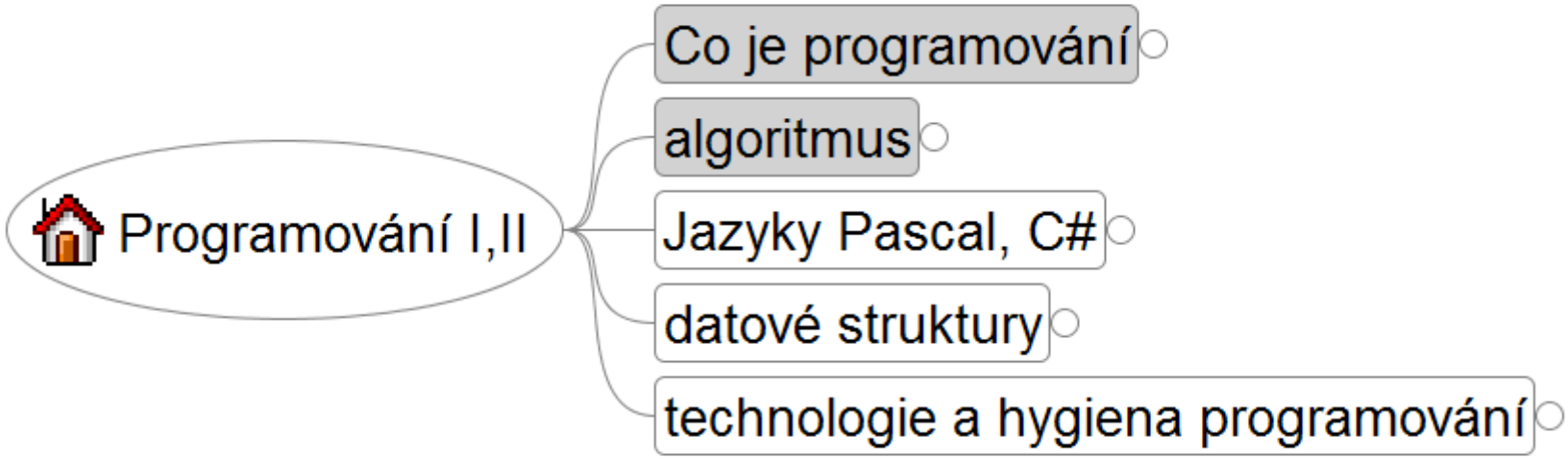
NPRG030 Programování I

RNDr. Tomáš Holan, Ph.D.

4.patro, dveře 404

http://ksvi.mff.cuni.cz/~holan/

Tomas.Holan@mff.cuni.cz


Programování I,II
Co je programování
algoritmus
Jazyky Pascal, C#
datové struktury
technologie a hygiena programování

Programování je…

- způsob, jak ovládat počítač
- umění řešit úlohy
  (a počítač nám v tom může pomoci)

- umění psát programy

? co je to program ?

- předpis, podle kterého počítač může provádět
  výpočet nějakého algoritmu

? co je to algoritmus ?

Příklad:

Algoritmus na sečtení dvou čísel zapsaných v desítkové soustavě.

1. Desítkové zápisy čísel umístíme pod sebe tak, aby jejich pravé okraje byly zarovnány
2. Delší z čísel doplníme zleva jednou nulou
3. Kratší z čísel doplníme zleva tolika nulami, aby byla obě čísla stejně dlouhá
4. Postupujeme zprava a ke každé dvojici číslic určíme číslici výsledku.
   Přitom tato číslice nezáleží jen na této dvojici, ale i na stavu výpočtu
   - Stavy jsou dva:
     - „s přenosem"
     - „bez přenosu"
   - na začátku je stav „bez přenosu"
   - výsledné číslice a stav lze určit například z tabulek:

Pro stav „bez přenosu":

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Pro stav „s přenosem":

	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0	1
2	3	4	5	6	7	8	9	0	1	2
3	4	5	6	7	8	9	0	1	2	3
4	5	6	7	8	9	0	1	2	3	4
5	6	7	8	9	0	1	2	3	4	5
6	7	8	9	0	1	2	3	4	5	6
7	8	9	0	1	2	3	4	5	6	7
8	9	0	1	2	3	4	5	6	7	8
9	0	1	2	3	4	5	6	7	9	0

výsledek „bez přenosu"
výsledek „s přenosem"

!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

ALGORITMUS NENÍ

„vysvětlit něco, co známe,
někomu, kdo to taky zná“

!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Příklad:

Eukleidův algoritmus

Úloha: Najít největšího společného dělitele dvou přirozených čísel (NSD(A,B)).

Postup1: Projít všechna čísla <= min(A,B)...

Postup2: (Eukleides)

- Dvojice A,B
- Když A<>B, vezmi novou dvojici: **to menší** a **rozdíl**
- Když A=B, je A=B=NSD původní dvojice

Příklad:

NSD a_1, a_2, … a_n

Postup:

- Pro k=2, …, n označ D_k = NSD a_1, a_2, … a_k
- D_2 = NSD(a_1, a_2)
- D_{i+1} = NSD(D_i, a_{i+1})

Příklad:

Úloha: Nalezněte nejkratší cestu šachovým koněm z jednoho pole (třeba D3) na všechna ostatní pole.

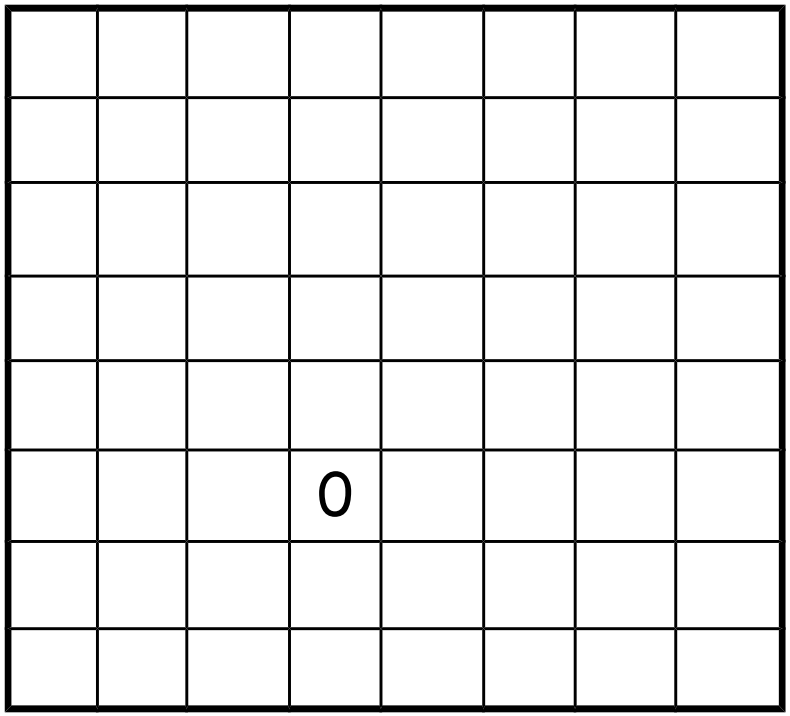
Postup:

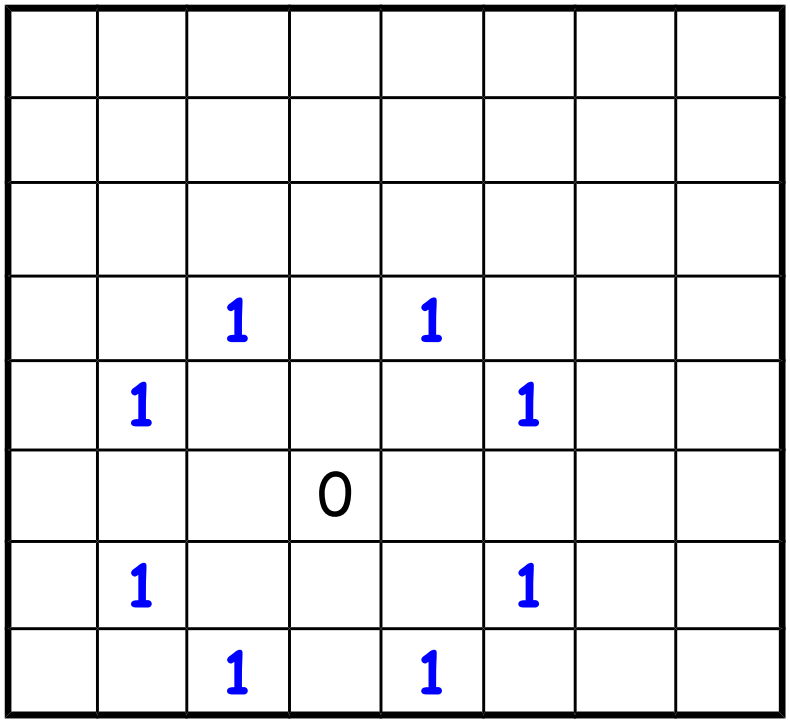
0. Na startovní pole zapiš číslo 0
1. Všechna **dosud neoznačená** pole dostupná jedním tahem z pole označeného 0 označ 1
2. Všechna **dosud neoznačená** pole dostupná jedním tahem z pole označeného 1 označ 2
3. … …… 2 …… 3

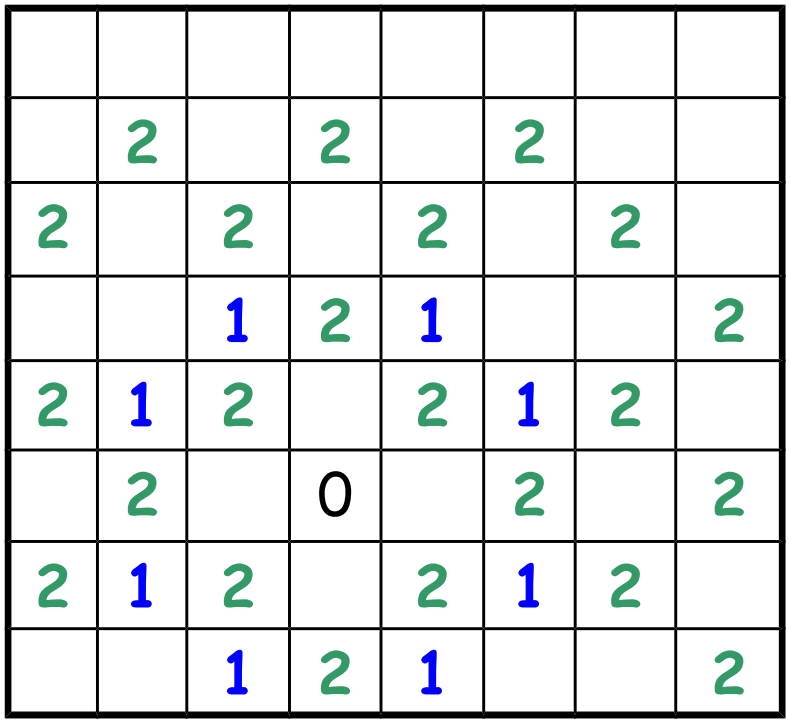
   … **atd.**

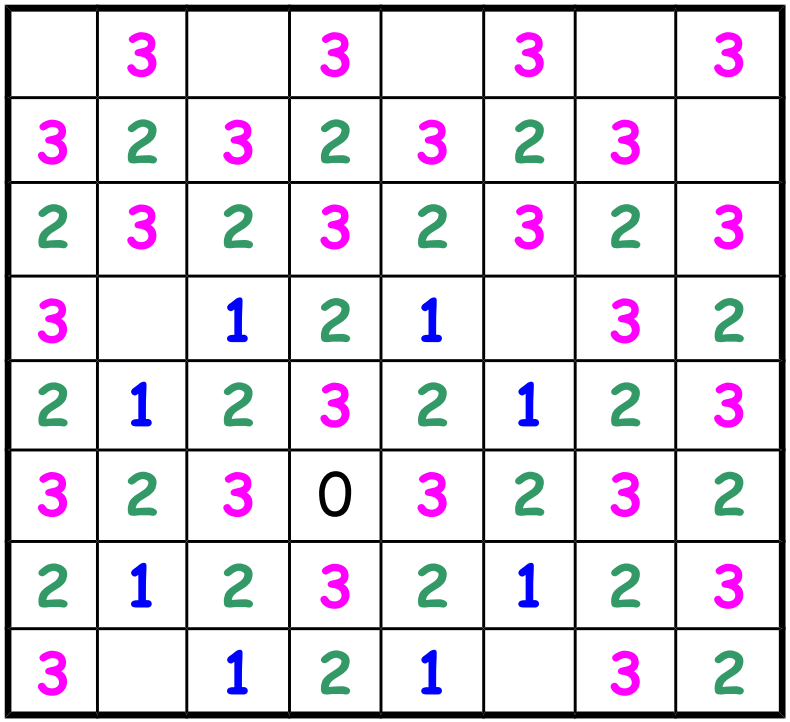
Nalezení cesty pak proveď „odzadu“:

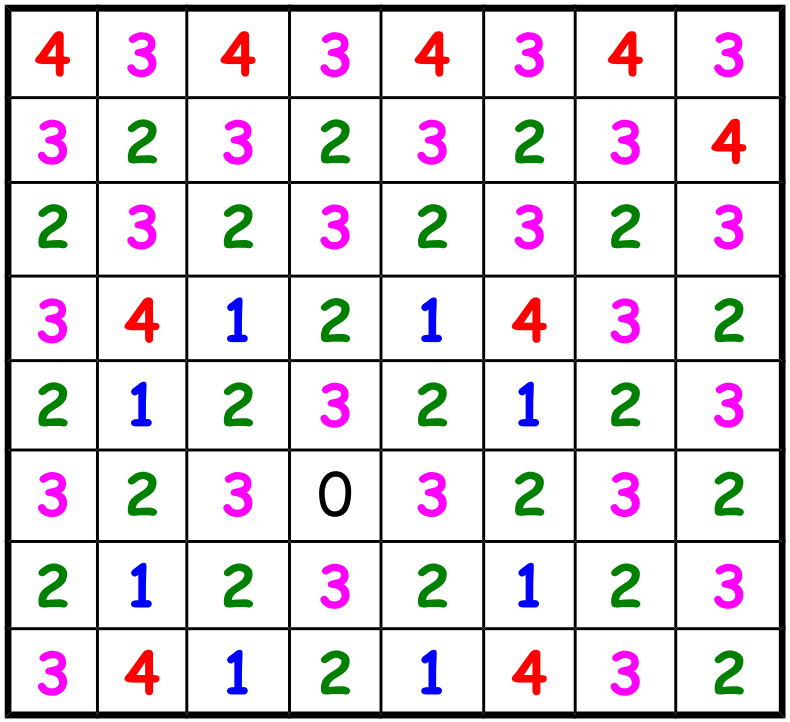
Je-li cílové políčko označené N, hledáme políčko dostupné z něj jedním tahem a označené N-1 … až k políčku 0.


0


1
1
1
1
0
1
1
1
1


2 2 2 2
2 2 2 2
1 2 1 2
2 1 2 2 1 2
2 0 2 2
2 1 2 2 1 2
1 2 1 2


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



„Algoritmus vlny"

Algoritmus

Posloupnost konečného počtu elementárních kroků vedoucí k vyřešení daného typu úloh

/Encyklopedický slovník/

Charakteristické vlastnosti:

- **hromadnost** (vstupní data, výstupní data)...daného typu úloh...
- **výsledek lze získat zcela mechanicky**...elementárních kroků...
- **konečnost**...konečného počtu...

bw_mapa.png

Příklad: (Theseus, Ariadna a Minotaurus)

Úloha:

- určitě najít Minotaura (je-li tam)
- po (ne)nalezení se bez bloudění vrátit

Označme chodby:

- červená prošel 2x (značky fixem)
- žlutá prošel 1x (natažená nit)
- zelená neprošel (ostatní)

Postup:

- Hledání začíná u Ariadny
- V každé místnosti postupuj odpředu podle tabulky:

1. je tu Minotaurus => STOP, našel
2. vede tu > 1 žlutá => zpět, motej nit, žlutá → červená
3. vede tu > 0 zelená => vpřed, rozmotávej, zelená -> žlutá
4. je tu Ariadna => STOP, Minotaurus neexistuje
5. je tu 1 žlutá => zpět, motej nit, žlutá → červená

Vlastnosti tohoto algoritmu:

a) vždy existuje nějaké pravidlo

b) skončí

c) cesta zpět se nebude křížit

d) nit neustále určuje cestu zpět

e) u Ariadny se zastaví,

jedině když neexistuje cesta k Minotaurovi

„Algoritmus prohledávání s návratem" (backtracking)

Na každém rozcestí, na které přijdeš, zjisti všechny možné cesty
a jednu z nich si vyber.

Takto postupuj, dokud to jde nebo dokud se nedostaneš do situace,
ve které jsi už byl.

Nemůžeš-li dál, vrať se na poslední rozcestí, kde ses rozhodoval –
- a rozhodni se jinak.

Ověřování správnosti algoritmu

Neexistuje universální metoda zaručující úspěch !

KOREKTNOST

Nebyla opomenuta žádná z možností

ČÁSTEČNÁ SPRÁVNOST

Skončí-li, dá dobrý výsledek.

KONEČNOST

Pro všechna přípustná data skončí.

Dokazování konečnosti

Stačí najít způsob,
jak každý stav výpočtu ohodnotit přirozeným číslem
a ukázat, že provedením jednoho kroku algoritmu
se tato hodnota zmenší.

Invariant

…je tvrzení, které platí po celou dobu výpočtu.

Kromě netypických výjimek
k ověření správnosti NIKDY NESTAČÍ
provést konečný počet výpočtů !

=> zkušební výpočty nejsou **pokus o důkaz**, ale **pokus o vyvrácení**.

Úloha: Zjistěte, zda číslo N je prvočíslo

Postup (Čínský algoritmus):

Zjisti, zda N je dělitelem čísla 2^N-2.

- Je-li, N je prvočíslo.
- Není-li, N není prvočíslo.

Příklad:

N=5 $2^5-2=30$ => 5 je prvočíslo

N=9 $2^9-2=510$ => 9 není prvočíslo

Úloha: Zjistěte, zda číslo N je prvočíslo

Postup (Čínský algoritmus):

Zjisti, zda N je dělitelem čísla 2^N-2.

↳Je-li, N je prvočíslo.

↳Není-li, N není prvočíslo.

Příklad:

N=5 $2^5-2=30$ => 5 je prvočíslo

N=9 $2^9-2=510$ => 9 není prvočíslo

Pro N=341 selže!

(Ten algoritmus je správný,
omezíme-li množinu přípustných dat na čísla <= 340.)

Programování

= Popisování složitějších algoritmických akcí
pomocí akcí jednodušších.

Příklad: Výpočet druhé mocniny přirozeného čísla
Záleží na tom, jaké jednodušší akce můžeme použít.

Verse 1: Umocni dané číslo na druhou.

Verse 2: Vynásob dané číslo sebou samým.

Verse 3: Sečti dané číslo tolikrát, kolik je samo.

Verse 4: Označ dané číslo N. Sečti N sčítanců rovných N.

Verse 5: … nejdříve nový pojem:

PROMĚNNÁ ……. možnost ukládat a vybírat mezivýsledky
Proměnné jsou pojmenovávány pomocí **IDENTIFIKÁTORŮ**,
obvykle složených z písmen a číslic, musí začínat písmenem.

Verse 5:

N, Počet a Suma jsou proměnné pro celá čísla.

1. Přečti do N hodnotu ze vstupu.
2. Do Suma dosaď 0.
3. Do Počet dosaď N
4. Dokud je hodnota proměnné Počet větší než 0, opakuj akce
   4.1 K hodnotě Suma přičti hodnotu proměnné N
   4.2 Hodnotu proměnné Počet zmenši o 1
5. Vypiš hodnotu proměnné Suma.

DEKLARACE PROMĚNNÝCH

= seznam proměnných a určení jejich **typů**

N, Pocet, Suma: integer

int N; int Pocet; int Suma;

Krušina, sedlák (baryton), Ludmila, jeho žena (soprán)

Jazyky pro zápis algoritmů (programovací jazyky).

PŘIŘAZOVACÍ PŘÍKAZ ukládá hodnotu do proměnné

```
x := v
```

```
x := N-7/2
y := y+1
```

PŘÍKAZ VSTUPU čte hodnotu ze vstupu a uloží ji do proměnné

```
read( x )
```

```
read( N )
read( A, B, C )
```

PŘÍKAZ VÝSTUPU zapíše hodnotu výrazu do výstupu

```
write( v )
```

```
write( N+1 )
```

Verse 6:

```
N, Pocet, Suma: integer
1. read( N )          { vstup }
2. Suma := 0          { počáteční hodnoty }
3. Pocet := N
4. Dokud je Pocet > 0, opakuj akce
  4.1. Suma := Suma + N
  4.2. Pocet := Pocet - 1
5. write( Suma )      { tisk výsledku }
```

KOMENTÁŘE

- zvyšují srozumitelnost
- nemají vliv na význam / běh

V jazyku Pascal: { cokoliv } (* cokoliv *)

Lze je napsat kamkoliv, kde smí být mezera

Řízení běhu programu

= v jakém pořadí se budou jednotlivé kroky/akce/příkazy provádět

Možnosti:

- **příkaz skoku**

  mění pořadí provádění příkazů

- **strukturované příkazy**

  vytvářejí složitější příkazy z jednodušších

PŘÍKAZ SKOKU určuje příští prováděnou instrukci

Ve většině jazyků má tvar

GOTO číslo příkazu

```
N, Pocet, Suma: integer
1: read( N )          { vstup }
2: Suma := 0          { pocatecni hodnoty }
3: Pocet := N
4: je-li Pocet = 0, pak GOTO 8
5: Suma := Suma + N
6: Pocet := Pocet - 1
7: GOTO 4
8: write( Suma )      { tisk vysledku }
```

Poznámka: řádka <-> příkaz

Poznámka: Dva druhy skoků:

- podmíněný
- nepodmíněný

- S těmito příkazy už bychom vystačili
- Když vynecháme deklarace, máme BASIC

STRUKTUROVANÉ PROGRAMOVÁNÍ

Edsger W. Dijkstra: Goto statement considered harmful, 1968

<= potřeba zvládat velké programy,
potřeba dělby práce

čím silnější prostředky máme k dispozici,
s tím větší kázní a obezřetností je musíme používat

nerespektování přirozených struktur
nebo dokonce jejich (násilná) likvidace
se v budoucnu projeví vážnými a nepředvídatelnými
(negativními) důsledky.

STRUKTUROVANÉ PROGRAMOVÁNÍ

vytváření složitějších (strukturovaných) příkazů
skládáním z jednodušších

Systém několika málo konstrukcí,
kterými lze vyjádřit všechny algoritmické konstrukce.
Přitom další rozšiřování už nic nepřidá.

TVRZENÍ

Strukturovaným příkazům rozumíme lépe
než (stejně složitým) příkazům nestrukturovaným.

PŘÍKLAD

↳Ber, dokud dávám!

↳Jestli prší, počkej, až pojede vlak.

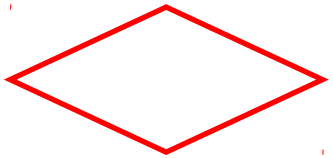
Odbočka:

VÝVOJOVÉ DIAGRAMY

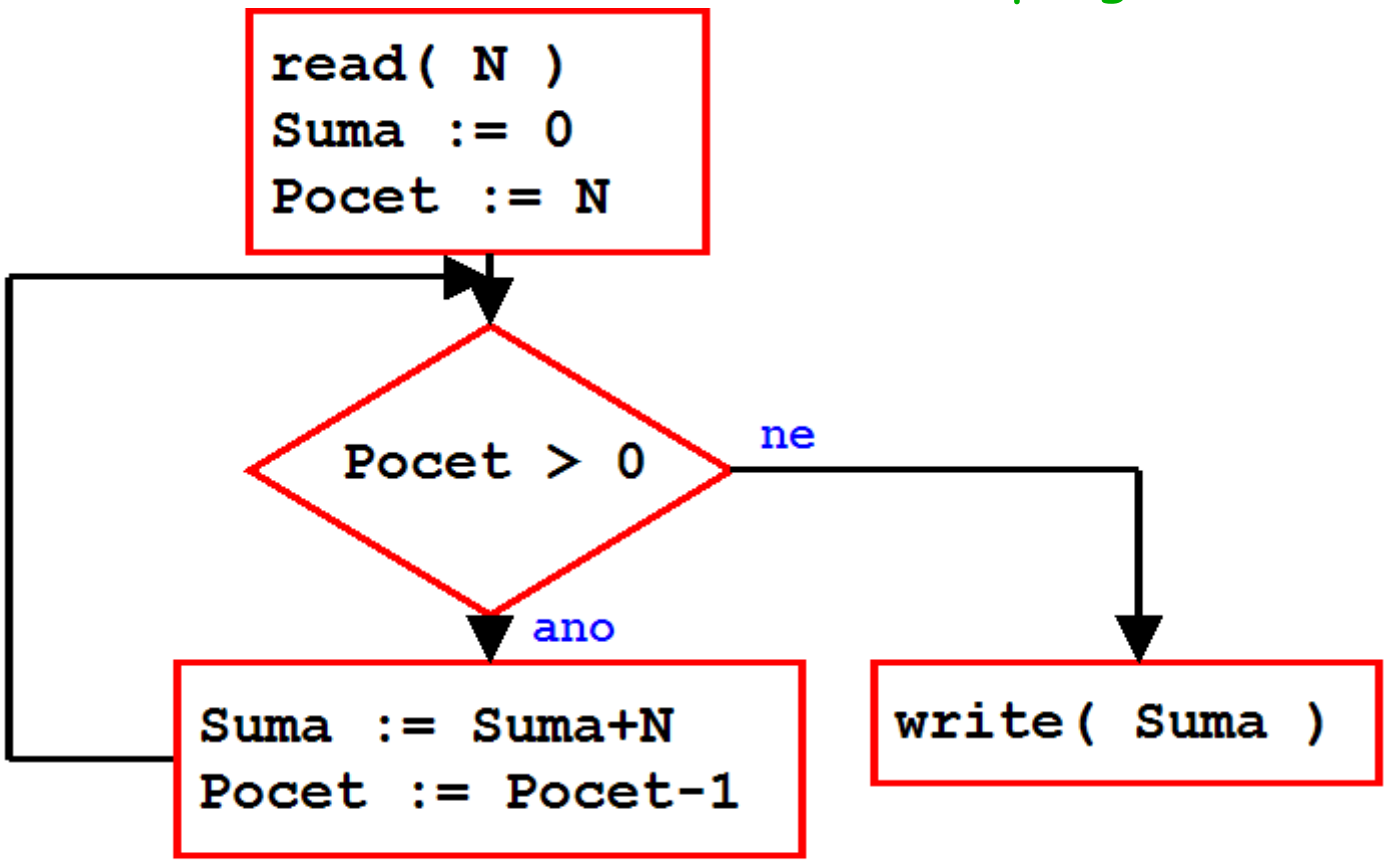
Historicky: Jeden z pokusů, jak překonat složitost programů.

Použijeme pro znázornění konstrukcí strukturovaného programování.

funkční blok
- akce



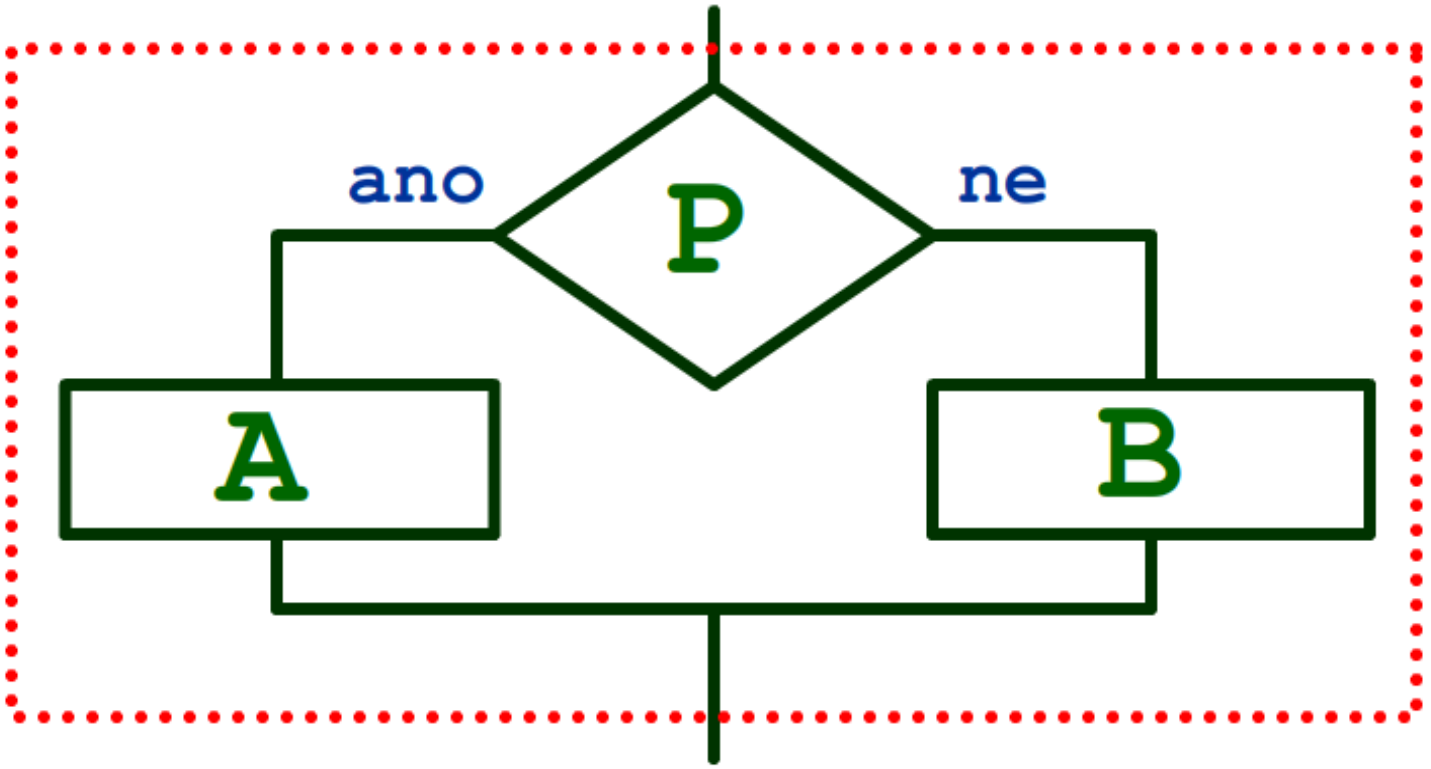
podmínkový blok
- větvení



Podmíněný příkaz

úplný:

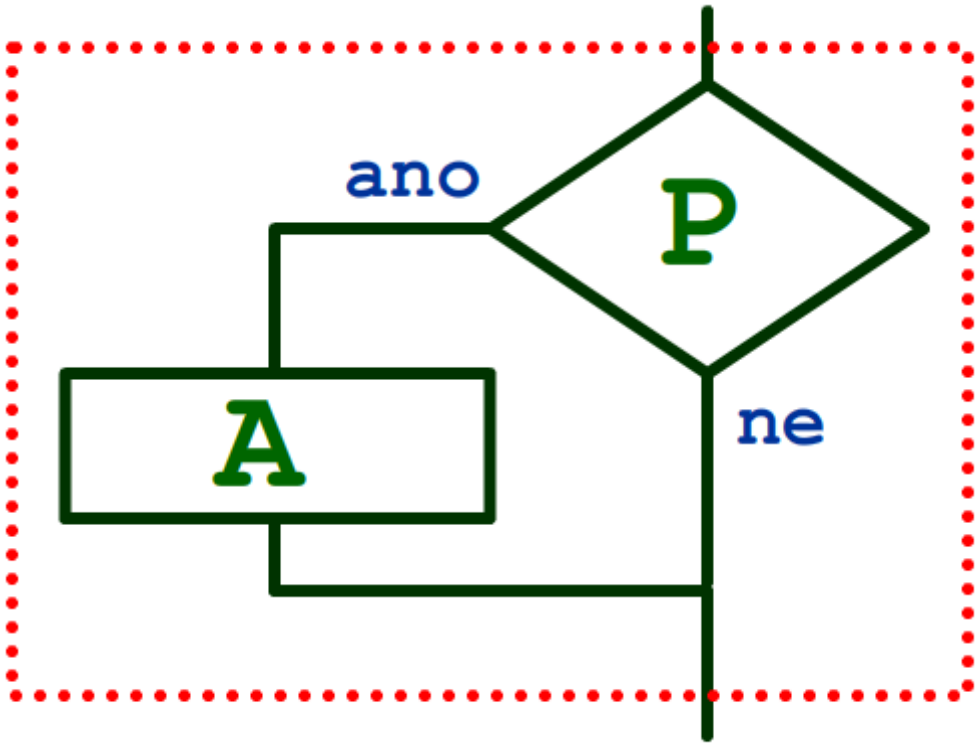
```
if P then A else B
```



Podmíněný příkaz

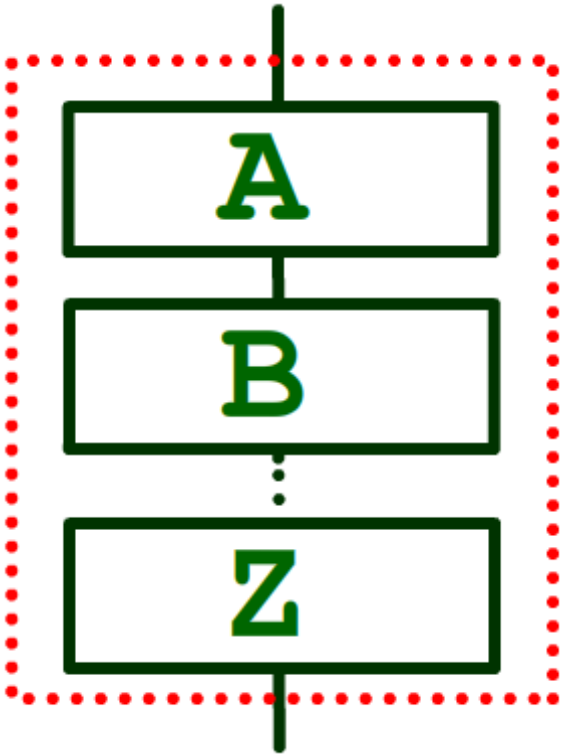
neúplný:

```
if P then A
```



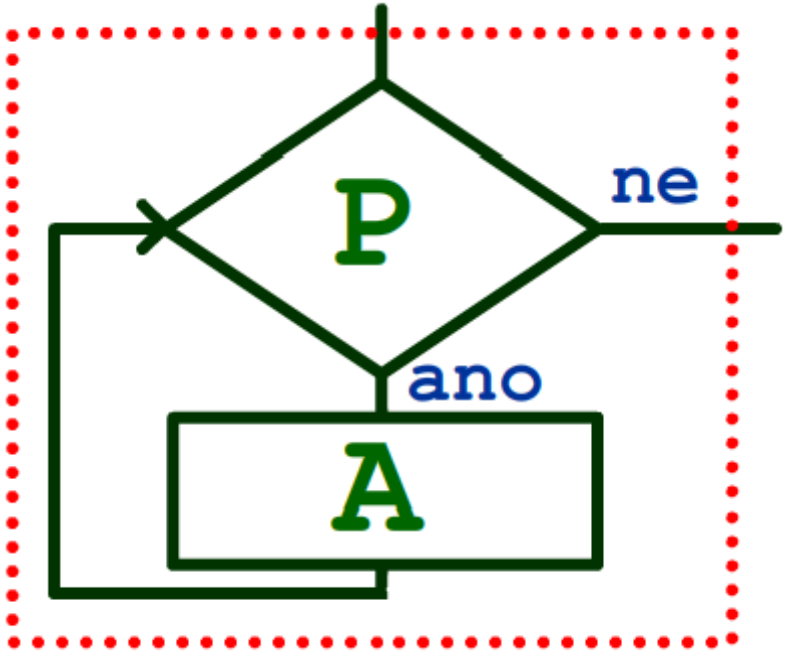
Složený příkaz

```
begin A;B;...Z end
```



Příkazy cyklu

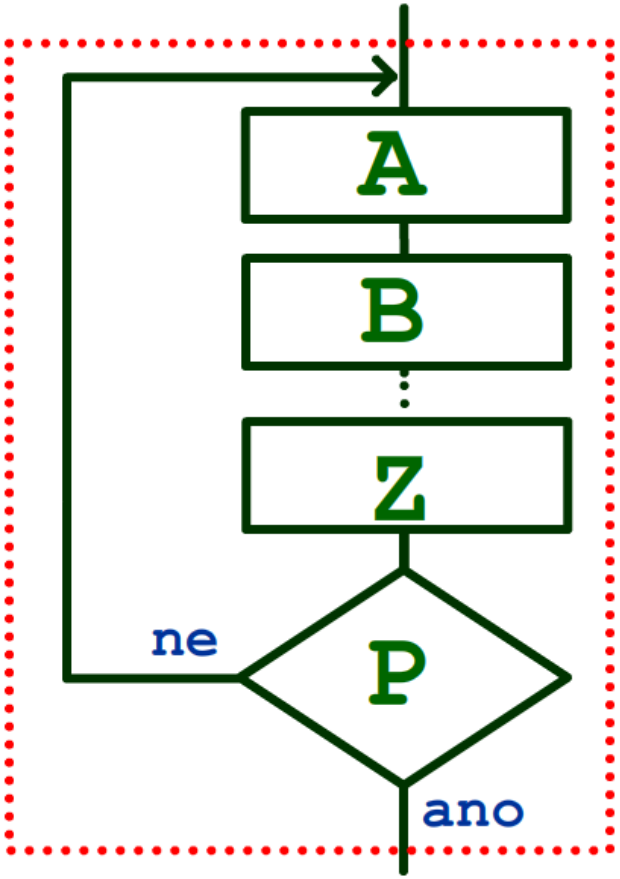
```
while P do A
```



kontrola PŘED provedením těla cyklu

Příkazy cyklu

```
repeat A;B;...Z until P
```



kontrola PO provedení těla cyklu


Programování I,II
Co je programování
algoritmus
zápis algoritmu
proměnná
elementární operace
řízení běhu programu
skok
strukturované programování
programovací jazyky
dokazování správnosti
korektnost
částečná správnost
konečnost
složitost
algoritmy
základní
sčítání dvou čísel v desítkové soustavě
Euklidův
algoritmus vlny
backtracking
Hornerovo schema
vyhledávání v poli
Eratosthenovo síto
prohledávání do hloubky x do šířky
programování her
heuristika