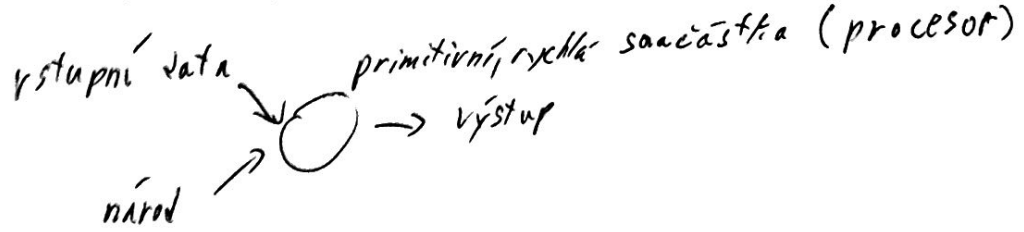
Programování pro fyziky I

- web `atf.mff.cuni.cz/~ledrinka`
  - poznámky k přednášce



- programovací jazyk Pascal

Algoritmus

= návod jak získat výsledek
- pomocí elementárních kroků (víme že nebudou ~~to~~ trvat dlouho)
- konečný počet kroků
- stejné zadání ⇒ stejný výsledek

Euklidův algoritmus

- 2 úsečky, jak zkonstruovat takovou
  = jak najít největší společný dělitel dvou celých čísel
- pokud úsečka AB neměří CD ...
- nalezneme tak, že odečítáme menší od většího

Národ č. 1 = Kameny
Národ č. 2 = Napsat čísla na tabulky

Programovací jazyk

```
program euki;
var a,b : integer;
begin
   a := 1998;
   b := 2916;
   while a <> b do
      if b > a then b := b - a else a := a - b;
   writeln(a);
end.
```

- žádné číslování kroků
  - jednoduchý text, slova, čísla, symboly

- Program : a) deklarace
  b) příkazy

- je algoritmus správně? ⇒ složité
- Komentáře { .... }

{ b ≠ 0 } a := a and b { ∃m: nové a + b = původní a }

Lazarus

- www.lazarus-ide.org
- prostředí pro výuku v Pascalu , koncovka .lpr

Soubor nový → Jednoduchý program

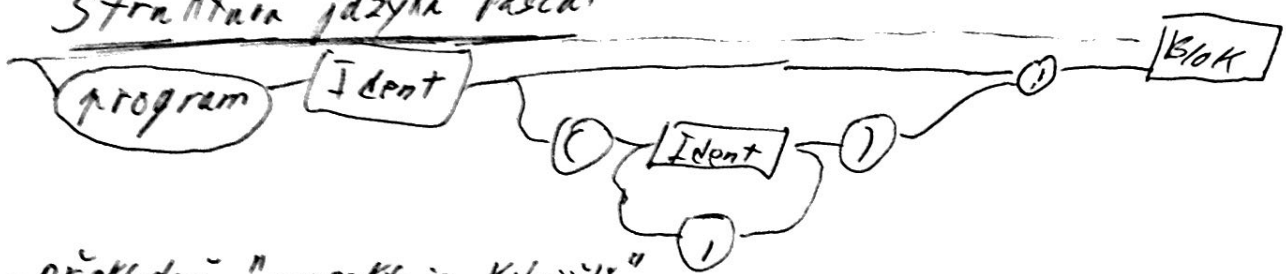
```
var a,b :integer;
```

```
var a: integer;
    b: integer;
```

```
var a: integer;
var b: integer;
```

- vstupní data psát do programu
  ⇒ překladač kontroluje

Struktura jazyka Pascal

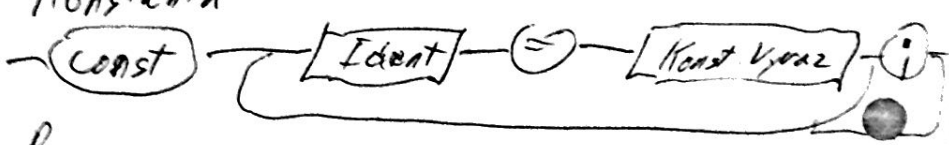


- překladač "respektuje kolejiště"

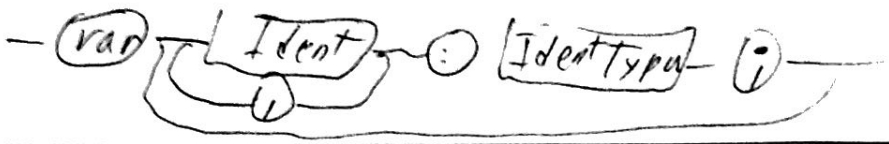
Blok

Deklarační Oddíl

Konstanta



Proměnné

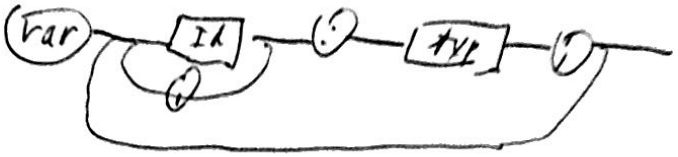


Programování - přednáška II

- začátek: program, identifikátor → blok, tečka
  - identifikátor: - nesmí začínat číslicí

- Blok: - deklarační oddíl { Konstanty, proměnné (+vlastní typy), funkce }
  - Složený příkaz

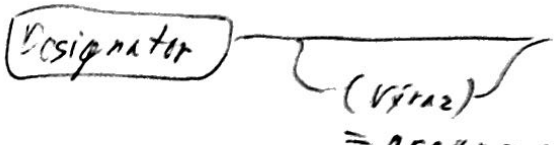
- Konstanty: `const, Horni_mez = 12;`
- Proměnné:



- Složený příkaz:

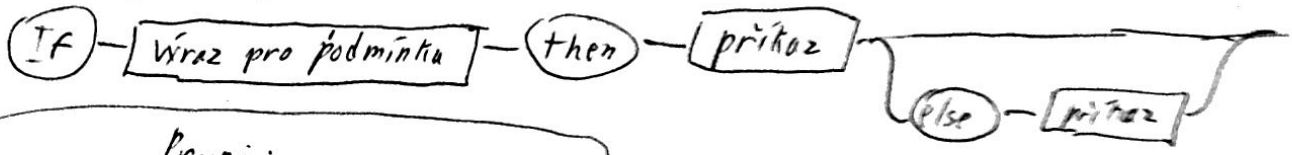
- Příkaz:

a) jednoduché { - volání procedury



- přiřazovací příkaz `x := 1;`

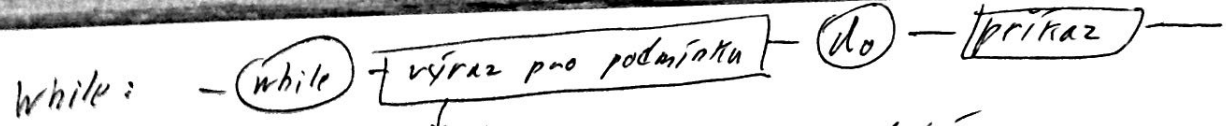
b) s strukturované:
- If, while, Repeat, For



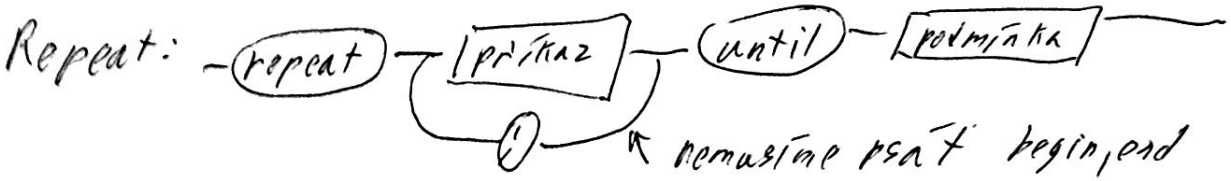

```
program Prvni;
  var a,b: integer;

begin
  a:=1;
  b:=2;
  if a>b then writeln('Hello');

  Readln;
end;
```

"Krátkozraký": ten. if a>b then
if a>10 b then ...

patří → else
K bližšímu else

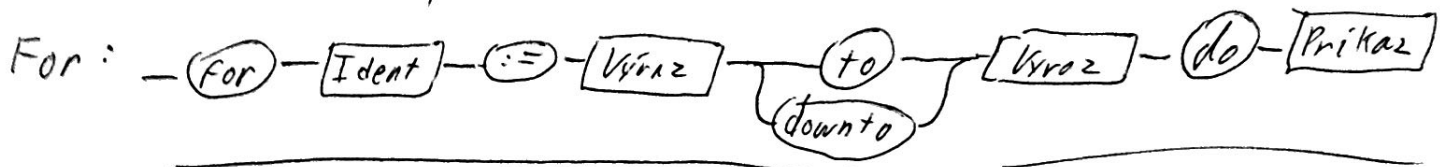
While: 

- na konci: výraz pro podmínku neplatí
- nejsou tady skoky (pozor na break!)

Repeat: 

↖ nemusíme psát begin, end

- vykoná příkaz, potom zkontroluje podmínku (rozdíl oproti while)

For: 


```
for a:=100 to 200 do begin
    soucet := soucet + i;
end;
```

pozn.: F2 přejmenuje proměnnou

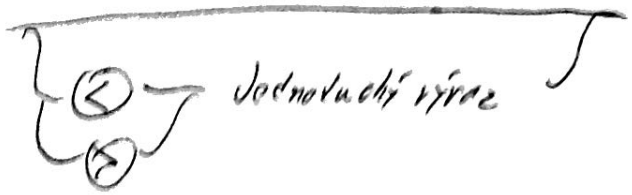
lze:

```
i := 1;
while (i<=N) do begin
    soucet := soucet + i;
    i := i + 1;
end;
```

- "cyklus" = inicializace + příkazy + podmínky
  (~~řídicí~~) (pozor na podmínky před for, while, repeat, ...)

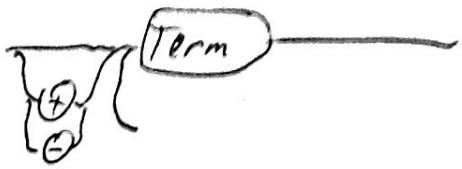
Výrazy:	
1) $\frac{x-1}{x+1} \mapsto y$	`y:=(x-1)/(x+1);`
2) $\frac{1}{2x}$	`1/(2*x)`, lze `1/2/x` (pozor / je 10× pomalejší než násob.)
3) $\frac{x-vt}{\sqrt{1-v^2/c^2}}$	`(x-v*t)/sqrt(1-sqr(v/c))`
4) $2\sin(x)\cos(x)$	`2*sin(x)*cos(x)`
5) $x \neq y$	`x<>y`

• Výraz: Jednoduchý výraz

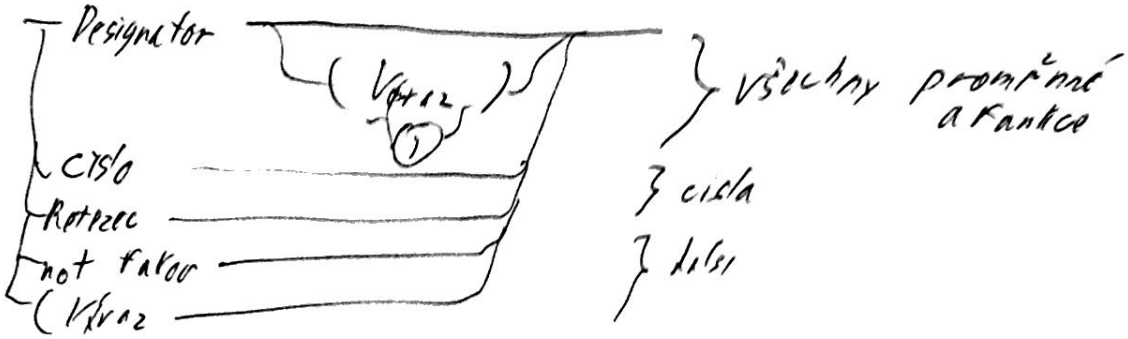


a > b > c | (a > b) and (b > c)

- Jednoduchý výraz:



- Faktor:

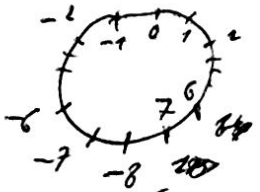


• násobení celých čísel => může přetéct
  - v dělení je výsledek vždy reálné číslo (v Pascalu)
  => psát ne 1/2 ale 1/2.0

• Round, Trunc - vrací celé číslo
• Abs, Sqr - vrací int pro integers

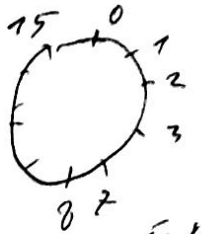
Programování - přednáška III

- celá čísla v počítači:



znaménka

NEBO



bez znaménka

- integer
- byte: 8 bitů, 2^8 komb., bez znaménka
- error {[R+}
- typy: shortint, smallint, longint, ..., int64, byte, word, longword

Procedury a funkce

cíl: nsd(a,b) → vrátí nsd(a,b)

```
function nsd(a,b: integer): integer;
begin ...
  nsd:=a;
end;
```

"formuláře"  parametry (mají typ!)

výsledek (pouze funkce) i procedury mají "side effect"

- algoritmus

a → 2
b = q → 7 → 5 → 3 → 1 ← q mod 7

(nejsou vrany po deklaraci)

```
function nsd(a,b: integer): integer;
var c: integer;
begin
  repeat
    c:=a mod b;
    a:=b;
    b:=c;
  until c=0;
  nsd:=a;
end;
```

← lokální proměnné

pozn. 10 mod 40 → 10

- "Lze psát" lokální funkce
  - deklarované pod funkcí
  - řídí parametry dané funkce i lokální proměnné

pozn. lze:

```
(a: integer; b: integer): integer;
```

• parametry funkcí: `funkce(var x: real): real;`

! odkaz na proměnnou !

```
var i,j: integer;
procedure Prohod (var a,b:integer);
var k: integer;
begin
  k := a;
  a := b;
  b := k;
end;
```

Programování - přednáška IV

zápis programu → strojový kód

- ~~lokální~~ × globální proměnné ← existují po celou dobu programu
  - jejich identifikátor si snadno zapleteme
  - na začátku programu se vynulují

×

lokální proměnné
- existují jen po dobu běhu funkce
- znovu vznikají a zanikají při každém volání
- nenalují se
- nepletou se s ostatními lokálními proměnnými

- Preferovat lokální (pokud jsou možné obě)!

- komentáře, rozumná jména proměnných
- ctrl + space → nabídne doplnění identifikátora

Výpočty s reálnými čísly

```
function ArcSin(sinus: real): real;
 var cosinus: real;
begin
  cosinus := sqrt(1-sqr(sinus));
  if cosinus=0 then if sinus>0 then ArcSin := Pi/2;
                                else ArcSin := -Pi/2;
               else ArcSin := ArcTan(sinus/cosinus);
end;
```

$y = tg(x)$ $w = \sin y$ $w \in \langle -1;1 \rangle$

$\arctan(x) = x$ $\arg y = \frac{w}{\sqrt{1-w^2}}$

Řady

- délka elipsy $L = 2\pi a \left[1 - \sum_{k=1}^{\infty} \left(\frac{1 \cdot 3 \cdot 5 \dots (2k-1)}{2 \cdot 4 \cdot 6 \dots 2k} \right)^2 \frac{\varepsilon^{2k}}{2k-1} \right]$ vyšetřovaný $\varepsilon = \frac{e}{a}$

$k=1 \quad \left[\left(\frac{1}{2}\right)^2 \frac{\varepsilon^2}{1} + \left(\frac{1 \cdot 3}{2 \cdot 4}\right)^2 \frac{\varepsilon^4}{3} + \left(\frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}\right)^2 \frac{\varepsilon^6}{5} + \dots \right]$

- další členy vypočítáme pomocí předchozích:

"f_2" $= r = \left(\frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}\right)^2$ } další člen $= \left(\frac{r \cdot (2k+1)}{(2k+2)}\right)^2 \cdot \frac{\text{moc\_e} \cdot \varepsilon^2}{7}$

$\text{moc\_e} = \varepsilon^6$ }

$r \cdot \left(\frac{9}{10}\right)^2 \cdot \frac{\text{moc\_e} \cdot \varepsilon^2}{7}$

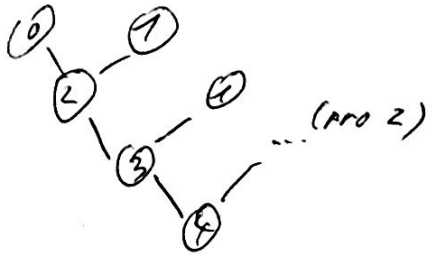
Rekurentní vztahy

- faktoriál: $f(0) = 1$
  $f(k) = k \cdot f(k-1)$
- lze volat funkci f ve funkci f

```
function faktorial(n: real): real;
begin
  if n=0 then faktorial := 1
         else faktorial := n . faktorial(n-1);
end;
```

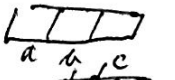
• musí být argument real / integer?

- Fibonacci $f(0) = 0$ 0 1 1 2 3 5 8 13 ...
  $f(1) = 1$
  $f(n) = f(n-1) + f(n-2)$

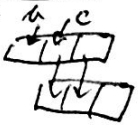


! příliš mnoho sčítání !
⇒ neaplikujeme rekurzi

⇒ "použití šablony" 0 1 1 2 3 5 8 13 ...



"tříbodová rekurze" } cyklus s 3 proměnnými



Př: $P_0(x) = 1$
$P_1(x) = x$
$P_n(x) = \frac{2n-1}{n} x P_{n-1}(x) - \frac{(n-1)}{n} P_{n-2}(x)$

Programování - přednáška V

Reálná čísla

w celé číslo [00 ... 0111] = 7

šířka ... určuje max. hodnotu

a) reálné číslo ⌈ spousta cifer
⌊ velká/malá čísla

→ "real" zabere 64 bit

- potřebujeme "inženýrský zápis" (2,4) 10^{6} ← exponent

(2,4) ← mantisa

$$\Rightarrow \boxed{x = s \cdot \frac{m}{2^e}}$$

s ... znaménko (1 bit)
m ... celé číslo (53 bit)
e ... celá mocnina (11 bit)

- násobení jednoduché
- sčítání horší

$\Rightarrow$ [+, - nejsou bezstrátové]

Př.: $1 - \sqrt{1+x^2}$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

- problém při vykreslování

$x_n = x_{n-1} + \Delta x$ } používejme for cyklus

GNUPLOT

[plot 'data.txt' with lines], [plot 'data.txt' u 1:3]

set style data lines

Př.: $\sum_{k=1}^{\infty} \frac{\sin(k^2 x)}{k^2}$ ← spojitá, pokud jde k do ∞ (ale skoro nikde nemá derivaci)

$\sum_{k=1}^{n} \frac{\sin(k^2 x)}{k^2}$

Př.: $\sin(\pi) \neq 0$, π je zaokrouhlené

Př.: $x^4 - 3x^3 + 2x^2 - 6x + 4$

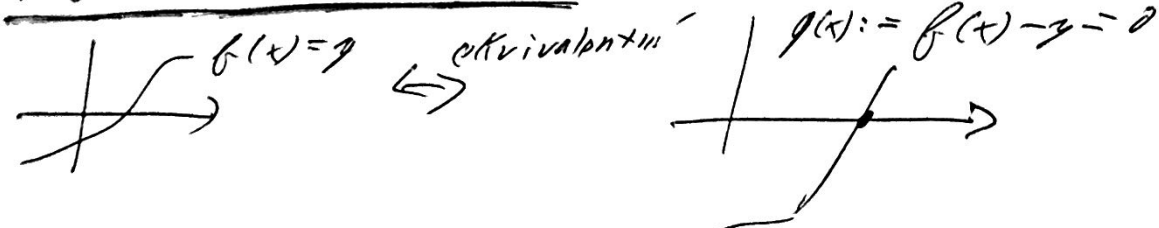
- správně: Hornerovo schéma

→ $(((((x-3).x + 2).x) - 6).x + 4$

[POZN.: Kontrolovat interval platnosti!]

- Jak spočítat $\sqrt{z}$?
  $x^2 - a = 0$, najdi $x > 0$
- nebo $\sin x = y$, najdi x
- nebo $x \cdot e^x = y$, najdi x
- nebo: - rozdíl  na stejné ploše

Hledání kořenů funkcí



- 1. možnost: vzít všechny reálná čísla
- 2. možnost: - najít 2 body s opačnými znaménky
  = Metoda půlení intervalu
    - najdi a, b: $f(a) \cdot f(b) < 0$
    - opakovaně půlím interval
      - v půlce spočtu funkční hodnotu
      - volím interval tak abych neztratil kořen

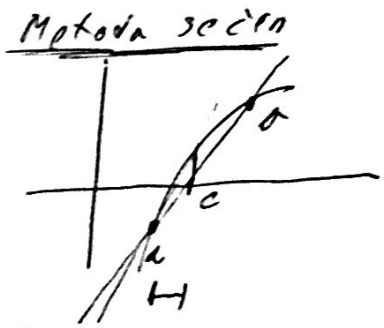
(- předpoklad: f je spojitá)

- Jak najít x_2 a, b ?

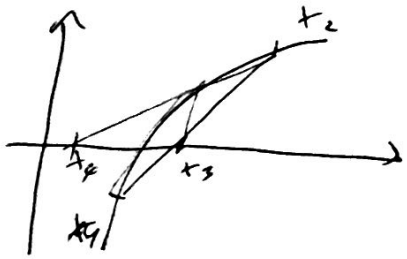

```
repeat
  c := (a+b)/2;
  if f(a).f(c) > 0 then a := c
                   else b := c;
until b-a < eps;
```

(V půlení zmenší $|a-b|$ na $\frac{1}{2}$
=> počet operací je zhruba $\log_2 N$)

Metoda sečen

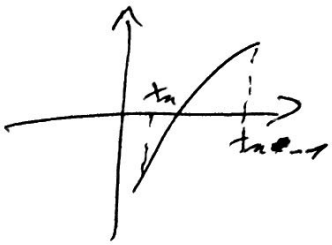


! pomalejší, ~~ale~~ bezpečné

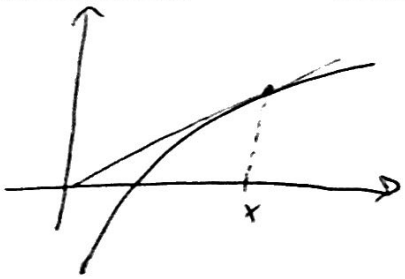


! rychlejší, můžem se ztratit !

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}}$$



Newtonova metoda (metoda tečen)



$$f(x+\Delta x) = f(x) + \Delta x \cdot f'(x)$$

první diferenciál funkce
= rovnice tečny ke grafu funkce

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
repeat
   dx := -f(x)/df(x);
   x := x + dx
until abs(dx) < eps;
```

Programování – přednáška VI

Typ char

```
var c : char;
for c := 'A' to 'Z' do
```

funkce: `ord(c)` → vrátí celé číslo
`char(i)` → vrátí char

- ordinální typy: – celá čísla
  – boolean
  – char
- definují se funkce: `succ(i)` / `pred(i)`

Typ interval

```
type poradoveCislo = 1..200;
var zakaznik : poradoveCislo;
```

Výčtový typ

```
type funkce = (Radovy, clenVydora, predseda);
```

- lze pouze přiřadit + pred, succ
  porovnat

(pozn.: Množiny : set of .... (ordinální typ))

Pole

```
const Dim = 3;
type tvektor3 = array [1..Dim] of reali
var a : tvektor3;
```

- Pic: – fotka, M×N pixelů
- pole `p2 := p1;` (pokud jsou stejného typu, kompatibilita!)
- `soucet(p2);` (jako argument
- `p1[1] := 7;` – přístup k prvku pole

+ const

```
procedure sectiV3(a,b : tvektor3; var c : tvektor);
begin
  c[1] := a[1] + b[1];
  c[2] := a[2] + b[2];
  c[3] := a[3] + b[3];
end;
```

- když připíšeme "const" k argumentům → nic se nekopíruje a nelze změnit hodnoty

Př.: `type tMatice3 = array [1..Dim] of tVektor3;`

`var A : tMatice3;`

a_{12} v matici: $\begin{pmatrix} a_{11} & a_{12} \\ \dots & \end{pmatrix}$

A[1]

platí A[1,2] = A[1][2] => A[1] je první řádek

=> "matice uložené po řádcích"

```
var faktorial : array [0..170] of real;
    Binomial  : array [0..1000, 0..1000] of real;
```

2 prvočíslo => i násobky 2 jsou

3 prvočíslo => násobky atd...

atd...

```
program Sito;
const N = 5 000 000
var Mazdelitel : array [0..N] of boolean
    i,p : integer;

begin
 p := 2;

 repeat
   i := p+p;
   while i <= N do begin
     Mazdelitel[i] := true;
     i := i+p;
   end;
 p := p+1
 while Mazdelitel[p] do p := p+1;

until p*p > N;
end.
```

Programování - přednáška VII

- pole ⊤ celek / individuum
- "off by one"

- algoritmus: JeTam?
  - vstup: pole, hodnota
  - výstup: ano/ne

- funkce

```
low(pole)
high(pole)
```

```
type seznam = array [1..5] of char;

function JeTam (seznam : tSeznam): boolean;
                znak : char
begin JeTam:=false;
for i:= low (seznam) to high (seznam) do begin
   if (seznam[i] = znak) then begin
     JeTam := true;
   end;
end;
end
```

- Předávání pole jako argumentu funkce:

```
function JeTam ( const pole : tSeznam ) : boolean;
```

  - const ⇒ může číst hodnoty, ale nemůže je měnit
  - nepředává hodnoty, ale odkaz na dané pole

- Klíčové slovo exit

```
if (nasel) then begin
    JeTam:= true;
    exit;
end;
```

exit ← skončí další loop v dané funkci

- Proměnná délka pole:

```
function JeTam (const pole : array of char; znak : char): boolean;
```

⇒ index prvního prvku je 0

Typ záznam

```
type tVektor = record
     x, y, z : real;
end;
var a, x : tVektor;
```

```
type tProdukt = record
  Chzacka : array [1..2] of char;
  A, Z : integer;
  PoloRo : real;
  ...
end;
var produkt : tProdukt;
```

← deklarace proměnné

```
produkt.A := -1;
```

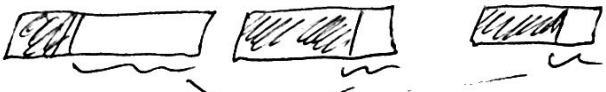
← přístup k hodnotě

POZN:

- komplexní čísla:

```
z.Re , z.Im
```

- Packed record
  - record má různě dlouhé prvky
  - počítač provede alignment (zarovnání) => vloží se mezery mezi proměnné



mezery

- packed:

...

- Specialita Pascala (a Javascriptu)

```
with z do
begin
   Re := 0;
   Im := 1;
end;
```

původně: { číslo `p[i].A` `p[i].B` ; psát `with ...`

- Příkaz Case

```
Case n of
  0 : f := 1;
  1 : f := x;
  ...
  4 : f := x*x;
else
  f := exp(ln(x)*n)
end;
```

- pouze na ordinální typy
- také lze

```
'A'..'Z' : f := x
```

- Inicializace proměnných

```
var pt: ^seznam = 'abcde';
```

```
const tabulkaFaktorialu : array[0..5] of integer = (1,1,2,6,24,120);
```

const – víme že se hodnoty nebudou měnit

```
var kratky : ^SeznamCisel = (pocet:2; cisla:(1,2,3,4));
```

- Variantní záznam

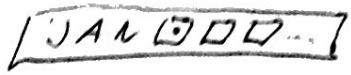
```
type ttvar = (tobdelnik, ttrojah., tkruh);
type atvar = record
               ...
             case druh: ttvar of
               tobdelnik:
               ttrojah:
               tkruh:
           ...
```

Programování – přednáška VIII

- Pole < rozsah / typ prvku
- Pole jako seznam, např: obrácený řetězec
  - rozměry pole ... nejhorší (nejlepší) případ

Jak určit délku?

1. zarážka



- napíšeme funkce pro důležité operace
- obvykle:

```
while not (zarážka)
begin
...
end;
```

(var cmd : array[1..max] of char;)

```
cmd := 'ping';
```

← automaticky přivádí zarážku

2. alternativa: uložit délku řetězce mimo

Pole + info o délce (lze uložit jako

(POZN: preferuje tuto variantu)

```
type THodnoty = record
       Pocet : integer
       Data : array[1..Max] of real;
end;
```

```
with konkretni_seznam do
    for i := low(pole) to delka do
```

3. Dynamická pole

```
var Hodnoty : array of real;
```

```
SetLength(Hodnoty, N);
```

- začíná na indexu 0
  končí na indexu N−1

(POZN: nepoužívat)

! nevíme "kde bydlí" !

- setlength při zvětšování velikosti pole někdy musí překopírovat celý obsah
- nové prvky mají nedefinované hodnoty

- pole jako argument:

```
function Cetam (const pole : array of char): real;
  ...
  for i := low(pole) to high(pole) do
  ...
```

  - pole začíná od 0
  - high vrací N-1 (počet prvků - 1) (low dá nula)

- lze psát:

```
writeln (rozsahHodnot ([0,3,4,-2,7,10]));
```

POZN

```
{$RANGECHECKS ON}
```

Řetězce

- konstanta, např. ve writeln
- datový typ: `var s : string;` `chrabr : array of char;`

```
S := 'abc';
t := s;
writeln(s);
```

```
t := copy (s, 2, 3);
```
- začíná 2. znakem
- kopíruje 3 znaky

```
Delete (s, 1, 4);
```

```
t := '..' + s;
```

```
Insert ('xx', t, 2)
```

```
Pos ('23', t)
```
- nalezne polohu řetězce '23' v t

```
Length (t)
```

POZN: nepracovat s textem v Pascalu
- použít regular expression