
KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

**Jan Laštovička**
jan.lastovicka@upol.cz
www.inf.upol.cz/lide/jan-lastovicka
17. listopadu 12, 771 46 Olomouc

Základy programování pro IT 1

# 2. Program a jeho logika

verze z 30. září 2024

## 1 Program

Posloupnost pod sebou napsaných příkazů:

```
statement1
⋮
statementn
```

nazýváme **programem**. Například:

```
x = y + 1
y = x * 2
```

je program tvořený dvěma příkazy.

Program se v prostředí **vykoná** tak, že se v prostředí postupně vykonají jeho příkazy. Například výše uvedený program se v prostředí:

| | |
|---|---|
| y | 1 |

vykoná tak, že se nejprve vykoná příkaz:

```
x = y + 1
```

který změní aktuální prostředí na:

| | |
|---|---|
| y | 1 |
| x | 2 |

a poté se vykoná příkaz:

```
y = x * 2
```

který provede změnu aktuálního prostředí:

| | |
|---|---|
| y | 4 |
| x | 2 |

**Typ prostředí** udává proměnné a typ jejich hodnot v prostředí. Například uvažujme typ prostředí s proměnnými `x` a `y`, jejichž hodnoty jsou celá čísla. Prostředí

| | |
|---|---|
| y | 1 |
| x | 2 |

je právě popsaného typu.

Program udává typ prostředí, pro které může být vykonán. Například program

```
x = y + 1
y = x * 2
```

musí být vykonán v prostředí s číselnou proměnnou `y`.

# 2 Programová trojice

(**Programová**) **trojice** je

$$\{precondition\}$$
$$program$$
$$\{postcondition\}$$

kde *precondition* a *postcondition* jsou podmínky nazývané **prekondice** a **postkondice** a *program* je program.

Například:

```
{x == 1}
x = x + 1
{x == 2}
```

Také programová trojice určuje typ prostředí, pro které dává smyls. Například trojice

```
{x == 1}
y = z + 2
{x == 1}
```

dává smysl pro prostředí s číselnými proměnným `x` a `z`.

Programová trojice

$$\{\mathit{precondition}\}\ \mathit{program}\ \{\mathit{postcondition}\}$$

je **pravdivá**, jestliže pro každé prostředí *env* typu určeného trojicí platí, že

- pokud *env* splňuje prekondici *precondition*, pak se po vykonání programu *program* v prostředí *env* dostaneme do prostředí, které splňuje postkondici *postcondition*.

Následují příklady.

1. Trojice

   ```
   {True}
   x = 1
   {x == 1}
   ```

   je pravdivá. Poté co nastavíme v libovolném prostředí `x` na jedna, bude v tomto prostředí mít `x` hodnotu jedna.

2. Trojice

   ```
   {False}
   x = 1
   {x == 2}
   ```

   je triviálně pravdivá, protože podmínka `False` není nikdy splněna. Tedy neexistuje prostředí, které by splňovalo prekondici. Program tedy nemůžeme nikdy vykonat.

3. Trojice

   ```
   {x == 1}
   y = x + 1
   x = y
   {x == 2}
   ```

je pravdivá. Zde stačí program spustit v jediném prostředí:

```
x | 1
```

program skončí v prostřed:

```
x | 2
y | 2
```

které splňuje podmínku `x == 2`.

4. Trojice

```
{x == 1}
x = x + x
y = x
{y == 3}
```

je nepravdivá, protože program spuštěný v prostředí:

```
x | 1
```

skončí v prostředí:

```
x | 2
y | 2
```

kde podmínka `y == 3` není splněna.

5. Trojice:

```
{x == 1}
x = x + y
y = x
{y == 2}
```

je nepravdivá. Program spuštěný v prostředí:

```
x | 1
y | 2
```

skončí v prostředí:

```
x | 3
y | 3
```

kde podmínka `y == 2` není splněna.

6. Trojice

```
{x == 0}
x = x + 1
{True}
```

je triviálně pravdivá, protože postkondice je vždy splněna.

7. Trojice

```
{True}
x = y
{x == y}
```

je pravdivá, přestože program dává smysl jen pro prostředí s proměnnou `y` a zde po vykonání programu bude splněno, že `x == y`.

# 3 Síla podmínek

Také podmínky udávají typ prostředí, kde dávají smysl. Například `x <= y` určuje typ prostředí s číselnými proměnnými `x` a `y`.

Podmínka *`condition1`* je **silnější než** podmínka *`condition2`*, jestliže

- pro každé prostředí *`env`* vyhovujícího typu platí, že jestliže *`env`* splňuje podmínku *`condition1`*, pak splňuje i podmínku *`condition2`*.

Například podmínka `x < y` je silnější než podmínka `x <= y`.

Podmínka *`condition1`* je **slabší než** podmínka *`condition2`*, jestliže podmínka *`condition2`* je silnější než podmínka *`condition1`*. Například `(x % 2) == 0` je slabší než `(x % 4) == 0`.

Podmínky *`condition1`* a *`condition2`* jsou **ekvivalentní**, jestliže podmínka *`condition1`* je zároveň silnější i slabší než podmínka *`condition2`*. Například podmínky `x == x` a `True` jsou ekvivalentní, ale `x < y` a `x <= y` ekvivalentní nejsou.

# 4 Důkazy pravdivosti trojic

K důkazu, že zadaná trojice je pravdivá, používáme následující pravidla.

## 4.1 Pravidlo přiřazení

Vezměme proměnnou *`variable`*, výraz *`expression`* a podmínku *`condition`*, pak platí:

```
{precodition}
variable = expression
{condition}
```

kde *precodition* vznikne z *condition* nahrazení všech výskytů *variable* za (*expression*).

Například:

```
{(x + 1) == 2}
x = x + 1
{x = 2}
```

po nahrazení prekondice za ekvivalentní jednodušší podmínku dostáváme:

```
{x == 1}
x = x + 1
{x = 2}
```

Další příklad:

```
{y == y}
x = y
{x == y}
```

po úpravě:

```
{True}
x = y
{x == y}
```

## 4.2 Pravidlo zesílení prekondice

Pokud

```
{precondition1}
program
{postcondition}
```

a *precondition2* je silnější než *precondition1*, pak

```
{precondition2}
program
{postcondition}
```

Například z

```
{True}
x = 1
{x = 1}
```

lze odvodit

```
{x = 2}
x = 1
{x = 1}
```

protože podmínka `x = 2` je silnější než podmínka `True`.

## 4.3 Pravidlo oslabení postkondice

Pokud

```
{precondition}
program
{postcondition1}
```

a *precondition2* je slabší než *precondition1*, pak

```
{precondition}
program
{postcondition2}
```

Například z

```
{True}
x = 1
{x == 1}
```

lze odvodit

```
{True}
x = 1
{(x == 1) or (x == 2)}
```

protože podmínka `(x == 1) or (x == 2)` je slabší než podmínka `x == 1`. Jestliže `x` je jedna, pak jistě platí, že `x` je jedna nebo dva.

### 4.4 Pravidlo skládání programů

Pokud

$$\{precondition\}$$
$$program1$$
$$\{condition\}$$

a

$$\{condition\}$$
$$program2$$
$$\{postcondition\}$$

pak

$$\{precondition\}$$
$$program1$$
$$program2$$
$$\{postcondition\}$$

Například z

```
{True}
x = 1
{x == 1}
```

a

```
{x == 1}
y = 2
{(x == 1) and (y == 2)}
```

plyne

```
{True}
x = 1
y = 2
{(x == 1) and (y == 2)}
```

## 5 Ukázky důkazů

1. Chceme dokázat pravdivost trojice:

```
{x > 0}
x = x + 1
{x > 0}
```

Použijeme axiom přiřazení na příkaz `x = x + 1` a postkondici `x > 0`. Obdržíme trojici:

```
{(x + 1) > 0}
x = x + 1
{x > 0}
```

Ekvivalentní úpravou zjednodušíme prekondici:

```
{x > -1}
x = x + 1
{x > 0}
```

Podmínka `x > 0` je jistě silnější než `x > -1`. Proto můžeme použít pravidlo zesílení prekondice a dostat požadovanou trojici:

```
{x > 0}
x = x + 1
{x > 0}
```

2. Chceme dokázat pravdivost trojice:

```
{(x == x0) and (y == y0)}
t = x
x = y
y = t
{(y == x0) and (x == y0)}
```

Začneme vezmeme poslední příkaz `y = t` a postkondici `(y == x0) and (x == y0)`. Pomocí axiomu přiřazení odvodíme trojici:

```
{(t == x0) and (x == y0)}
y = t
{(y == x0) and (x == y0)}
```

Pokračujeme prostředním příkazem `x = y`. Víme, že jeho postkondice musí být `(t == x0) and (x == y0)`. Opět použijeme axiom přiřazení a dostáváme:

```
{(t == x0) and (y == y0)}
x = y
{(t == x0) and (x == y0)}
```

Obdržené trojice můžeme pomocí pravidla skládání složit:

```
{(t == x0) and (y == y0)}
x = y
y = t
{(y == x0) and (x == y0)}
```

Podobně postkondice prvního příkazu `t = x` musí být `(t == x0) and (y == y0)`. Proto axiom přiřazení vede na:

```
{(x == x0) and (y == y0)}
t = x
{(t == x0) and (y == y0)}
```

Složením posledních dvou trojic obdržíme kýženou trojici:

```
{(x == x0) and (y == y0)}
t = x
x = y
y = t
{(y == x0) and (x == y0)}
```

3. Trojici

```
{x >= 0}
x = x - 1
y = x
{y >= 0}
```

dokázat nelze, protože je nepravdivá. Stačí najít jedeno prostředí, kde je splněna prekondice a po skončení programu není splněna postkondice. Konkrétně pokud program spustíme v prostředí:

| | |
|---|---|
| x | 0 |

dostaneme se do stavu

| | |
|---|---|
| x | -1 |
| y | -1 |

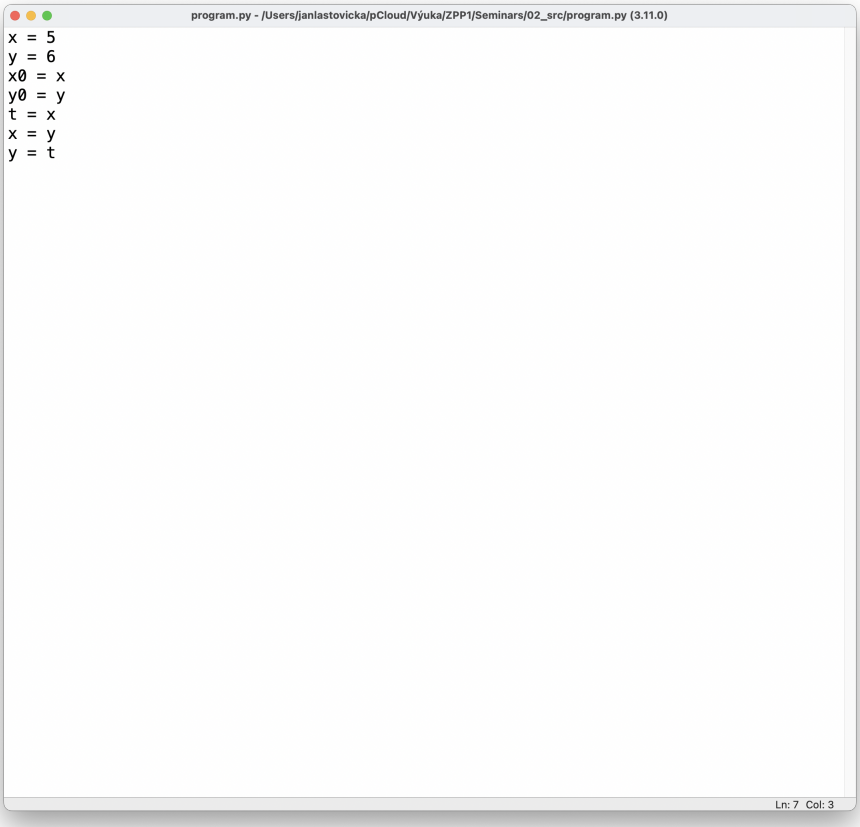
kde není splněna postkondice `y >= 0`.

# 6 Spouštění programu

Zatím jsme používali interpret v **interaktivním režimu**. Interpret čekal na zadání našeho příkazu, pak jej vykonal a znovu čekal na další příkaz. Interpret můžeme také použít tak, aby vykonal program zapsaný v souboru. Takový soubor ztotožňujeme s programem, který obsahuje. Nejprve vytvoříme jednoduchý program. Volbou `File / New File` hlavní nabídky otevřete nové okno. Napište do něj následující obsah:

```
x = 5
y = 6
x0 = x
y0 = y
t = x
x = y
y = t
```

Nyní volbou `File / Save` uložte obsah do souboru na disk pod názvem `program.py`. Přípona souboru `py` prozrazuje operačnímu systému, že se jedná o program napsaný v jazyce Python. Právě jste vytvořili svůj první program. Okno editoru s programem zachycuje Obrázek 1.




Obrázek 1: Okno editoru `IDLE` s prvním programem.

Program můžete spustit volbou `Run / Run Module` z hlavní nabídky nebo stiskem klávesy `F5`. Spuštění programu vede k jeho vykonání v **prázdném prostředí**; to je prostředí, které neobsahuje žádnou proměnnou. Po skončení vykonávání programu bude aktuální prostředí vypadat následovně.

| | |
|---|---|
| x | 6 |
| y | 5 |
| x0 | 5 |
| y0 | 6 |
| t | 5 |

Po skončení vykonávání programu se interpret přepne do interaktivního režimu a v okně `Shell` můžeme zjišťovat hodnoty proměnných:

```
>>> x
6
>>> y
5
>>> (y == x0) and (x == y0)
True
```

Program začíná nastavením hodnot vstupním proměnným. V našem případě jsou to příkazy:

```
x = 5
y = 6
```

Počet vstupních proměnných a jejich možné hodnoty určuje popis programu. Popis našeho programu by mohlo znít:

> Na vstupu jsou dány libovolné hodnoty v proměnných `x` a `y`. Program prohodí obsah proměnných.

Dále následuje nastavení pomocných proměnných, které slouží ke kontrole správnosti programu:

```
x0 = x
y0 = y
```

Zbytek tvoří samotný program:

```
t = x
x = y
y = t
```

Pro zvýšení čitelnosti můžeme do programu dodávat komentáře. Vše do konce řádku za znakem mřížky (`#`) interpret ignoruje. Také můžeme části programu oddělit prázdnými řádky. Prázdné řádky interpret přeskakuje. Zvýšíme čitelnost našeho programu:

```
# Program, který prohazuje hodnoty vstupních proměnných.

# Vstupní proměnné:
x = 5
y = 6

# Pomocné proměnné:
x0 = x
y0 = y

# Program

# Prekondice: (x == x0) and (y == y0)
t = x
x = y
y = t
# Postkondice: (y == x0) and (x == y0)
```

Aby bylo možné v programu tisknout hodnoty, zavedeme si *příkaz tisku*:

```
print(value1, value2, ...)
```

kde *value1*, *value2*, ... jsou libovolné hodnoty. Příkaz tisku se vykoná tak, že vytiskne hodnoty *value1*, *value2*, ... oddělené mezerou a odřádkuje. Například:

```
>>> print(2, 4, 6)
2 4 6
>>>
```

Můžeme například upravit náš program tak, aby tiskl hodnotu prekondice, postkondice a proměnných, které nás zajímají:

```
# Program, který prohazuje hodnoty vstupních proměnných.

# Vstupní proměnné:
x = 5
y = 6

# Pomocné proměnné:
x0 = x
y0 = y
```

```
# Program:

print((x == x0) and (y == y0))

t = x
x = y
y = t

print((y == x0) and (x == y0))
print(x, y)
```

Vykonání programu vytiskne:

```
True
True
6 5
```

# Otázky a úkoly k semináři

1. U následujících trojic rozhodněte, zda jsou pravdivé. V kladném případě trojici dokažte, v záporném napište, v kterém prostředí je pravdivost porušena.

   (a)

   ```
   {x > 0}
   x = x - 1
   {x > 0}
   ```

   (b)

   ```
   {(x % 2) == 0}
   x = x + 2
   {(x % 2) == 0}
   ```

   (c)

   ```
   {(x == x0) and (y == y0)}
   x = y
   {(y == x0) and (x == y0)}
   ```

   (d)

   ```
   {True}
   y = x + 1
   z = y * 2
   {z == 2 * (x + 1)}
   ```

2. Napište program tak, aby následující trojice byla pravdivá.

```
{(x == x0) and (y == y0)}
program
{(x == x0 + y0) and (y == x0 - y0)}
```

   V programu nesmíte používat proměnné `x0` a `y0`. Správnost programu dokažte. Program spusťte v interpretu.