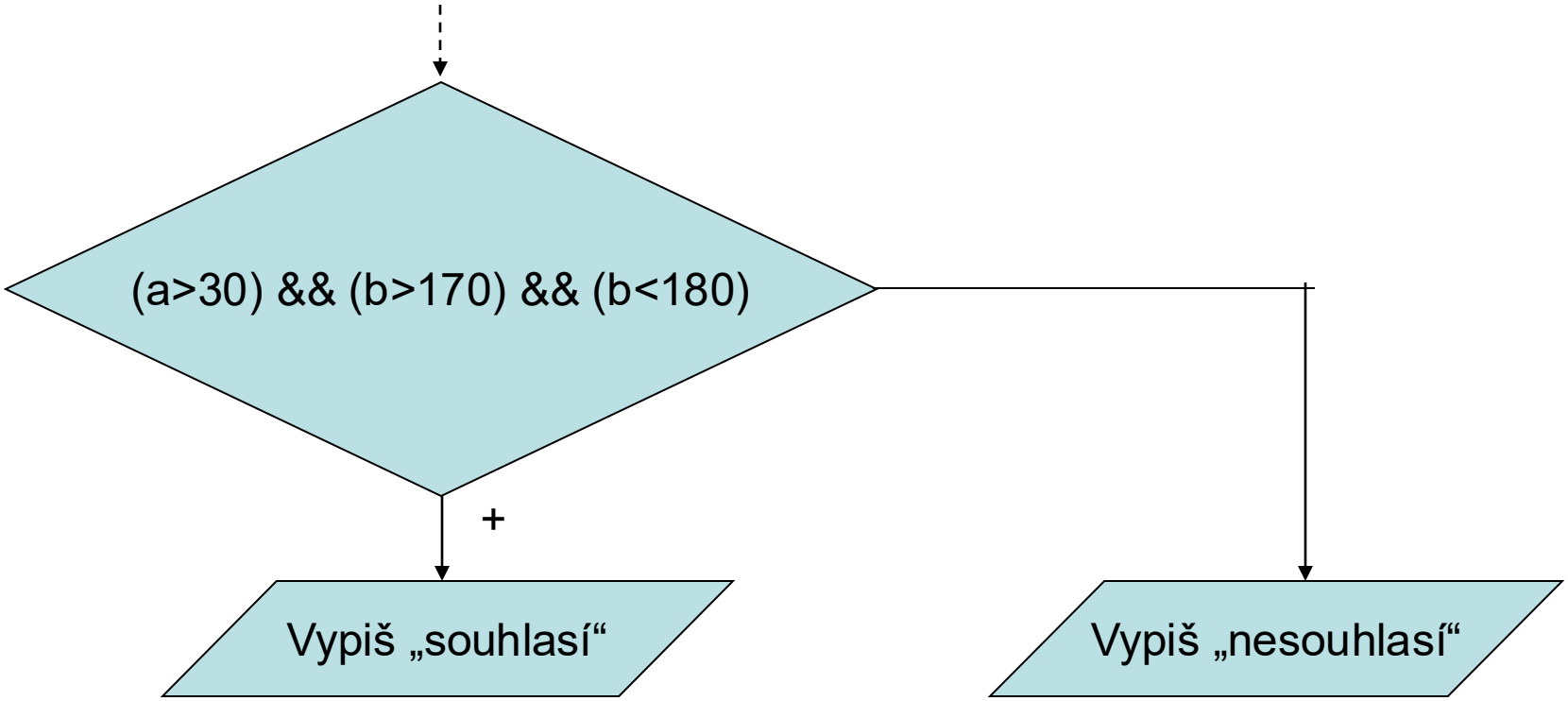
Skládání log. výrazů pomocí log. spojek

ZAPÍŠEME JINAK POMOCÍ LOG. PROMĚNNÉ

Př.

Známe věk pacienta a [roky] a výšku pacienta b [cm].

Chceme algoritmus, který určí, zda je pacient starší než 30 let a zároveň velký mezi 170 a 180 cm.



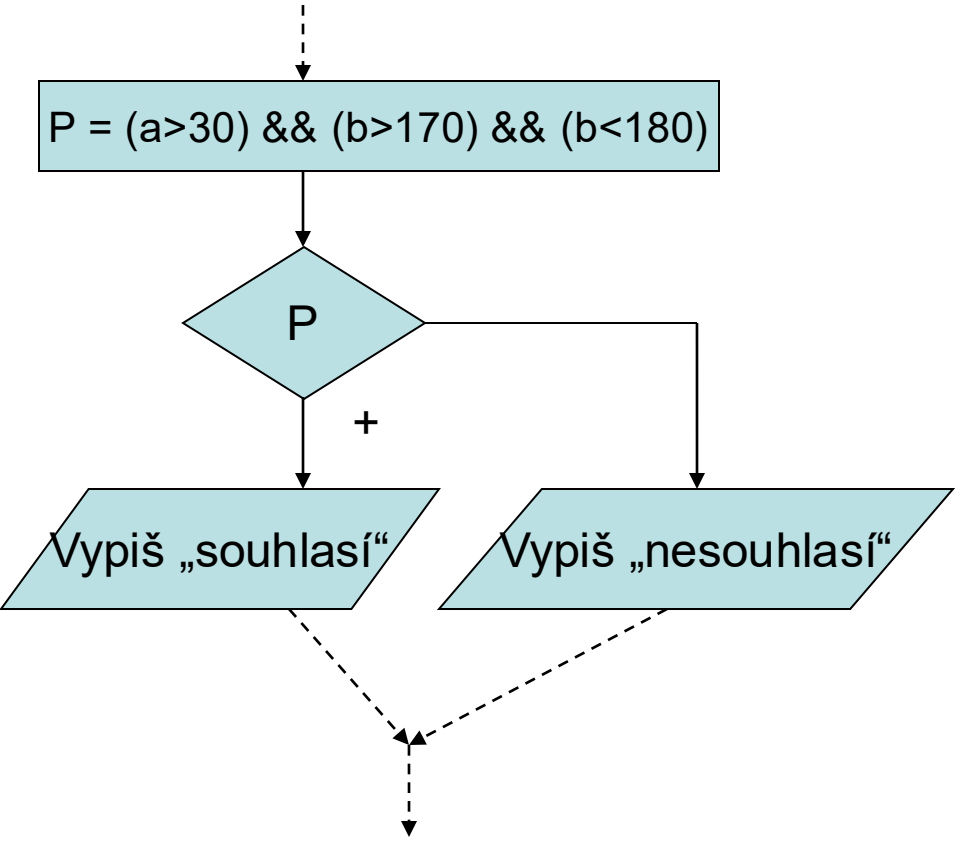
DP 1.6 – varianta dokonč.

Skládání log. výrazů pomocí log. spojek

ZAPSÁNO JINAK POMOCÍ LOG. PROMĚNNÉ

Př.

Známe věk pacienta a [roky] a výšku pacienta b [cm].
Chceme algoritmus, který určí, zda je pacient starší než 30 let a zároveň velký mezi 170 a 180 cm.



DP 1.6 – varianta dokonč.

```
import java.util.Scanner;

class DP23_varianta2
{
	public static void main(String args[])
	{
		double a, b;  boolean P;

		Scanner sc = new Scanner(System.in);
		a = sc.nextFloat();
		b = sc.nextFloat();

		P = a>30 && b>170 && b<180 ;

		if ( P ) {
			System.out.printf("Zaradit.\n" );
		} else {
			System.out.printf("Nezarazovat.\n" );
		}

	}

}
```

logický typ – úpravy a operace

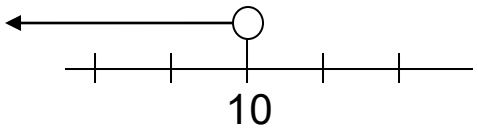
D.CV. – prohlédnout popř. nahledat SŠ zdroj….

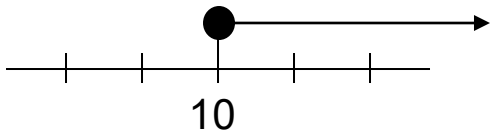
- základní zákony Booleovské algebry

komutativní	P or Q=Q or P	P and Q = Q and P
asociativní	(P or Q) or R = P or (Q or R)	(P and Q) and R = P and (Q and R)
distributivní	(P and Q) or R = (P or R) and (Q or R)	(P or Q) and R = (P and R) or (Q and R)
neutrálnost	P or false = P	P and true = P
agresivita	P or true = true	P and false = false
idempotence	P or P = P	P and P = P
zákon o vyloučeném třetím	P or not P = true	P and not P = false
de Morganovy zákony	not (P or Q) = not P and not Q	not (P and Q) = not P or not Q
negace negace	not not P = P	

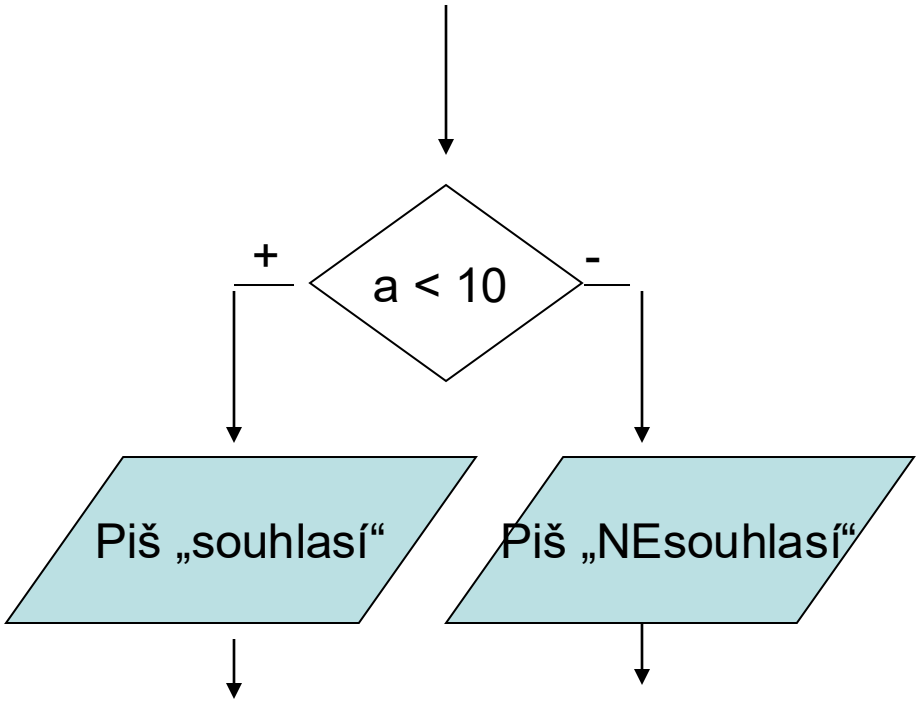
Pro nás důležitá odvozená vlastnost např. …. ekvivalence podmínek ! ! !
vlastně algebraický doplněk původní podmínky

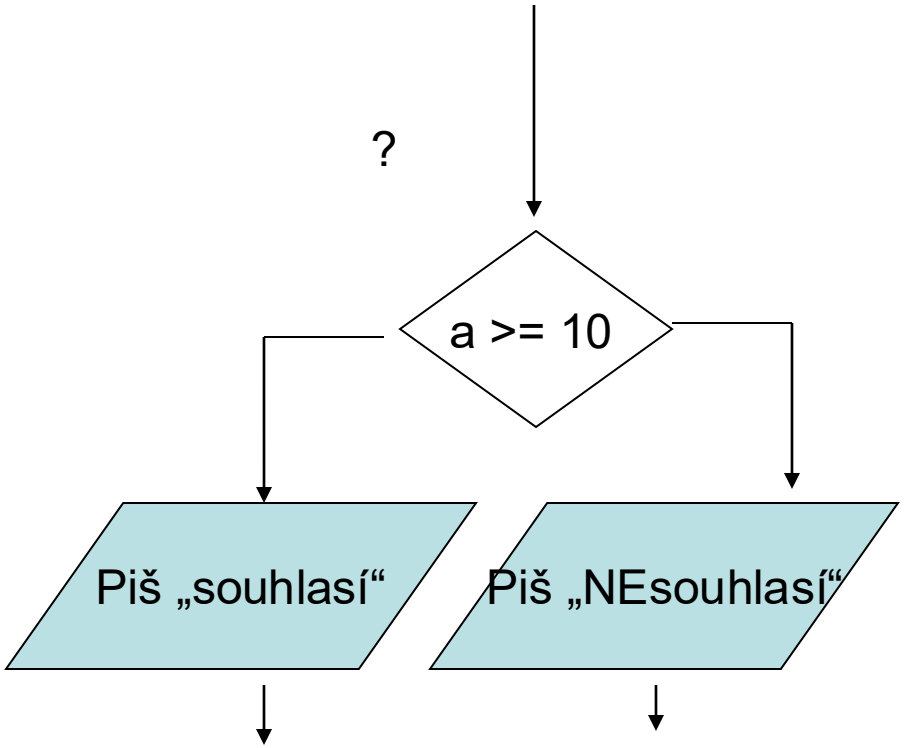
Př. Je a menší než deset ?





Ekvivalentní podmínky, doplněk a < 10 ←→ ! (a >= 10)





Vícenásobný přepínač v Javě – switch-case

```
switch ( výraz ) {
   case hodnota1: SEKVENCE1; break;
   case hodnota2: SEKVENCE2; break;
   case hodnota3: SEKVENCE3; break;

      . . . atd . . .

   default: SEKVENCE;
}
```

Vícenásobný přepínač v Javě – switch-case


```
import java.util.Scanner;

class DPswitch
{
        public static void main(String args[])
        {
                System.out.print("Vlozte pismenko znamky: ");
                Scanner sc = new Scanner(System.in);
                char c = sc.next().charAt(0);
                System.out.println("vlozeno "+c);

                switch (c) {

                        case 'A': System.out.println("vyborne"); break;
                        case 'B': System.out.println("velmi dobre"); break;
                        case 'C': System.out.println("dobre"); break;
                        case 'D': System.out.println("uspokojive"); break;
                        case 'E': System.out.println("dostatecne"); break;
                        case 'F': System.out.println("neprospel"); break;
                        default: System.out.println("neznama znamka");

                }

        }

}
```

Pozn.: samostatně si prostudujte klíčové slovo **break**

Cykly

(while, do-while a for) v Javě

Cykly

cyklus = opakovaně prováděná část algoritmu

tělo cyklu = opakující se kroky uvnitř cyklu

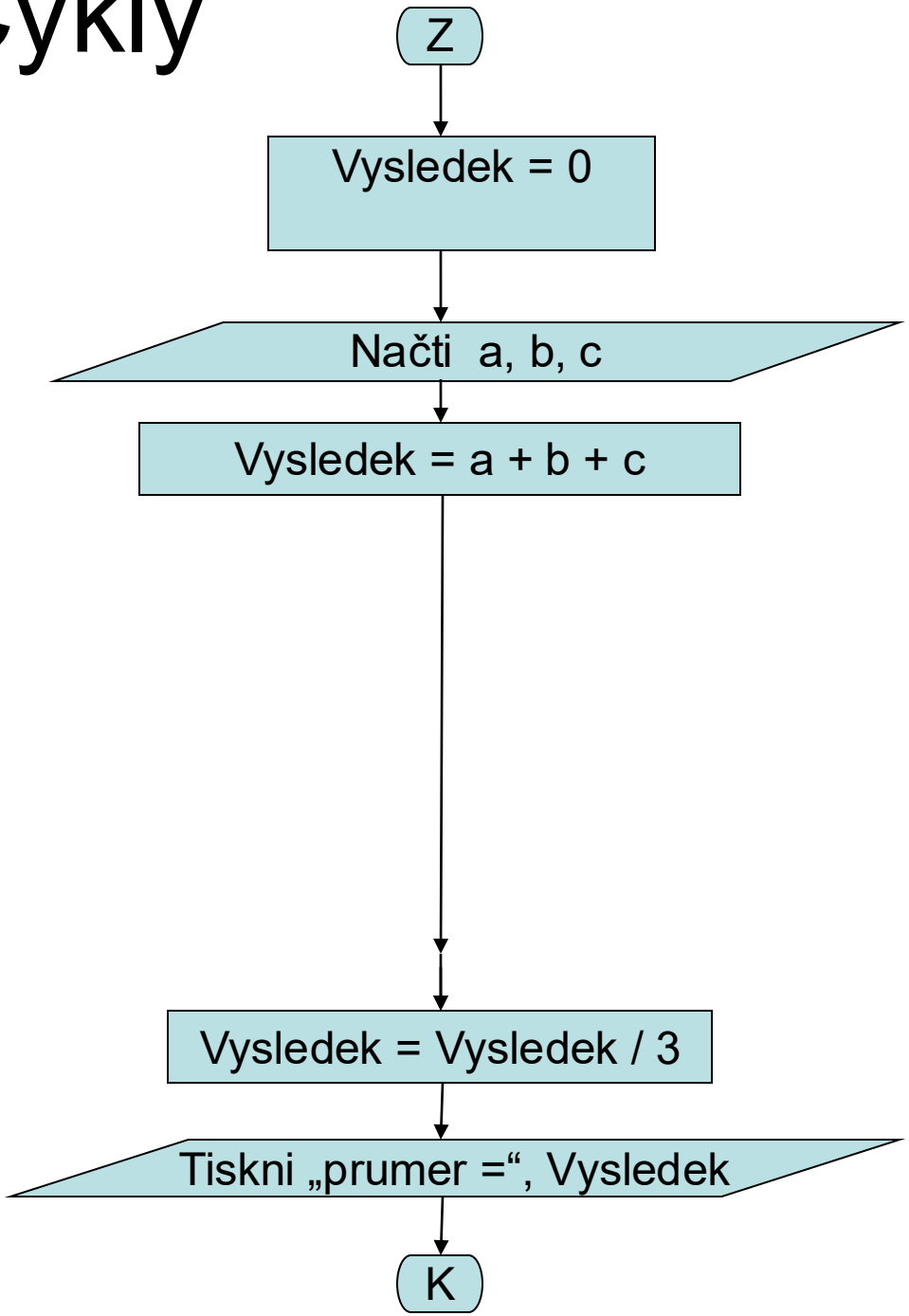
DP 1.8 (A)

Cykly

---> k čemu nám jsou dobré cykly ?

Motivační př.:

algoritmus pro výpočet aritm. průměru 3 celých čísel.



DP 1.8 (B)

Cykly

---> k čemu nám jsou dobré cykly ?

Motivační př.:

algoritmus pro výpočet
aritm. průměru N celých čísel.
(N>=1)

$$\frac{\sum_{P=1}^{N} \alpha_P}{N}$$

DP 1.8 (A)

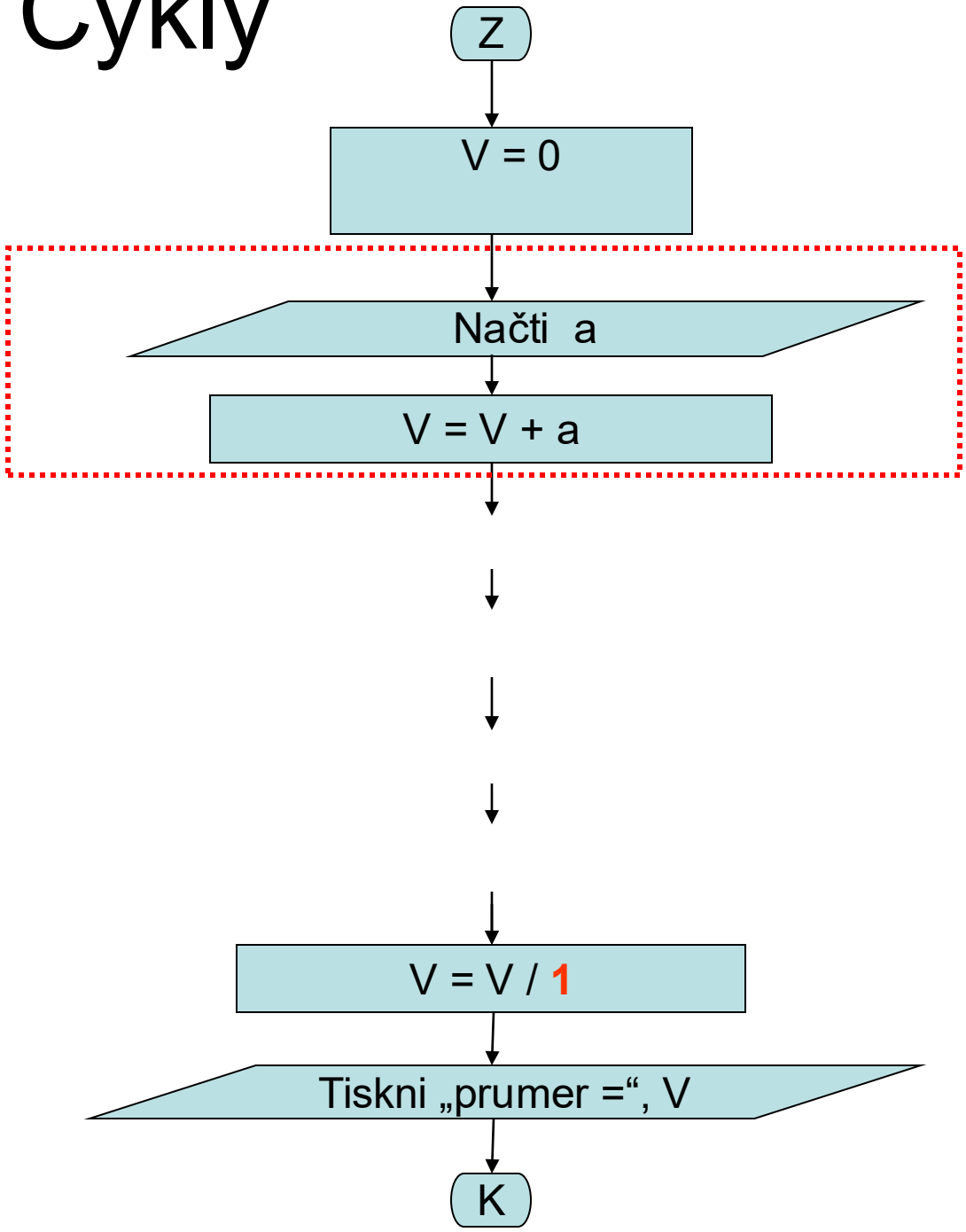
Cykly

---> k čemu nám jsou dobré cykly ?

Motivační př.:

algoritmus pro výpočet aritm. průměru 3 celých čísel.

Zatím by bylo 1



DP 1.8 (A)

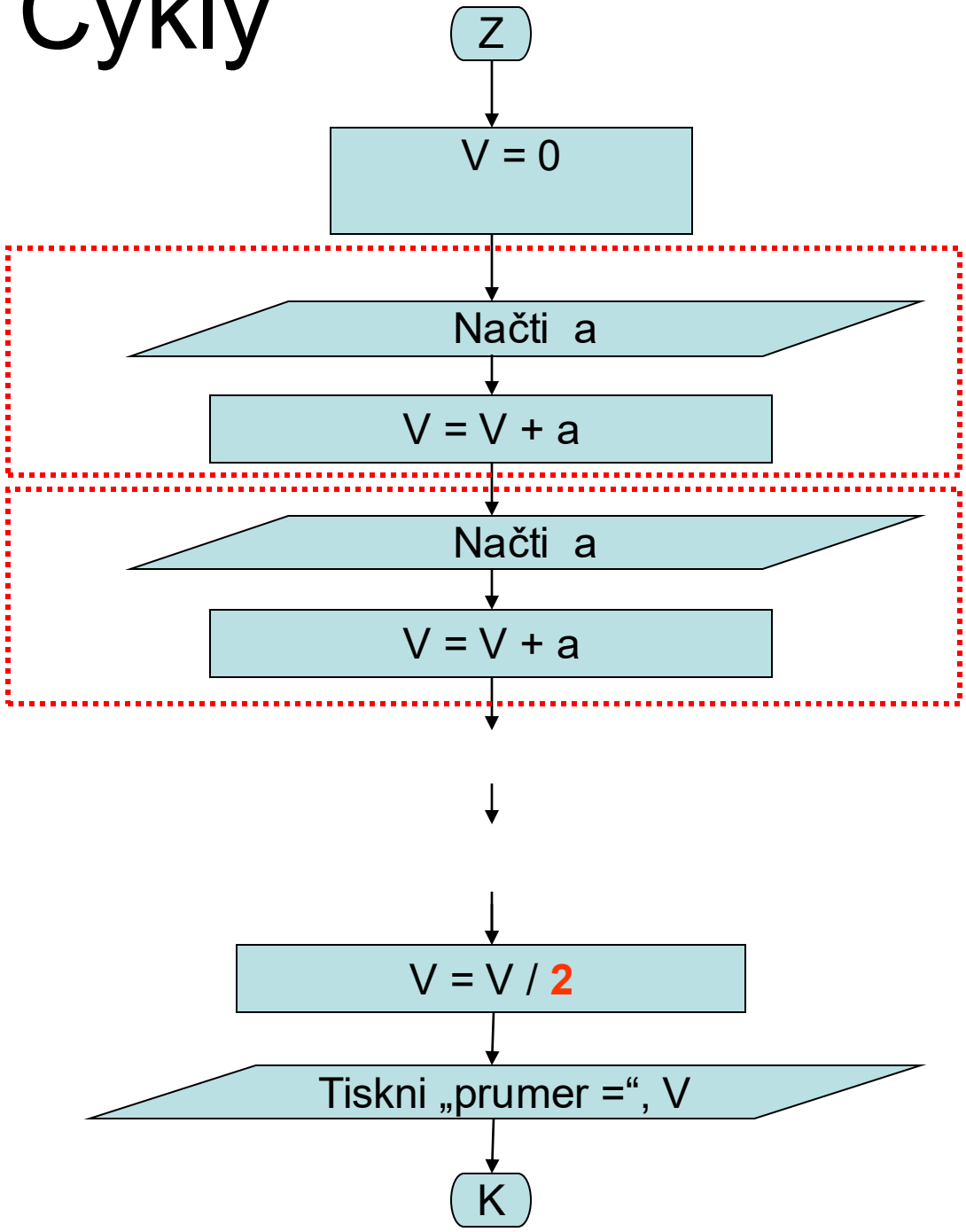
Cykly

---> k čemu nám jsou dobré cykly ?

Motivační př.:

algoritmus pro výpočet aritm. průměru 3 celých čísel.

Zatím by byly 2 Jednalo by se o správný průměr ze dvou čísel.



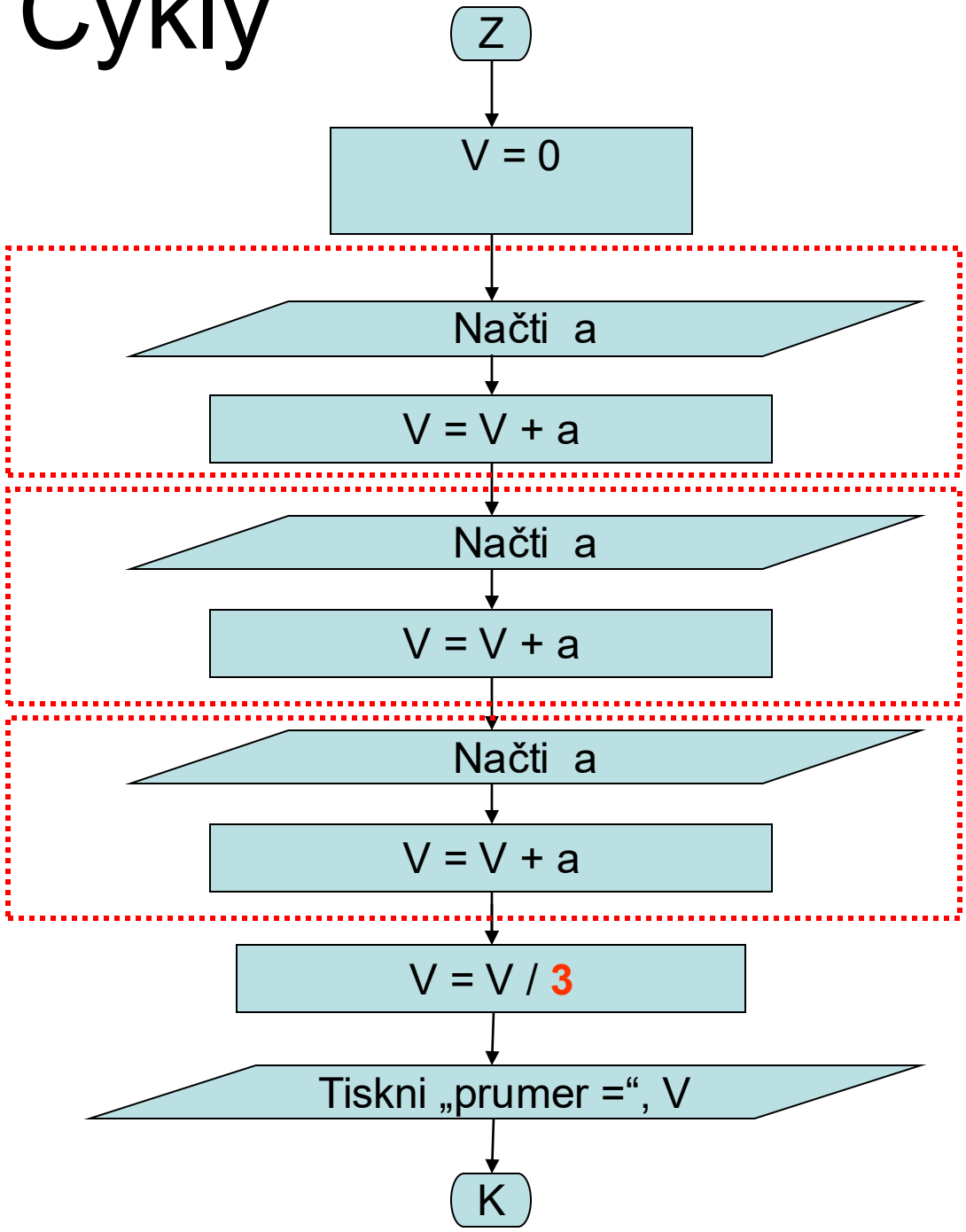
DP 1.8 (A)

Cykly

---> k čemu nám jsou dobré cykly ?

Motivační př.:

algoritmus pro výpočet aritm. průměru 3 celých čísel.



DP 1.8 (B)

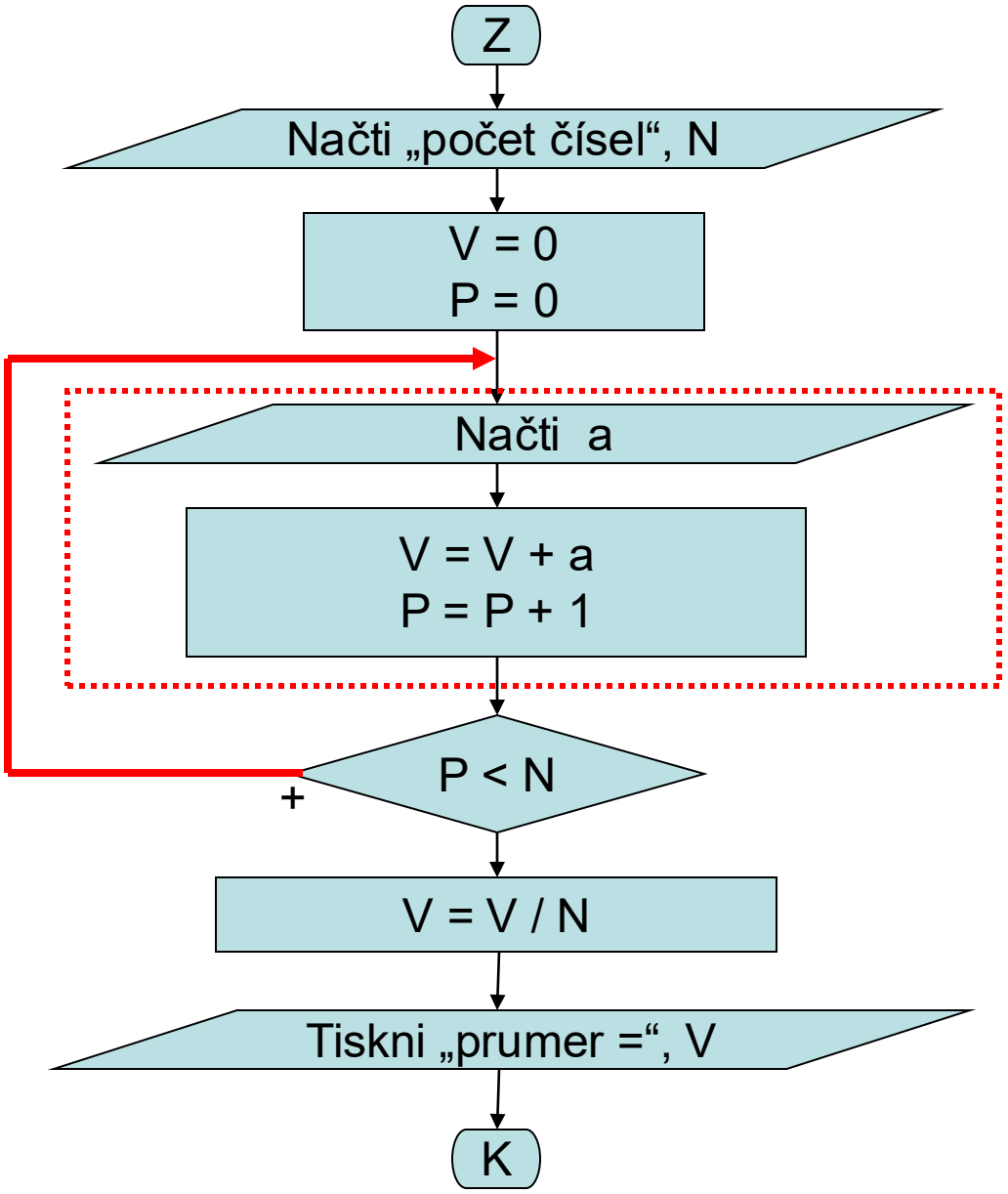
Cykly

---> k čemu nám jsou dobré cykly ?

Motivační př.:

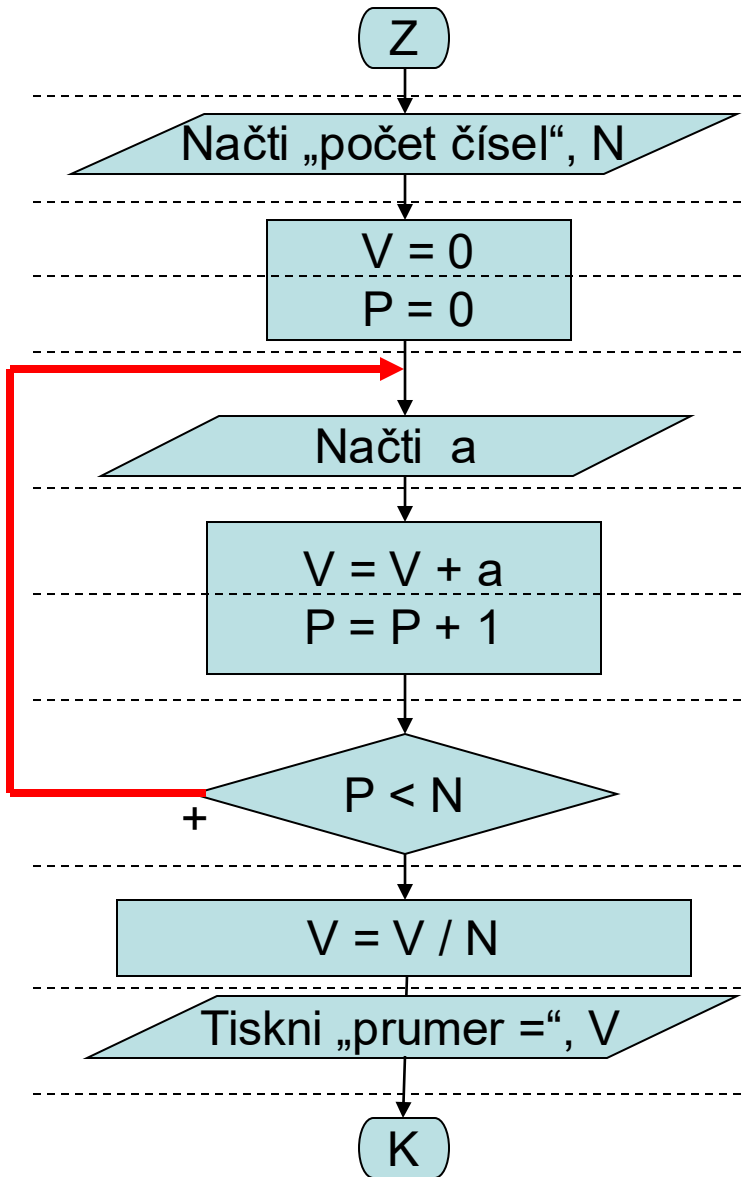
algoritmus pro výpočet
aritm. průměru N celých čísel.
(N>=1)

$$\frac{\sum_{P=1}^{N} \alpha_P}{N}$$



DP 1.8 (B)
pokrač.

pro vstup: 6, 1, 2



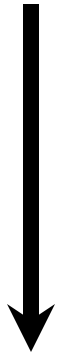
TRASOVACÍ TABULKA

1. průchod				2. průchod				3. průchod			
N	V	P	a	N	V	P	a	N	V	P	a
?	?	?	?								
3	?	?	?								
3	0	?	?								
3	0	0	?								
3	0	0	6	3	6	1	1	3	7	2	2
3	6	0	6	3	7	1	1	3	9	2	2
3	6	1	6	3	7	2	1	3	9	3	2
3	6	1	6	3	7	2	1	3	9	3	2
								3	3	3	2
								3	3	3	2

Cykly

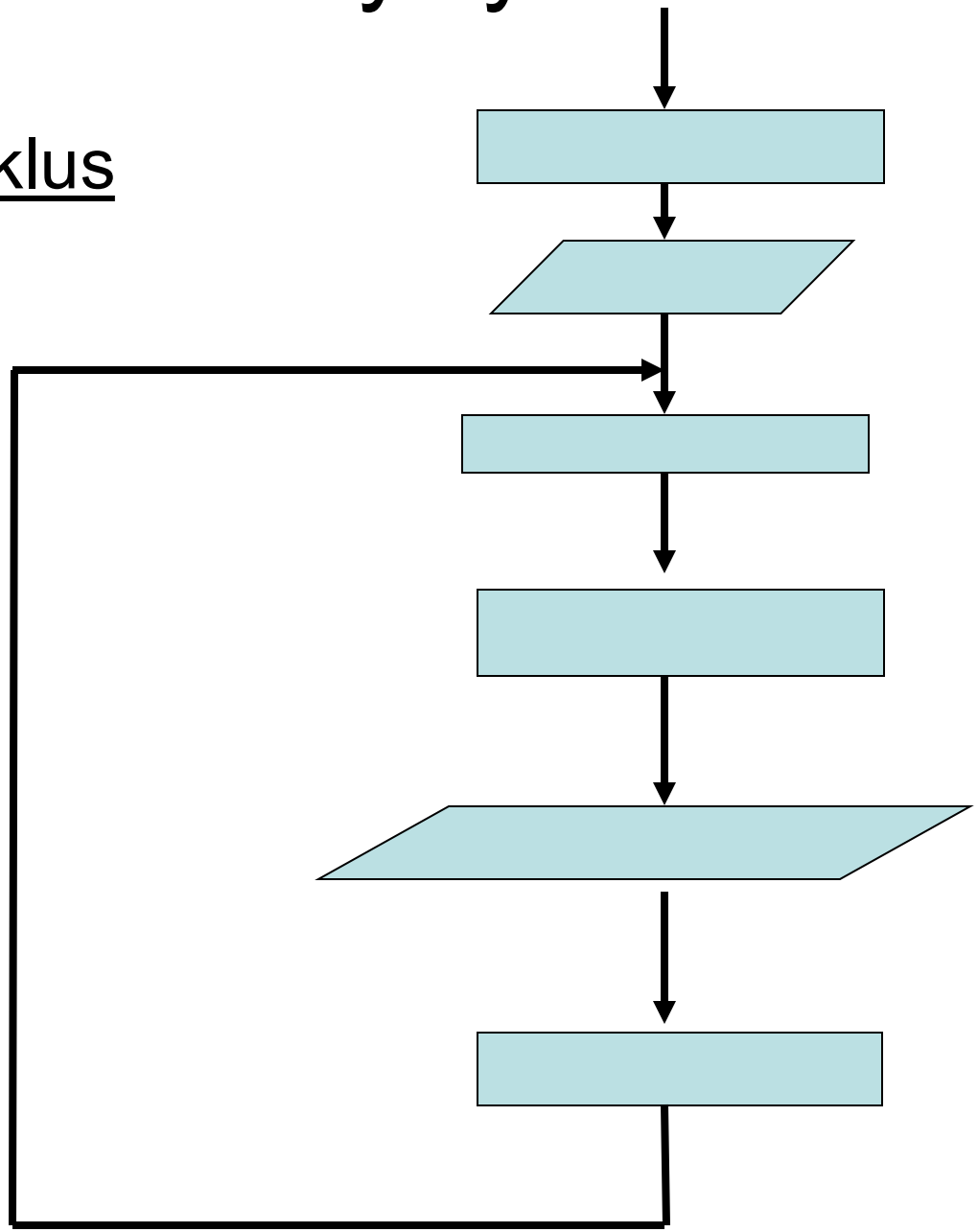
Spec. případ ☺

Nekonečný cyklus



V praxi
stěžejní operace:

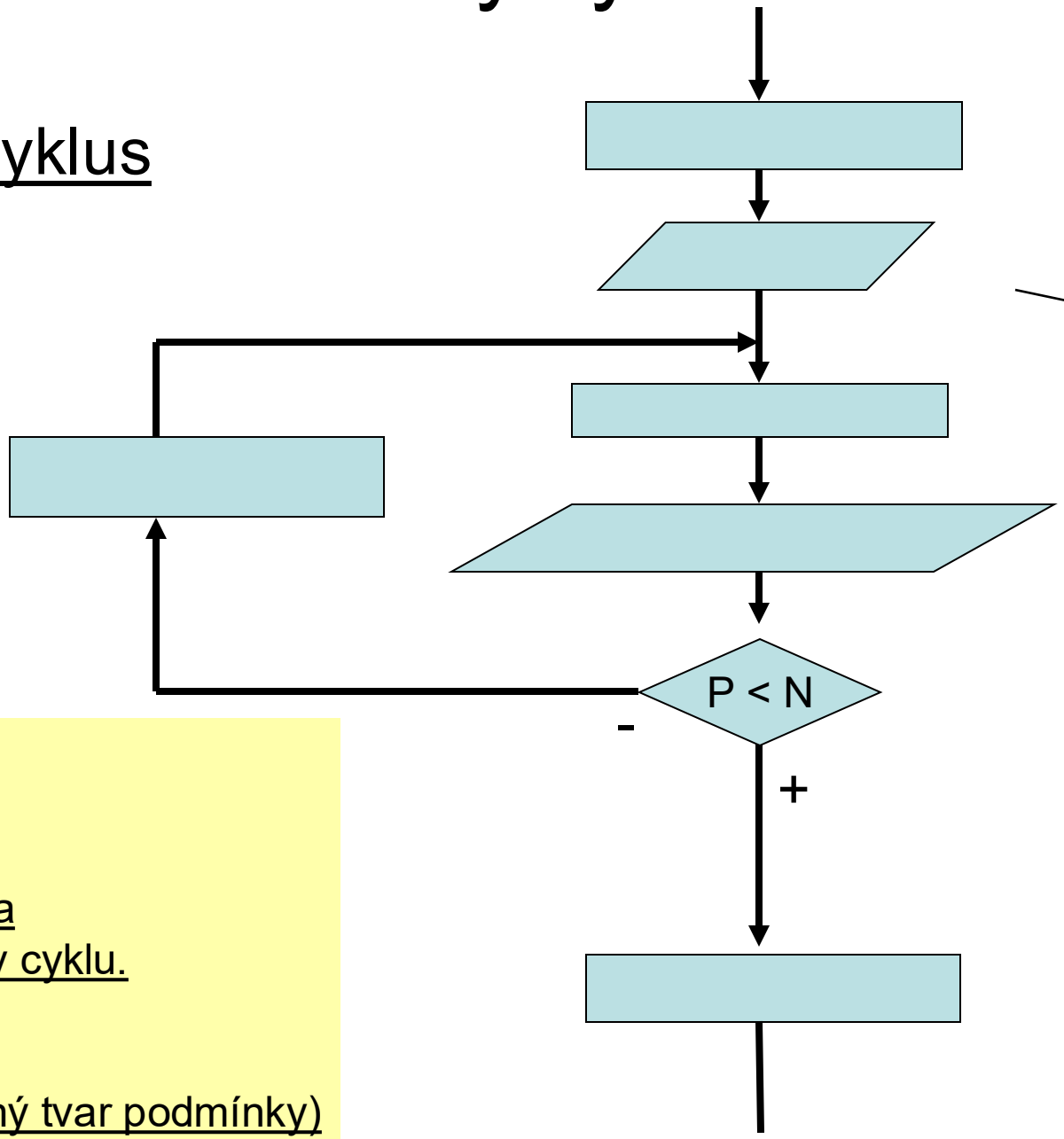
řídící podmínka cyklu.



Cykly

Spec. případ ☺

„divoký“ cyklus



My chceme
Programovat
„strukturovaně“

V praxi
též důležitá:

Správná poloha
řídící podmínky cyklu.

(a dále i správný tvar podmínky)

Cykly

Proto třeba dodržet jisté konvence a algoritmus naformulovat do některé z „typizovaných“ podob:

typy cyklů:

- s řídicí podmínkou před tělem
- s řídicí podmínkou za tělem

} indukční

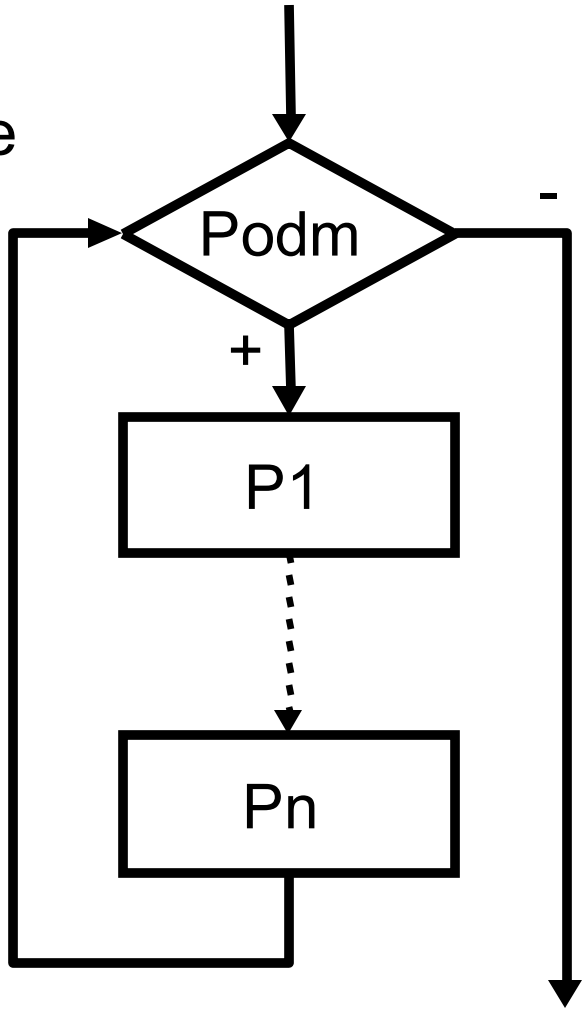
- s pevným počtem opakování

} iterační

Cykly

Cyklus s podmínkou před tělem

KONVENCE:
k ukončení dojde v případě, že řídicí podmínka není splněna



„while cyklus“

```
while( PODM ){

      P1 ;

      . . .

      Pn ;

}
```

Důsledek:
tělo cyklu se nemusí vykonat ani jednou

Cykly

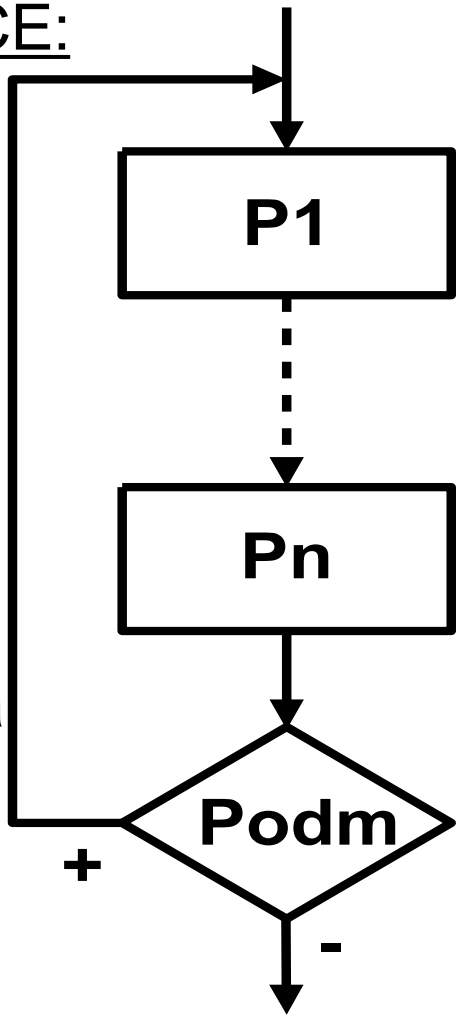
Cyklus s podmínkou za tělem

STEJNÁ KONVENCE:

k ukončení dojde v případě, že řídicí podmínka není splněna

Důsledek:

tělo cyklu se vykoná alespoň jednou



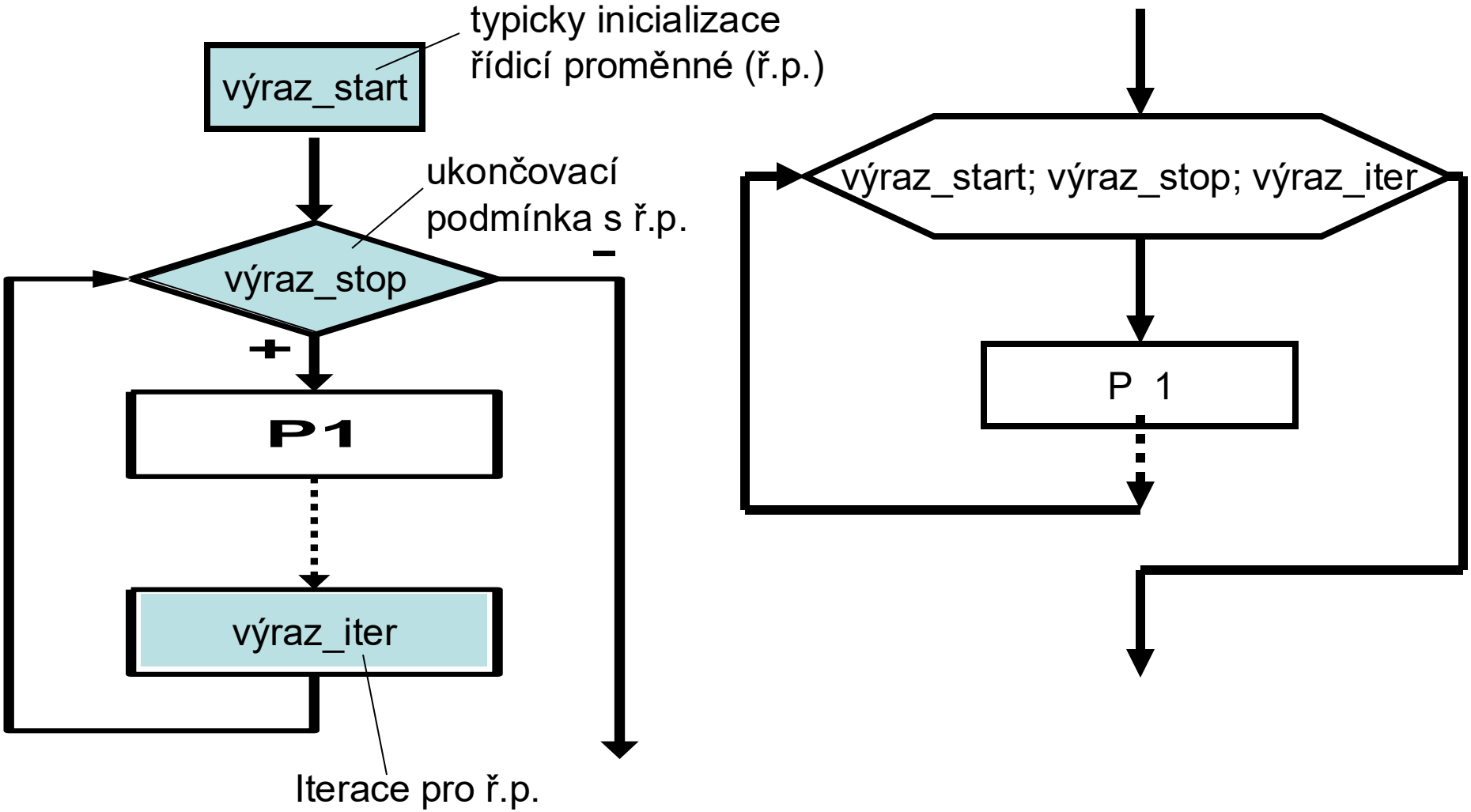
„do-while cyklus“

```
do {

   P1 ;

   . . .

   Pn ;



} while( PODM );
```

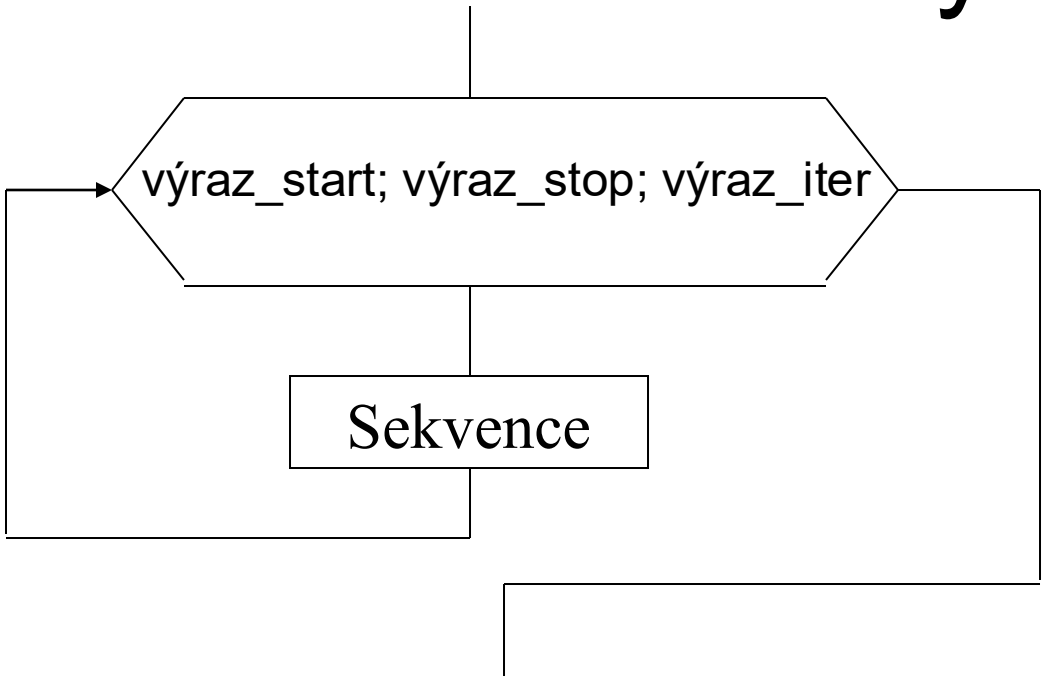
Cykly

Cyklus s pevným počtem opakování

…. FOR cyklus



FOR cyklus




```
for( výraz_start; výraz_stop; výraz_iter ){

      Sekvence;

}
```

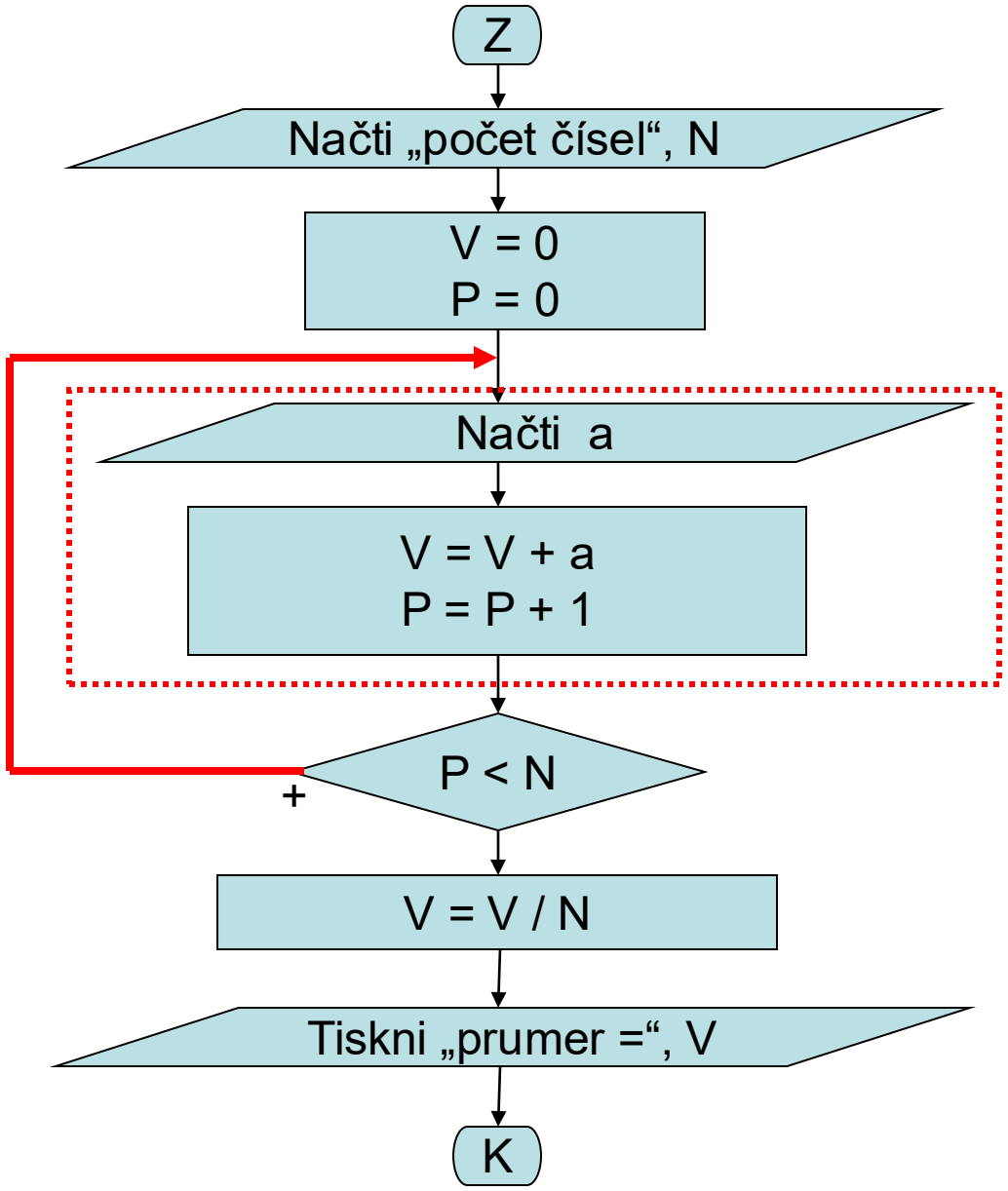
DP 1.8 (B)

Konečně můžeme dokončit 1.8 . . .

Motivační př.:

algoritmus pro výpočet
aritm. průměru N celých čísel.
(N>=1)

$$\frac{\sum_{P=1}^{N} \alpha_P}{N}$$



DP 1.8 (B) varianta s do-while cyklem

DP 1.8 (C) varianta s while cyklem

DP 1.8 (D) varianta s for cyklem

DP 1.8 (B)

Konečně můžeme dokončit 1.8 B . . .

```
import java.util.Scanner;

class DP24B
{
        public static void main(String args[])
        {
                int N, P; double V, a;

                V=0; P=0; // není nutné, jsou automaticky nulovány

                Scanner sc = new Scanner(System.in);

                System.out.print("Vlozte pocet cisel: ");
                N = sc.nextInt();

                do {    System.out.print("Vlozte cislo " + (P+1) + ": ");
                        a = sc.nextFloat();
                        V=V+a; P=P+1; // zkuste si += a ++

                } while ( P<N ) ;

                V = V / N;
                System.out.printf("Prumer je %f\n", V);

        }
}
```

DP 1.8 (C)

```
import java.util.Scanner;

class DP24B
{
	public static void main(String args[])
	{
		int N, P; double V, a;

		V=0; P=0; // není nutné, jsou automaticky nulovány

		Scanner sc = new Scanner(System.in);

		System.out.print("Vlozte pocet cisel: ");
		N = sc.nextInt();

		while ( P<N ) {
			System.out.print("Vlozte cislo " + (P+1) + ": ");
			a = sc.nextFloat();
			V=V+a; P=P+1;
		}

		V = V / N;
		System.out.printf("Prumer je %f\n", V);

	}

}
```

DP 1.8 (D)

```
import java.util.Scanner;

class DP24B
{
		public static void main(String args[])
		{
				int N, P; double V, a;

				V=0; // není nutné, jsou automaticky nulovány

				Scanner sc = new Scanner(System.in);

				System.out.print("Vlozte pocet cisel: ");
				N = sc.nextInt();

				for ( P=0; P<N; P=P+1 ) {
						System.out.print("Vlozte cislo " + (P+1) + ": ");
						a = sc.nextFloat();
						V=V+a;
				}

				V = V / N;
				System.out.printf("Prumer je %f\n", V);

		}

}
```

DP 1.9 A

Napište algoritmus, který vypíše všechna celá čísla v intervalu 0 až 3 a jejich druhé mocniny.

Požadovaný tvar výstupu:

0 --- 0

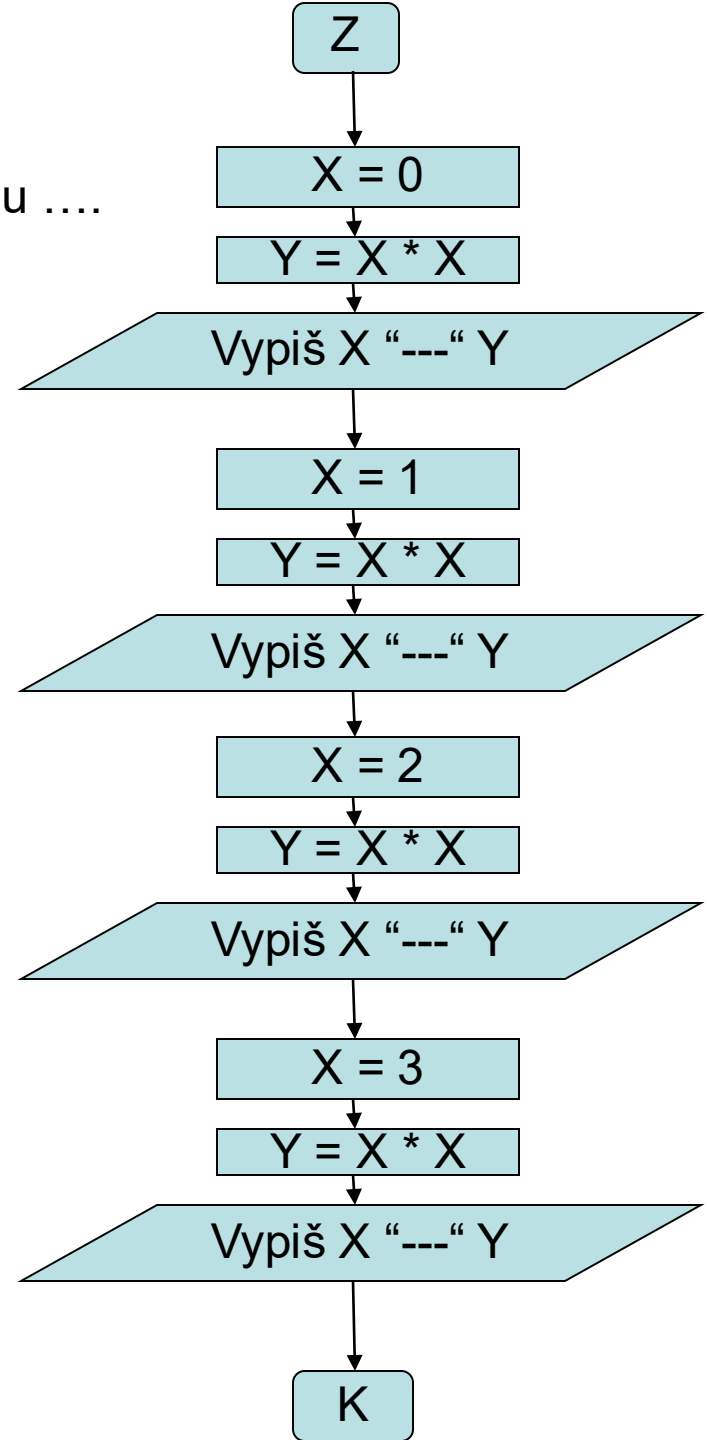
1 --- 1

2 --- 4

3 --- 9

DP 1.9 A
pokrač.

možno i bez použití cyklu ….



Napište algoritmus, který vypíše všechna celá čísla v intervalu 0 až 100 a jejich druhé mocniny.

Požadovaný tvar výstupu:

0 --- 0

1 --- 1

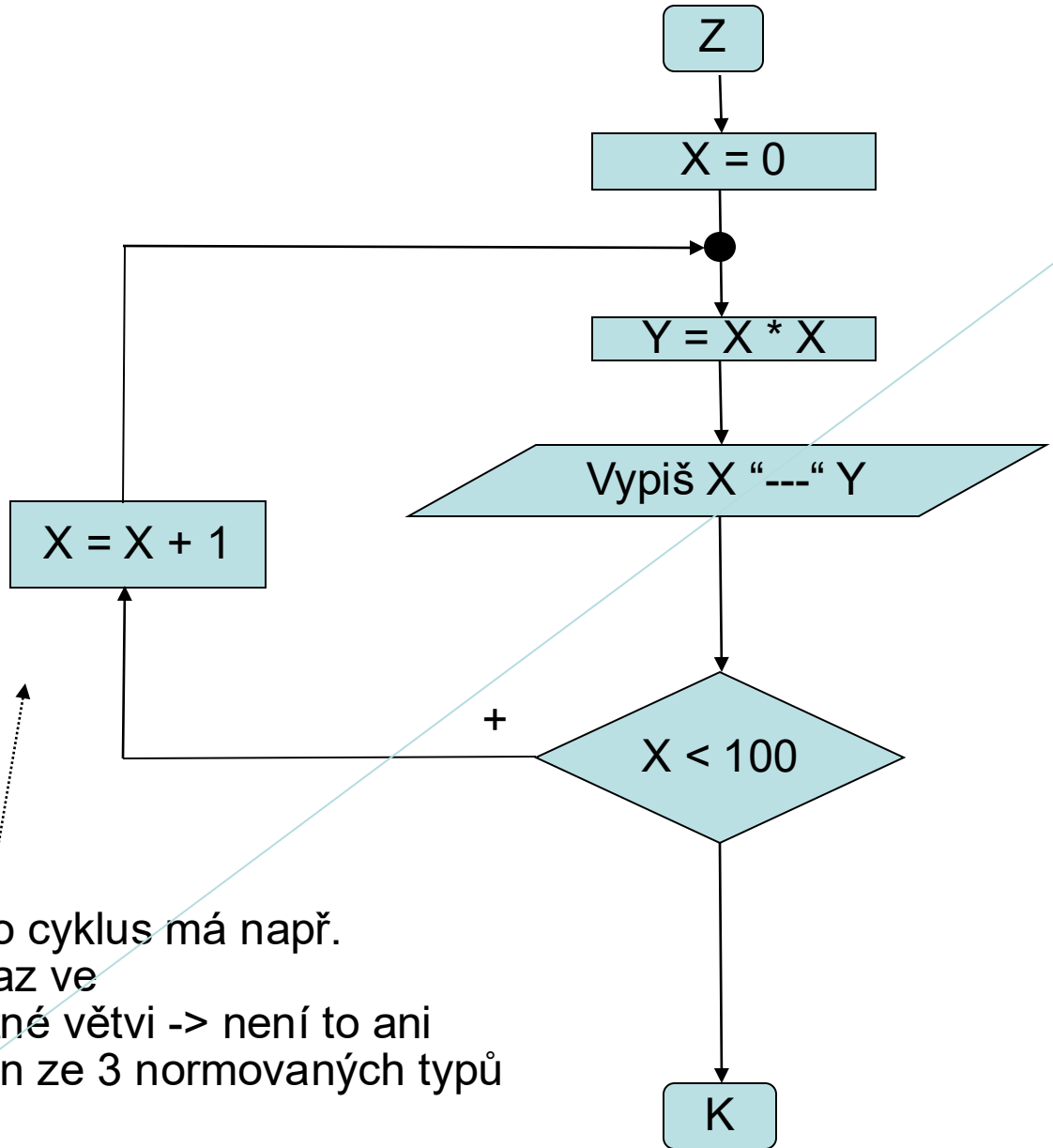
2 --- 4

atd.

100 --- 10000

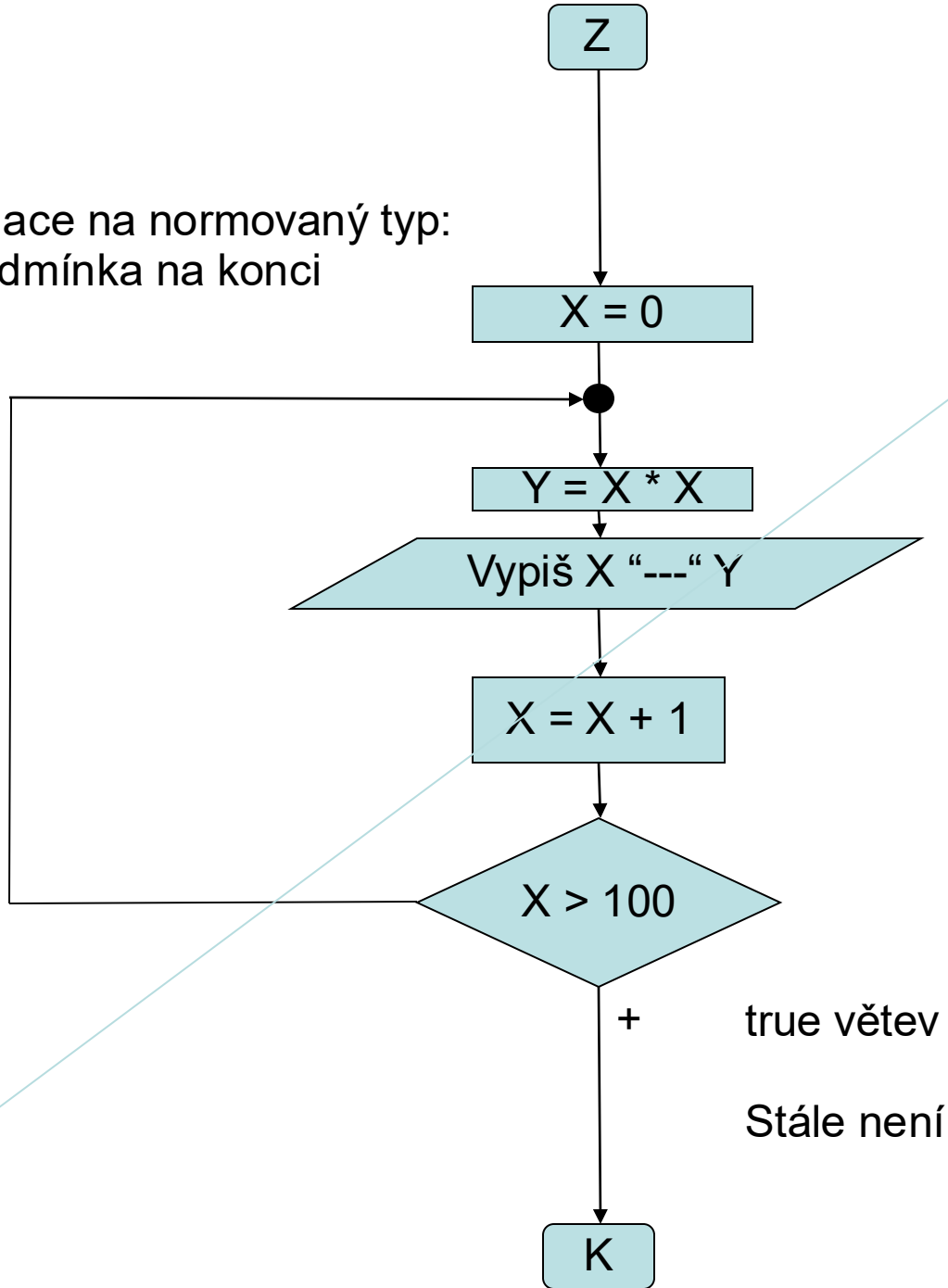
DP 1.9 B pokrač.

Pozor také na řešení, která neodpovídají uvedeným 3 základním konstrukcím
(většinou je lze na ně transformovat):



tento cyklus má např.
příkaz ve
zpětné větvi -> není to ani
jeden ze 3 normovaných typů

Transformace na normovaný typ:
krok 1: podmínka na konci

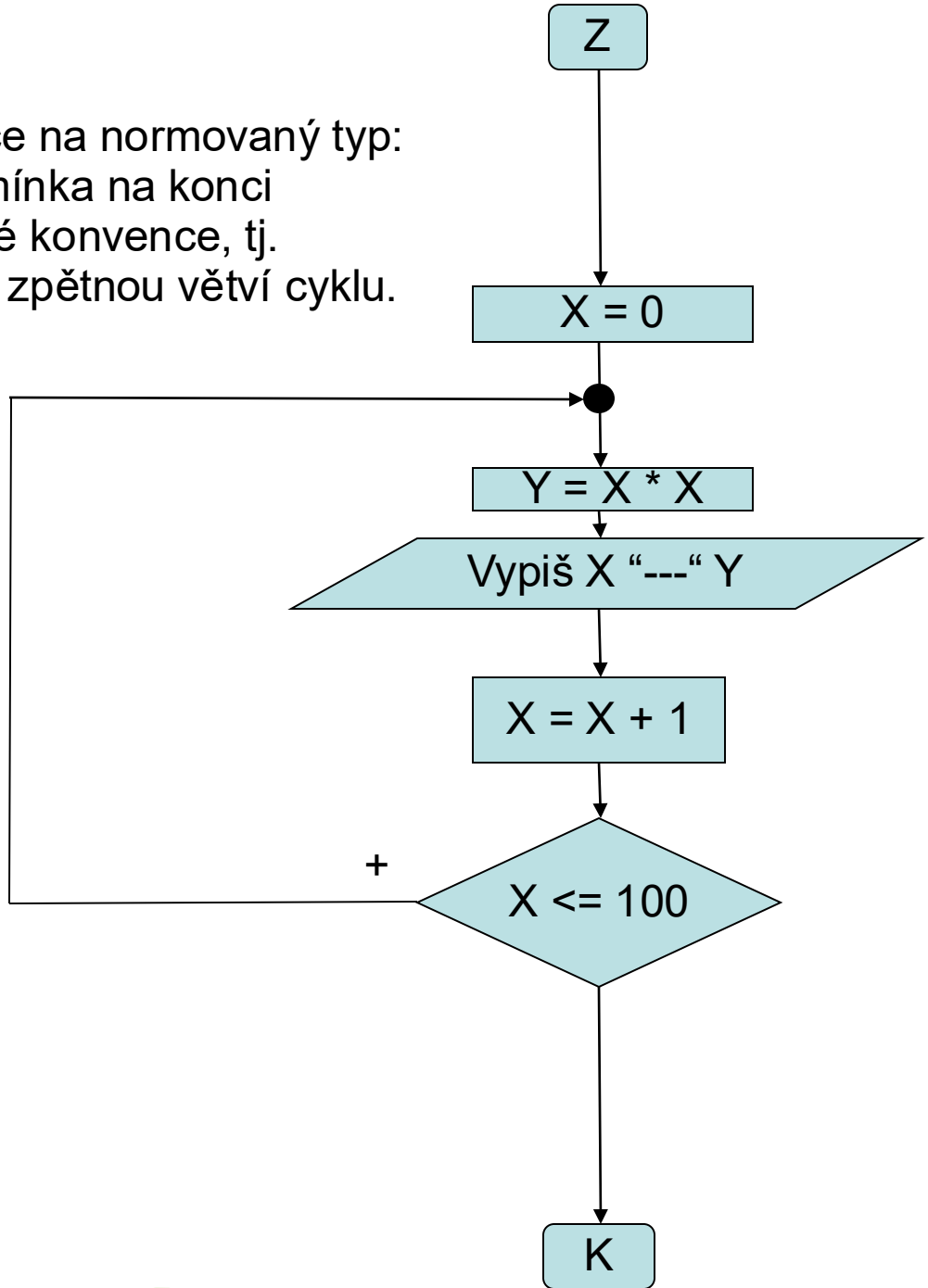


true větev nevede na opakování cyklu.

Stále není v Javě normovaný typ

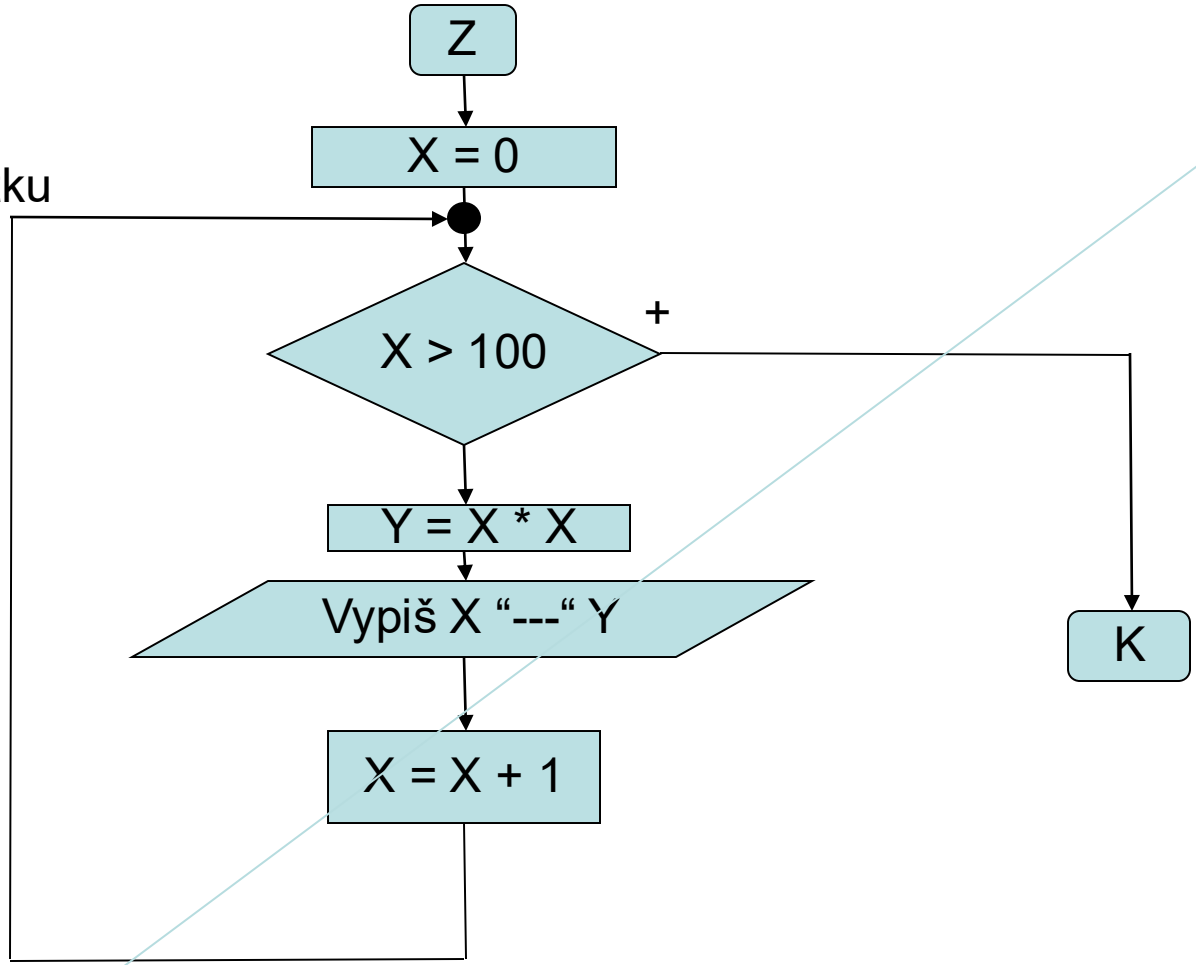
DP 1.9 B
pokrač.

Transformace na normovaný typ:
krok 2: podmínka na konci
dle zavedené konvence, tj.
true větev je zpětnou větví cyklu.



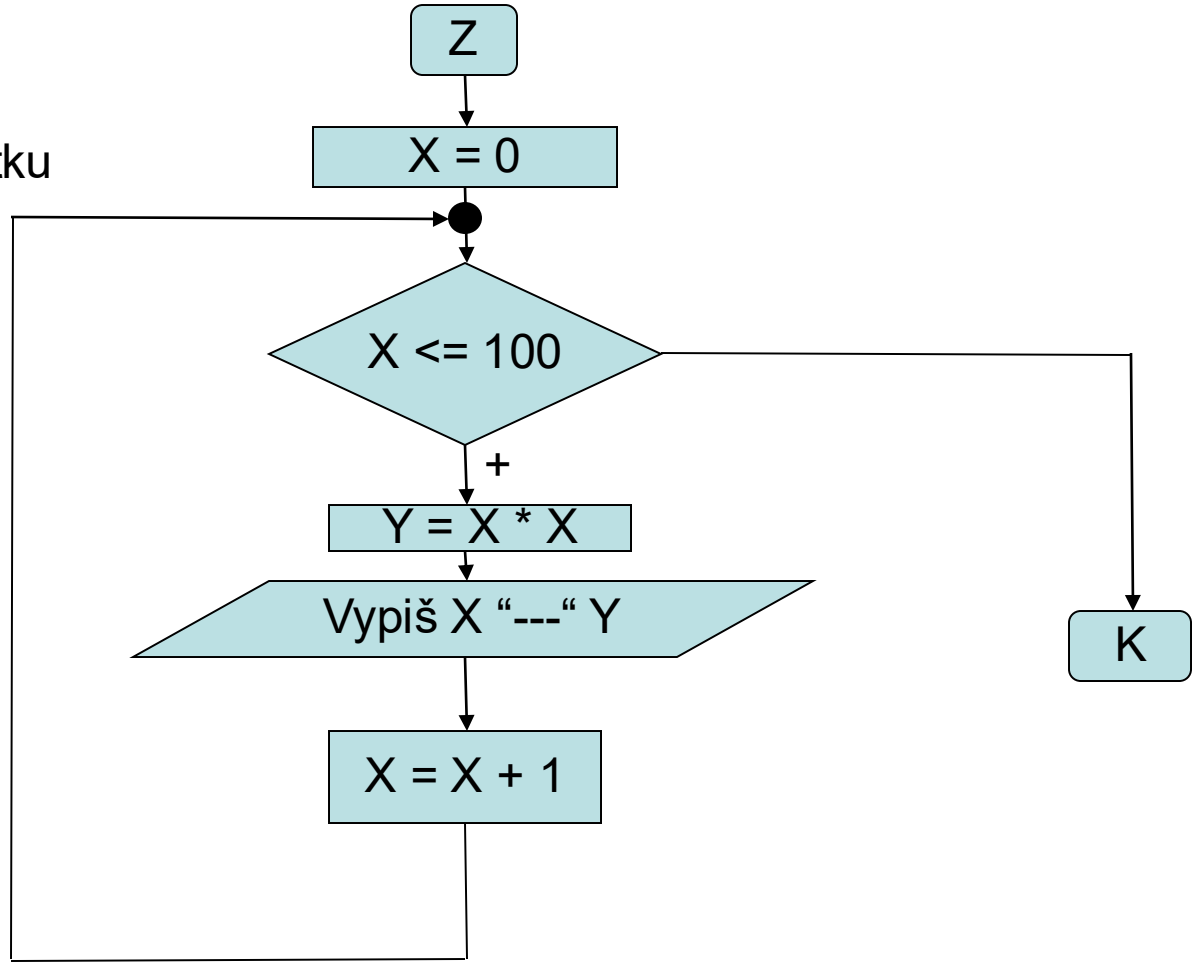
DP 1.9 C
pokrač.

Jiná verze -
- podmínka na začátku



DP 1.9 C
pokrač.

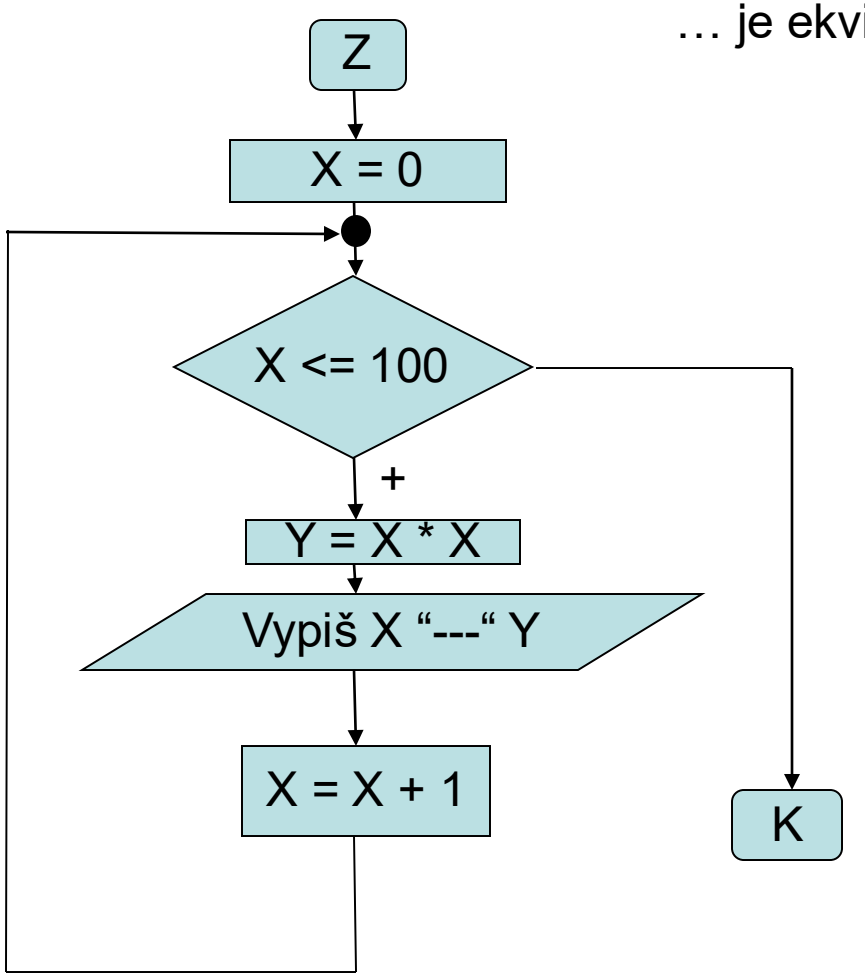
Podmínka na začátku
dle konvence ..

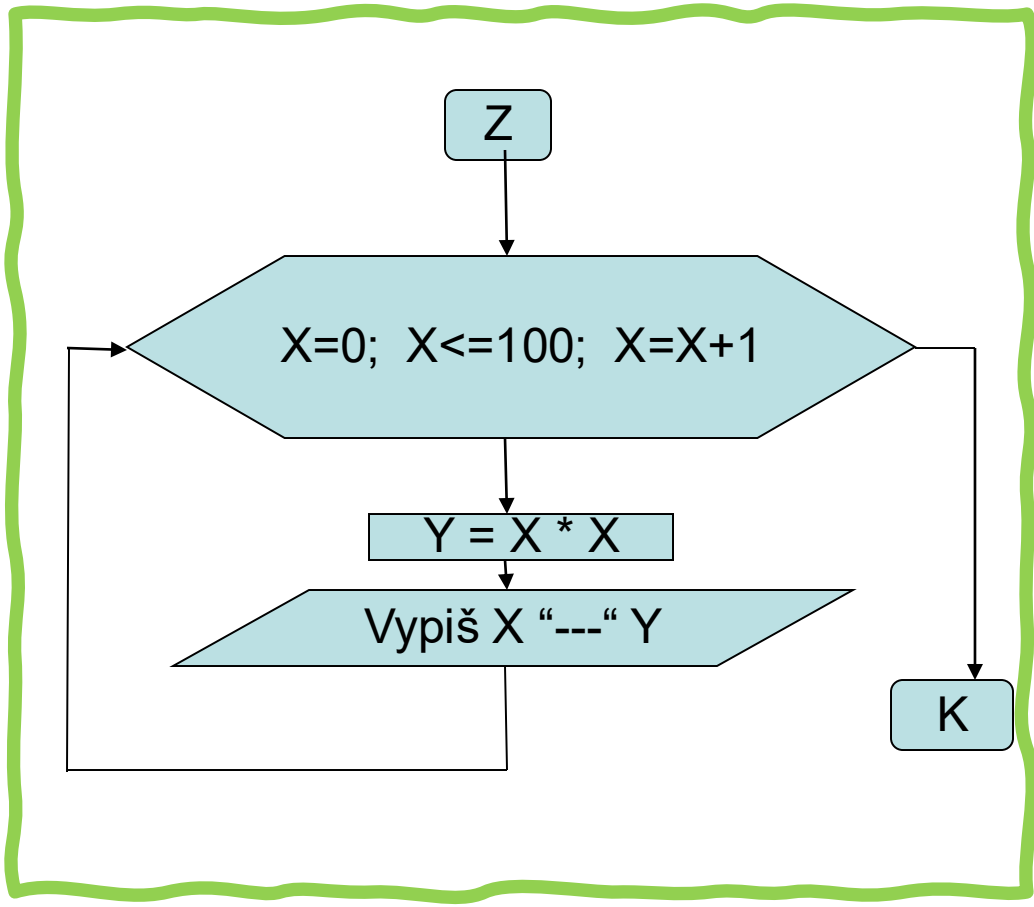


DP 1.9 D
pokrač.

Cyklus s pevným počtem opakování (tkzv. for cyklus)

… je ekvivalentní cyklu s podmínkou na začátku





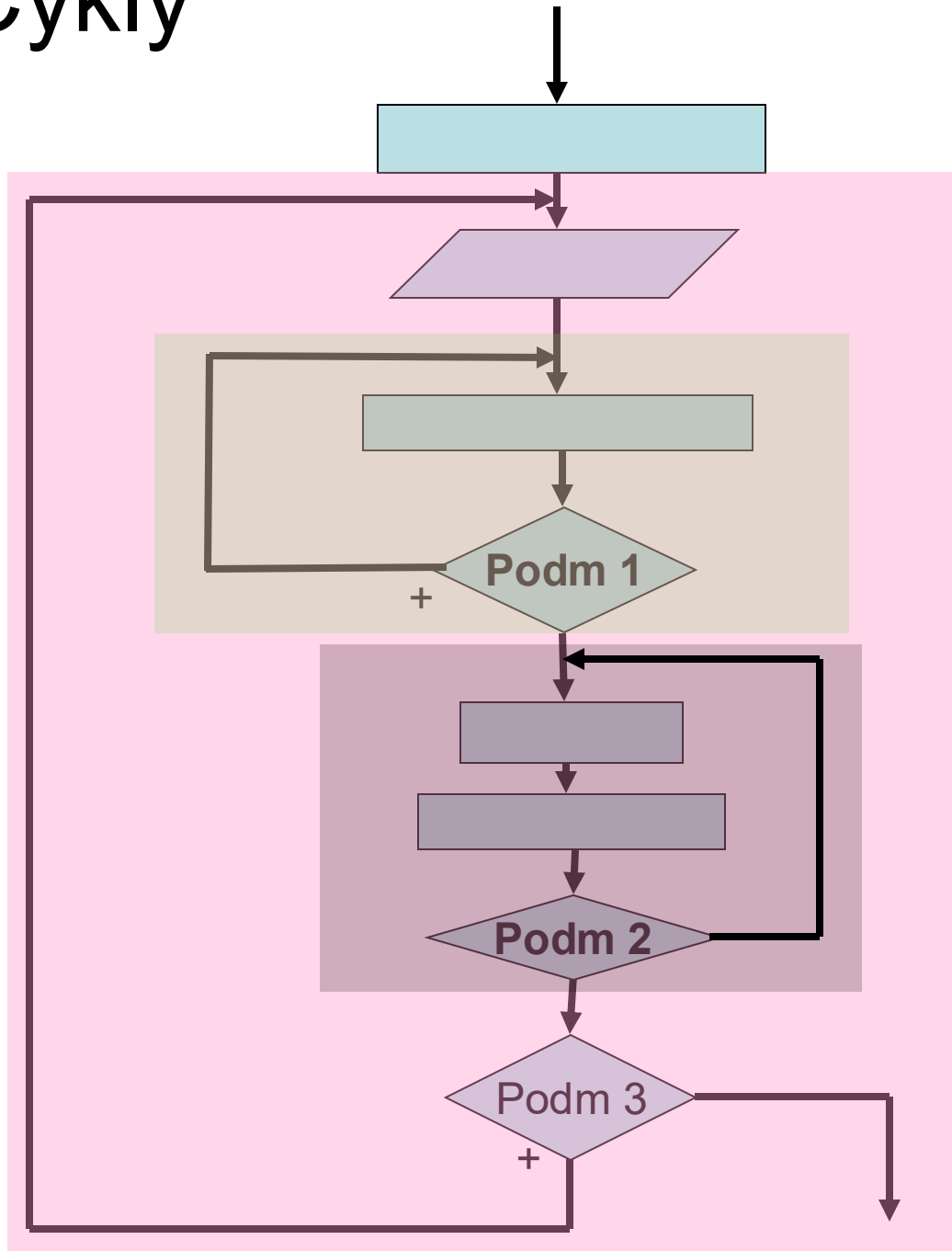
(konec demonstračního př. 2.5)

Cykly

Pozn:

a) vícenásobně vnořené cykly

b) samostatně si prostudujute klíčové slovo **continue**



Další demonstrační úlohy

DP 1.10

Na vstupu jsou zadána 4 celá čísla. Napište algoritmus, který najde a vypíše nejmenší z nich. Odladťe také program v Javě,

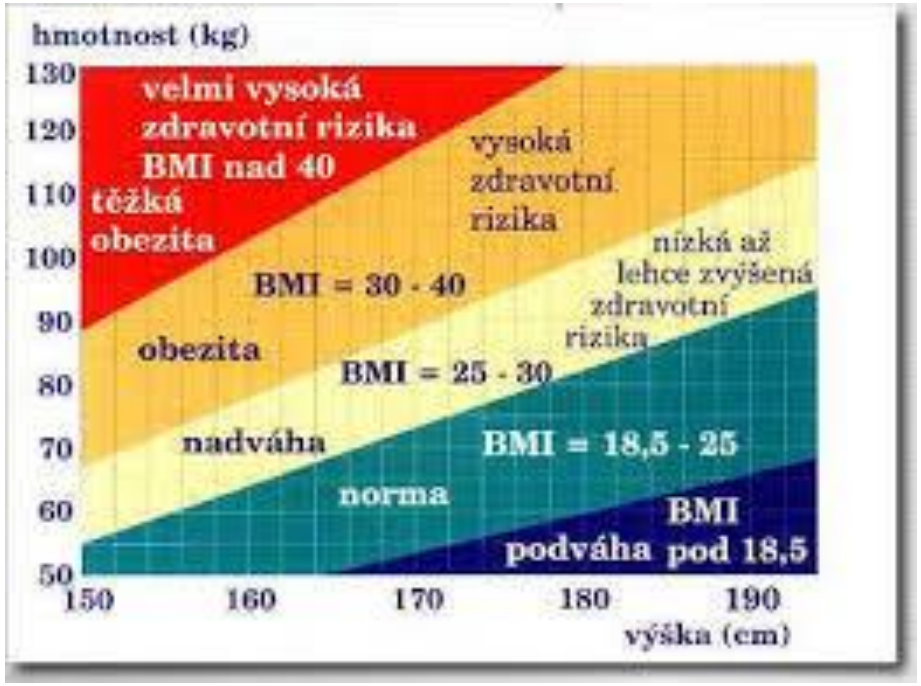
kterým algoritmus otestujete.

DP 1.11 A

Sestavte algoritmus a napište a odlaďte v Javě program pro výpočet Body Mass Indexu.

$$BMI = \frac{\text{hmotnost (kg)}}{\text{výška}^2\,\text{(m)}}$$

Výsledek vypište s přesností na 1 desetinné místo.



DP 1.11 B

Sestavte algoritmus a napište a odlaďte v Javě program pro výpočet Body Mass Indexu a klasifikaci osob podle něj.

$$BMI = \frac{\text{hmotnost (kg)}}{\text{výška}^2\,\text{(m)}}$$

Tabulka hodnot BMI	
Podváha	< 18,5
Normální hmotnost	18,5–25,0
Nadváha	25,1–30,0
Obezita	30,1–40,0
Morbidní obezita	> 40

Odhalte, co je v zadání špatně a upravte podle toho jeho specifikaci.

DP 1.12

Na vstup jsou postupně zadávána celá kladná čísla. Po zadání jiného čísla (např. -1) na vstup se načítání ukončí a vypíše se hodnota největšího z předchozích kladných čísel.

Příklad vstupu: 5 2 3 4 20 30 40 -1

Výstup k danému příkladu vstupu: 40

Funkce v Javě - úvod

- uživatelem definované funkce
- předávání parametrů hodnotou

Funkce – motivační příklad

Vyjdeme z následujícího kódu ….

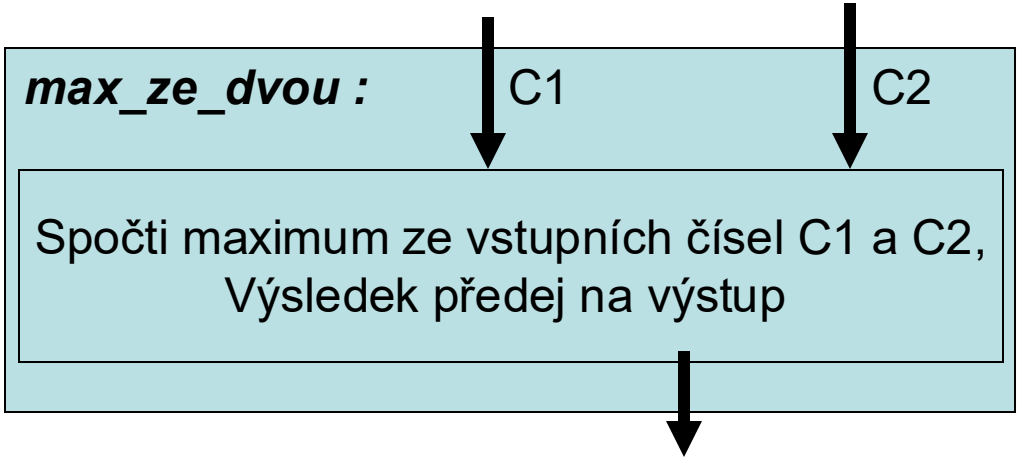
```
public static void main(String args[])
{    int x = 10; int y = 20; int v, w;

     if (x>y) v=x;
          else v=y;
     // co je teď ve v ?

     if (x>5) w=x;
          else w=5;
     // co je teď ve w ?

}
```

Najděte části kódu se stejnou činností

Funkce – motivační příklad

Našli jsme místa, vhodná pro řešení „podprogramem“ ….

```
public static void main(String args[])
{    int x = 10; int y = 20; int v, w;

     if (x>y) v=x;
          else v=y;
     // co je teď ve v ?

     if (x>5) w=x;
          else w=5;
     // co je teď ve w ?

}
```



C1, C2 … argumenty funkce
Výsledek … návratová hodnota

max_ze_dvou …. Identifikátor fce

Funkce – motivační příklad

```
public static void main(String args[])
{    int x = 10; int y = 20; int v, w;

     if (x>y) v=x;
          else v=y;
     // co je teď ve v ?

     if (x>5) w=x;
          else w=5;
     // co je teď ve w ?

}
```

funkce s identifikátorem max_ze_dvou

dvěma vstupními parametry int

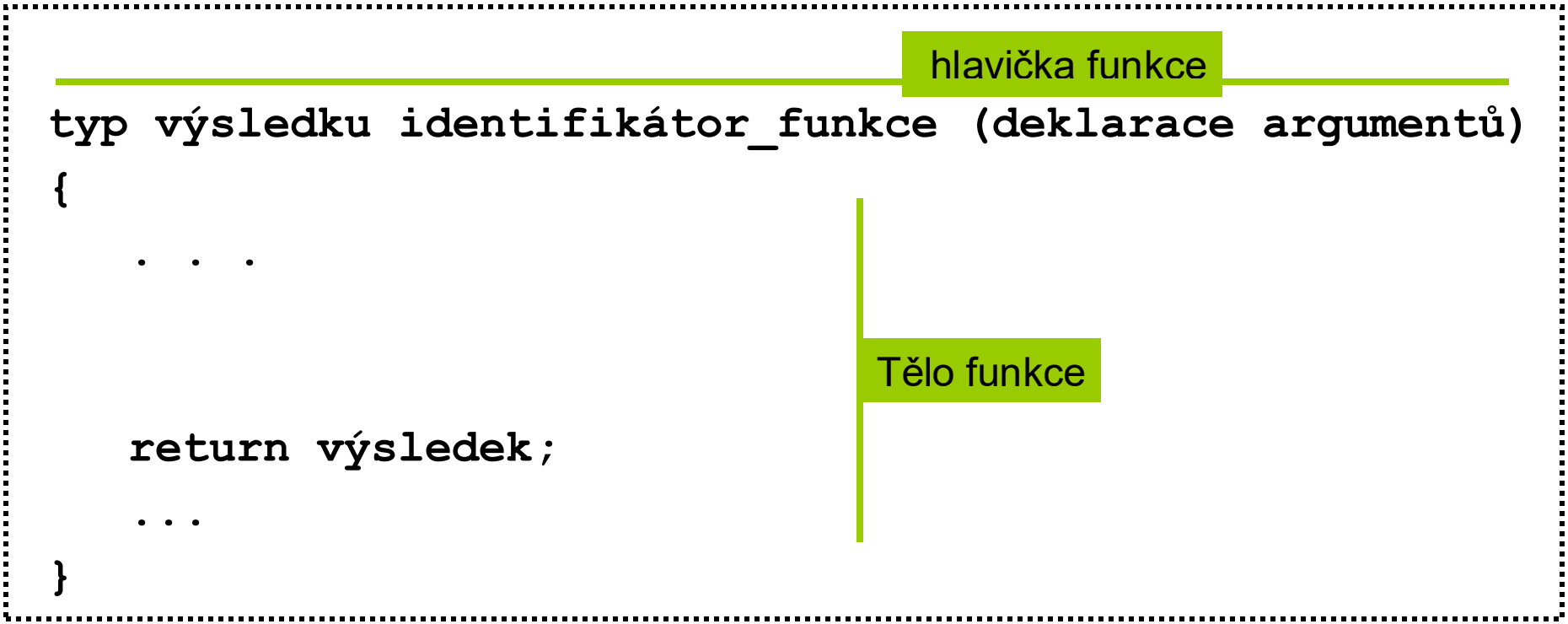
a návratovou hodnotou typu int

```
public static void main(String args[])
{    int x = 10; int y = 20; int v, w;

     v = max_ze_dvou(x,y);   // co bude ve v ?
     w = max_ze_dvou(x,5);   // co bude ve w ?
}
```

Výsledný zápis … opakovaně voláme podprogram max_ze_dvou

Funkce v Javě

- Jeden ze základních stavebních kamenů (metody tříd)
- Obecný zápis (definice) funkce je



Funkce – použití (volání funkce)

- Obecný tvar volání funkce

s uložením návratové hodnoty:

```
návratová_hodnota = identifikátor_funkce (argumenty) ;
```

bez uložení návratové hodnoty:

```
identifikátor_funkce (argumenty) ;
```

Fce – motivační př. – dokončení

→ Napíšeme funkci, která vrátí větší ze dvou zadaných čísel. Čísla budou typu **`int`** a budou to argumenty funkce. Maximum bude návratovou hodnotou.

```
/* definice funkce, která vrátí větší ze dvou celých čísel */
public static int max_ze_dvou (int a, int b)
{
     if (a>b) return(a);
          else return(b);
}
/* příklad použití námi definované funkce */
public static void main(String args[])
{    int x = 10; int y = 20; int v, w;
     v = max_ze_dvou(x,y);   // co bude ve v ?
     w = max_ze_dvou(x,5);   // co bude ve w ?
}
```

tkzv. **PŘEDÁVÁNÍ PARAMETRU HODNOTOU**

DP 1.13

```
import java.util.Scanner;


class Fce
{
		public static int max_ze_dvou(int a, int b)
		{
				if (a>b) return a; else return b;

		}

		public static void main(String args[])
		{
				int a, b;
				System.out.print("Vlozte 2 cela cisla: ");
				Scanner sc = new Scanner(System.in);
				a = sc.nextInt(); b = sc.nextInt();
				System.out.printf("vlozeno a=%d b=%d \n", a, b);
				System.out.printf("vetsi je %d \n", max_ze_dvou(a, b) );


		}

}
```

DP 1.13

```
import java.util.Scanner;


class Fce
{
		public static int max_ze_dvou(int a, int b)
		{
				// TERNARNI OPERATOR

		}

		public static void main(String args[])
		{
				int a, b;
				System.out.print("Vlozte 2 cela cisla: ");
				Scanner sc = new Scanner(System.in);
				a = sc.nextInt(); b = sc.nextInt();
				System.out.printf("vlozeno a=%d b=%d \n", a, b);
				System.out.printf("vetsi je %d \n", max_ze_dvou(a, b) );


		}

}
```

DP 1.13

```
import java.util.Scanner;

class Vypocty {

		public int max_ze_dvou(int a, int b)
		{
				if (a>b) return a; else return b;

		}
}


class Fce
{
		public static void main(String args[])
		{
				int a, b;
				System.out.print("Vlozte 2 cela cisla: ");
				Scanner sc = new Scanner(System.in);
				a = sc.nextInt(); b = sc.nextInt();
				System.out.printf("vlozeno a=%d b=%d \n", a, b);

				Vypocty v = new Vypocty();
				System.out.printf("vetsi je %d \n", v.max_ze_dvou(a, b) );

		}

}
```

Funkce – potlačení parametrů a návratové hodnoty - **void**

Fce bez návatové hodnoty
=
tkzv. **PROCEDURA**

Funkce s potlačenou návratovou hodnotou
(v těle mohu použít return, ale bez parametru)

```
void idfunkce(deklarace argumentů)
```

Funkce bez parametrů (při volání funkce nutno použít prázdné závorky)

```
typ výsledku idfunkce()
```

Hlavička funkce bez parametrů (při volání funkce nutno použít prázdné závorky)

```
void idfunkce()
```

tkzv. **VEDLEJŠÍ EFEKT** funkce – v těle funkce se ovlivní (také) něco jiného, než její argumenty a návratová hodnota. V OOP se tohoto postupu často používá – metody manipulují s daty v rámci třídy.

Př.

```
/* definice funkce bez parametrů a bez návrat. hodnoty */
public static void tiskni_pozdrav ()
{
      System.out.println("zdravime Vas");

}

/* příklad použití definované funkce – 5x vytiskne pozdrav */
public static void main(String args[])
{
  int i;

  for(i=0; i<5; i++) tiskni_pozdrav();

}
```

```
class Vypocty {

        private int hodnota = 0;

        public void nastav_hodnotu (int value)
        {
                this.hodnota = value;
        }

        public int zjisti_hodnotu ()
        {
                return this.hodnota;
        }

}
```

```
Vypocty v = new Vypocty();
System.out.printf("hodnota je %d \n", v.zjisti_hodnotu() );

v.nastav_hodnotu(20);
System.out.printf("hodnota je %d \n", v.zjisti_hodnotu() );
```

DP 1.14

Předávání parametrů hodnotou

Experimenty s rozsahem platnosti proměnných

```
import java.util.Scanner;

class Fce_rozsah
{
        public static int max_ze_dvou(int a, int b)
        {       int x;
                if (a>b) x=a; else x=b;
                a=100; b=200;
                System.out.printf("2 a=%d b=%d \n", a, b); // co se vytiskne ?
                return x;
        }

        public static void main(String args[])
        {
                int a=10, b=20, v;
                System.out.printf("1 a=%d b=%d \n", a, b); // co se vytiskne ?
                v = max_ze_dvou(a, b);
                System.out.printf("vetsi je %d \n", v );
                System.out.printf("3 a=%d b=%d \n", a, b); // co se vytiskne ?
        }
}
```

DP 1.14

```
import java.util.Scanner;

class Fce_rozsah
{
        static int a, b ;

        public static int max_ze_dvou(int a, int b)
        {       int x;
                if (a>b) x=a; else x=b;
                a=100; b=200;
                System.out.printf("2 a=%d b=%d \n", a, b); // co se vytiskne ?
                return x;
        }

        public static void fn()
        {
                System.out.printf("4 a=%d b=%d \n", a, b); // co se vytiskne ?
        }

        public static void main(String args[])
        {
                int a=10, b=20, v;
                System.out.printf("1 a=%d b=%d \n", a, b); // co se vytiskne ?
                v = max_ze_dvou(a, b);
                System.out.printf("vetsi je %d \n", v );
                System.out.printf("3 a=%d b=%d \n", a, b); // co se vytiskne ?

                fn();
        }
}
```

DP 1.14

```
import java.util.Scanner;

class Fce_rozsah
{
        static int a, b ;

        public static int max_ze_dvou(int a, int b)
        {       int x;
                if (a>b) x=a; else x=b;
                a=100; b=200;
                System.out.printf("2 a=%d b=%d \n", a, b); // co se vytiskne ?
                return x;
        }

        public static void fn()
        {
                System.out.printf("4 a=%d b=%d \n", a, b); // co se vytiskne ?
        }

        public static void main(String args[])
        {
                int v;
                System.out.printf("1 a=%d b=%d \n", a, b); // co se vytiskne ?
                v = max_ze_dvou(a, b);
                System.out.printf("vetsi je %d \n", v );
                System.out.printf("3 a=%d b=%d \n", a, b); // co se vytiskne ?

                fn();
        }
}
```

Samostatné průběžné úlohy

(SPÚ)

pro blok 1

SPÚ 1.1

Sestavte algoritmus a program pro výpočet délky přepony pravoúhlého trojúhelníka.

Vstupní údaje: délky obou odvěsen (reálná čísla)

Výstupní údaj: délka přepony

Výsledek vypište s přesností na 4 desetinná místa.

SPÚ 1.2

Sestavte algoritmus a následně napište a odlaďte program v Javě, který převede zadanou rychlost v m/s na mi/h.

Výsledek vypište na 2 desetinná místa.

Pomůcka: 1 mi = 1609,35 m

Testovací data např:
1500 m/s = 3355.40 mi/h
0.96 m/s = 2.15 mi/h

....

SPÚ 1.3 A

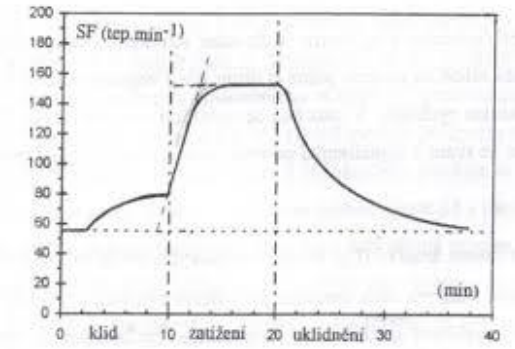
Ruffierova zkouška tělesné zdatnosti, založená na měření tepové frekvence před a po zátěži.
POSTUP:
- nejprve vsedě změřte na zápěstí počet tepů TF1
- proveďte 30 dřepů v pravidelném tempu 1 dřep za sekundu
- ihned po výkonu usedněte a změřte počet tepů TF2
- v klidu seďte a uklidňujte se po dobu 1 minuty
- pak změřte počet tepů TF3

Sestavte algoritmus a napište a odlaďte v Javě program pro výpočet tzv. Ruffierova indexu (RI) podle vztahu:

RI= [(TF1 + TF2 + TF3) - 200]/10

Výsledek vypište s přesností na 1 desetinné místo.

Čím vyšší hodnota Ruffierova indexu, tím horší kondice. V některé z dalších úloh pak vytvoříme program, který bude člověka klasifikovat podle hodnoty RI do několika kategorií.



SPÚ 1.3 B

Sestavte algoritmus a napište a odlaďte v Javě program pro klasifikaci osob podle tzv. Ruffierova indexu (RI) spočteného dle vztahu (viz úloha CV 1.3 A) :

RI= [(TF1 + TF2 + TF3) - 200]/10

Index	Zdatnost
nižší než 0	výborná
0,1 - 5	velmi dobrá
5,1 - 10	průměrná
10,1 - 15	podprůměrná
vyšší než 15	nedostatečná

Odhalte, co je v zadání špatně a upravte podle toho jeho specifikaci.

SPÚ 1.4

Napište algoritmus který určí, zda zadaná tři čísla mohou být stranami pravoúhlého trojúhelníka.

Odlaďte také program v Javě,

Kterým algoritmus otestujete. Pořadí, v jakém vstupují délky stran, není definováno !

SPÚ 1.5

Úloha:

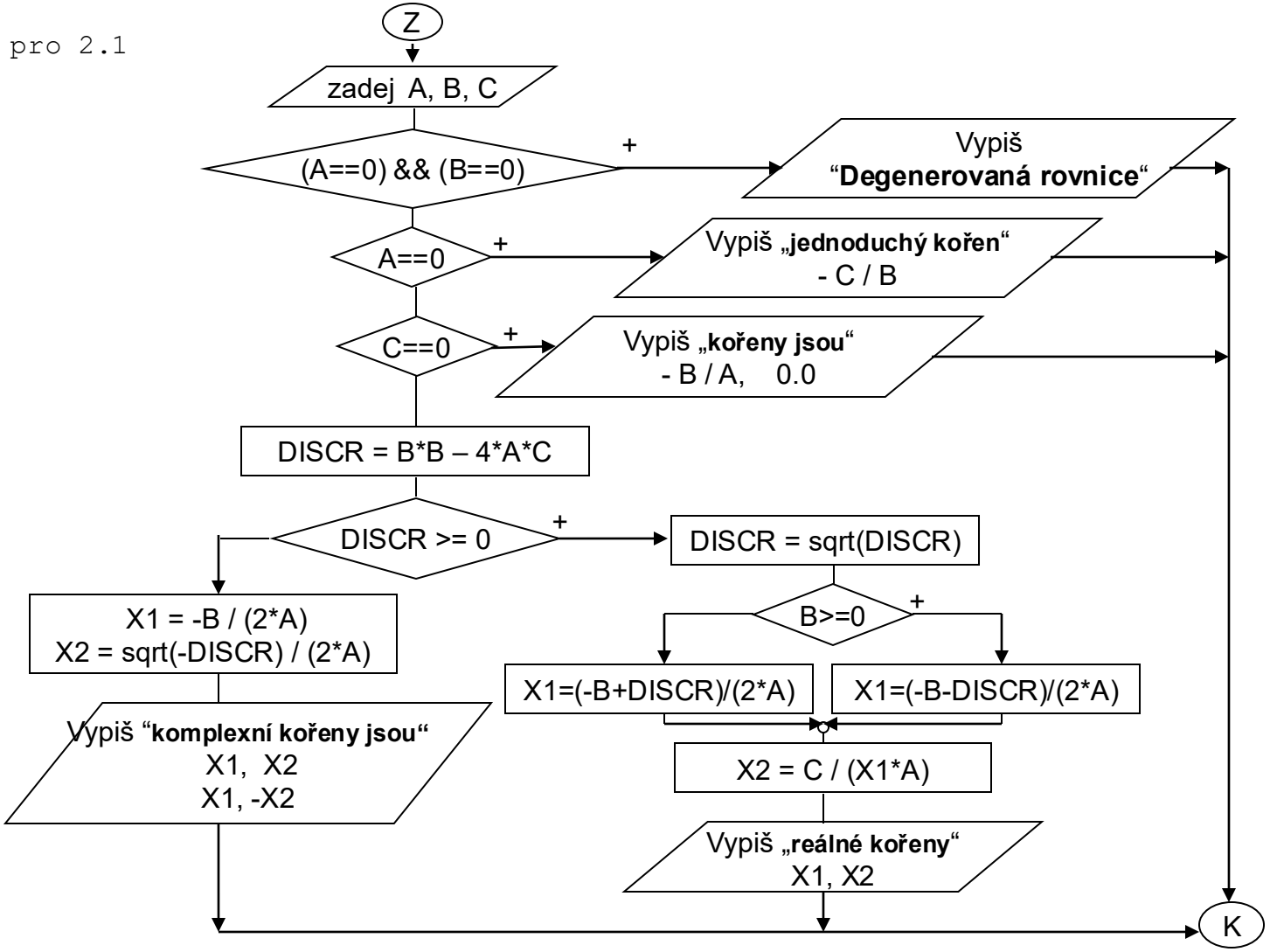
Jsou známy koeficienty a,b,c kvadratické rovnice $ax^2+bx+c=0$. Navrhněte algoritmus který určí **reálné kořeny** této rovnice. Pokud rovnice reálné kořeny nemá, tuto skutečnost rozpoznejte a oznamte.

Odladťe také program v Javě, kterým algoritmus otestujete.

1 1 -6 ------ 2 -3

……..

Algoritmus pro 2.1



SPÚ 1.6

Na vstupu je dána sekvence kladných celých čísel zakončená nulou.

Navrhněte algoritmus a napište program, který určí, kolik čísel mezi nimi je lichých a zároveň dělitelných třemi.

Př. vstup: 4 5 8 9 12 0

výstup: 1

SPÚ 1.7

Na vstupu je dána sekvence tělesných teplot pacientů (ve °C a s rozlišením na desetiny stupně) vzorkovaná po 10 minutách a jejich aktuální tepová frekvence v leže v klidu (v tepech za minutu, rozlišení na celé tepy). Vstup je ukončen vložením nulové hodnoty.

Napište algoritmus a odlaďte program v Javě, který zjistí a vypíše, kolik z těchto pacientů mělo tělesnou teplotu mezi 36 a 37 °C (bez těchto mezních hodnot) a zároveň tepovou frekvenci mezi 55 a 85 tepy za vteřinu (včetně těchto mezních hodnot). Vypište také, kolik procent z celkového počtu pacientů to tvořilo (výpis s přesností na celá procenta).

Příklad vstupních dat:

36.3	76
35.9	84
37.2	78
39.8	113
35.7	49
36.7	81
0	0

Výstup pro tato vst. data:

Pocet vzorku splnujicich definovane rozmezi
2

Procento pacientu splnujicich def. rozmezi
33

SPÚ 1.8

Doplňte předchozí úlohu 1.7 o následující funkce:

a) výpis hodnoty nejvyšší teploty u všech pacientů,
b) výpis průměrné tepové frekvence pacientů, u kterých je splněna sada podmínek z úlohy 3.1

Příklad vstupních dat:

36.3	76
35.9	84
37.2	78
39.8	113
35.7	49
36.7	81
0	0

Výstup pro tato vst. data:

Nejvyssi telesna teplota 39.8 st. C
Prumerna tep. frekvence relevantnich pac. 79 tepu/s

2. (příští) blok

Zopakování algoritmizace č. II.

- miniprojekt z látky 1. bloku (max. 1 hodina)
- pole, zásobník, fronta, spojový seznam v Javě (dynamické datové struktury s příklady implementací a použití)
- odhad (asymptotické) časové složitosti vybraných algoritmů
- vybrané algoritmy řazení a vyhledávání

Zakončení : krátký test, případně miniprojekt na začátku dalšího bloku

Zdroje

- https://www.itnetwork.cz/java/zaklady/java-tutorial-uvod-do-jazyka-java
- https://www.fi.muni.cz/~tomp/pb162/printable/01_flow_control.html
- https://www.interval.cz/clanky/naucte-se-javu-datove-typy/
- https://www.kiv.zcu.cz/~rohlik/vyuka/uur/UUR-10.html
- Algoritmizace a programování – sylaby přednášek a cvičení (moodle-vyuka.cvut.cz)

3. blok

Grafy, stromy

- Pojmy graf, strom, popis pomocí matice sousednosti, incidenční matice, ohodnocené grafy
- Orientované, neorientované grafy
- Pojem kostra grafu
- Zopakování maticového násobení, které je nezbytné pro algoritmy
- Algoritmy hledání nejkratší cesty, Floydův algoritmus, ukázka Bellman-Fordova, Dijkstrova algoritmu

Zakončení : krátký test případně miniprojekt na začátku dalšího bloku - procedurální implementace Floydova algoritmu, alternativně ověření jeho znalosti jinou formou

4. blok

Grafové algoritmy

- Implementace grafových algoritmů ve zvoleném jazyce pro řešení reálných problémů (průchod do šířky, průchod do hloubky)
- Hledání nejkratší vzdálenosti a využití ohodnocených grafů, tj. např. model železnice a hledání nejlevnějších jízdenek,
- Využití objektů pro členění složitějších algoritmů
- Praktická ukázka implementace stromu

Zakončení : krátký test případně miniprojekt na začátku dalšího bloku

5. blok

Vybrané numerické algoritmy

- kořeny nelin. rovnic a polynomů
- numerická integrace a derivace
- numerické řešení obyč. dif. rovnic
- aproximace a interpolace, metoda nejmenších čtverců
- metody řešení soustavy lin. rovnic

Zakončení : krátký test případně miniprojekt na začátku dalšího bloku

6. blok

Paralelní výpočty, supercomputing

- Metody paralelizace algoritmů, supercomputing
- Implementace a měření času vybraného algoritmu
- Huffmanův algoritmus a pojmy z teorie informace, kódy

Zakončení : krátký test případně miniprojekt na začátku dalšího bloku

7. blok

Gramatiky, automaty

- Definice pojmů gramatiky + automaty a jejich popis s využitím teorie grafů
- Využití automatů při reálných aplikacích (Mealy, Moore), pak implementace zvoleného automatu v programovacím jazyce
- Markovovské řetěžce jako speciální případ automatů - ukázka na praktické úloze, v jednoduhcé podobě zvládnutelné

8. blok - rezerva