Základy algoritmizace

3. Řízení běhu programu

doc. Ing. Jiří Vokřínek, Ph.D.
Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Základy algoritmizace

- Dnes:
  - Řízení výpočtu
  - Větvení
  - Cykly
  - Výjimky

Problém, algoritmus a program

Příklad: Najít největšího společného dělitele čísel 6 a 15

- Víme co musí platit pro číslo *d*, aby bylo největším společným dělitelem čísel *x* a *y*
- Známou deklarativní znalost o problému můžeme využít pro návrh výpočetního postupu, např.
  1. Nechť máme nějaký odhad čísla *d*
  2. Můžeme ověřit, zda-li *d* splňuje požadované vlastnosti
  3. Pokud ano, jsme u cíle
  4. Pokud ne, musíme *d* vhodně modifikovat a začít znovu
- Výpočetní problém chceme vyřešit využitím konečné množiny primitivních operací počítače
- Konkrétní úlohu pro čísla 6 a 15 zobecňujeme pro „libovolná“ čísla *x* a *y*, pro které navrhneme algoritmus
- Algoritmus následně přepíšeme do programu využitím konkrétního programovacího jazyka

Algoritmus a program

- Algoritmus je postup řešení třídy problémů
- Algoritmus je recept na výpočetní řešení problému
- Program je implementací algoritmu s využitím zápisu příkazů programovacího jazyka
- Program je posloupnost instrukcí počítače

- Předpokládáme, že náš problém lze výpočetně řešit a je výpočetně zvladatelný

Problémy v předmětu ZAL takové jsou, ale v praktických problémech tomu tak nemusí být vždy. Můžeme narazit na problém jak úlohu vůbec formulovat, nebo na limity výpočetního výkonu.

Řešení problému

- Množina primitivních instrukcí počítače je relativně malá
  - Práce s číselnými hodnotami (v operační paměti počítače)

    Odkazované jmény deklarovaných proměnných
  - Výpočetní operace (výrazy)

    Binární nebo unární operace, tj. čtení jednoho nebo dvou číselných hodnot z paměti, provedení operace a zápis výsledku do paměti.
  - Testování hodnot proměnných (podmínky a větvení výpočtu)

    Pokud podmínka platí, vykonej instrukci, jinak udělej něco jiného nebo nic.
  - Skoky na provedení konkrétní posloupnosti instrukcí v závislosti na splnění podmínky

    „Program counter“ (PC) jako ukazatel z jaké adresy v paměti čte počítač instrukce pro vykonání.
- Tyto instrukce se objevují ve své abstraktní podobě
  - V zápisu algoritmu např. jako bloky vývojového diagramu
  - V zápisu programu jako příkazy a vyhrazená klíčová slova

Příklad – slovní popis výpočtu

- Úloha

  Najděte největšího společného dělitele čísel 6 a 15.

  Co platí pro společného dělitele čísel?

- Řešení

  Návrh postupu řešení pro dvě libovolná přirozená čísla

  Definice vstupu a výstupu algoritmu

  - Označme čísla *x* a *y*
  - Vyberme menší z nich a označme jej *d*
  - Je-li *d* společným dělitelem *x* a *y* končíme
  - Není-li *d* společným dělitelem pak zmenšíme *d* o 1 a opakujeme test až *d* bude společným dělitelem *x* a *y*

- Symboly *x*, *y* a *d* reprezentují proměnné (paměťové místo) ve kterých jsou uloženy hodnoty, které se v průběhu výpočtu mohou měnit

Příklad – slovní popis algoritmu

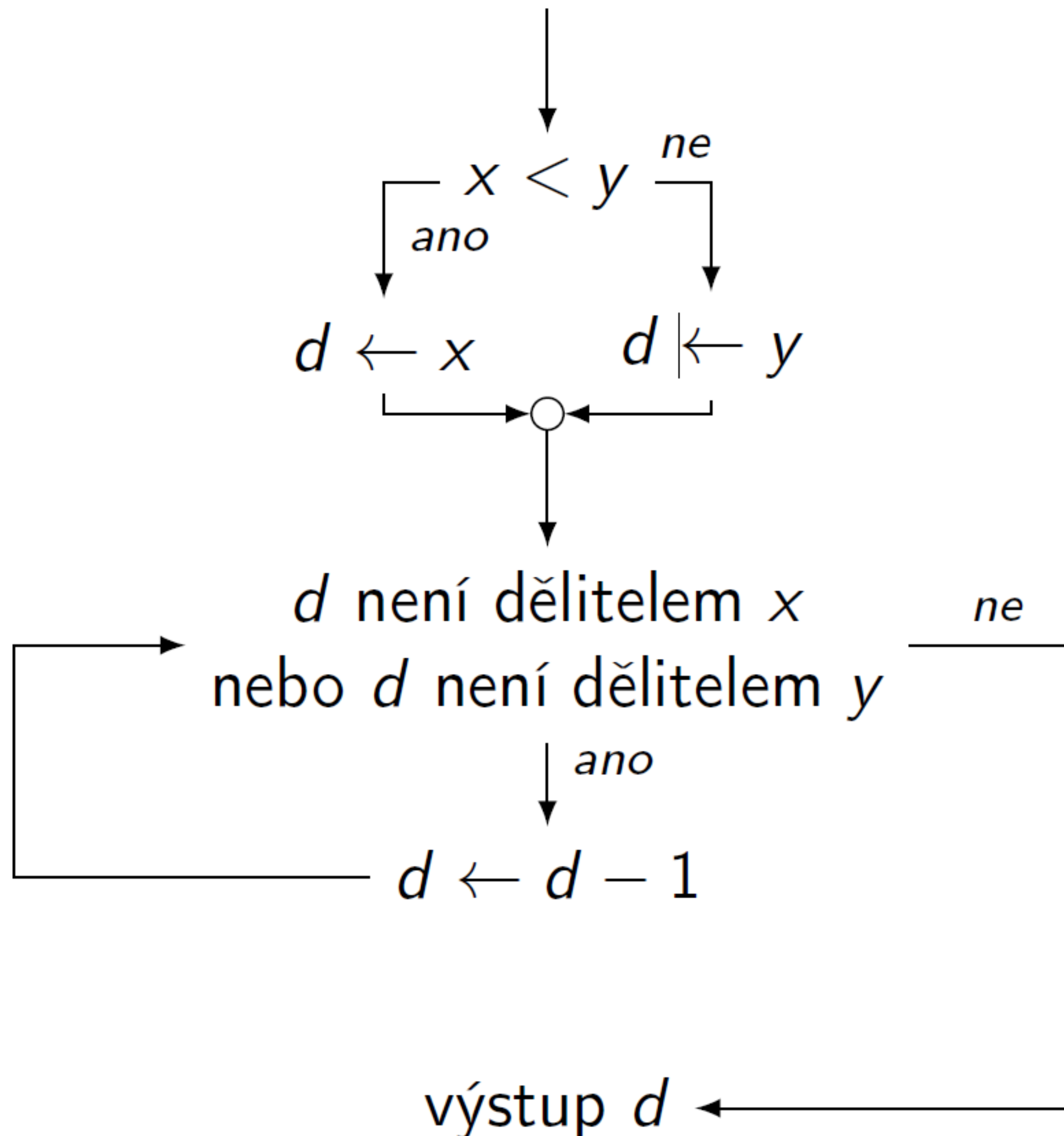
- Úloha

  Najděte největší společný dělitele přirozených čísel *x* a *y*.

- Řešení
  - Vstup: dvě přirozená čísla *x* a *y*
  - Výstup: přirozené číslo *d* – největší společný dělitel *x* a *y*
  - Postup:
    1. Je-li $x < y$, pak *d* nastav na hodnotu *x*, jinak na hodnotu *y*
    2. Pokud *d* není dělitelem *x* nebo *d* není dělitelem *y* opakuj krok 3, jinak proveď krok 4
    3. Zmenši *d* o 1
    4. Výsledkem je hodnota *d*

Algoritmus = výpočetní postup jak zpracovat vstupní data a určit (vypočítat) požadované výstupní hodnoty (data) s využitím elementárních výpočetních instrukcí a pomocných dat.

Příklad – grafický popis algoritmu



Příklad – algoritmus v pseudokódu

- Zápis algoritmu využitím klíčových a dobře pochopitelných slov

Algoritmus 1: Nalezení největšího společného dělitele

Vstup: x, y – kladná přirozená čísla
Výstup: d – největší společný dělitel x a y

if $x < y$ **then**
    $d \leftarrow x$;
else
    $d \leftarrow y$;
while d *není dělitelem* x *nebo* d *není dělitelem* y **do**
    $d \leftarrow d - 1$;
return d

Neodpovídá přesně zápisu programu v konkrétním programovacím jazyce, ale je čitelný a lze velmi snadno přepsat

Příklad – program v Pythonu

```
def getGreatestCommonDivisor(x, y):
    if (x<y) :
        d = x
    else:
        d = y
    while ((x % d != 0) or (y % d != 0)):
        d = d - 1
    return d
```

```
def getGreatestCommonDivisor(x, y):
    d = x if x < y else y
    while x % d or y % d: d -= 1
    return d
```

Řízení výpočtu

Základní výpočetní operace

- Aritmetické operátory

  + - * / % ** //

- Porovnávací (relační) operátory

  == != <> > < >= <=

- Přiřazení

  = += -= *= /= %= //=

- Bitové operace

  & | ^ ~ << >>

- Logické operátory

  and or not

- Operátory příslušnosti

  in not in

- Operátory identity

  is is not

Řízení výpočtu

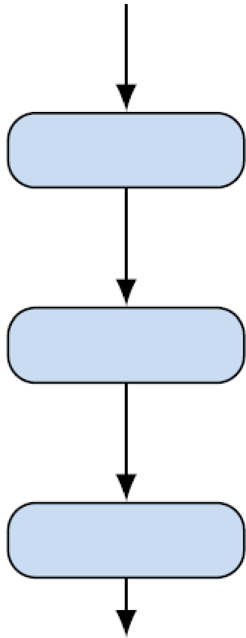
- Strukturované příkazy
  - **Složený příkaz** – posloupnost příkazů
  - **Blok** – posloupnost deklarací a příkazů vymezena odsazením

V jiných jazycích např. závorkami {}

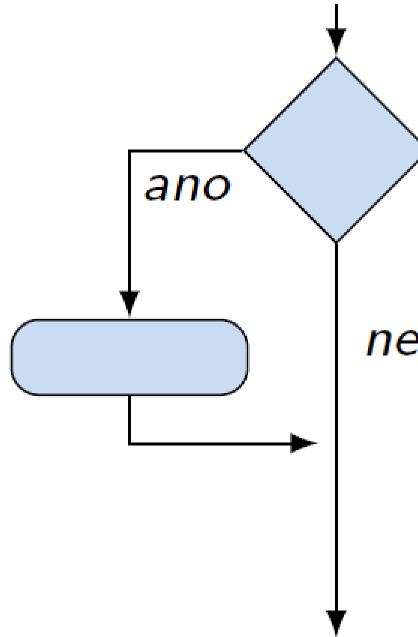
- Základní řídicí struktury
  - **Posloupnost** – předepisuje postupné provedení dílčích příkazů
  - **Větvení** – předepisuje posloupnost v závislosti na splnění určité podmínky
  - **Cyklus** – předepisuje opakované provedení posloupnosti (bloku) v závislosti na splnění určité podmínky

Řízení výpočtu

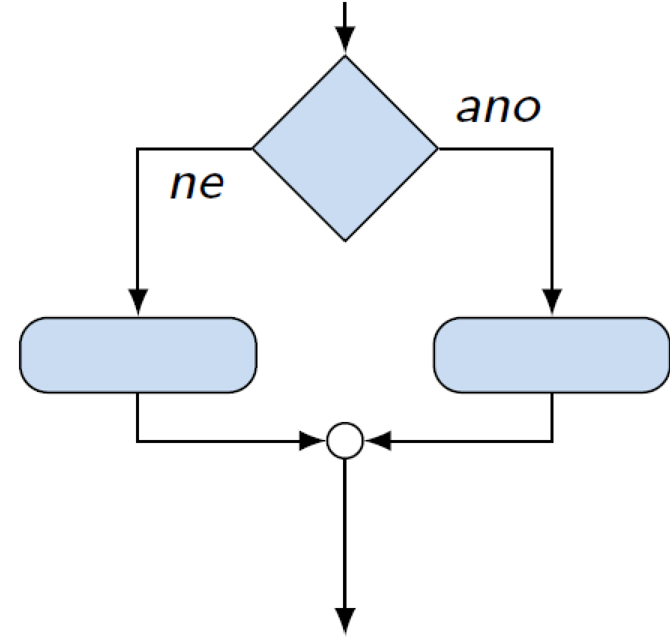
- Sekvence



- Podmínka if

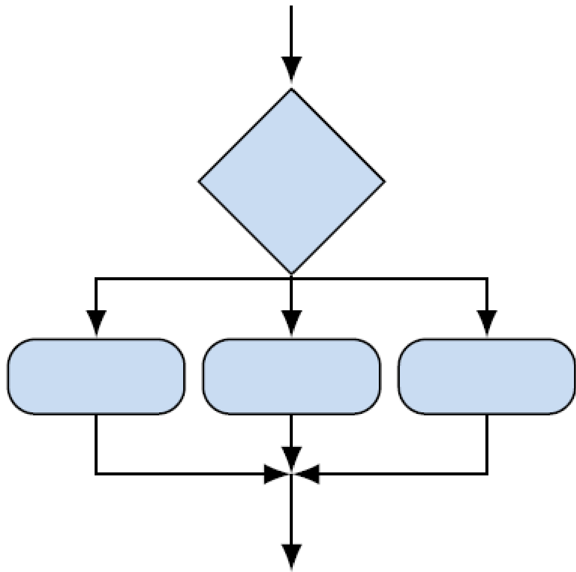


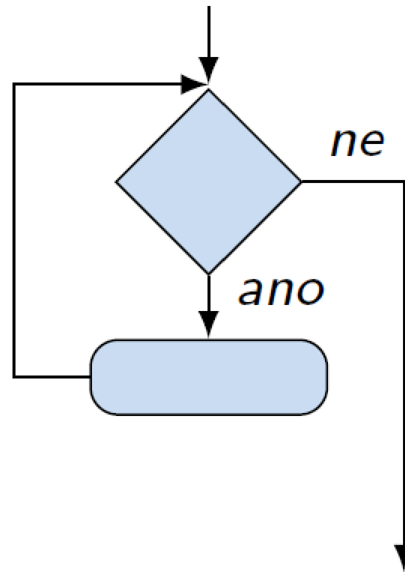
- Podmínka if-else

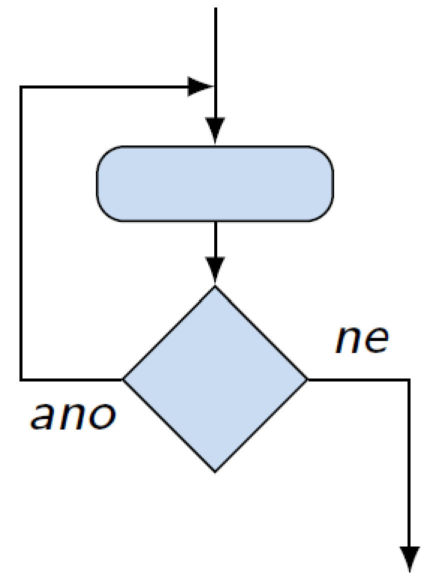


Řízení výpočtu

- Větvení switch*
- Cyklus for a while
- Cyklus do*







**v Pythonu není*

Větvení

Větvení

- Příkaz if umožňuje větvení programu na základě podmínky

```
if expression:
   statement(s)
```

- Podmínka je logický výraz, jehož hodnota je typu Boolean
  - Každý objekt lze testovat – za false je považováno:
    - None, False
    - Nula numerického výrazu
    - Prázdná sekvence „“, (), []
    - Vše ostatní je True

Porovnej:

```
while x % d != 0 or y % d != 0: d -= 1
while x % d or y % d : d -= 1
```

- Příkaz je příkaz, respektive blok

Větvení

- if…elif…else – volitelné další volby

```
if expression:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

- Příklad

```
if (x<y):
    tmp = x
    x = y
    y = tmp
```

```
if (x<y):
    min = x
    max = y
else:
    min = y
    max = x
```

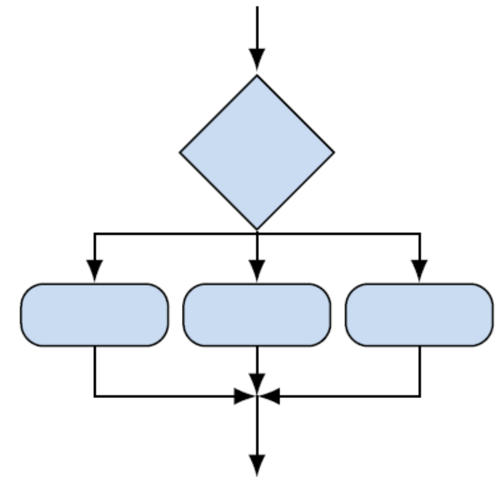
Větvení

- Přepínač switch umožňuje větvení programu na základě různých hodnot výrazu výčtového typu

Není v pythonu přímo podporován

- Základní tvar (Java)

```
switch (výraz) {
    case konstanta1: příkazy1; break;
    case konstanta2: příkazy2; break;
    ...
    case konstantan: příkazyn; break;
    default: příkazydef; break;
}
```



- Vypočte se hodnota výrazu a provedou se příkazy odpovídající konstantě s identickou hodnotou

Větvení

- Switch příklad – napište program, který podle čísla dne vytiskne na obrazovky jeho jméno

```
dayOfWeek = 2

if dayOfWeek==1:
    print("Monday")
elif dayOfWeek==2:
    print("Tuesday")
elif dayOfWeek==3:
    print("Wednesday")
elif ...
elif dayOfWeek==7:
    print("Sunday")
else:
    print("Invalid day")
```

```
dayOfWeek = 2

week = {
    1:"Monday",
    2:"Tuesday",
    3:"Wednesday",
        ...
    7:"Sunday"
}

print(week.get(dayOfWeek,
"Invalid day"))
```

Větvení

- Switch – dictionary a funkce
  - S anonymní funkcí

```
dayOfWeek = 2
week = {
    1:lambda: print("Monday"),
    2:lambda: print("Tuesday"),
    7:lambda: print("Sunday")
}[dayOfWeek]()
```

  - Obecně

```
dict = {
    1:lambda x,y: x+y,
    2:lambda x,y: x-y,
    3:lambda x,y: x*y
}
oper, a, b = 1,10,5
result = dict[oper](a,b)
```

```
dict = {
    1:foo1,
    2:foo2,
}

dict[oper](arguments)
```

Cykly

Cykly

- Cyklus while
  - Základní tvar cyklu – běží dokud je podmínka *true*

```
while expression:
   statement(s)
```

Cyklus nemusí proběhnout ani jednou!

  - Nekonečná smyčka

```
while True:
   statement(s)
```

  - Else – vykonán pokud podmínka přestane platit (jen jednou)

```
while expression:
   statement(s)
else:
   statement(s)
```

Cykly

- Příklad

```
a=0
while a<10:
    print(a)
    a+=1

else:
     print("on false")
```

```
0
1
2
3
4
5
6
7
8
9
on false
```

- Přerušení běhu break

```
a=0
while a<10:
    print(a)
    a+=1
    if a>5: break
else:
     print("on false")
```

```
0
1
2
3
4
5
```

Cykly

- Cyklus for
  - Používá se pro iterace v daném rozsahu
  - Obecně je ekvivalentem zápisu

```
a=0
while a<10:
    doSomething()
    a+=1
```

  - Příklad Java

```
for (int i=0; i<10; i++) {
   doSomething()
}
```

  - V Pythonu přes seznamy, řetězce, range

```
for a in range(10):
    doSomething()
```

Cykly

- range je funkce generující posloupnosti celých čísel
  - range(5) vygeneruje posloupnost 0, 1, 2, 3, 4
  - range(1,10) posloupnost zacinající 1 a končící 9!
  - obecně

```
range([start], stop[, step])
```

  - Všechny parametry celá čísla, mohou být záporná

- Příklady cyklů

```
for a in range(0,10,2): print(a)

list = (1,2,3,4,5)
for a in list: print(a)

for s in "Nejaky retezec": print(s)
```

Cycly

- Řízení cyklů – for i while
  - else – prováděno při nesplnění podmínky (tedy na konci)
  - break – ukončí cyklus (přeskočí i else)

  - continue – „restartuje“ cyklus, pokračuje na další iteraci

```
for s in "Nejaky retezec":
    if s == " ": continue
    print(s)
```

  - pass – *null statement*, nedělá nic

```
for s in "Nejaky retezec":
    if s == " ": pass
    else: print(s)
```

Cykly

- Konečnost algoritmu – pro přípustná data v konečné době skončí
- Aby byl algoritmus konečný musí každý cyklus v něm uvedený skončit po konečném počtu kroků
- Zacyklení – jeden z nejčastějších důvodů neukončení programu
  - Program vykonává cyklus, jehož podmínka není nikdy splněna
  - Příklad

```
while i != 0:
    j = i - 1
```

Neprovede se ani jednou, nebo nikdy neskončí

Cycky

- Základní pravidlo pro konečnost cyklu
  - Provedením těla cyklu se musí změnit hodnota proměnné použité v podmínce cyklu
  - Ne vždy zaručuje úspěch

```
while i != 0: i -= 1
```

Konečnost cyklu závisí na hodnotě proměnné před vstupem do cyklu

  - Vstupní podmínky musí zajistit příkazy předcházející cyklu
  - Zabezpečený program testuje přípustnost dat

  - Cyklus *for* je robustní proti zacyklení

```
for a in range(0, 10, -1):
    print("run")
print("stop")
```

Výjimky

Výjimky

- K řízení výjimečných stavů programu
- Událost, která vzniká při běhu programu a přeruší jeho *normální* tok
- Výjimka musí být *ošetřena*, nebo program končí
- Příklad výjimky
  - ZeroDivisionError, IndexError, IOError, TypeError

```
a= 1 + "1"
```

```
Traceback (most recent call last):
  File „program.py", line 1, in <module>
    a= 1 + "1"
TypeError: unsupported operand type(s) for +: 'int' and 'str'

Process finished with exit code 1
```

Výjimky

- Zachycení a ošetření výjimek

```
try:
    block of operations
except Exception1:
    handle Exception1 type here
except Exception2:
    handle Exception2 type here
else:
    execute this block in case of no exception
```

- Příklad

```
try:
    a= 1 + "1"
except:
    print("Something wrong")
else:
    print("Everything is OK")
```

Výjimky

- Zachycení více výjimek

```
try:
    block of operations
except(Exception1[, Exception2[,...ExceptionN]]]):
    if there is any exception from given list
    then execute this block
else:
    if there is no exception execute this block
```

- try-finally
  - Nezachytí výjimku, není ovlivněno except a else
  - Blok finally se vždy provede

```
try:
    block of operations
    this will be skipped due to any exception
finally:
    this would be allways executed
```

Pokud nastala výjimka, „vyvolá“ se po bloku `finally`

Výjimky

- try-finally
  - Příklad práce se souborem

```
try:
    fh = open("testfile", "w")
    try:
        fh.write("This is my test file.")
    finally:
        print("Going to close the file")
        fh.close()
except IOError:
    print("Error: can\'t find file or read data")
```

Výjimky

- Výjimky s argumenty
  - Umožňují předat informaci o důvodu chyby
  - Jedna hodnota nebo tuple

(error string, error number, error location)

```
try:
   You do your operations here
except ExceptionType as argument:
   You can print value of argument here
```

- Příklad

```
def getIntVal(var):
   try:
      return int(var)
   except ValueError as argument:
      print("No numbers in\n", argument)

getIntVal("xyz")
```

Výjimky

- Vyvolávání výjimek

```
raise Exception("my exception")

Traceback (most recent call last):
  File "program.py", line 1, in <module>
    raise Exception("my exception")
Exception: my exception

Process finished with exit code 1
```

- Po zachycení lze opět vyvolat

```
try:
    a=1+"1"
except:
    print("Something wrong")
    raise
```

Výjimky

- Obecná pravidla
  - Zachytáváme jen to, co umíme napravit
  - Napravujeme na správném místě a správným způsobem
  - Nepoužíváme pro ošetření očekávaných stavů

    výjimka = něco výjimečného
  - Vyvarujeme se „tichému“ zachycení

```
try:
    a=1+"1"
except:
    pass #TODO: something wrong?
```

Základy algoritmizace

- Dnes:
  - Řízení výpočtu
  - Větvení
  - Cykly
  - Výjimky

Příště shrnutí, datové struktury, algoritmy