ALGORITMY A PROGRAMOVÁNÍ (ÚVOD)

aneb pohled do programátorské kuchyně

Algoritmy a algoritmizace

Algoritmus je postup nebo návod, jak řešit nějakou libovolnou úlohu (např. kuchařka, návod na použití, obsluhu, matematický výpočet ap.). Algoritmus musí být srozumitelný – používáme pouze takové kroky, které vykonavatel algoritmu (člověk, stroj, procesor) zná a umí je vykonat (např. „Přijeď večerním rychlíkem do Ostravy, čekám v Porubě na nádraží.“ – instrukce pro dospělého x dítě).

Algoritmus musí být také ***jednoznačný*** - při nejednoznačnosti může být provedena jiná činnost, než jakou měl na mysli autor algoritmu („Strouhej 3 dny staré housky.“). Nesrozumitelný algoritmus jeho vykonavatel nemůže provést a dá to obvykle najevo (počítač chybovým hlášením, člověk protestem). Při nejednoznačnosti může být provedena jiná činnost, než byla zamýšlena.

Každý algoritmus je řešen pomocí řady elementárních kroků na různých úrovních podrobnosti.

Algoritmus musí být ***konečný*** – k řešení dané úlohy musíme dojít během konečného počtu kroků (jednotlivých elementárních příkazů). Za algoritmus nebudeme považovat takový návod, podle kterého bychom úlohu řešili nekonečně dlouho.

Algoritmus má vlastnost ***hromadnosti*** (obecnosti) – jeden algoritmus je možno použít k řešení velkého množství úloh stejného typu, ale s různými vstupními daty (např. kvadratická rovnice). Pro úlohy řešené na počítači je typické, že se stejné postupy opakují pro různá data.

Řešením úlohy pro daná (přípustná) vstupní data jsou nějaká výstupní data. Před tím než začneme vytvářet nějaký algoritmus řešení dané úlohy, musíme si ujasnit, jaká jsou vstupní data, jaká mají být výstupní data a dále vztahy, kterými jsou vstupní a výstupní data svázána (např. kvadratická rovnice - vstupními daty jsou koeficienty A, B, C, výstupními daty jsou hledané kořeny kvadratické rovnice vztahy tvoří výpočet diskriminantu a další použité vzorce).

Efektivnost nebo také ***složitost*** algoritmu souvisí s tím, kolik operací musíme provést, abychom konkrétní úlohu vyřešili. Zjednodušeně řečeno, čím více operací, tím složitější a tedy méně efektivní algoritmus. Protože provedení každé operace trvá určitou dobu, lze složitost (efektivnost) algoritmu měřit celkovým časem potřebným k vyřešení úlohy. Často se dostáváme do rozporu, zda zvolit přístup, při kterém vytvoříme efektivní (rychlejší) algoritmus za cenu zvýšených nároků na kapacitu paměti, nebo přístup, při němž ušetříme část kapacity paměti za cenu snížení efektivnosti (rychlosti) algoritmu (např. rekurzivní procedury a funkce).

Etapy programátorské práce

1. ***Definice problému***
   - jasná a přesná definice toho, co má program řešit
   - určení vstupních a výstupních údajů
   - specifikace výjimečných situací (například požadované reakce na chybná vstupní data)
   - analýza řešitelnosti
2. ***Nástin řešení***
   - rozklad složitějšího problému na jednodušší podproblémy
   - například výběr vhodné numerické metody
3. ***Sestavení algoritmu***
   - vytváří se algoritmus řešení
   - opakovaně se používá principu rozkladu složitějších problémů na jednodušší problémy
   - navrhují se datové struktury, s nimiž bude program pracovat

4. ***Kódování programu***
   - zápis algoritmu do konkrétního programovacího jazyka
5. ***Ověřování správnosti algoritmu***
   - program, který produkuje nějaké výsledky, ještě nemusí být správný
   - program ověřujeme na vhodné množině zkušebních vstupních dat, pro něž známe správné výsledky
   - chyby:
     a) syntaktické (formální) – objeví se při překladu programu, vznikají chybným zapsáním syntaxe jazyka
     b) sémantické – výsledky z počítače se neshodují se správnými výsledky (algoritmus není v pořádku
     c) logické – objeví se při výpočtu (např. dělení nulou, nekonečný cyklus)

Zápis algoritmů

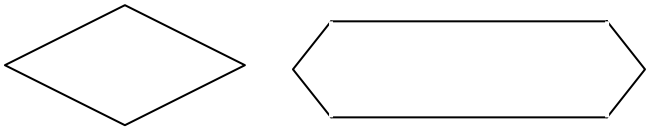
1. ***Přirozený jazyk***.
2. ***Vývojové diagramy*** (definována množina značek, do kterých se zapisuje činnost, která se má v daném místě algoritmu provést).
3. ***Programovací jazyk*** (počítače rozumějí algoritmům zapsaným v některém z programovacích jazyků – Pascal, Delphi, C++, C#, Visual Basic, Java a tisíce dalších).

Značky pro vývojové diagramy

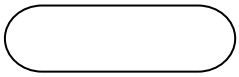
Zpracování – zapisují se akce, které se mají provést

Vstup/výstup – zápis vstupních a výstupních dat

Rozhodování, větvení, přepínání – zapisujeme podmínku, jejíž splnění nebo nesplnění rozhoduje o dalším kroku řízení

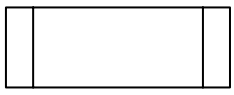


Mezní značka – označuje začátek nebo konec algoritmu



Spojka – přechod na jinou část nebo z jiné části algoritmu

Předem definovaná činnost (specifikovaná jinde a v daném VD nerozpracovávaná, např. podprogram)



Programovací jazyk Pascal

Algoritmus je posloupnost příkazů. Algoritmus zapsaný v programovacím jazyce je ***program***.

Počítač sám o sobě dokáže provádět jen jednoduché aritmetické a logické operace, které musí být zaznamenány ve vnitřní paměti ve dvojkovém kódu (strojovém kódu počítače).

Tvořit programy ve strojovém kódu je činnost velmi pracná a namáhavá. Pro urychlení procesu tvorby programů byly vytvořeny programovací jazyky. Základní smysl programovacího jazyka je v tom, že na jedné straně umožňuje vytvářet programy efektivněji, na druhé straně jsou programy více srozumitelné člověku.

K přeložení programu zapsaného v programovacím jazyce do strojového kódu slouží speciální programy – kompilátory.

Programovací jazyk Pascal byl vytvořen na zásadách strukturovaného programování. Díky tomu dovoluje psát programy srozumitelně a přehledně. Jeho používání ovlivňuje logický přístup k řešení problémů na počítači.

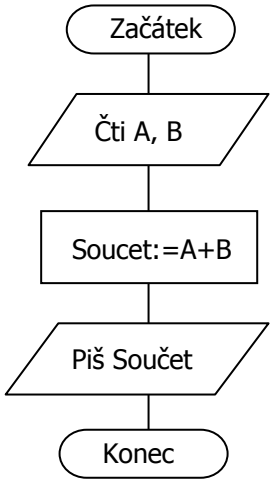
Program v jazyce Pascal je z formálního hlediska text vytvořený podle tzv. syntaktických pravidel jazyka (gramatiky jazyka). Tato pravidla stanoví, jaké symboly lze v textu použít a jak tyto symboly v rámci programu řadit.

Mezi symboly patří: písmena (bez rozlišení malých a velkých písmen), číslice 0 až 9, speciální symboly (+ - * / ( ) = < >...) a tzv. ***klíčová slova*** (například program, var, begin, end,…). Klíčová slova jsou slova vyhrazená pro daný programovací jazyk. Mají přesně stanovený význam a způsob použití. Viz dále.

Př.:

Napište program, který vypočte součet dvou libovolných celých čísel.

Vývojový diagram:



Program:

```
program Secti;
{program secte 2 libovolna cela cisla}
var
  A, B:integer;
  Soucet:real;

begin
write ('Zadej A: ');
readln(A);
write('Zadej B: ');
readln(B);
Soucet:=A+B;
writeln(A,'+',B,'=',Soucet);
end.
```

Program v jazyce Pascal má 3 základní části:

Hlavičku programu – která začíná klíčovým slovem **program** a obsahuje jméno programu (nejlépe je volit jméno souboru, ve kterém je program uložen) a je ukončena středníkem

Deklarační část – kde se deklarují (definují) proměnné, konstanty, procedury, funkce a další prvky jazyka, které se budou v programu používat

Příkazovou část – která popisuje vlastní postup řešení úlohy. Začíná vždy klíčovým slovem **begin**, za kterým následuje posloupnost příkazů oddělovaných středníky a je vždy ukončena klíčovým slovem **end** a tečkou.

program Nazev;

deklarační část

begin

příkazová část

end.

Komentář

Komentář slouží pro zvýšení srozumitelnosti programu. Uvádí se ve složených závorkách {...} nebo (*...*).

V jazyce Pascal není žádný závazný formát pro psaní zdrojového programu. Zdrojový text lze libovolně dělit na řádky a jednotlivé konstrukce lze oddělovat libovolným počtem mezer. Toho využíváme při psaní programu a píšeme ho co nejpřehledněji.

Identifikátor

Identifikátor (pojmenování) může obsahovat písmena, číslice nebo podtržítko. Začíná vždy písmenem. Počítač (překladač) nerozlišuje malá a velká písmena. Délka identifikátoru není omezena, avšak konkrétní překladač může rozeznávat pouze prvních N znaků (obvykle 32 nebo 64).

Nesmí se shodovat s klíčovým slovem. (Klíčová slova jsou slova vyhrazená pro daný programovací jazyk. Mají přesně stanovený význam a způsob použití a jako identifikátory je proto nelze použít – např. PROGRAM, VAR, BEGIN, END a některá další.)

Každý identifikátor musí být nejprve deklarován a teprve potom použit. Identifikátor volíme tak, aby jednoznačně a výstižně vyjadřoval to, co označuje. ***Deklarace*** slouží k zavedení a pojmenování určitých součástí programu.

Např.:

`A, B, Soucet, CelkovySoucet, Koren_rovnice, X1,X2,` apod.

Proměnná

Proměnná je buňka nebo více buněk v paměti, které slouží k uchování nějaké hodnoty (čísla, znaku ap. v závislosti na typu proměnné) po dobu běhu programu. Obsah buňky se během výpočtu může měnit.

Abychom s touto buňkou v paměti mohli pracovat, zvolíme si pro ni vhodné jméno – identifikátor, který potom pro označení té které proměnné používáme.

Proměnná musí být nejprve deklarována před tím, než bude v programu použita, aby se pro ni v paměti vymezilo místo.

Deklarace je v podstatě výčet všech proměnných, které se budou v programu používat. Deklarací proměnné určujeme jméno proměnné a typ hodnot, které se budou do proměnné ukládat. Deklarace zajistí vymezení patřičně velké oblasti paměti pro uložení hodnoty daného typu.

Jaké hodnoty buňka může obsahovat a jaká bude její velikost specifikuje typ proměnné. Typ proměnné specifikuje množinu přípustných hodnot, které mohou být do proměnné ukládány.

Typem proměnné jsou současně určeny i operace, které lze pro tuto proměnnou předepisovat.

Základní typy proměnných

integer

real

boolean

char

Pro každou proměnnou se v paměti vymezí místo, jehož velikost odpovídá jejímu typu.

Deklarace proměnných začíná slovem **var** a potom následuje výčet jednotlivých proměnných s uvedením jejich typu.

var
Jméno1, Jméno2, ...:typ1;
Jméno3, Jméno4, ...:typ2;
...

Např.:

```
var
  A, B:integer;
  Soucet:real;
  ...
```

Deklarací sdělujeme počítači, že budeme v programu pracovat např. se symbolickou veličinou A, jejímž obsahem bude v každém okamžiku realizace programu celé číslo. Proměnné lze přiřadit pouze takovou hodnotu, která patří do množiny přípustných hodnot specifikované typem proměnné.

Deklarací se proměnným nedefinují jejich hodnoty.

Proměnné lze určit její obsah (naplnit ji) buď přečtením hodnoty příkazem **read**/**readln**, nebo dosazením hodnoty přiřazovacím příkazem. Dokud není do proměnné hodnota uložena, je její obsah nedefinován.

Typ INTEGER

Proměnné typu integer je v paměti počítače vymezena buňka o velikosti 2 B. Typ integer je konečná, souvislá podmnožina celých čísel. Rozsah této podmnožiny je <-32 767, 32 766> (tj. 215-1, 1 bit je na znaménko). Rozsah čísla je omezený z důvodu omezenosti buňky v paměti, která je vyhrazena pro daný typ.

Standardní konstanta MAXINT udává největší možné celé číslo v absolutní hodnotě zobrazitelné počítačem (závisí na konkrétní implementaci). Pro libovolné N typu integer tedy platí

-MAXINT <= N <= MAXINT

Operace s výsledkem rovněž typu integer:

+, -, *, DIV (celočíselné dělení), MOD (zbytek pro celočíselném dělení)

Relační operace:

=, <>, <, >, <=, >= (výsledkem je hodnota TRUE, pokud je relace splněna, nebo FALSE, pokud splněna není).

Standardní funkce:

ABS(I) – absolutní hodnota čísla I (výsledek je rovněž typu integer)
SQR(I) – druhá mocnina čísla I (výsledek je rovněž typu integer)
ODD(I) – zda I je liché číslo (výsledek je typu boolean – viz dále)

Výsledky operací jsou shodné s matematickými pouze pokud výsledek leží v intervalu <-MAXINT, MAXINT > – jinak dochází k chybě zvané přeplnění.

Typ REAL

Proměnné typu real je v paměti počítače vymezena buňka o velikosti 6 B (5 B je vyhrazeno pro mantisu, 1 B na exponent, 1 bit je na znaménko; mantisa udává přesnost čísla – zhruba na 7-14 desítkových číslic, exponent jeho rozsah – často je to -10^{75} až $+10^{75}$.) Typ real je konečná podmnožina reálných čísel.

Pro oddělení celé a desetinné části se používá desetinná tečka.

Např.:
$56{,}432*10^{7}$ → 5.643200000E08
$145{,}28*10^{-5}$ → 1.452800000E-03

Operace s výsledkem rovněž typu real:

+, -, *, /

Relační operace:

=, <>, <, >, <=, >= (výsledkem je hodnota TRUE, pokud je relace splněna, nebo FALSE, pokud splněna není).

Standardní funkce s výsledkem typu real:

ABS(R) – absolutní hodnota čísla I (výsledek je rovněž typu integer)
SQR(R) – druhá mocnina čísla I (výsledek je rovněž typu integer)
SIN(R) – sinus R (argument je v radiánech)
COS(R) – cosinus R (argument je v radiánech)
ARCTAN(R) – arcustangens R (argument je v radiánech)
LN(R) – ln R (přirozený)
EXP(R) – e^R
SQRT(R) – druhá odmocnina R

Výsledek operace musí opět ležet v rozsahu zobrazitelných čísel. Výsledek je nepřesný vzhledem k omezenému počtu zobrazitelných míst mantisy.

Konverze typu INTEGER a REAL

Hodnota typu integer se na typ real převádí automaticky. Rozsah hodnot typu real je vždy větší než rozsah hodnot typu integer, a proto je vždy definován. Pro převod hodnot typu real na hodnoty typu integer jsou definovány dvě standardní transformační funkce:

TRUNC(R) – jejíž výsledkem je celé číslo, které vznikne odseknutím desetinné části hodnoty R

Např.:
```
trunc(125.2)=125
trunc(83.7) =83
trunc(-28.3)=-28
trunc(-96.5)=-96
```

ROUND(R) – jejímž výsledkem je zaokrouhlená hodnota R

Např.:
```
round(125.2)=125 Nelze trunc('7')
round(83.7) =84 round('2')
round(-28.3)=-28
round(-96.5)=-97
```

Typ BOOLEAN

Proměnné typu boolean je v paměti počítače vymezena buňka o velikosti 1 B.

Typ boolean je množina logických hodnot, které se označují předdefinovanými identifikátory TRUE a FALSE. Logické hodnoty jsou tedy dvě a vyjadřují dva stavy: splněno/nesplněno. V různých kontextech se jim říká různě, např. ano/ne, 1/0, pravda/nepravda.

X	Y	X and Y	X or Y	not X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Konstanty a proměnné logického typu se používají méně často než konstanty a proměnné číselného typu. Velmi často se však používají logické výrazy (podmínky) – viz dále.

Typ CHAR

Hodnotami tohoto typu jsou znaky. Množina hodnot tohoto typu je dána konkrétním typem počítače.

Na počítačích PC IBM kompatibilních se používá tzv. ASCII kódová tabulka (American Standard Code for Information Interchange).

Posloupnosti znaků se říká řetězec znaků. Přímé zápisy znaků v programu se dávají do apostrofů.

Např.:
```
write('Zadej A: ');
write('Zadej B: ');
writeln('Soucet cisel ',A,' a ',B,' je ',Soucet)
```

Př.:
Sestavte program, který na obrazovku vypíše písmena abecedy.
```
program Abeceda;
var
  P:char;
begin
for P:='A' to 'Z' do writeln(P) {write(P), write(P,' ')}
end.
```

Přiřazovací příkaz

Každá příkazová část programu napsaného v jazyce Pascal začíná klíčovým slovem **begin** a končí slovem **end.** (s tečkou).

Program se skládá z příkazů oddělených ***středníkem***. Základním a nejjednodušším příkazem je přiřazovací příkaz. ***Přiřazovací příkaz*** slouží k přiřazení hodnoty proměnné.

proměnná := přímo hodnota/proměnná/obecně výraz;

Na levé straně od přiřazovacího znaménka **:=** se uvádí proměnná, do níž se hodnota dosazuje, a na pravé straně výraz, jehož hodnota má být přiřazena této proměnné.

Např.:
```
A:=1;
B:=A;
Soucet:=A+B;
Rychlost:=Vzdálenost/Cas;
```
Typ hodnoty uvedené na pravé straně přiřazovacího příkazu musí odpovídat typu proměnné na levé straně přiřazovacího příkazu (tzv. kompatibilita podle přiřazení). Typy si odpovídají tehdy, jsou-li totožné, nebo je-li možná vnitřní (automatická) konverze typů (typu integer na typ real).

Výraz

Pomocí výrazů se vypočte nějaká hodnota z více hodnot a proměnných. Hodnota se získá při provádění programu tzv. vyhodnocením výrazu.

Výrazy obsahují ***operandy*** (datové objekty – hodnota nebo proměnná), ***operátory*** (např. matematická znaménka, viz dále), popř. ***kulaté závorky***. Operátor předepisuje operace, které mají být provedeny s operandy při vyhodnocení výrazu. Závorky umožňují změnit pořadí operací při vyhodnocování výrazu, jež je jinak určeno prioritou operátorů a pořadím zapsaných operací (zleva doprava) při stejné prioritě operátorů.

Např.:
```
Soucet:=A+B;
Prumer:=(A+B)/2;
```

Vyhodnocování výrazů

Vyhodnocování výrazu probíhá při provádění programu a jeho výsledkem je hodnota výrazu.

Např.:
A, B jsou typu integer. Potom se následující výraz 4 * A + B div 3 – 1 vyhodnotí takto:
```
((4 * A) + (B div 3)) - 1
```
Obsahuje-li výraz závorky, provede se vyhodnocení takto:
```
4 * ( ( A + ( B div 3 ) ) - 1 )
```
Použití závorek ve výrazu umožňuje změnit pořadí operací při vyhodnocování výrazu, které je jinak určeno prioritou operátorů. Závorky se mohou použít i tam, kde sice nemění prioritu operací (a jsou tedy nadbytečné), ale mohou zlepšit čitelnost zapsaného výrazu.

Výraz musí být sestaven tak, aby ve všech krocích vyhodnocování vznikaly výsledky přípustného typu vzhledem k operátorům.

Př.:
`A < 3 and 3 <= B` ... nesprávně zkonstruovaný výraz, správně: `(A < 3) and (3 <= B)`

Operátory

Každý datový typ určuje nejen množinu přípustných hodnot, ale současně také množinu operací, které je možno s datovými objekty daného typu předepisovat. Operace lze předepisovat pomocí operátorů nebo použitím standardních funkcí.

Operátory jsou základní symboly jazyka s přesně vymezeným významem. Každý z operátorů může být použit jen pro určitý typ operandů. Proto rozlišujeme operátory pro:

- aritmetické operace: +, -, *, /, div, mod
- relační operace: =, <>, <, >, <=, >=
- logické operace: and, or, not (unární – předepisují operaci s jedním operandem: +, -, not)

Operátorem se předepisuje operace s jedním nebo dvěma operandy. Některé operátory umožňují použití jen jednoho typu operandu, jiné je možno použít pro operandy více typů (např. operátor * pro typy integer a real).

Pro typ integer a real (tedy číselné typy) můžeme předepisovat aritmetické a relační operace, ale nelze pro ně předepisovat operace logické (např. 5 and 3 nebo not 5.3).

Pro typ boolean (logický typ) můžeme předepisovat relační a logické operace, ale nelze pro ně předepisovat operace aritmetické (nelze např. sčítat true + false).

Např.:

```
5 div 3 = 1     3 div 4 = 0     5 mod 3 = 2     3 mod 4 = 3
```

Nelze:

```
3.0 div 5       3 mod 5.0       3 mod 0         5 div 0
```

Relační (porovnávací):

Používají se pro všechny jednoduché typy (integer, real, boolean i char).

Porovnávají se vždy hodnoty stejného typu nebo se provede implicitní konverze I → R (integer na real; pokud se argumenty liší, překladač se je nejprve pokusí sjednotit pomocí implicitní konverze, pokud to není možné dojde k chybě). Výsledek porovnávání je typu boolean.

Např.:

```
3 > 5 → false          3.0 < 5.0 → true          3 < 5.0 → true
3 <= 5 → true          3.0 >= 5.0 → false        3.0 >= 5 → false

false < true → true        'A' >= 'D' → false
false <= true → true       'G' <> 'H' → true
false <> true → true       'B' <= 'D' → true
```

Nelze: true < 'A' 'A' < 50 false = 0

Pozor u typu REAL při testu na rovnost! Vlivem výpočtu může dojít k nepřesnosti řádově 0.0000001 a rovnost nikdy nenastane.

1 / 3 * 3 <> 1 1 / 3 = 0,33333... 0,33333... * 3 = 0,99999...

Tedy např. místo A = B musíme psát abs(A-B) < EPS, kde EPS je vhodné malé kladné číslo.

Pozn.:

0 < X <= Y → (X > 0) and (X <= Y)

Př.:

Za pomoci přiřazovacího příkazu a relačních operací zjistěte, které ze tří čísel A, B a C je největší. (Bez použití podmíněného příkazu.)

```
AJeNejvetsi:=(A>B) and (A>C);
BJeNejvetsi:=(B>A) and (B>C);
CJeNejvetsi:=(C>A) and (C>B);
CJeNejvetsi:=not AJeNejvetsi and not BJeNejvetsi);
```

Logické:

Výsledek logické operace je vždy typu boolean.

X	Y	X and Y	X or Y	not X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Priorita operátorů

1. not
2. *, -, div, mod, and
3. +, -, or
4. =, <>, <, >, <=, >=

Pokud si nejsme jisti prioritou operátorů, použijeme kulaté závorky.

Příkaz výstupu (procedura)

Z programu je možné číst z různých zařízení a také na různá zařízení zapisovat. Standardně je vstupním zařízením klávesnice a výstupním obrazovka.

Zápis na obrazovku obstarává procedura **writeln** (zapiš). Má tvar

writeln(seznam parametrů);

Parametry jsou identifikátory nebo řetězce znaků. Jednotlivé parametry se oddělují čárkou. Řetězce znaků musí být uvedeny v apostrofech.

Např.:

```
writeln('Soucet cisel',A,'a',B,'je',Soucet);
```

Je možno použít 3 tvary parametru:

writeln(vystupující údaj);

writeln(vystupující údaj: počet znaků);

writeln(vystupující údaj: počet znaků: počet desetin. míst);

Vystupující údaj se zadává výrazem, jehož typ může být buď integer, real, char, boolean nebo řetězec znaků. Počet znaků a počet desetinných míst jsou výrazy typu integer.

Udáváme-li celkový počet míst a počet desetinných míst, musíme rezervovat z celé části čísla jedno místo na desetinnou tečku.

Pro výstup platí následující pravidla:

a) Pokud neurčíme celkový počet znaků, vystoupí tolik znaků, kolik jich obsahuje hodnota výrazu.

b) Je-li v parametru určen větší počet znaků než je nutné, doplní se vystupující hodnoty zleva mezerami.

c) Je-li v parametru určen menší počet znaků, než je nutné, vystoupí v případě typu integer a real minimální nutný počet znaků a z hodnot ostatních typů pouze zadaný počet znaků zleva.

d) Je-li při výstupu hodnoty typu real v parametru určen počet desetinných míst, vystoupí číslo sestávající z celé a desetinné části, v opačném případě vystoupí číslo v semilogaritmickém tvaru (zapsané pomocí mantisy a exponentu).

Např.

```
writeln('Soucet cisel ',A,' a ',B,' je ',Soucet:5:2);
```

Pokud např. A = 12 345, B = 6 789 a Soucet = 19 134 vystoupí při (lomítko / v zápisu nahrazuje mezeru)

```
writeln(A);                  12345
writeln(A:7);                //12345
writeln(A:2);                12345
writeln(Soucet:12);          ////.19134E05
writeln(Soucet:3);           .19134E05
writeln(Socet:10:3);         //19134.000
writeln('ABECEDA');          ABECEDA
writeln('ABECEDA':2);        AB
writeln('ABECEDA':10);       //ABECEDA
```

Př.:

```
program Secti;
{program secte 2 libovolna cela cisla}
var
  A, B:integer;
begin
write ('Zadej A: ');
readln(A);
write('Zadej B: ');
readln(B);
writeln('Soucet cisel ',A,' a ',B,' je ',A+B)
end.
```

Příkaz writeln má za následek výstup uvedených parametrů a ukončení řádky (další výstup v programu by byl uveden na nové řádce).

Druhý příkaz pro výstup z programu je příkaz **write**. Má tvar:

write(seznam parametrů);

Tento příkaz neprovádí ukončení řádky (další výstup v programu by byl uveden na stejné řádce jako předchozí).

Př.:

Sestavte program, který na obrazovku vypíše písmena abecedy.

```
program Abeceda;
var
  P:char;
begin
writeln('- Abeceda - ');
for P:='A' to 'Z' do write(P, ' '); {writeln(P), write(P)}
readln
end.
```

Stejného účinku jako má příkaz writeln se dosáhne použitím příkazů write(seznam parametrů); writeln;.

Příkaz vstupu (procedura)

Čtení z klávesnice obstarává procedura **readln** (čti). Má tvar

readln(proměnná);

Kolik znaků se tímto příkazem přečte závisí na typu proměnné:

a) Je-li typu integer nebo real, přečte se posloupnost znaků tvořící dekadické číslo s případným znaménkem a přiřadí se příslušné proměnné. Mezery a oddělovače řádků se při čtení ignorují.

b) Je-li typu char, přečte se jediný znak a jeho hodnota se přiřadí proměnné. Přečte-li se při tom oddělovač řádků, do proměnné se uloží znak mezera.

Např.:

```
readln(A); A = 12345
readln(B); B = 6789
```

Při čtení několika údajů bezprostředně za sebou lze v příkazu readln uvést seznam příslušných proměnných oddělených čárkou.

Např.:

```
readln(A,B); A = 12345, B = 6789
```

Příkaz readln se provádí tak, že zbytek řádky, ze které čteme se vynechá a následujícím příkazem se začne číst od začátku nové řádky. Příkaz **read** čte stále ze stejné řádky, dokud nenarazí na oddělovač řádky. Pak teprve přechází na čtení nové řádky.

Např.:

```
readln(A); má stejný účinek jako read(A);
readln;
readln(A,B); má stejný účinek jako read(A,B);
readln;
```

Konstanty

Konstanta je hodnota známá již při psaní programu a během programu se nemění (např. g = 9.8m/s, Î = 3.14 ap.). Při každém spuštění programu zůstává stejná (konstantní, neměnná).

Konstanta se používá tak, že se buď na příslušném místě v programu zapíše přímo její hodnota (tzv. literál), nebo ji můžeme nejprve pojmenovat a potom v programu na příslušném místě používat její jméno. Konstantu deklarujeme (pojmenováváme) v deklaraci konstant. Pojmenováním konstanty se zvyšuje srozumitelnost a zjednodušuje se modifikovatelnost programu (je tím umožněno změnit konstantu - např. zvýšit její přesnost - změnou deklarace konstanty, tj. změnou na jediném místě programu.)

Číselnou konstantu si překladač umístí rovnou do překladového kódu, na místo, kde se s ní pracuje (na rozdíl od proměnné, které je vymezena buňka v paměti).

Deklarace konstant začíná slovem **const** a potom následuje výčet jednotlivých konstant s přiřazením (=) konkrétní hodnoty každé konstantě.

const
Jméno1 = hodnota1;
Jméno2 = hodnota2;
...

Např.:
```
const
pi=3.14;
PocetRadek=30;
Jmeno='Josef Novak';
```

V programu deklarace konstant předchází deklaraci proměnných.

Př.:
Sestavte program, který vypočte obvod a plochu kruhu o poloměru R.

```
program ObvodAPlochaKruhu;
const
Pi=3.14;
var
R, Obvod, Plocha:real;
begin
write('Zadej polomer R: ');
readln(R);
Obvod:=2*Pi*R;
Plocha:=Pi*R*R;
writeln('Obvod kruhu o polomeru', R:7:2 ,' je',Obvod:7:2);
writeln('Plocha kruhu o polomeru', R:7:2 ,' je',Plocha:7:2)
end.
```

Př.:
V předcházejícím příkladu zvyšte přesnost Ludolfova čísla PI na 3.141596.

Standardní konstantou v programovacím jazyce Pascal je konstanta MAXINT. Tato konstanta udává největší možné celé číslo zobrazitelné počítačem. Hodnota -MAXINT udává nejmenší zobrazitelné celé číslo. Konstanta MAXINT je předem definovaná, to znamená, že ji nedeklarujeme, ale můžeme ji rovnou použít.

Priorita deklarací (obvykle)

label
const
type
var
procedure
function

Standardní funkce

Standardní funkce je dílčí algoritmus definovaný jazykem. Její deklarace se v programu neuvádí. Každá standardní funkce může být použita jen pro určité datové typy a získaná funkční hodnota je také vždy určitého typu. Podle typu funkční hodnoty mluvíme o funkcích aritmetických, logických apod.

Standardní funkce se v programu nedeklarují, ale zapisují se přímo až v místě použití.

Aritmetické

ABS(X), **SQR**(X), **SQRT**(X), **LN**(X), **EXP**(X), **SIN**(X), **COS**(X), **ARCTAN**(X)

Př.:
```
abs(5) = 5          sqr(5) = 25         sqrt(4) = 2.0
abs(-5) = 5         sqr(-5) = 25        sqrt(4.0)= 2.0
abs(-3.0)= 3.0      sqr(-3.0)= 9.0      sgrt(2) = 1.4142136
```

```
ln(2) = 0.693       sin(2) = 0.9092974
ln(2.0) = 0.693     cos(2) = -0.4161468
exp(0.693)= 2.0     arctan(2)= 1.1071487
```

Logické

ODD(X) testuje, zda argument je liché číslo. Argument musí být typu integer, výsledkem je logická hodnota.

Př.:

```
odd(5)= true        odd(-3)= true
odd(0)= false       odd(-4)= false      odd(4)= false
```

Nelze: `odd(3.0), odd(false), odd('5')`!

Ordinální typ

Ordinální typ tvoří taková množina hodnot, kdy je každé hodnotě jednoznačně přiřazeno tzv. ordinální číslo. ***Ordinální číslo*** je číselná reprezentace každé hodnoty ordinálního typu v počítači. U ordinálního typu se dají vyjmenovat všechny jeho hodnoty a platí pro něj standardní funkce

ORD(X) – zjišťuje ordinální číslo dané hodnoty X

SUCC(X) – zjišťuje bezprostředního následovníka dané hodnoty X

PRED(X – zjišťuje bezprostředního předchůdce dané hodnoty X

CHR(X) – je inverzní funkce k funkci ord a zjišťuje znak odpovídající danému ordinálnímu číslu (argument funkce je typu integer a funkční hodnota je typu char)

Např.:

```
ord('1') = 49         chr(65) = A
ord('A') = 65         chr(49) = 1
ord(' ') = 32         chr(32) =

succ('P') = Q         pred('Q') = P
succ(5) = 6           pred(6) = 5
succ('5') = 6         pred('6') = 5
succ(-5) = -4         pred(-5) = -6
succ(285) = 286       pred(285) = 284
succ(0) = 1           pred(0) = -1

ord(false) = 0        ord(true) = 1
succ(false) = true    pred(true) = false
```

Nelze: `succ(true), pred(false)`!

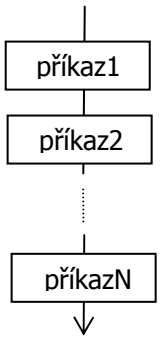
Př.:

Napište program, který vypíše ordinální číslo zadaného znaku a znak odpovídající zadanému ordinálnímu číslu.

```
program VratHodnotu;
var
  c, d:integer;
  z, a:char;
begin
write('Zadej znak: ');
readln(z);
writeln('Ordinalni cislo znaku','"',z,'"',' je ',ord(z));
write('Zadej ordinalni cislo: ');
readln(c);
writeln('Znak pro ordinalni cislo ',c, ' je ',chr(c))
end.
```

Složený příkaz

Vývojový diagram:



Má tvar

```
begin
příkaz1;
příkaz2;
...
příkazN
end;
```

Složený příkaz je uzavřen mezi **begin** a **end**, které tvoří jakési závorky a činí tak příkaz nedělitelným. Na složený příkaz se potom díváme jako na jeden celistvý příkaz.

Jednotlivé příkazy tvořící složený příkaz se oddělují středníkem.

Např.:

```
begin
Obvod:=2*A+2*B;
Obsah:=A*B;
Uhlopricka:=sqrt(sqr(A)+sqr(B))
end;
```

Složený příkaz se často používá jako součást podmíněného příkazu a příkazu cyklu. Konkrétní použití složeného příkazu si ukážeme v následujících příkladech.

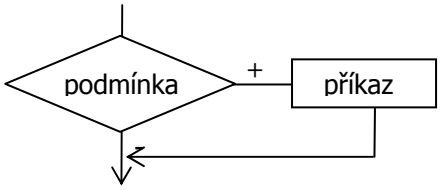
Podmíněný příkaz IF

Podmíněný příkaz umožňuje větvení programu do dvou alternativ. Vyhodnocuje se podmínka a na jejím základě se rozhoduje o dalším postupu.

Podmíněný příkaz může být obecně použit na všech místech příkazové části programu, kde potřebujeme podmínit další postup, tzn. vybrat v závislosti na hodnotě podmínky (splněna/nesplněna) jeden ze dvou příkazů.

a) Neúplný podmíněný příkaz

Vývojový diagram:



Má tvar

if podmínka **then** příkaz;

Pozn.: Jestliže platí podmínka, potom vykonej příkaz.

Podmínku tvoří logický výraz. Příkaz může být jeden nebo posloupnost příkazů (složený příkaz).

Např.:

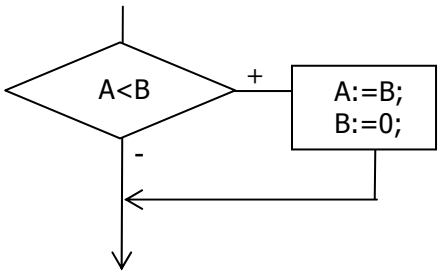
```
if A<0 then A:=abs(A);

if (A<0) or (B<0) then
  begin
  A:=abs(A);
  B:=abs(B)
  end;
```

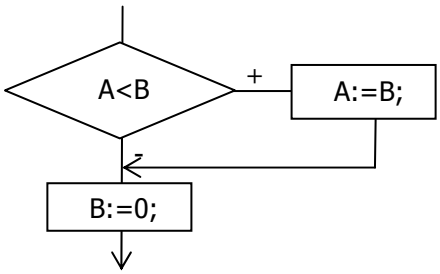
Pokud zapomeneme v podmíněném příkazu posloupnost příkazů zapsat jako složený příkaz (uzavřít ji mezi **begin** a **end**), obdržíme zápis se zcela jiným významem.

Např.:

```
if A<B then begin A:=B; B:=0 end;
```

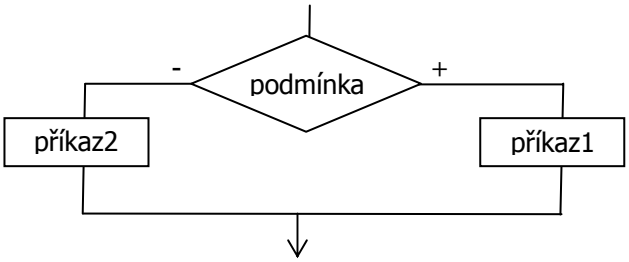



```
if A<B then A:=B; B:=0;
```



b) Úplný podmíněný příkaz

Vývojový diagram:



Má tvar

if podmínka **then** příkaz1 **else** příkaz2;

Pozn.: Jestliže platí podmínka, potom vykonej příkaz1, jinak vykonej příkaz2.

Podmínku opět tvoří logický výraz. Dílčími příkazy bývá často posloupnost příkazů (složený příkaz). Za posledním příkazem před **else** nesmí být středník.

Měli bychom vždy dodržovat zásadu, že všechny vstupující údaje zkontrolujeme před jejich vlastním zpracováním (většinou ihned po jejich přečtení) a případnou chybu opravíme.

Př.:

Sestavte program, který vypočte obvod a plochu kruhu. Na případná nepřípustná vstupní data reagujte příslušným výstupem.

```
program ObvodAPlochaKruhu;
const
  Pi=3.14;
var
  R, Obvod, Plocha:real;
begin
write('Zadej polomer R: ');
readln(R);
if R>0
  then
    begin
    Obvod:=2*Pi*R;
    Plocha:=Pi*R*R;
    writeln('Obvod kruhu je',Obvod:7:2);
    writeln('Plocha tohoto je',Plocha:7:2)
    end
  else
    writeln('Chybne zadani vstupnich dat')
end.
```

Př.:

Sestavte program, který vypočte obvod a plochu obdélníka a délku jeho úhlopříčky. Na případná nepřípustná vstupní data reagujte příslušným výstupem.

```
program Obdelnik;
var
  A, B, Obvod, Plocha, Uhlopricka:real;
begin
write('Zadej delku strany A: ');
readln(A);
write('Zadej delku strany B: ');
readln(B);
if (A>0) and (B>0)
  then
    begin
    Obvod:=2*A+2*B;
    Plocha:=A*B;
    Uhlopricka:=sqrt(sqr(A)+sqr(B))
    writeln('Obvod obdelnika je',Obvod:7:2);
    writeln('Plocha obdelnika je',Plocha:7:2);
    writeln('Uhlopricka obdelnika je',Uhlopricka:7:2)
    end
  else
    writeln('Chybne zadani vstupnich dat');
end.
```

Za else může být jakýkoli příkaz, tedy i opět podmíněný.

Podmíněné příkazy se mohou do sebe vnořovat, což umožňuje určit úsek programu, který bude proveden z více než dvou možných úseků.

Př.:

Sestavte program, který zjistí, která ze tří zadaných různých celočíselných hodnot A, B, C je největší, a tuto největší hodnotu vypíše.

```
program Maximum1;
var
  A, B, C:integer;
begin
writeln('Zadej tri ruzna cela cisla');
write('A: ');
readln(A);
write('B: ');
readln(B);
write('C: ');
readln(C);
if A>B
  then
    if A>C
      then writeln('A = ', A, ' je nejvetsi')
      else writeln('C = ', C, ' je nejvetsi') {kdyz A<=C}
  else
    if B>C
      then writeln('B =', B,' je nejvetsi')
      else writeln('C =', C,' je nejvetsi') {kdyz B<=C}
end.
```

Př.:

Sestavte program, který zjistí, která ze tří zadaných různých celočíselných hodnot A, B, C je největší, a tuto největší hodnotu vypíše. Řešte bez vnořených podmíněných příkazů.

```
program Maximum2;
var
  A, B, C, Max:integer;
begin
writeln('Zadej tri ruzna cela cisla');
write('A: ');
readln(A);
write('B: ');
```

```
readln(B);
write('C: ');
readln(C);
Max:=A;
if B>Max then Max:=B;
if C>Max then Max:=C;
writeln(Max,' je nejvetsi')
end.
```

Př.:
Sestavte program, který zjistí, zda jsou 2 libovolná celá čísla kladná nebo záporná a na zjištěnou skutečnost bude reagovat příslušným výstupem.

```
program KladneCiZaporne;
var
  A, B:integer;
begin
write('Zadej 2 cela cisla');
write('A: ');
readln(A);
write('B: ');
readln(B);
if (A>0) and (B>0)
  then writeln('A i B jsou kladna')
  else
    if (A<0) and (B<0)
      then writeln('A i B jsou zaporna')
      else
        if (A>0) and (B<0)
          then writeln('A je kladne a B je zaporne')
          else
            if (A<0) and (B>0)
              then writeln('A je zaporne, B je kladne')
end.
```

Podmíněný příkaz CASE (výběrový příkaz)

Tento příkaz umožňuje větvení výpočtu do několika alternativ v závislosti na hodnotě výrazu. Má tvar

case výraz **of**
konstanta1 : příkaz1;
konstanta2 : příkaz2;
...
konstantaN : příkazN;
else příkazN+1
end;

Hodnota výrazu musí být typu integer, boolean nebo char.

Konstanty musí být stejného typu jako výraz. Žádná konstanta nesmí být uvedena více než jednou. Příkaz case se provádí takto: nejprve se vyhodnotí výraz a pak se provede dílčí příkaz před nímž je uvedena dílčí konstanta označující stejnou hodnotu jako má výraz. Dílčím příkazem může opět být složený příkaz. Pokud více různým konstantám odpovídá stejný příkaz, mohou být tyto uvedeny vedle sebe odděleny čárkami.V tomto jediném případě se před else píše středník.

Př.
Sestavte program, který tiskne název dne v týdnu v závislosti na jeho pořadovém čísle.

```
program JmenoDne;
var
  den:integer;
begin
write('Zadej poradove cislo dne: ');
readln(den);
case den of
```

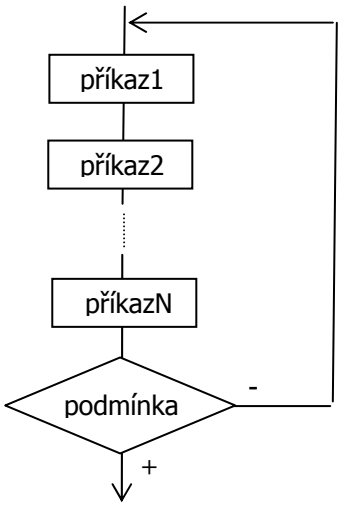
```
  1: writeln('Pondeli');
  2: writeln('Utery');
  3: writeln('Streda');
  4: writeln('Ctvrtek');
  5: writeln('Patek');
  6: writeln('Sobota');
  7: writeln('Nedele');
  else writeln('Chybne zadani poradoveho cisla dne')
  end
end.
```

Příkaz cyklu REPEAT (příkaz s podmínkou na konci)

Příkazy cyklu umožňují opakovaně provádět určitý příkaz nebo posloupnost příkazů. Počet opakování u tohoto cyklu je dán podmínkou. Příkazy se budou opakovat tak dlouho, dokud nebude splněna podmínka.

Má tvar

repeat příkaz **until** podmínka**;**



repeat
příkaz1;
příkaz2;
...
příkazN
until podmínka**;**

Pozn.: Opakuj, dokud neplatí (opakuj, dokud nebude splněna podmínka).

Tento příkaz se provádí tak, že se nejprve provedou všechny příkazy (tělo cyklu) potom se vyhodnotí výraz udávající podmínku ukončení cyklu a je-li hodnota tohoto výrazu false, opakuje se tato činnost znovu. Provádění příkazu repeat končí až tehdy, když výraz udávající podmínku má hodnotu true.

Př.:

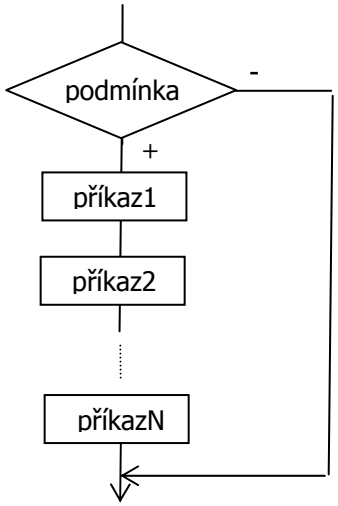
Sestavte program, který tiskne 10 celých čísel vzestupně počínaje číslem 1.

```
program Vzestupne1;
const
  Pocet=10;
var
  N:integer;
begin
writeln('Cisla serazena vzestupne pomoci repeat');
N:=0;
repeat
  N:=N+1;
  writeln(N)
  until N>=Pocet
end.
```

Příkaz cyklu WHILE (příkaz s podmínkou na začátku)

Má tvar

while podmínka **do** příkaz**;**




```
while podmínka do
  begin
  příkaz1;
  příkaz2;
  ...
  příkazN
  end;
```

Pozn.: Dokud platí (podmínka), dělej.

Zajišťuje opakované provádění příkazu tak dlouho, dokud je splněna uvedená podmínka. Příkaz cyklu while se používán v případech, kdy počet opakování není předem znám. Počet opakování může být též nulový (pokud podmínka hned na začátku není splněna).

Příkazem může být opět složený příkaz (uzavřené mezi begin a end).

Tento příkaz se provádí takto: Jestliže podmínka má hodnotu true (je splněna), provede se tělo cyklu (příkaz). Provádění příkazu while končí, když výraz udávající podmínku má hodnotu false. Má-li tento výraz hodnotu false už na začátku provádění příkazu, pak se příkaz uvedený za do neprovede ani jednou.

Př.:

Sestavte program, který tiskne 10 celých čísel vzestupně počínaje číslem 1.

```
program Vzestupne2;
const
  Pocet=10;
var
  N:integer;
begin
writeln('Cisla serazena vzestupne pomoci while');
N:=0;
while N<Pocet do
  begin
  N:=N+1;
  writeln(N)
  end
end.
```

Příkaz cyklu FOR (cyklus s určeným počtem opakování)

Používá se tehdy, je-li předem znám počet opakování.

U tohoto cyklu se udává interval hodnot, pro který se má opakování provádět. Hodnoty, které mají být postupně dosazovány do řídící proměnné se určují uvedením počáteční a koncové hodnoty. Hodnoty se do řídící proměnné cyklu dosazují postupně automaticky.

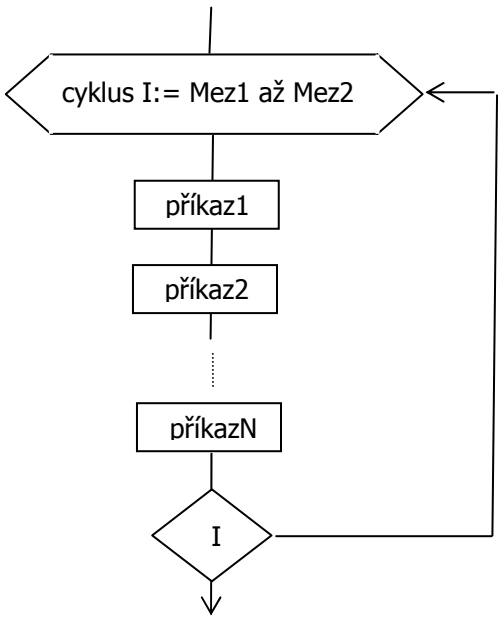
Má tvar

for I:= Mez1 **to** Mez2 **do** příkaz**;**

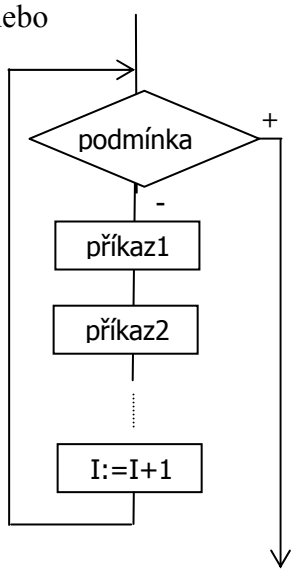
nebo

for I:= Mez1 **downto** Mez2 **do** příkaz**;**

kde I je parametr (řídící proměnná) cyklu (typu integer), Mez1 a Mez2 jsou počáteční a konečná hodnota parametru (obecně to mohou být výrazy).



nebo



for I:=Mez1 **to/downto** Mez2 **do**
begin
příkaz1;
příkaz2;
...
příkazN
end;

Pozn.: Opakuj pro.

Příkaz for se provádí takto: Nejprve se vypočte hodnota výrazů Mez1 a Mez2, potom se parametru cyklu přiřadí počáteční hodnota (Mez1) a porovná se s konečnou hodnotou (Mez2). Jestliže parametr cyklu není větší než Mez2, provede se tělo cyklu. Pak se parametr cyklu zvýší o 1 a celý proces se opakuje. Ukončení cyklu nastává tehdy, když je parametr cyklu větší než Mez2.

Při použití downto místo to se parametr cyklu zmenšuje o 1.

Př.:

Sestavte program, který tiskne 10 celých čísel

a) vzestupně počínaje číslem 1,

ad a)

```
program Vzestupne3;
const
  Pocet=10;
var
  N:integer;
begin
writeln(CCisla serazena vzestupne po-
moci for');
for N:= 1 to Pocet do
  writeln(N)
end.
```

b) sestupně počínaje číslem 10.

ad b)

```
program Sestupně;
const
  Pocet=10;
var
  N:integer;
begin
writeln('Cisla serazena sestupne po-
moci for');
for N:= Pocet downto 1 do
  writeln(N)
end.
```

Literatura:

David Morkes: **Základy programování** (učebnice pro střední školy), Computer Press, Brno1998

Tomáš Hála: **Pascal** (učebnice pro střední školy), Computer Press, Brno 1999

Josef Jinoch, Karel Müller, Josef Vogel**: Programování v jazyku Pascal**, SNTL 1988

Martin Kvoch: **Programování v Turbo Pascalu 7.0**, Kopp, České Budějovice 1993

Martin Kvoch, Jaroslav Jančík: **Sbírka úloh z jazyka Pascal**, Kopp, České Budějovice 1995

Zpracovala: Ing. Simona Martínková, březen 2007