INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Výukový materiál zpracován v rámci projektu

EU peníze školám

Programování

Petr Martiník

Registrační číslo projektu: CZ.1.07/1.5.00/34. 0418
Číslo klíčové aktivity: III/2
Název klíčové aktivity: Inovace a zkvalitnění výuky prostřednictvím ICT
Číslo materiálu: VY_32_INOVACE_261-280
Datum: 31. 10. 2013

Vzdělávací oblast:	Informatika a informační a komunikační technologie
Tematická oblast:	Algoritmizace úloh
Předmět:	Informatika a výpočetní technika
Výstižný popis způsobu využití, případně metodické pokyny:	Většinou jde o texty se záznamem výkladu v hodinách. Mohou být využity k opakování nebo k samostatnému studiu. Doplněny jsou příklady na procvičení nebo rozšíření učiva.
Klíčová slova	Algoritmus, program, Pascal, řídící struktury, efektivita algoritmu
Druh učebního materiálu:	Pracovní sešit

Autorem materiálu a všech jeho částí, není-li uvedeno jinak, je Petr Martiník.

Obsah

Programování 5

1 Tvorba programu (VY_32_INOVACE_261) 6

1.1 Zadání 6

1.2 Analýza 6

1.3 Algoritmus 6

1.4 Testování 7

1.5 Program 7

1.6 Ladění 8

1.7 Optimalizace 10

1.8 Uživatelská úprava programu 11

1.9 Dokumentace 11

1.10 Údržba 11

2 Úprava programu (VY_32_INOVACE_262) 12

2.1 Čekání na stisk klávesy 12

2.2 Příkaz výstupu - write 12

2.3 Formátování výstupů 15

2.4 Příkaz vstupu - read 16

3 Výrazy – operace (VY_32_INOVACE_263) 18

4 Výrazy – funkce (VY_32_INOVACE_264) 21

5 Výrazy - vyhodnocování výrazů (VY_32_INOVACE_265) 22

6 Výrazy - zápis výrazů (VY_32_INOVACE_266) 24

7 Výrazy - podmínky a negace (VY_32_INOVACE_267) 27

8 Strukturované příkazy (VY_32_INOVACE_268) 29

9 Zápis programu (VY_32_INOVACE_269) 31

10 Testování programu (VY_32_INOVACE_270) 34

11 Podmíněný příkaz (VY_32_INOVACE_271) 40

12 Mocnina, cyklus (VY_32_INOVACE_272) 43

13 Spoření (VY_32_INOVACE_273) 48

14 Číslice (VY_32_INOVACE_274) 51

15 Číslice 2 (VY_32_INOVACE_275) 53

16 Posloupnosti (VY_32_INOVACE_276) 55

17 Posloupnosti – maxima (VY_32_INOVACE_277) 58

18 Dělitelnost (VY_32_INOVACE_278) 60

19 Dělitelnost 2 (VY_32_INOVACE_279) 63

20 Dělitelnost 3 (VY_32_INOVACE_280) 65

21 Literatura 67

Programování

Tento materiál obsahuje většinou texty se záznamem výkladu z hodin, které mohou sloužit k opakování nebo samostatné přípravě žáků. Součástí většiny témat jsou úkoly k procvičení a rozšíření učiva.

Je určen pro výuku základů programování, které je vyučováno povinně pro všechny žáky v tercii nebo kvartě šestiletého studia. Může sloužit i k úvodnímu samostatnému opakování ve volitelném předmětu programování.

Cílem je naučit žáky tvorbě jednoduchých a efektivních programů v jazyce Pascal. Předpokládá se znalost základních pojmů z oblasti programování (algoritmus, program, vývojový diagram, efektivita programu, konstanty a proměnné, přiřazovací příkaz, programovací jazyky a překladače). Ve výuce se používá programovací jazyk Pascal a vývojové prostředí Bloodshed Dev-Pascal.

Každé téma je určeno na jednu výukovou hodinu. Rozsah a časová náročnost cvičení a úkolů pro přípravu žáků je závislý na výběru učitele.

1 Tvorba programu (VY_32_INOVACE_261)

V této úvodní kapitole si popíšeme etapy tvorby programu. Jde o popis kroků, které by vás měly vést k vytvoření jednoduchého několikařádkového programu v jazyce Pascal. Při tvorbě větších projektů jsou jednotlivé kroky rozsáhlejší a složitější. Při popisu jednotlivých kroků budeme tvořit program pro výpočet obsahu čtverce.

1.1 Zadání

Zadání je zásadní etapa, ve které se zadavatel (neodborník na programování) musí domluvit s programátorem (neodborníkem na danou problematiku), což bývá mnohdy obtížné. My budeme mít zadání jednoduché, bude formulováno například takto:

Sestavte program pro výpočet obsahu čtverce o straně a.
Součástí zadání mohou být i požadavky na vzhled a chování programu na obrazovce.

1.2 Analýza

V tomto kroku studujeme danou problematiku, hledáme a volíme potřebné metody, postupy a vzorce (ten náš pro výpočet obsahu čtverce snad znáte). Závěrem analýzy by měla být tzv. specifikace algoritmické úlohy, tj. popis vstupů a výstupů. V našich programech by specifikace měla obsahovat identifikátory používaných proměnných, jejich významy a datové typy, u vstupů i popis případné vstupní podmínky (tj. pro jaké vstupní hodnoty úlohu řešíme). Reakcí programu na hodnoty, které nesplňují vstupní podmínku, se nebudeme zabývat. Kromě vstupních a výstupních proměnných je dobré si podobně popsat i další použité (pomocné) proměnné. Specifikace naší úlohy může vypadat takto.

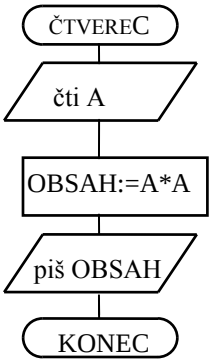
```
Vstupy:
      A     velikost strany čtverce real, A>0
Výstupy:
      OBSAH obsah čtverce           real
```

Specifikace může obsahovat i popis způsobu zadávání dat a požadavky na formu a vzhled výstupů.

1.3 Algoritmus

V této etapě vlastně úlohu vyřešíme. Navrhneme postup řešení, který ale nebudeme přímo psát v konkrétním programovacím jazyku, ale v nějaké přehlednější podobě, ve které nemusíme dodržovat přesná syntaktická pravidla programovacího jazyka. Použijeme vývojové diagramy.

Každý algoritmus se v podstatě skládá ze tří částí. V první zadáme vstupní hodnoty, v další vypočítáme výstupy a uložíme je do výstupních proměnných a nakonec vypíšeme výstupy. Ne každý algoritmus je ale tak jednoduchý jako tento úvodní.



1.4 Testování

Správnost vytvořeného algoritmu bychom měli otestovat. Pro vstupy, pro které známe výsledky, bychom měli simulovat činnost algoritmu a závěry porovnat s očekávanými výsledky. Naše programy jsou zatím tak jednoduché, že testování vynecháme a vrátíme se k němu v kapitole 11.

1.5 Program

V této etapě zvolíme programovací jazyk (my už máme zvolen Pascal) a zapíšeme algoritmus v tomto jazyce.

Program v jazyce Pascal má tří části: hlavičku, deklarační část a příkazovou část.

Hlavička má tvar:

```
program identifikátor;
```

Do hlavičky patří i připojení programových jednotek (uses), které bude popsáno v další kapitole.

Deklarační část popisuje (deklaruje) objekty, se kterými bude program pracovat. Každý objekt musí být v Pascalu nejprve deklarován a potom teprve použit. Deklarovat lze různé objekty (návěští, konstanty, typy, proměnné, procedury a funkce), v našich programech budeme deklarovat pouze proměnné. Deklaraci proměnných zahájíme slovem var a pro každou proměnnou napíšeme její identifikátor a typ podle následující ukázky. Proměnné stejného typu je možné seskupit do seznamu.

```
var
 N: integer;
 A,B,C: real;
 R,S: string;
 P,Q: boolean;
```

Příkazová část popisuje vlastní výpočetní proces, tj. co se s deklarovanými objekty bude dělat. Začíná slovem begin a je ukončena slovem end a tečkou (.). Obsahuje příkazy oddělené středníkem. Celý program tedy může vypadat takto:

```
program CTVEREC;
var
 A,OBSAH: real;
begin
 read(A);
 OBSAH:=A*A;
 write(OBSAH)
end.
```

1.6 Ladění

Až v této etapě jdeme k počítači a ve vývojovém prostředí námi zvoleného jazyka program editujeme (tj. zapíšeme do počítače).

Vlastní ladění znamená odstraňování chyb v programu. Chyby mohou být syntaktické a logické. Syntaktické chyby jsou chyby v zápisu programu (program není napsán podle syntaktických pravidel jazyka). Vývojové prostředí chyby odhalí a informuje nás o nich (kde je chyba a jaká je to chyba). Ne vždy to bude naprosto srozumitelné a jednoznačné, ale hledání a odstraňování syntaktických chyb je vcelku jednoduché. Pokud je program syntakticky v pořádku, je možné ho přeložit (kompilovat) a spustit.

Horší to bude s druhým typem chyb. Logické chyby jsou chyby ve výpočtu. Program něco spočítá, ale výsledky jsou chybné. Může se také stát, že výpočet není ukončen nebo se program přeruší. Tyto chyby se hledají hůře a k jejich odstranění je nutné program upravit nebo řešit zcela znovu.

Základní obsluha vývojového prostředí Bloodshed Dev-Pascal

V této části stručně popíšeme základní funkce vývojového prostředí Dev-Pascal potřebné pro odladění jednoduchého programu.

Nový program vytvoříme v nabídce File – New Source file (nebo klávesovou zkratkou Ctrl+U nebo tlačítky na panelu nástrojů). Otevře se okno pro zápis zdrojového kódu programu. Obsah okna vymažte.

Zapište zdrojový kód programu Čtverec, při zápisu je rozumné se přepnout na anglickou klávesnici (nebudeme používat znaky s háčky a čárkami, naopak budeme často potřebovat znaky, které jsou pouze na anglické klávesnici). Při editaci ale nezapomeňte, že znaky hledáte na anglické klávesnici.

```
program CTVEREC;
var A, Obsah: real;
```

```
begin
 read(A);
 Obsah:=A*A;
 write(Obsah)
end.
```

Pro ukládání programu platí stejná pravidla jako při ukládání v MS Wordu a jiných programech. Ukládá se pomocí File – Save Unit (uložit), File – Save Unit As (uložit jako). Ještě neuložený program má název Untitled. V okně pro uložení nastavte adresář, zkontrolujte v dolní části typ souboru (Pascal source file) a zadejte název programu. Je třeba dodržet pravidla pro MS DOS, tj. maximálně 8 znaků, bez háčků a čárek, bez mezer (tj. např. ctverec). Příponu pas nepište, program si ji doplní sám. Dále je možné ukládat pomocí Save Unit nebo klávesovou zkratkou Ctrl+S (soubor se pouze přepíše) nebo Save Unit As (v okně je možno změnit název nebo umístění souboru). Na disku se vytvoří soubor s příponou pas. Je to tzv. čistý text, tj. textový soubor, který lze vytvořit i v libovolném jiném editoru čistého textu (např. v Poznámkovém bloku).

Pro překlad (kompilaci) programu je funkce Execute – Compile (Ctrl+F9) nebo tlačítko na Panelu nástrojů. Před překladem dojde k syntaktické kontrole programu (tj. zda zapsaný program odpovídá syntaktickým pravidlům Pascalu) a v případě, že je syntakticky v pořádku, dojde i k jeho překladu. Výsledek kontroly je uveden v okně. Pokud jsou v programu syntaktické chyby, objeví se počet chyb v položce Total errors a v dolní části prostředí jsou chyby popsány na záložce Compiler (řádek, sloupec, popis). Tlačítkem Continue se vrátíme do editačního okna, tlačítko Execute pro spuštění programu je nedostupné. Syntaktické chyby je třeba odstranit. Pokud je program syntakticky v pořádku, dojde k jeho překladu (na disku vznikne soubor exe). Tlačítkem Continue se vrátíme do editačního okna, tlačítkem Execute můžeme program spustit.

Program lze spustit pomocí Execute – Run nebo jinými způsoby (klávesová zkratka F9, tlačítko na panelu nástrojů, tlačítko Execute z okna s výsledkem kompilace, spuštění exe souboru v Průzkumníkovi). Před spuštěním je nutné vždy provést překlad. Lze použít i funkci Execute - Compile And Run (Ctrl+F10, panel nástrojů), která přeloží a pak spustí program (pokud je syntakticky v pořádku).

Po spuštění programu se otevře okno s textovým prostředím MS DOSu (černé pozadí). Program již běží, zpracovává se příkaz read(A), což se projevuje tím, že program zobrazil blikající kurzor a čeká na zadání hodnoty. Zadejte z klávesnice hodnotu (velikost strany čtverce) a potvrďte ji klávesou Enter. Po zadání program přiřadil zadanou hodnotu do proměnné A.Tím skončil příkaz read(A) a hned proběhly další příkazy. Proměnné Obsah se přiřadila hodnota výrazu A*A a potom se na obrazovku vypsala hodnota proměnné Obsah příkazem write(Obsah). Tím program skončil a přepnul se automaticky zpět do vývojového prostředí, takže jsme si ani nepřečetli výsledek.

Abychom si mohli prohlédnout výsledky, přidáme do programu příkaz, který zajistí, že program před ukončením počká, až uživatel stiskne libovolnou klávesu. Toto zajistíme konstrukcí repeat until keypressed, kterou umístíme jako poslední příkaz před end (nezapomeňte doplnit za předchozí příkaz středník). Tato konstrukce ale funguje pouze v případě, že je připojena knihovna (unit) crt, což zajistíme zápisem uses crt do hlavičky programu za název programu. Upravený program tedy vypadá takto:

```
program CTVEREC;
uses crt;
var A, Obsah: real;
begin
 read(A);
 Obsah:=A*A;
 write(Obsah);
 repeat until keypressed
end.
```

Po úpravě program přeložte a spusťte znovu. Program počítá výsledek správně, ale vzhled programu a komunikace s uživatelem není dobrá. Toto upravíme v příští hodině.

Práci s vývojovým prostředím ukončíme File - Exit Dev-Pascal nebo jinými prostředky. Pokud od posledního uložení došlo ke změně zdrojového textu, budete vyzváni k uložení souboru.

Podívejte se v Průzkumníku do adresáře, do kterého jste program ukládali. Při práci vznikly 3 soubory se stejným názvem a s příponami pas (zdrojový kód), ow a exe (přeložený spustitelný soubor, pro naše programy nebude důležitý). Pro orientaci si v Průzkumníkovi zapněte zobrazení přípon souborů. Uchovávat budeme pouze soubory pas, soubory ow a exe budeme mazat.

Po spuštění programu Dev-Pascal se otevře poslední editovaný program. Ten můžeme zavřít a můžeme buď vytvořit nový program nebo otevřít už existující program (File – Open Project or file...). Otevřeno může být i více oken s programy.

1.7 Optimalizace

Optimalizací se rozumí taková úprava programu, jejímž cílem je zlepšení efektivity výpočtu. Budeme se tedy snažit, aby program běžel co nejrychleji a spotřeboval co nejméně paměti. Program by ale měl zůstat přehledný a srozumitelný.

Program Čtverec je tak jednoduchý, že v něm žádnou možnost optimalizace nenajdeme. Časovou a paměťovou efektivitou se budeme zabývat v kapitolách 13 – 20.

1.8 Uživatelská úprava programu

Program by v prvé řadě měl správně řešit danou úlohu. Pro běžné uživatele je nutné program upravit. Upravit bychom měli reakce na chybné vstupy uživatele, což není vždy jednoduché a už jsme domluveni, že to v našich programech nebudeme dělat. Úprava vzhledu na obrazovce je ale nezbytná. V následující kapitole si probereme prostředky pro úpravu programu.

1.9 Dokumentace

Ke každému programu by měla být zpracována dokumentace. Kromě uživatelské dokumentace (nápověda, uživatelské a referenční příručky,...) také dokumentace programátorská (popis řešení a programu). V našich úlohách byste ke každému programu měli mít zadání, specifikaci, vývojový diagram a program v jazyce Pascal bez uživatelských úprav.

1.10 Údržba

Programy z praxe (ať už komerční nebo volně šiřitelné) by se měly upravovat i po uvedení do provozu nebo na trh. Updaty a upgrady odstraňují z programů chyby, reagují na změnu požadavků, případně jen program vylepšují. Tato etapa nám odpadá.

2 Úprava programu (VY_32_INOVACE_262)

V minulé hodině jsme odladili program Čtverec. Otevřete zdrojový soubor s programem Čtverec a spusťte tento program. Po spuštění zadejte požadovanou hodnotu a prohlédněte si obrazovku. Program je funkční, ale komunikace s uživatelem je nedostačující. Doplníme program tak, aby komunikace s uživatelem byla přijatelnější. Po každém kroku program spusťte, abyste sledovali, jak se úprava vyvíjí.

```
program CTVEREC;
uses crt;
var A, Obsah: real;
begin
 read(A);
 Obsah:=A*A;
 write(Obsah);
 repeat until keypressed
end.
```

2.1 Čekání na stisk klávesy

Čekání na stisk libovolné klávesy před ukončením běhu programu jsme vyřešili už v minulé kapitole. Pokud pouze doplníte konstrukci repeat until keypressed, došlo by k syntaktické chybě. Pascal nezná funkci keypressed. Pro všechny příkazy musí být někde popsáno, co se má přesně vykonat. Jednotlivé příkazy jsou popsány v tzv. jednotkách (unitech). Aby program mohl příslušný příkaz používat, musí být k programu připojena jednotka, ve které je tento příkaz definován. Každá jednotka má svůj název (např. crt, dos, graph, printer,…). Základní příkazy jsou definovány v jednotce system, která je k programu připojena automaticky. Funkce keypressed je ale v jednotce crt, takže ji musíme k programu připojit. Jednotky se připojují v hlavičce programu, konkrétně chceme-li připojit jednotku crt, napíše v hlavičce programu uses crt;. Pokud bychom chtěli připojit jednotek více, napíšeme za slovem uses jejich seznam oddělený čárkami (např. uses crt, dos, graph;).

2.2 Příkaz výstupu - write

Dále budeme chtít, aby se na obrazovku psaly nějaké texty. Budeme chtít, aby vzhled na uživatelské obrazovce vypadal např. takto:

```
Obsah ctverce
*************

Velikost strany: 5

Obsah ctverce o strane 5.0 je 25.00.
```

Chceme-li, aby se něco vypsalo na obrazovku, použijeme příkaz write. Zatím jsme vypsali pouze hodnoty proměnných (např. write(A) – vypsala se hodnota proměnné A), chceme-li napsat na obrazovku libovolný text, použijeme v závorce řetězec - write(‘Obsah ctverce’). Znak apostrofu (‘) je na anglické klávesnici vlevo od klávesy Enter dole (nahoře jsou uvozovky). Podtržení docílíme výpisem podtrhávacích znaků, např. hvězdiček.

```
program CTVEREC;
uses crt;
var A, Obsah: real;
begin
 clrscr;
 write('Obsah ctverce');
 write('*************');
 read(A);
 Obsah:=A*A;
 write(Obsah);
 repeat until keypressed
end.
```

Program tiskne požadované texty, ale na nesprávných místech. Cokoli má Pascal napsat, napíše to vždy od pozice kurzoru. Příkaz write vždy napíše požadovaný text a ponechá kurzor za posledním napsaným znakem. Proto se podtržení vypsalo vedle nadpisu a také při zadání hodnoty kurzor blikal za poslední hvězdičkou. Místo příkazu write můžeme použít příkaz writeln, který funguje stejně jako příkaz write, ale nakonec přesune kurzor na začátek nového řádku. Toto nám v obou případech vyřeší problém. Příkaz writeln může být také samostatně bez závorky a parametrů, potom pouze přesune kurzor na nový řádek. Tím můžeme vynechat řádek.

```
program CTVEREC;
uses crt;
var A, Obsah: real;
begin
 clrscr;
 writeln('Obsah ctverce');
 writeln('*************');
 writeln;
 read(A);
 Obsah:=A*A;
 write(Obsah);
 repeat until keypressed
end.
```

Před každým zadáním hodnot příkazem read by se měla na obrazovce zobrazit stručná srozumitelná výzva k zadání hodnot. Tady použijeme write (nikoli writeln), protože chceme, aby kurzor zůstal za dvojtečkou. Všimněte si mezery v řetězci za dvojtečkou, která způsobí, že kurzor nebude blikat těsně za dvojtečkou, ale o jednu pozici dále.

```
program CTVEREC;
uses crt;
var A, Obsah: real;
begin
 clrscr;
 writeln('Obsah ctverce');
 writeln('*************');
 writeln;
 write('Velikost strany: ')
 read(A);
 Obsah:=A*A;
 write(Obsah);
 repeat until keypressed
end.
```

Tisk výsledků by měl být vždy s komentářem. Nemusí to být ve větě jako v našem příkladu. Věta, kterou chceme vytisknout, se skládá z 5 částí – třikrát text a dvakrát hodnota proměnné (v příkladu vytištěno tučně). Příkaz write může v závorce obsahovat více výrazů oddělených čárkami. Nahradíme tedy příkaz write(Obsah) kombinací řetězců a proměnných (všimněte si opět mezer uvnitř řetězců aby na sebe nebyly nalepené texty a čísla). Předtím ještě vynecháme řádek.

```
program Ctverec;
uses crt;
var A, Obsah: real;
begin
 clrscr;
 writeln('Obsah ctverce');
 writeln('*************');
 writeln;
 write('Velikost strany: ')
 read(A);
 Obsah:=A*A;
 writeln;
 write('Obsah ctverce o strane ',A,' je ', Obsah, '.');
 repeat until keypressed
end.
```

Můžete ještě zkusit upravit program tak, aby nadpis a podtržení bylo uprostřed obrazovky.

Příkaz výstupu má obecný tvar write(seznam výrazů), kde seznam výrazů jsou výrazy oddělené čárkou. Výrazy se vyhodnocují a jejich hodnoty se vypisují postupně na obrazovku. Nejčastěji budeme používat řetězce (write('AHOJ')), proměnné (write(A)), aritmetické výrazy (write(A+B)) a jejich kombinace.

Pokud budou mít proměnné hodnoty A=5, B=3, vypíšou následující příkazy texty uvedené v pravém sloupci.

```
write(A)                    5
write(A,B)                  5 3
write('AHOJ')               AHOJ
write(A+B)                  8
write('A+B')                A+B
write('Soucet je',A+B)      Soucet je 8
write(A, '+', B, '=', A+B)  5 + 3 = 8
```

Výpis začíná vždy od pozice kurzoru. Příkaz write(seznam výrazů) vypíše hodnoty výrazů a ponechá kurzor za posledním vypsaným znakem. Příkaz writeln(seznam výrazů) vypíše hodnoty výrazů a přesune kurzor na začátek nového řádku. Příkaz writeln pouze přesune kurzor na nový řádek.

2.3 Formátování výstupů

U všech výrazů v příkazu write (nebo writeln) lze určit na kolik pozic se má hodnota vytisknout na obrazovku (tj. kolik znaků se má použít). U výrazů typu real se navíc dá určit počet desetinných míst (tím se zbavíme semilogaritmického tvaru).

Počet pozic se určí tak, že za výraz napíšeme dvojtečku a celé nezáporné číslo (můžeme tedy i nulu). Toto číslo určuje počet požadovaných pozic. U čísel se do pozic započítávají všechny znaky, tedy i desetinná tečka, znaménko -, oddělovač E.

Pokud je toto číslo větší, než počet pozic potřebných k zápisu hodnoty výrazu, doplní se mezery zleva. Pokud je toto číslo menší než počet pozic potřebných k zápisu čísla, potom se u typů integer a real toto číslo ignoruje (tj. použije se takový počet pozic, který je k zápisu čísla potřeba). To znamená, že hodnoty integer zůstanou stejné, hodnoty real se zobrazí v semilogaritmickém tvaru se zkrácenou (zaokrouhlenou) mantisou.

Pokud se chceme zbavit semilogaritmického tvaru, napíšeme ještě jednu dvojtečku a za ní další celé nezáporné číslo. Tím určujeme počet desetinných míst. Toto lze ale použít pouze u hodnot typu real. Číslo se potom zapíše v desetinném tvaru zaokrouhlené na požadovaný počet desetinných míst.

Pro hodnoty A=123, X=23.5678 se vytisknout texty uvedené ve druhém sloupci.

```
write(A:6)       123                 doplnily se 3 mezery zleva
write(A:1)    123                    počet pozic se ignoroval
write(X:20)       2.3567800000E+01   doplnily se mezery zleva
write(X:9)    2.36E+01               zkrátila se mantisa (zaokrouhlení)
write(X:2)    2.4E+01                počet pozic se ignoroval
write(X:0)    2.4E+01                počet pozic se ignoroval
write(X:10:2)      23.57             2 desetinná místa, doplněno 5 mezer
write(X:0:2)  23.57                  2 desetinná místa, žádné mezery navíc
write(X:0:0)  24                     na jednotky, žádné mezery navíc
```

Upravte v programu CTVEREC tisk závěrečné výstupní věty tak, aby se hodnota A tiskla v desetinném tvaru zaokrouhlená na 1 desetinné místo a hodnota proměnné Obsah na 2 desetinná místa. Na počtu pozic nezáleží (to znamená za první dvojtečkou použijte 0). Vyzkoušejte i jiné formátovací hodnoty.

2.4 Příkaz vstupu - read

Příkaz vstupu má obecný tvar read(seznam proměnných), kde seznam proměnných jsou identifikátory proměnných oddělené čárkou (read(A), read(A,B), read(X,Y,Z)).

Při zpracovávání tohoto příkazu počítač přeruší činnost a očekává zadání hodnot (signalizuje to blikajícím kurzorem). Uživatel zadá hodnoty. Po zadání počítač přiřadí zadané hodnoty proměnným v seznamu.

Uživatel zadává hodnoty z klávesnice. Pokud jich zadává více, oddělují se mezerou. Během zadávání lze použít Backspace k mazání. Zadávání se ukončí a potvrdí klávesou Enter.

Uživatel musí zadat odpovídající hodnoty (tj. hodnoty odpovídajících typů). Např. pokud je proměnná typu integer, nesmí zadat desetinné číslo, pokud je proměnná integer nebo real, nesmí zadat text. Pokud zadá špatně, dojde k přerušení programu.

Může se stát, že uživatel zadá více nebo méně hodnot, než je požadováno. Mějme například příkaz read(A,B,C), tzn. očekává se zadání 3 hodnot. Pokud uživatel zadá méně hodnot (např.: 2 5), přiřadí se tyto hodnoty postupně proměnným (A:=2, B:=5) a očekává se zadání chybějících hodnot (opět bliká kurzor, uživatel musí zadat poslední hodnotu). Počítač se tedy ptá tak dlouho, až zadáme správný počet hodnot.

Pokud uživatel zadá více hodnot (např.: 2 5 7 4 3), přiřadí se v našem případě první tři hodnoty postupně proměnným A,B,C a ostatní nadbytečné hodnoty se zapamatují pro další použití. Pokud se později v programu vyskytne příkaz read, počítač se nebude ptát uživatele, ale použije tyto zapamatované nadbytečné hodnoty.

Také příkaz read může být ve tvaru readln(seznam proměnných). Příkaz read(seznam proměnných) si případné nadbytečně zadané hodnoty zapamatuje a později použije. Příkaz readln(seznam proměnných) případné nadbytečné hodnoty zahodí (tj. nepamatuje si je pro další použití a pokud narazí na další příkaz vstupu, zeptá se uživatele).

Může se také použít příkaz readln bez seznamu výrazů a bez závorky. Tento příkaz pouze čeká na stisk klávesy Enter.

Při úpravě programu zadávejte každou hodnotu zvlášť, tj. např. místo read(A,B,C) použijte read(A); read(B); read(C). Používejte readln(seznam výrazů) místo read(seznam výrazů). Vyhnete se problémům se zadáváním zejména řetězců.

Sestavte program pro výpočet obvodu a obsahu obdélníka o stranách a,b. Upravte program tak, aby měl na monitoru následující vzhled:

```
Obvod a obsah obdelnika
-----------------------

Velikost strany a : 4
                b : 2

Obdelnik ma strany 4.0 cm a 2.0 cm.
Obvod je   12.00 cm.
Obsah je    8.00 cm2.
```

Tučně zobrazené hodnoty jsou vstupní nebo výstupní.

Na konci program čeká, až uživatel stiskne libovolnou klávesu.

Vzorce pro výpočet obvodu a obsahu obdélníka a jejich zápis v Pascalu:

$o = 2(a+b)$ 2*(A+B)

$S = ab$ A*B

3 Výrazy – operace (VY_32_INOVACE_263)

Výrazy jsou zápisy posloupnosti operací. Mohou obsahovat:

- konstanty a proměnné (operandy),
- operátory, funkce,
- kulaté závorky.

Každý výraz musíme zapsat do řádku jako posloupnost znaků, což může být u matematických výrazů problematické.

Každý výraz má typ a hodnotu.

Typ výrazu je typ hodnoty, která je výsledkem vyhodnocení výrazu. Je dán deklarací proměnných nebo zápisem konstant (přímým nebo v deklaraci) a použitými operacemi a funkcemi.

Hodnota výrazu je dána v okamžiku provádění příkazu, který obsahuje výraz. Každá proměnná obsažená ve výraze musí mít předem definovanou hodnotu (přiřazovacím příkazem nebo příkazem vstupu), jinak má hodnotu náhodnou (v některých překladačích „nula“).

Konstanty a proměnné už umíme používat. Závorky se používají pro upřednostnění operace. Používají se pouze kulaté, mohou se do sebe vnořovat, neměli bychom používat nadbytečné závorky.

Jazyk Pascal používá operace unární (mají 1 operand) a binární (2 operandy). Operace rozdělíme do několika skupin.

Aritmetické operace jsou operace pro práci s čísly

+	sčítání
-	odčítání (resp. unární změna znaménka)
*	násobení
/	dělení
div	celočíselné dělení
mod	zbytek při celočíselném dělení

Zajímavé a možná pro vás nové jsou operátory pro dělení. Výsledek dělení 13:4 lze zapsat buď desetinným číslem (13:4=3,25) nebo se zbytkem (13:4=3 zb. 1). Výsledky Pascalských operací jsou 13/4=3,25, 13 div 4=3, 13 mod 4= 1.

U všech operací bychom měli vědět, jakého typu mohou být operandy (některé operace nejde s určitými typy provést) a jakého typu výsledky. Pro aritmetické operátory jsou tyto informace v následujících tabulkách:

operand	- (změna znaménka)
integer	integer
real	real

1. operátor	2. operátor	+ - *	/	div mod
integer	integer	integer	real	integer
integer	real	real	real	nelze
real	integer	real	real	nelze
real	real	real	real	nelze

Řetězcové operace jsou operace pro práci s texty (řetězci). V Pascalu je jediná řetězcová operace + s významem spojování řetězců. Je to binární operace, operandy mohou být typu řetězec nebo znak (char), výsledek typu řetězec a vznikne spojením dvou řetězců do jednoho. U této operace záleží na pořadí operandů.

Relační operace jsou operace pro porovnání hodnot: <, >, <=, >=, =, <>. Porovnávají vždy dvě hodnoty, jsou to tedy binární operace. Výsledek je typu boolean (určuje, jestli dané rovnost či nerovnost platí – true nebo ne – false).

Porovnávat lze:

- číselné typy mezi sebou,
- řetězce a char mezi sebou – neporovnává se podle délky řetězce, ale podle lexikografického uspořádání (podle „abecedy“, přesněji podle kódů znaků),
- boolean – false<true.

Logické operace jsou unární not (negace), a binární and (disjunkce), or (konjunkce) a xor (výlučné nebo). Operandy jsou typu boolean, výsledek také typu boolean.

Hodnotu výsledků operací určíme podle tabulek, které byste měli znát z matematiky.

X	not X
true	false
false	true

X	Y	X and Y	X or Y	X xor Y
true	true	true	true	false
true	false	false	true	true
false	true	false	True	true
false	false	false	False	false

Všechny binární operace jsou komutativní (nezáleží na pořadí operátorů). Jsou také asociativní, takže pro více operandů nemusíme psát závorky. Operace and je pravdivá, když jsou pravdivé všechny operandy. Operace or je pravdivá, když je pravdivý alespoň jeden operand. Operace xor v případě, že je pravdivý právě jeden operand.

4 Výrazy – funkce (VY_32_INOVACE_264)

Funkce „počítají“ a vracejí hodnotu. Jejich obecný tvar je: identifikátor(parametry). Jazyk Pascal obsahuje velké množství předdefinovaných funkcí pro práci s čísly, řetězci i dalšími datovými typy. Vlastní funkce si může definovat i uživatel. Z dostupných funkcí vybereme pouze některé aritmetické pro práci s čísly tak, abychom mohli realizovat základní matematické výpočty. Parametr (dále označený X) bude u těchto funkcí vždy pouze jeden (obecně jich může být více). Může jím být libovolný výraz přípustného typu (v našem výběru integer nebo real). Přehled funkcí je v tabulce:

abs(x)	absolutní hodnota	
sqr(x)	druhá mocnina	
sqrt(x)	druhá odmocnina	x>0
sin(x)	sinus	x v radiánech
cos(x)	kosinus	x v radiánech
trunc(x)	odseknutí desetinné části	trunc(2.37)=2 trunc(3.7)=3
round(x)	zaokrouhlení na jednotky	round(2.37)=2 round(3.7)=4
odd(x)	test na liché číslo	pro x liché je hodnota true, pro x sudé hodnota false

Tak jako u operací nás zajímají přípustné typy parametrů a typy výsledků.

typ parametru	abs sqr	sqrt sin cos	trunc round	odd
integer	integer	real	integer	boolean
real	real	real	integer	nelze

5 Výrazy - vyhodnocování výrazů (VY_32_INOVACE_265)

Každý výraz uvedený v programu se při běhu programu vyhodnocuje. Znamená to, že se postupně vykonávají jednotlivé operace a funkce. Musí být stanovena přesná pravidla, která určují pořadí vykonávání jednotlivých operací. Tato pravidla se v různých jazycích mohou lišit.

V Pascalu se nejprve vyhodnocují výrazy v závorkách počínaje od nejvnitřnějších. Stejnou prioritu mají funkce.

Dále rozhoduje priorita operátorů. Jsou rozděleny do 4 skupin od nejvyšší po nejnižší prioritu:

- unární operátory: not + -
- multiplikativní operátory: * / div mod and
- aditivní operátory: + - or xor
- relační operátory: = <> > < <= >=

Pokud je výrazu více operátorů se stejnou prioritou nebo obsahuje více závorek za sebou, řídíme se pravidlem zleva doprava (nejprve se vyhodnocuje to, co je v zápise více vlevo).

Při práci s výrazy byste měli umět:

- rozhodnout o přípustnosti výrazu,
- určit pořadí vyhodnocování operací a funkcí,
- určit typ výrazu,
- určit hodnotu výrazu.

Kromě hodnoty výrazu lze všechno určit ve fázi ladění programu. Hodnotu lze vypočítat až při běhu programu, protože je závislá na konkrétních hodnotách operandů. V dalších příkladech budeme vše určovat najednou.

Vyhodnoťte výrazy a určete jejich typ. Určete, které výrazy nelze vyhodnotit (jsou nepřípustné). Předpokládejme deklarace var A,B:real;I,J,K:integer;T:string; a hodnoty proměnných A=5.0, B=2.0, I=9, J=5, K=2, T=‘KONEC’.

20-2*3*3+4/2+3	
10+3*(2-4)+3.0	
A-I/J*K	
A*7+I div 2-J mod 3	
A*B-abs(A-B-3)	
T+’!’	
A+trunc(A/K+1)-round(K)	
A<J	
A div 2	
T=‘KONEC’	
A+T	
(A<3) and (A-B>0) and (I<A)	
odd(I+3) or (I<3)	
odd(A+B)	
odd(K-trunc(A/3))	

6 Výrazy - zápis výrazů (VY_32_INOVACE_266)

Výrazy se zapisují do řádku jako posloupnost znaků. Tím mohou nastat určité problémy oproti pravidlům, na které jste zvyklí u matematických výrazů.

Zapište v jazyku Pascal výrazy.

$2x+y$	2*X+Y	Znak násobení (*) musíme psát
$(a+b)c$	(A+B)*C	
$\frac{a+b}{c}$	(A+B)/C	Čitatel musí být v závorce
$\frac{a}{b+c}$	A/(B+C)	Jmenovatel musí být v závorce
$\frac{a.b}{c}$	A*B/C	Závorka v čitateli je zbytečná
$\frac{a}{b.c}$	A/(B*C) A/B/C	Ve jmenovateli závorka musí být nebo lze zapsat jako složený zlomek
a^3-b^2	A*A*A-B*B A*sqr(A)-sqr(B)	Obecná mocnina neexistuje
a^{-2}	1/sqr(A) 1/(A*A) 1/A/A	Záporný exponent změníme na kladný a přemístíme do jmenovatele
$\left(2x+\frac{a}{b}\right)^2$	sqr(2*X+A/B) (2*X+A/B)*(2*X+A/B)	Druhá možnost je správně, ale je neefektivní (zbytečně počítá dvakrát totéž)
$\sin^2 x$	sqr(sin(X)) sin(X)*sin(X)	Na druhou je sinus
$\sin x^2$	sin(X*X) sin(sqr(X))	Na druhou je x
$\sin^2 x^2$	sqr(sin(X*X)) sqr(sin(sqr(X)))	Na druhou je obojí

Ve výrazech se můžeme setkat s některými matematickými problémy.

V Pascalu není definována obecná mocnina. Pomůžeme si buď násobením, funkcí sqr, nebo si obecnou mocninu nadefinujeme jako funkci.

Z goniometrických funkcí jsou definovány pouze funkce sinus a kosinus. Funkce tangens a kotangens můžeme zapsat pomocí známých vztahů $tgx = \frac{\sin x}{\cos x}$, $\cot gx = \frac{\cos x}{\sin x}$.

Argumenty goniometrických funkcí je nutné zadat v radiánech. Pokud je úhel ve stupních, převedeme ho na radiány podle vztahu $x = \frac{\alpha\pi}{180}$. Hodnota π je v Pascalu definována jako funkce s identifikátorem PI.

Zapište v jazyce Pascal výrazy

$\frac{2x+3y}{2a}$	
$3\left[\left(a+\frac{x}{y}\right)^2-1\right]$	
$\frac{7{,}5y^2+4{,}8y}{6{,}62x^2+5}$	
$\frac{\frac{1}{a}+\frac{1}{b}}{c+d}$	
$\sqrt{\frac{a}{b+c}}$	
$\sin^2\alpha+\cos^2\beta$	
$\frac{1{,}2-\sqrt{3a+b}}{2}$	
$\frac{\lvert a+b\rvert}{x-0{,}5}$	
$\sqrt{1+tg^2\gamma}$	
$\cos\left\lvert(2x+p+2)\sqrt{x^2+px+q}\right\rvert$	

$a^2 - c + \cfrac{a}{b.c + \cfrac{c}{d + \cfrac{e}{f}}}$	
$\frac{-b - \sqrt{b^2 - 4ac}}{2a}$	
$\frac{4}{3}\pi r^3$	
$\frac{v}{3}\left(P_1 + P_2 + \sqrt{P_1 P_2}\right)$	
$\sqrt{1 - \sin(180° - \beta)}$	

7 Výrazy - podmínky a negace (VY_32_INOVACE_267)

Podmínky jsou výrazy typu boolean. Jejich hodnota je pravda (true) nebo nepravda (false). Musí tedy obsahovat operace a funkce, jejichž výsledkem je hodnota boolean. Jsou to relační a logické operace a funkce odd. Pomocí nich zapíšeme některé podmínky, které jsme zvyklí vyslovovat jinak.

Zapište v Pascalu podmínky.

$x \in \langle 0,1)$	
$x \notin (5,10\rangle$	
$x \in \{2,3,4\}$	
$x \notin \{1,2\}$	
$a \neq \pm 1$	
C je dělitelné 5	
B je sudé	
X je liché číslo větší než 1	
C je dělitelné 2 nebo 3	

Při tvorbě programů často napíšeme podmínku a budeme ji potřebovat negovat (tedy zapsat tak, aby její pravdivostní hodnoty vycházely obráceně). Podmínku lze jednoduše negovat tak, že ji napíšeme do závorky a před závorku napíšeme operátor not. Přehlednější (a efektivnější) bude podmínku skutečně negovat. K tomu použijeme následující pravidla pro relační a logické operátory.

Relační operátory tvoří dvojice ($=\ \neq$, $<\ \geq$, $>\ \leq$). První z operátorů je vždy negací druhého a obráceně.

Pro negaci negace použijeme pravidlo not not A = A.

Operace and a or se negují podle De Morganových zákonů: not (A and B) = not A or not B, not (A or B) = not A and not B.

Napište negace výrazů.

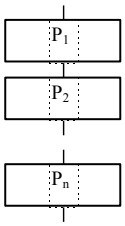
výraz	negace
A>0	A<=0
A+B>=C	A+B<C
X*Z=Z*A	X*Z<>Z*A
(A>0) and (B>0)	(A<=0) or (B<=0)
(A=0) or (B=0)	(A<>0) and (B<>0)
(X*Y<>0) and (X>0) and (Y>0)	(X*Y=0) or (X<=0) or (Y<=0)
not(A>B)	A>B

8 Strukturované příkazy (VY_32_INOVACE_268)

Prozatím jsme programy vytvářeli pouze z jednoduchých příkazů (přiřazovací, read, write). Abychom mohli naprogramovat i složitější výpočty, potřebujeme další příkazy, které umožní programy větvit nebo opakovat jejich části. K tomu jsou definovány řídící struktury, které se zapisují pomocí strukturovaných příkazů. Je dokázáno, že níže uvedených pět struktur stačí na poskládání libovolného programu (dokonce by stačily pouze tři).

V programování lze použít následující řídící struktury:

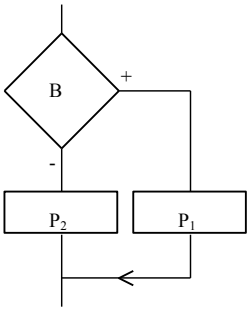
Posloupnost




```
begin P1;P2;...;Pn end
```

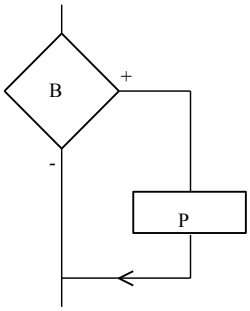
Větvení

úplné




```
if B then P1 else P2
```

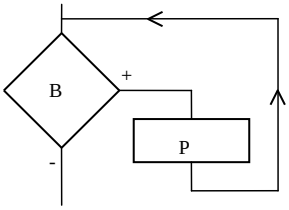
neúplné




```
if B then P
```

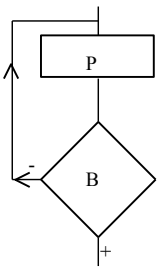
Cyklus

while




```
while B do P
```

repeat




```
repeat P until B
```

Zápisy $P, P_1, P_2, ... P_n$ v obdélnících znamenají libovolný příkaz (jednoduchý nebo strukturovaný), B je podmínka (výraz typu Boolean).

Posloupnost jsou příkazy, které se vykonávají po sobě v daném pořadí.

Podmíněný příkaz umožňuje rozvětvit program podle toho, zda bude nebo nebude splněna podmínka.

Cyklus je opakovaná činnost. Rozdíl mezi cyklem while a repeat je v tom, jestli podmínka, která rozhoduje o tom, zda se má cyklus ukončit nebo opakovat, je umístěna na začátku nebo na konci. Cyklus while má podmínku na začátku a tělo cyklu (příkazy, které se mají opakovat) tudíž nemusí proběhnout ani jednou. Cyklus repeat má podmínku na konci a tělo cyklu proto vždy alespoň jednou proběhne.

Kromě uvedených struktur jsou ještě v Pascalu k dispozici vícenásobné větvení case a cyklus for.

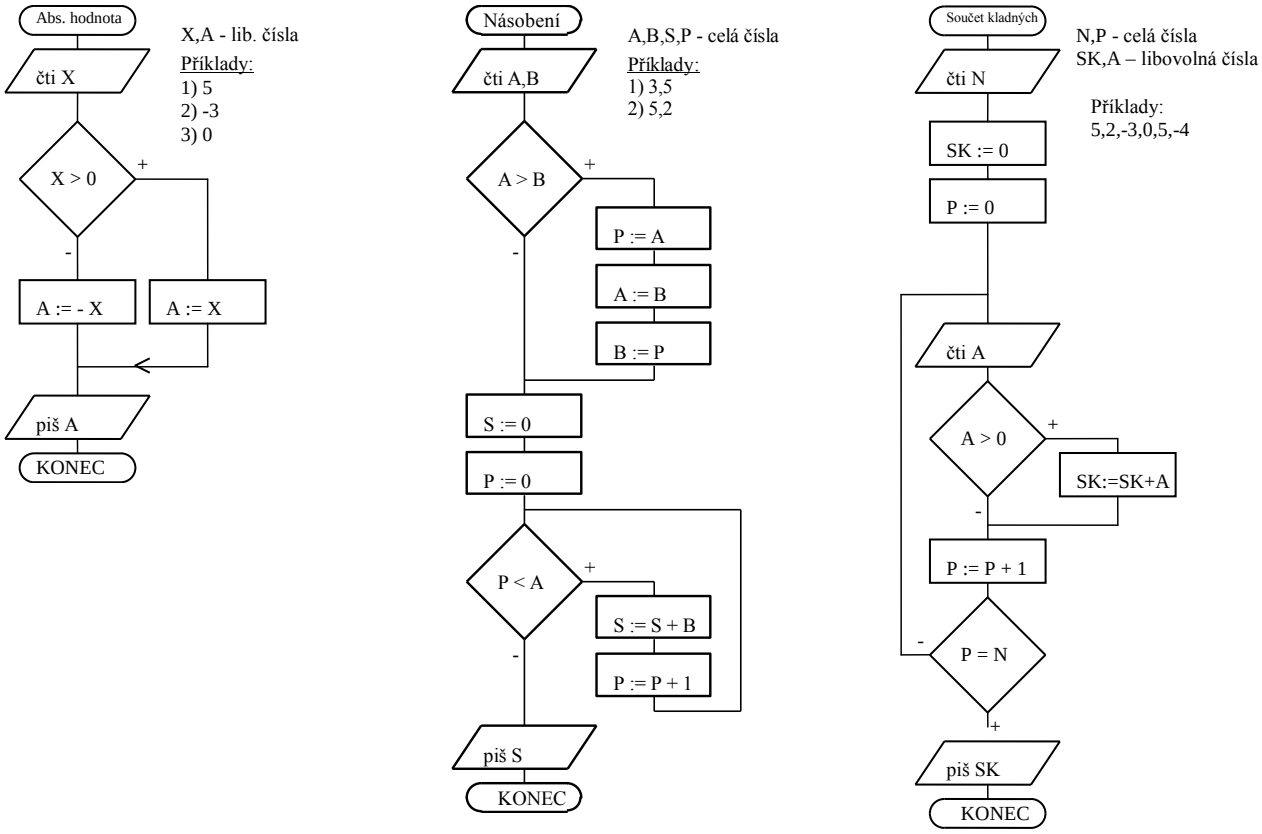
Místo obdélníků s příkazy si můžeme představit jakýkoli jednoduchý příkaz, ale také libovolnou z výše uvedených struktur. Všimněte si toho, že každá struktura má jeden začátek a jeden konec a lze ji tedy zaměnit s kterýmkoliv obdélníkem. Tímto způsobem lze do sebe struktury vnořovat. Programátorská metoda, které se říká strukturované programování, vyžaduje použití pouze těchto základních struktur a jejich vzájemné vnořování. Je dokázáno, že každý algoritmus lze tímto způsobem vytvořit.

U každé struktury je uveden její obecný zápis v jazyce Pascal. Ten použijeme při psaní programu, pouze doplníme konkrétní příkazy a podmínky. Pascal vyžaduje, aby byl program navržen podle zásad strukturovaného programování, jinak bychom program nezapsali.

9 Zápis programu (VY_32_INOVACE_269)

V kapitole 1 jsme napsali jednoduché programy, které obsahovaly pouze posloupnosti jednoduchým příkazů. Zde si ukážeme, jak zapsat programy, které obsahují větvení a cykly. Abychom program mohli zapsat v Pascalu, musí být vytvořen podle zásad strukturovaného programování, které je popsáno v předchozí kapitole. Pro úspěšný zápis musíme rozlišit, z jakých struktur se program skládá a jak jsou do sebe vnořeny.

Zapište v Pascalu algoritmy vyřešené ve formě vývojových diagramů.



Zatím nás nemusí zajímat, co dané algoritmy řeší. Pro úspěšný zápis v Pascalu potřebujeme pouze typy proměnných, které jsou uvedené v zadání. Připravené příklady použijeme až v další kapitole.

```
program ABSHODNOTA;
var
 X,A: real;
begin
 read(X);
 if X>0 then A:=X else A:=-x;
 write(A)
end.
```

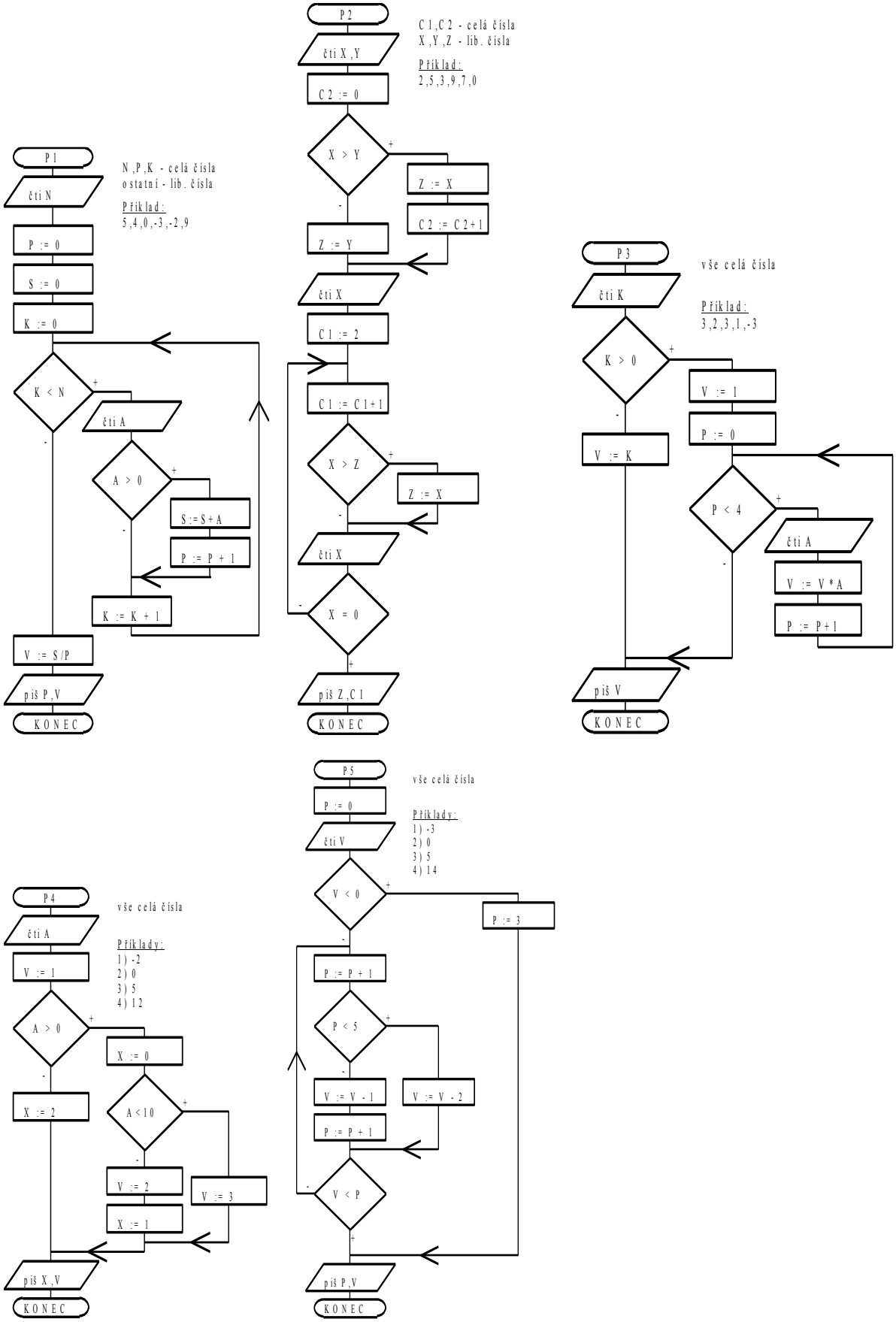
```
program NASOBENI;
var
 A,B,S,P: integer;
begin
 read(A,B);
 if A>B then begin P:=A; A:=B; B:=P end;
 S:=0; P:=0;
 while P<A do begin S:=S+B; P:=P+1 end;
 write(S)
end.
```

```
program SOUCET;
var
 N,P: integer;
 SK,A: real;
begin
 read(N);
 SK:=0; P:=0;
 repeat
  read(A);
  if A>0 then SK:=SK+A;
  P:=P+1
 until P=N;
 write(SK)
end.
```

Pokud je v příkazu vnořena posloupnost více příkazů, musíme je uzavřít do příkazové závorky begin-end, kterou uvnitř cyklu repeat můžeme vynechat.

Před else nesmí být nikdy středník.

Zapište v jazyce Pascal algortimy:

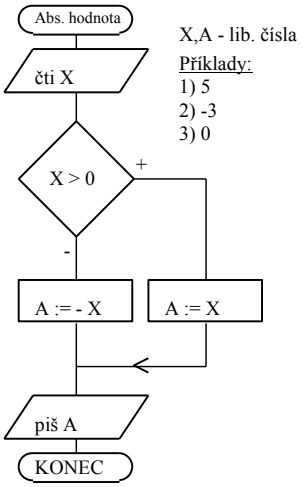


Řešte podobně jako tři ukázkové příklady. Výsledky jsou uvedeny na konci další kapitoly.

10 Testování programu (VY_32_INOVACE_270)

Správnost sestaveného algoritmu bychom měli testovat. Testování budeme dělat v testovacích (simulačních) tabulkách. Připravíme si vstupní hodnoty, pro které známe výsledky a budeme simulovat běh programu (vykonávat operace ve správném pořadí). Do tabulky budeme zapisovat postupné změny hodnot proměnných. Platná bude vždy poslední napsaná hodnota. Pro otestovaní správnosti algoritmu bychom si měli připravit více sad vstupních hodnot. Určitě tak, abychom se do každé části algoritmu alespoň v jednom příkladu dostali. Kromě běžných hodnot bychom měli testovat pro hodnoty okrajové, program někdy funguje např. pro všechna přirozená čísla, ale pro okrajovou hodnotu 1 je chybně. Abychom mohli správnost algoritmu ověřit, musíme už samozřejmě vědět, co má program spočítat.

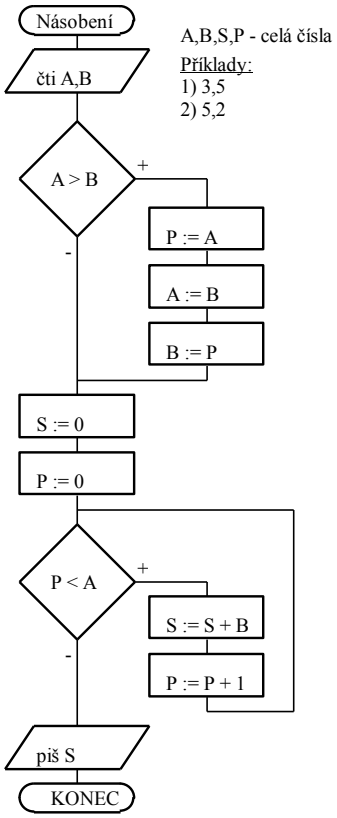
Testujte algoritmy z minulé kapitoly pro zadané hodnoty. K testování použijte uvedené příklady.



Algoritmus má počítat absolutní hodnotu zadaného čísla. Pro připravené příklady mají být výsledky 1) 5, 2) 3, 3) 0.

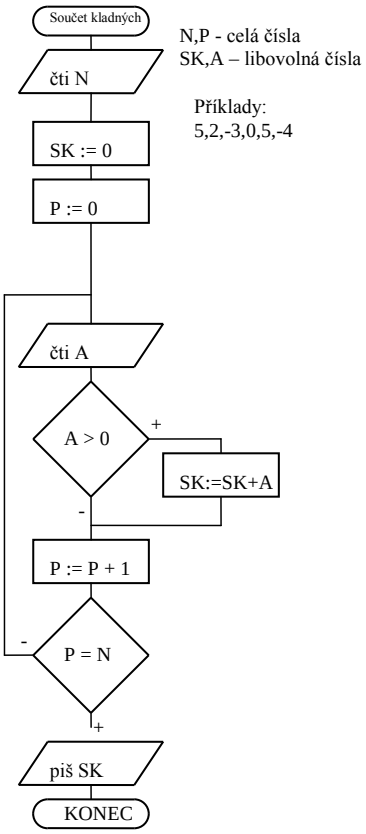
X	5	-3	0
A	**5**	**3**	**0**

Výsledky jsou v pořádku, algortimus počítá absolutní hodnotu.



Tento algoritmus má vynásobit dvě celá nezáporná čísla. Pro uvedené příklady by mělo vyjít: 1) 15, 2) 10.

A	3	5 2
B	5	2 5
P	0 1 2 3	5 0 1 2
S	0 5 10 **15**	0 5 **10**



Tento algoritmus má z řady zadaných čísel vybrat kladná a určit jejich součet. Jako první číslo se ale zadává počet zadávaných hodnot. V příkladu 5, 2, -3, 0, 5, -4 znamená první číslo 5 počet následujících hodnot. Výsledek (součet kladných) se počítá z hodnot 2, -3, 0, 5, -4. Výsledek by tedy měl vyjít 7.

N	5
SK	0 2 **7**
P	0 1 2 3 4 5
A	2 -3 0 5 -4

Simulujte v testovací tabulce algoritmy zadané v předchozí kapitole a určete výstupy.

Řešení je uvedeno včetně zápisu programu.

```
program P1;
var
 N,P,K: integer;
 A,S,V: real;
begin
 read(N);
 P:=0;
 S:=0;
 K:=0;
 while K<N do
  begin
   read(A);
   if A>0 then begin
                S:=S+A;
                P:=P+1
               end;
   K:=K+1
  end;
 V:=S/P;
 write(P,V)
end.
```

N	5					
P	0	1	2			
S	0	4	13			
K	0	1	2	3	4	5
A	4	0	-3	-2	9	
V	V					

```
program P2;
var
 C1,C2: integer;
 X,Y.Z: real;
begin
 read(X,Y);
 C2:=0;
 if X>Y then begin
              Z:=X;
              C2:=C2+1
             end
        else Z:=Y;
 read(X);
 C1:=2;
 repeat
  C1:=C1+1;
  if X>Z then Z:=X;
  read(X)
 until X=0;
 write(Z,C1)
end.
```

X	2	3	9	7	0
Y	5				
C2	0				
Z	5	9			
C1	2	3	4	5	

```
program P3;
var
 K,V,P,A: integer;
begin
 read(K);
 if K>0 then begin
              V:=1;
              P:=0;
              while P<4 do
               begin
                read(A);
                V:=V*A;
                P:=P+1
               end
             end
        else V:=K;
 write(V)
end.
```

K	3				
V	1	2	6	6	-18
P	0	1	2	3	4
A	2	3	1	-3	

```
program P4;
var
 A,V,X: integer;
begin
 read(A);
 V:=1;
 if A>0
  then begin
        X:=0;
        if A<10
         then V:=3
         else begin
               V:=2;
               X:=1
              end
       end
  else X:=2;
 write(X,V)
end.
```

A	-2	0	5		12	
V	1	1	1	3	1	2
X	2	2	0		0	1

```
program P5;
var
 P,V: integer;
begin
 P:=0;
 read(V);
 if V<0
  then P:=3
  else repeat
        P:=P+1;
        if P<5 then V:=V-2
               else begin
                     V:=V-1;
                     P:=P+1
                    end
       until V<P;
 write(VP)
end.
```

P	0 3	0 1	0 1 2	0 1 2 3 4 5 6
V	-3	0 -2	5 3 1	14 12 10 8 6 5

11 Podmíněný příkaz (VY_32_INOVACE_271)

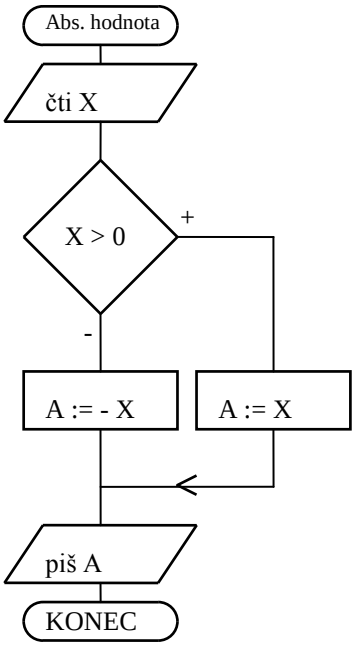
Podmíněným příkazem se samostatně nebudeme příliš zabývat. Projdeme pouze výpočet absolutní hodnoty a na něm si připomeneme jednotlivé etapy při tvorbě algoritmu, které jsou podrobněji popsány v kapitole 1. V závěru se domluvíme na základních pravidlech uživatelské úpravy programu.

Sestavte program pro výpočet absolutní hodnoty čísla X (bez použití funkce ABS).

Specifikace

```
Vstup:
      X     zadané číslo      real
Výstup:
      A     absolutní hodnota real
```

Vývojový diagram



Testování

Pro vstup 5 má vyjít 5, pro -3 výstup 3 a pro mezní hodnotu 0 má vyjít také 0. Testovací tabulka nám potvrdí správnost algoritmu.

X	5	-3	0
A	**5**	**3**	**0**

Program

```
program ABSHODNOTA;
```

```
var
 X,A: real;
begin
 read(X);
 if X>0 then A:=X else A:=-x;
 write(A)
end.
```

Dokumentace

Ke každému programu, který budeme probírat, se snažte mít dokumentaci, která může vypadat jako následující obrázek.

Sestavte program pro výpočet absolutní hodnoty zadaného čísla X. (Bez použití funkce ABS.)

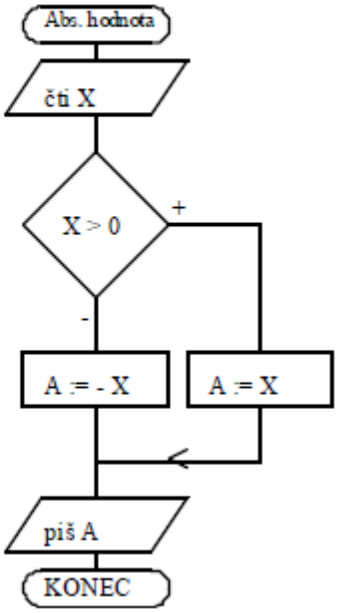
Vstup:

X	zadané číslo	real

Výstup:

A	absolutní hodnota	real

```
program ABSHODNOTA;
var X,A:real;
begin
  read(X);
  if X>0 then A:=X
         else A:=-X;
  write(A)
end.
```



Pro zpracování příkladů na počítači se domluvíme na základních pravidlech pro uživatelskou úpravu programu:

- čekání na stisk klávesy na konci,
- nadpis,
- zadávání hodnot s textovou výzvou,
- komentované výstupy,
- výstupy typu real v desetinném tvaru,
- výstupy typu boolean rozepsat (např místo write(B), kde B je proměnná typu Boolean napsat do upraveného programu např. if B then write(‘Ano’) else write(‘Ne’)),
- v textech nepoužívat háčky a čárky,

- stručné a jasné texty, bez chyb,
- nepoužívat barvy a zvuky.

12 Mocnina, cyklus (VY_32_INOVACE_272)

V tomto příkladu si

- zopakujeme postup při tvorbě programu,
- ukážeme, jak postupovat při tvorbě cyklu,
- vyřešíme problém Mocnina,
- ukážeme jak zajistit, aby nějaká činnost proběhla n-krát,
- ukážeme, jak převést cyklus while na repeat a naopak.

Sestavte program pro výpočet mocniny a^n (a je libovolné číslo, n je celé nezáporné číslo).

V rozboru úlohy si ujasněme, že mocninu lze spočítat jako opakovaný součin, tedy např. 25=2x2x2x2x2=32, kde násobíme 5x, obecně a^n=a x ... x a, kde násobíme n-krát. Chce se po nás tedy, aby se po zadání např. 2,5 dostali výsledek 32, pro hodnoty 5,1 výsledek 5 a pro dvojici 8,0 hodnotu 1. Rozbor úlohy zakončíme specifikací (tedy popisem vstupů a výstupů).

```
Vstup:
      A     základ      real
      N     exponent    integer, N>=0
Výstup:
      M     mocnina     real
```

Ve specifikaci píšeme názvy proměnných, jejich význam a typ. U vstupů také vstupní podmínku pokud je vstupní hodnota nějak omezena. Dále řešíme pouze za předpokladu, že je vstupní podmínka splněna. Vždy bychom se měli snažit, aby výsledky byly uložené ve výstupních proměnných (tedy ne pouze vypsané na obrazovce). Pojďme už vytvořit algoritmus.

Z rozboru vyplývá, že budeme opakovaně násobit, to znamená, že budeme cosi opakovat, takže budeme tvořit cyklus. Pokud tvoříme cyklus, je rozumně postupovat v pořadí:

- tělo cyklu – tj. činnost, která se opakuje,
- volba typu cyklu a vyslovení podmínky – tj. zajištění, aby cyklus proběhl tolikrát, kolikrát má a aby byl včas ukončen,
- nastavení počátečních hodnot,
- vše ostatní.

Základem (tělem cyklu) je opakování násobení, kdy násobíme vždy předchozí mocninu číslem A a zapamatujeme si výsledek jako novou mocninu (M:=M*A).

Musíme zajistit, aby se tato činnost opakovala n-krát. To uděláme tak, že si jako pomocnou proměnnou zavedeme počítadlo P, které po každém násobení zvětšíme o 1 (doplňme za specifikaci Pomocné proměnné: P ... počítadlo ... integer). Cyklus potom končí, až bude P=N. Na typu cyklu nebudeme zatím příliš přemýšlet a zvolíme cyklus repeat.

```
repeat
 M:=M*A;
 P:=P+1
until P=N;
```

Dalším krokem je nastavení počátečních hodnot. Každá proměnná, která je použita ve výrazu by měla mít předtím přiřazenu hodnotu, jinak se počítá s hodnotou náhodnou.Zkontrolujeme, jestli při prvním vstupu do cyklu budou mít všechny proměnné ve výrazech přiřazeny hodnoty. Výrazy jsou pro nás pravé strany přiřazovacích příkazů a celé podmínky. Z toho vyplývá, že by měly mít přiřazenu hodnotu proměnné M, A, P a N. Proměnné A a N jsou vstupy, takže hodnotu budou mít, M a P musíme před cyklem nastavit: P nastavíme na 0 (pokud jsme ještě nenásobili je počet násobení 0), M nastavíme na 1.

```
P:=0;
M:=1;
repeat
 M:=M*A;
 P:=P+1
until P=N;
```

Nakonec přidáme vše ostatní, což je v tomto případě pouze vstup a výstup hodnot.

```
read(A,N);
P:=0;
M:=1;
repeat
 M:=M*A;
 P:=P+1
until P=N;
write(M)
```

Algoritmus je zde zapsán přímo v Pascalu (zatím bez deklarací), v hodinách ho řešíme v podobě vývojového diagramu.

Následuje testování algoritmu v testovací tabulce. Pro testování si nachystáme příklady, pro které známe výsledky. Měli bychom si nachystat tolik příkladů, abychom se do každé části algoritmu dostali alespoň v jednom příkladu, dále bychom měli mít kromě „normálních hodnot“ nachystané testovací příklady pro mezní nebo okrajové hodnoty, u kterých by mohl být problém se správností algoritmu (v našem případě určitě pro N=0). Vyzkoušíme tedy výše uvedené příklady pro dvojice a výsledky: 2,5 – 32; 5,1 – 5; 8,0 – 1. Testovací tabulku doporučuji také dělat pro pochopení řešení příkladu.

Pro první dva příklady testování vychází správně, pro třetí příklad ne. Ve třetím příkladu se nám podařilo program tzv. zacyklovat (tj. teoreticky se bude cyklus opakovat donekonečna a nikdy neskončí). Proměnnou P=0 totiž zvýšíme o 1 na P=1 a pak už ji budeme pouze zvětšovat, takže nikdy nemůže být rovna N=0. Algoritmus je tedy správně pro všechny hodnoty kromě N=0.

Problém je v tom, že připustíme, aby tělo cyklu vůbec proběhlo. Mohli bychom to řešit podmínkou if N>0 před cyklem, ale lepší bude změnit typ cyklu. Rozdíl mezi cykly repeat a while je v tom, že cyklus repeat má podmínku na konci a cyklus while podmínku na začátku. Tělo cyklu repeat tedy vždy alespoň jednou proběhne, tělo cyklu while nemusí proběhnout ani jednou. A to je přesně to, co potřebujeme, změníme tedy cyklus repeat na while.

Změna jednoho typu cyklu na druhý je vcelku jednoduchá. Tělo cyklu opíšeme, cyklus zaměníme a podmínku negujeme (případně ji ještě můžeme upravit, pokud se nad programem více zamyslíme). Po změně typu cyklu tedy algoritmus vypadá takto:

```
read(A,N);
P:=0;
M:=1;
while P<>N do
 begin
  M:=M*A;
  P:=P+1
 end;
write(M)
```

Podmínku stačí napsat ve tvaru while P<N.

Můžete vyzkoušet v testovací tabulce, že teď už je algoritmus funkční.

V dalším kroku napíšeme celý program

```
program MOCNINA;
var A,M: real; N,P: integer;
begin
read(A,N);
P:=0;
M:=1;
while P<N do
 begin
  M:=M*A;
  P:=P+1
 end;
write(M)
end.
```

V základním zápise nebudeme používat uživatelské úpravy programu (tzn. aby to na obrazovce nějak vypadalo).

Až nyní jdeme na počítač, ladíme, doplňujeme uživatelské úpravy, to ale nyní nebudeme řešit

Z každého příkladu byste měli mít:

- zadání,
- specifikaci,
- algoritmus v podobě vývojového diagramu,
- program v Pascalu (bez uživatelských úprav).

Sestavte program pro výpočet součinu a x b pomocí sčítání (a,b jsou celá nezáporná čísla).

Máme vypočítat součin, aniž bychom použili násobení (nebo dělení). Podobně jako u mocniny převedeme výpočet na nižší operaci – 4 x 5 = 5+5+5+5 (nebo to můžeme i obrátit na 5 x 4 = 4+4+4+4+4, protože násobení je komutativní). Úspornější je první varianta, protože v ní sčítáme pouze 4x. Obecně výpočet vypadá takto: a x b = b+...+b, kde sčítáme a-krát. Je to tedy velmi podobné úloze na výpočet mocniny, pouze místo opakovaného násobení opakujeme sčítání a změní se názvy proměnných.

```
Vstup:
      A,B   činitelé    integer, A,B>=0
Výstup:
      S     součin      real
```

V řešení zvolíme přímo cyklus while. Počáteční hodnota S musí být na začátku nastavena na 0.

```
read(A,B);
P:=0;
S:=0;
while P<A do
 begin
  S:=S+B;
  P:=P+1
 end;
write(S)
```

Otestujte pro příklady: 3,5 – 15; 5,3 – 15; 1,4 – 4; 3,0 – 0.

Přidáme ještě úpravu, která zajistí časovou efektivitu. Cyklus se vždy opakuje A-krát, bylo by rozumné, aby v proměnné A byla vždy menší z hodnot A,B (je rozdíl, jestli sčítáme 3 krát 1000 nebo 1000 krát 3). Před cyklem zjistíme, která z proměnných A,B je menší a pokud to nebude A, tak proměnné zaměníme.

```
read(A,B);
if A>B then begin P:=A; A:=B; B:=P end;
P:=0;
S:=0;
while P<A do
 begin
  S:=S+B;
  P:=P+1
 end;
write(S)
```

Sestavte program pro výpočet mocniny a^n (n celé číslo).

Sestavte program pro výpočet součinu a.b pomocí sčítání (a,b celá čísla).

Sestavte program pro celočíselné dělení dvou přirozených čísel pomocí odčítání.

Sestavte program, který převede údaj v sekundách na hodiny, minuty a sekundy pomocí odčítání.

Sestavte program, který převede údaj v hodinách na týdny, dny a hodiny pomocí odčítání.

Sestavte program pro výpočet faktoriálu čísla N. (Faktoriál čísla N se zapisuje N! a je to součin všech kladných celých čísel od 1 do N, tj. například 6!=1.2.3.4.5.6=720). Dále je definováno, že 0!=1.

13 Spoření (VY_32_INOVACE_273)

Na účet s úrokem 5% bude vložena částka A Kč. Kolik bude na účtu za n let.

Všude budeme předpokládat, že úrok se nemění a počítá se na konci každého roku. Částka se vloží jednou a dále se bude pouze úročit. V cyklu budeme simulovat průběh jednotlivých let a sledovat stav na účtu. Částku na účtu si budeme pamatovat a průběžně počítat v proměnné C (která bude i výstupem). Každoročně tedy k částce C přičteme 5% z této částky, tj vynásobíme ji 1,05 (C:=1.05*C), protože C+5% z C=C+5*C/100=C*(1+5/100)=C*1.05.

```
Vstup:
      A     vložená částka    real; A>0
      N     počet let         integer, N>0
Výstup:
      C     částka za n let   real
```

Bude se opakovat příkaz C:=1.05*C. Opakovat se má n krát, podobně jako v příkladu na mocninu zavedeme počítadlo.

```
read(A,N);
C:=A;
P:=0;
repeat
 C:=1.05*C;
 P:=P+1
until P=N;
write(C)
```

U příkladů na spoření nebudeme dělat testovací tabulky.

Můžeme nechat cyklus repeat, protože cyklus alespoň jednou musí proběhnout (předpokládáme N>0).

Mohli bychom ušetřit proměnnou C a přímo měnit proměnnou A (která by byla zároveň výstupem), také P bychom nemuseli používat a zmenšovat N (N:=N-1) až by N bylo 0, obě původní hodnoty by se ale zapomněly.

Na účet s úrokem U% bude vložena částka A Kč. Kolik bude na účtu za n let.

Přibyl jeden vstup navíc (úrok si zadáme – např. 3,5% jako 3.5), jinak zůstává příklad stejný.

```
Vstup:
      A     vložená částka    real; A>0
      N     počet let         integer, N>0
      U     úroková míra      real, U>0
Výstup:
      C     částka za n let   real
```

Jedinou změnou je každoroční změna C - C+U% z C=C+U*C/100=C*(1+U/100) a více s tím neuděláme.

```
read(A,N,U);
C:=A;
P:=0;
repeat
 C:=(1+U/100)*C;
 P:=P+1
until P=N;
write(C)
```

Výpočet uvnitř cyklu není příliš efektivní a na takovéto výpočty bychom měli dát zvláště pozor, protože se v cyklu opakují. Hodnota 1+U/100 se počítá při každém průchodu cyklem, ale je stále stejná, protože hodnota U se nemění. Tuto hodnotu si spočítáme pouze jednou před cyklem, uložíme do pomocné proměnné K a uvnitř cyklu už budeme násobit pouze touto hodnotou.

```
read(A,N,U);
C:=A;
P:=0;
K:=1+U/100;
repeat
 C:=K*C;
 P:=P+1
until P=N;
write(C)
```

Na účet s úrokem U% bude vložena částka A Kč. Za kolik let bude na účtu částka X Kč?

Úloha se obrací. Zadáváme kolik chceme mít na účtu a výstupem je počet let, proměnná C se stává pomocnou.

```
Vstup:
      A     vložená částka    real; A>0
      U     úroková míra      real, U>0
      X     cílová částka     real, X>A
Výstup:
      P     počet let   integer

read(A,U,X);
C:=A;
P:=0;
K:=1+U/100;
repeat
 C:=K*C;
 P:=P+1
until C>=X;
write(P)
```

Na účet s úrokem U% bude každoročně vložena částka A Kč. Kolik bude na účtu za n let?

Na účet s úrokem U% bude každoročně vložena částka A Kč. Za kolik let bude na účtu částka X Kč?

Změna v těchto úlohách je oproti úlohám předchozím v tom, že se částka ukládá každoročně (vždy stejná). Každý rok tedy dochází ke dvěma operacím – přidání částky (předpokládejme, že na začátku roku) a úročení (předpokládejme, že na konci roku). Stačí tedy shrnout tyto 2 operace a místo C:=K*C počítat C:=K*(C+A). Vše ostatní zůstává stejné, pouze počáteční hodnotu C je třeba nastavit na 0 (jinak by se na začátku částka A započítala dvakrát).

Podobné úlohy by šly řešit pro jakoukoli pravidelnou procentuální změnu hodnoty (zvýšení nebo snížení) - např. za jak dlouho klesne hodnota výrobku na polovinu, jestliže jeho cenu obchodník sníží každý den o 5 %. V případě procentuálního poklesu ceny by se počítalo K:=1-U/100.

Všechny úlohy by šly řešit efektivněji jedním přiřazovacím příkazem podle vzorců z finanční matematiky.

Kromě cyklů while a repeat lze také v Pascalu (a nejen v něm) použít cyklus for. Rozdíl mezi cykly while a repeat je v tom, že tělo cyklu while nemusí proběhnout, zatímco tělo cyklu repeat vždy alespoň jednou proběhne (tělo cyklu for nemusí proběhnout). Cykly while nebo repeat je nutné použít, pokud dopředu počet opakování cyklu neznáme. Cyklus for je možné použít, pokud dopředu známe počet průchodů cyklem (tj. kolikrát se cyklus bude opakovat). Cyklus for nepřináší nic nového, šel by zapsat pomocí cyklů while nebo repeat a větvení, jeho zápis je však přehlednější a také jeho průběh je časově efektivnější. Nebudeme si uvádět jeho syntaxi v Pascalu (museli bychom probrat ještě nějaké pojmy navíc), ale pouze budeme u každého příkladu uvažovat, jestli by šel cyklus for použít. Z dosavadních příkladů by cyklus for šel použít u příkladů: mocnina, násobení, kolik za n let, nešel by použít u příkladů za kolik let částka X.

Hodnota zařízení C Kč se každoročně sníží o P %, jaká bude jeho hodnota za n let?

Hodnota zařízení C Kč se každoročně sníží o P %, za kolik let klesne jeho hodnota na polovinu původní hodnoty?

14 Číslice (VY_32_INOVACE_274)

Ve všech příkladech na číslice budeme předpokládat, že vstupy budou celá nezáporná čísla. Všechny proměnné budou typu integer (úlohy by šly řešit také pomocí řetězců). Zásadní bude postup, jak projít zápisem čísla zprava doleva a zjistit jednotlivé číslice.

Sestavte program pro výpočet ciferného součtu zadaného čísla.

```
Vstup:
      A     zadané číslo            integer; A>=0
Výstup:
      S     ciferný součet čísla A  integer
```

Pro číslo 2374 má vyjít jako výsledek 16. Budeme procházet číslo zprava doleva. Poslední číslici vypočítáme výrazem A mod 10 (zbytek při dělení 10). Aby při dalším průchodu byla na konci další číslice, zbavíme se poslední číslice příkazem A:=A div 10. Toto budeme opakovat až projdeme všechny číslice, tj. až bude hodnota A rovna 0. Základní algoritmus průchodu číslicemi v čísle:

```
repeat
 C:=A mod 10;
 A:=A div 10
until A=0;
```

Při každém průchodu je příslušná číslice v proměnné C. Doplníme pouze zpracování této hodnoty, aby se vypočítal ciferný součet.

```
read(A);
S:=0;
repeat
 C:=A mod 10;
 S:=S+C;
 A:=A div 10
until A=0;
write(S)
```

Doporučuji vyzkoušet v testovací tabulce (pro výše uvedený vstup a také pro nějaké jednociferné číslo a pro nulu).

Mohli bychom ušetřit proměnnou C a psát S:=S + A div 10. Uvědomme si, že v proměnné A se nepamatuje původní hodnota. Cyklus repeat může zůstat (není třeba převádět na while), dokonce by v některých dalších příkladech mohl být cyklus while chybný. Cyklus for použít nelze, protože dopředu neznáme počet opakování cyklu (je to vlastně počet cifer zadaného čísla). Pokud budete zadávat příliš dlouhá čísla, budou vám vycházet podivné výsledky. To je proto, že výpočet přesáhne maximální hodnotu typu integer. Pro vstupní číslo by bylo vhodnější použít jiný celočíselný datový typ s větším rozsahem.

Sestavte program, který určí počet cifer v zápise čísla.

Chceme určit, kolikaciferné číslo je. Pro číslo 2374 má vyjít 4, pro 12 - 2, 5 – 1, 0 – 1.

```
Vstup:
      A     zadané číslo      integer; A>=0
Výstup:
      P     počet cifer       integer
```

Tady nemusíme vůbec zpracovávat jednotlivé číslice, ale pouze zkracovat číslo a počítat, kolikrát jsme to udělali.

```
read(A);
P:=0;
repeat
 P:=P+1;
 A:=A div 10
until A=0;
write(P)
```

Opět vyzkoušejte v testovací tabulce.

Cyklus repeat musí zůstat, v cyklu while by byla chyba pro nulu (vyšla by nulaciferná).

Sestavte program, který určí kolikrát se v zápise čísla vyskytuje číslice 5.

Pro 25145 má vyjít 2, pro 2402 – 0.

```
Vstup:
      A     zadané číslo            integer; A>=0
Výstup:
      P     počet výskytů číslice 5 integer
```

Podobně jako v prvním příkladu budeme procházet číslice a pokud to bude 5 zvětšíme počitadlo.

```
read(A);
P:=0;
repeat
 C:=A mod 10;
 if C=5 then P:=P+1;
 A:=A div 10
until A=0;
write(P)
```

Tady by mohl být cyklus while, ale pokud bychom počítali počet nul, bylo by to s cyklem while špatně pro nulu.

15 Číslice 2 (VY_32_INOVACE_275)

Sestavte program, který určí, zda se v zápise čísla vyskytuje číslice 5.

Pro 25154 je odpověď ano, pro 2402 ne. Výstupem je logická hodnota pravda nebo nepravda, tedy hodnota typu boolean.

```
Vstup:
      A     zadané číslo      integer; A>=0
Výstup:
      B     je tam pětka?     boolean
```

Na začátku hodnotu B nastavíme na false (není tam pětka) a pokud na nějakou v čísle narazíme, překlopíme B na true.

```
read(A);
B:=false;
repeat
 C:=A mod 10;
 if C=5 then B:=true;
 A:=A div 10
until A=0;
write(B)
```

Tento postup není efektivní, protože pokud už na nějakou pětku narazíme, nemusíme testovat dále (a tento algoritmus prochází všechna čísla až do konce). Ukončíme cyklus buď když projdeme celým číslem (A=0) nebo když už narazíme na pětku (B=true).

```
read(A);
B:=false;
repeat
 C:=A mod 10;
 if C=5 then B:=true;
 A:=A div 10
until (A=0) or (B=true);
write(B)
```

Zkusíme ještě cyklus upravit z hlediska efektivnější práce s hodnotami typu boolean. Napsali jsme podmínku B=true. Ta je pravdivá (tj. má hodnotu true), když B je true a není pravdivá (tj. má hodnotu false), když B je false, hodnota podmínky je tedy vždy stejná jako hodnota B, takže místo B=true stačí psát pouze B (ušetřili jsme operaci). Další zlepšení lze zavést do zápisu podmíněného příkazu if C=5 then B:=true, místo tohoto příkazu můžeme napsat jediný přiřazovací příkaz B:= C=5, tj. hodnotu výrazu C=5 (boolean) přiřadíme do proměnné B.

```
read(A);
B:=false;
repeat
 C:=A mod 10;
 B:=C=5;
 A:=A div 10
until (A=0) or B;
write(B)
```

Pozor! Pokud bychom nechali v podmínce ukončení pouze A=0 (bez B), byla by úloha špatně. Počítala by, jestli číslo začíná pětkou.

Při náhradě příkazu if C=5 then B:=true příkazem B:= C=5 jsme se dopustili nepřesnosti, příkazu B:=C=5 totiž odpovídá příkaz if C=5 then B:=true else B.=false, což je rozdíl.

Nakonec ještě pro zajímavost nejefektivnější řešení

```
read(A);
while (A>0) and (A mod 10 <>5) do A:=A div 10;
B:=A mod 10 = 5;
write(B)
```

Sestavte program, který určí, kolikrát se v zápise čísla vyskytuje cifra C.

Sestavte program, který určí, zda se v zápise čísla vyskytuje cifra C.

Sestavte program, který v zápise čísla určí součet cifer na lichých pozicích (počítáno odzadu).

Sestavte program, který v zápise čísla určí součet cifer na lichých pozicích (počítáno odpředu).

Sestavte program, který určí k-tou cifru zprava v zápise čísla.

Sestavte program, který určí k-tou cifru zleva v zápise čísla.

Sestavte program, který ze zadaného čísla vytvoří nové číslo, které bude mít opačné pořadí číslic.

Sestavte program, který ze zadaného čísla vytvoří nové číslo, které bude mít zaměněnou první a poslední číslici.

16 Posloupnosti (VY_32_INOVACE_276)

Na vstupu bude uživatel zadávat řadu hodnot (jejich počet může být pokaždé jiný). Programy mají z těchto hodnot něco určit (např. je sečíst, určit počet kladných hodnot,...).

Sestavte program, který sečte n zadaných hodnot.

Pro zadaná čísla 2,4,1,5,3 se má vypočítat 15. Protože nevíme, kolik hodnot budeme mít, nemůžeme si všechny hodnoty pamatovat a až po jejich zadání počítat výsledek. To ale nepotřebujeme, výsledek můžeme počítat průběžně po zadání každé jednotlivé hodnoty (zkuste si představit, co byste dělali a počítali, kdyby vám tyto hodnoty někdo diktoval). Každou zadanou hodnotu vždy okamžitě zpracujeme a už si ji nemusíme pamatovat, v programu to bude znamenat, že můžeme postupně hodnoty načítat do stejné proměnné, pro všechny hodnoty tedy zavedeme pouze jednu vstupní proměnnou A. Specifikace je zatím neúplná, později ji rozšíříme.

```
Vstup:
      A     zadaná čísla      real
Výstup:
      S     součet čísel      real
```

Základem algoritmu (tělem cyklu) je opakování zadání hodnoty a jejího zpracování

```
read(A);
S:=S+A
```

V každém kroku se tedy bude zadávat jedna hodnota, musíme nějak zajistit, aby se včas skončilo (tj. až už uživatel žádné hodnoty nemá). To se dá zařídit různě, zvolíme jeden způsob. Na začátku zadáme, kolik máme hodnot (N) a potom zařídíme, aby se cyklus opakoval N-krát (pomocí počítadla, to už jsme dělali). Rozšíříme tedy vstupy ve specifikaci o proměnnou N.

```
Vstup:
      A     zadaná čísla      real
      N     počet čísel       integer, N>0
Výstup:
      S     součet čísel      real
```

Řada zadávaných hodnot tedy vypadá např. 5,2,4,1,5,3, přitom první hodnota je N (kolik mám čísel) a pak následuje N hodnot a pouze tyto hodnoty se zpracovávají, takže po zadání řady 5,2,4,1,5,3 má být výsledek 15 (nikoli 20).

```
read(N);
S:=0;
P:=0;
repeat
 read(A);
 S:=S+A;
 P:=P+1
until P=N;
write(S)
```

Vyzkoušejte v testovací tabulce.

Cyklus repeat můžeme nechat, podle vstupní podmínky vždy alespoň 1 hodnota bude zadána, šlo by použít cyklus for.

Podobně by se řešily příklady, které by z řady hodnot zjišťovaly jiné výstupy. Základ algoritmu by zůstal stejný.

```
read(N);
nastavení počátečních hodnot
P:=0;
repeat
 read(A);
 zpracování A
 P:=P+1
until P=N;
tisk výsledku
```

Sestavte program, který určí počet záporných z n zadaných hodnot.

```
Vstup:
      A     zadaná čísla      real
      N     počet čísel       integer, N>0
Výstup:
      PZ    počet záporných   integer
read(N);
PZ:=0;
P:=0;
repeat
 read(A);
 if A<0 then PZ:=PZ+1;
 P:=P+1
until P=N;
write(PZ)
```

Sestavte program, který určí součet lichých hodnot z n zadaných hodnot.

Vstupní hodnoty musí být integer.

```
Vstup:
      A     zadaná čísla      integer
      N     počet čísel       integer, N>0
Výstup:
      SL    součet lichých    integer
read(N);
SL:=0;
P:=0;
repeat
 read(A);
 if odd(A) then SL:=SL+A;
 P:=P+1
until P=N;
write(SL)
```

Sestavte program, který určí aritmetický průměr z n zadaných hodnot.

Aritmetický průměr se vypočítá jako součet hodnot dělený počtem hodnot. Řešíme jako příklad první příklad této kapitoly, ale součet je pomocná proměnná.

```
Vstup:
      A     zadaná čísla            integer
      N     počet čísel             integer, N>0
Výstup:
      AP    aritmetický průměr      integer

read(N);
S:=0;
P:=0;
repeat
 read(A);
 S:=S+A;
 P:=P+1
until P=N;
AP:=S/N;
write(AP)
```

17 Posloupnosti – maxima (VY_32_INOVACE_277)

Sestavte program, který určí největší z n zadaných hodnot.

```
Vstup:
      A     zadaná čísla      real
      N     počet čísel       integer, N>0
Výstup:
      MAX   největší číslo    real
```

Budeme si průběžně pamatovat dosud největší hodnotu (MAX). Po zadání nového čísla porovnáme toto číslo s hodnotou MAX a pokud bude větší, tak hodnotu MAX změníme. Na konci zůstane v proměnné MAX největší ze všech.

```
repeat
 read(A);
 if A>MAX then MAX:=A;
 P:=P+1
until P=N;
```

Problémem je nastavení počáteční hodnoty MAX. Nemůžeme ji nastavit např. na 0. Pokud by všechny hodnoty byly záporné, zůstala by v proměnné MAX 0, což je špatně. Do proměnné MAX bychom měli nastavit nejmenší možnou hodnotu, což by šlo, kdybychom o zadávaných hodnotách věděli něco bližšího (např. kdyby to byly výšky lidí v centimetrech, mohli bychom nastavit 0, pokud by to byly naměřené teploty vzduchu, mohli bychom dát nějakou zápornou hodnotu, která nikdy nemůže nastat, např. hodnotu absolutní nuly, obecně nejmenší možnou hodnotu nebo hodnotu menší). V naší úloze ale o hodnotách nic nevíme, takže nejmenší možné je mínus nekonečno a to do proměnné MAX dát nemůžeme, proto to musíme vyřešit jinak. První hodnotu si přečteme před cyklem a uložíme jí jako počáteční maximum do proměnné MAX. Zároveň ale musíme nastavit hodnotu počítadla P na 1 (nikoliv na 0), aby byl počet zadávaných hodnot v pořádku.

```
read(N);
read(A);
MAX:=A;
P:=1;
repeat
 read(A);
 if A>MAX then MAX:=A;
 P:=P+1
until P=N;
write(MAX)
```

Tento algoritmus po nás bude chtít vždy alespoň 2 hodnoty, což je v rozporu se vstupní podmínkou (ta předpokládá, že hodnota může být pouze 1). Změníme proto repeat na while.

```
read(N);
read(A);
MAX:=A;
P:=1;
while P<N do
```

```
 begin
  read(A);
  if A>MAX then MAX:=A;
  P:=P+1
 end;
write(MAX)
```

Pokud bychom hledali minimum, změnila by se pouze podmínka (if A<MIN then MIN:=A).

Sestavte program pro určení součtu kladných čísel z n zadaných čísel.

Sestavte program pro určení součtu záporných čísel z n zadaných čísel.

Sestavte program pro určení součtu sudých čísel z n zadaných čísel.

Sestavte program pro určení počtu záporných čísel z n zadaných čísel.

Sestavte program pro určení počtu lichých čísel z n zadaných čísel.

Sestavte program pro určení počtu sudých čísel z n zadaných čísel.

Sestavte program pro určení aritmetického průměru kladných čísel z n zadaných čísel.

Sestavte program pro určení aritmetického průměru záporných čísel z n zadaných čísel.

Sestavte program, který určí nejmenší hodnotu z n zadaných čísel.

Sestavte program, který určí nejmenší a největší hodnotu z n zadaných čísel.

Sestavte program, který určí největší hodnotu z n zadaných čísel a počet jejích výskytů.

Sestavte program, který určí nejmenší hodnotu z n zadaných čísel a počet jejích výskytů.

Sestavte program, který určí dvě největší hodnoty z n zadaných čísel.

Sestavte program, který určí dvě nejmenší hodnoty z n zadaných čísel.

Sestavte program, který určí největší hodnotu z n zadaných čísel a pořadí jejího prvního výskytu.

Sestavte program, který určí největší hodnotu z n zadaných čísel a pořadí jejího posledního výskytu.

Sestavte program, který určí nejmenší hodnotu z n zadaných čísel a pořadí jejího prvního výskytu.

Sestavte program, který určí nejmenší hodnotu z n zadaných čísel a pořadí jejího posledního výskytu.

18 Dělitelnost (VY_32_INOVACE_278)

V úlohách na dělitelnost budeme pracovat s celými kladnými čísly (tj. ne desetinná ani záporná ani 0). Pro test na dělitelnost použijeme operaci mod (zda zbytek po dělení je nulový).

Sestavte program, který vypíše všechny dělitele zadaného čísla.

```
Vstup:
      C     zadané číslo      integer; C>0
Výstup:
      D     dělitelé čísla C  integer
```

Zkoušíme dělitelnost čísla C všemi celými čísly od 1 do čísla C.

```
read(C);
D:=0;
repeat
 D:=D+1;
 if C mod D=0 then write(D);
until D=C;
```

Změna pořadí příkazů uvnitř cyklu ovlivní počáteční hodnotu D a podmínku.

```
read(C);
D:=1;
repeat
 if C mod D=0 then write(D);
 D:=D+1;
until D>C;
```

Mohli bychom vypisovat také obráceně, od největšího po nejmenší.

```
read(C);
D:=C+1;
repeat
 D:=D-1;
 if C mod D=0 then write(D);
until D=1;
read(C);
D:=C;

repeat
 if C mod D=0 then write(D);
 D:=D-1;
until D=0;
```

Všechny 4 verze mají stejnou časovou náročnost, tělo cyklu proběhne vždy C krát.

Lze řešit efektivněji, zkoušet dělitelnost pouze čísly od 1 do odmocniny z C (včetně) a vypisovat vždy dvojici C, C div D (pokud je C dělitelné D). Podmínka ukončení cyklu by tedy měla vypadat D>sqrt(C). V ní se ale porovnává celé číslo s hodnotou typu real, což může být nepřesné. Proto umocněním na tvar D*D>C dostaneme porovnávání hodnot typu integer, které je přesné. Pro čísla C, která jsou druhou mocninou nějakého celého čísla (např. 36, 25, 81,…) by se poslední dělitel vypsal dvakrát, proto je nutné vyjmout test odmocninou až za cyklus. Algoritmus by fungoval ve

tvaru repeat pro všechny hodnoty kromě 1 (vypsala by se dvakrát), proto je třeba cyklus repeat změnit na while.

```
read(C);
D:=1;
While D*D<C do
 begin
  if C mod D=0 then write(D, C div D);
  D:=D+1
 end;
if D*D=C then write(D)
```

Sestavte program, který určí počet dělitelů zadaného čísla.

Výstupem je proměnná P – počet dělitelů (integer), D je pouze pomocnou proměnnou. Místo výpisu dělitelů je budeme počítat.

```
read(C);
P:=0;
D:=1;
repeat
 if C mod D=0 then P:=P+1;
 D:=D+1;
until D>C;
write(P)
```

```
read(C);
P:=0;
D:=1;
While D*D<C do
 begin
  if C mod D=0 then P:=P+2;
  D:=D+1
 end;
if D*D=C then P:=P+1;
write(P)
```

Sestavte program, který určí počet vlastních dělitelů zadaného čísla.

Vlastní dělitelé jsou všichni dělitelé kromě tzv. samozřejmých dělitelů (tj. 1 a číslo samo). Oproti předchozímu příkladu stačí pouze upravit rozsah čísel, kterými zkoušíme dělitelnost od 2 do C-1, resp. od 2 do odmocniny z C (včetně). U čísel 1 a 2 by se program zacykloval, což by šlo řešit podmínkou D>C, ale pro číslo 2 by se započítal jeden dělitel, což je chybné. Proto je třeba i v první variantě změnit cyklus na while.

```
read(C);
P:=0;
D:=2;
while D<C do
 begin
  if C mod D=0 then P:=P+1;
  D:=D+1
 end;
write(P)
```

```
read(C);
P:=0;
D:=2;
while D*D<C do
 begin
  if C mod D=0 then P:=P+2;
  D:=D+1
 end;
if D*D=C then P:=P+1;
write(P)
```

Sestavte program, který vypíše všechny vlastní dělitele zadaného čísla.

Jednoduchou úpravou předchozího programu se vrátíme zpět k výpisu.

```
read(C);
D:=2;
while D<C do
 begin
  if C mod D=0 then write(D);
  D:=D+1
 end;
```

```
read(C);
D:=2;
while D*D<C do
 begin
  if C mod D=0 then write(D,C div D);
  D:=D+1
 end;
if D*D=C then write(D);
```

19 Dělitelnost 2 (VY_32_INOVACE_279)

Sestavte program, který určí, zda zadané číslo je prvočíslo.

Prvočísla jsou čísla, která mají pouze vlastní dělitele. Výjimkou je číslo 1, které sice nemá vlastní dělitele, ale není prvočíslem. Vstupem bude opět číslo C, výstupem proměnná boolean (označme ji B). Nejjednodušší (ale neefektivní) by bylo upravit program pro výpočet počtu vlastních dělitelů přidáním příkazu na konec if P>0 then B:=false else B=true (jednička by se musela ošetřit zvlášť). Efektivnější bude pracovat s proměnnou B přímo. Pokud už zjistíme, že je číslo něčím dělitelné, můžeme testování ukončit, neboť je jisté, že číslo není prvočíslem. Proměnnou B nastavíme na začátku na hodnotu true. Pokud bude něčím dělitelná, překlopíme ji na false a můžeme cyklus ukončit (upravíme podmínku přidáním B=true). Pokud ničím dělitelné nebude, dojdeme až po odmocninu z C a tam skončíme, v proměnné B zůstane true. Dělitelnost odmocninou nemusíme testovat zvlášť za cyklem, testujeme pouze dělitelnost a nemůže dojít k podobným problémům jako při výpisu a určování počtu dělitelů. Výsledek pro hodnotu C=1 je třeba určit zvlášť.

```
read(C);
B:=false;
if C>1 then
 begin
  B:=true;
  D:=2;
  While (D*D<=C) and (B=true) do
   begin
    if C mod D=0 then B:=false;
    D:=D+1
   end
 end;
write(B)
```

Sestavte program, který určí nejmenšího vlastního dělitele zadaného čísla.

Vstupem bude opět číslo C, výstupem proměnná MIN – nejmenší vlastní dělitel. Nejmenší vlastní dělitel nemusí existovat, což vyřešíme tak, že pro tento případ nastavíme MIN na hodnotu nula. Mohli bychom zvolit různé strategie hledání, zvolme nejefektivnější. Nastavíme MIN na 0. Budeme zkoušet dělitelnost všemi čísly od 2 do odmocniny z C (včetně). Jestliže bude číslo C některým z těchto čísel dělitelné, nastavíme MIN na toto číslo a můžeme cyklus ukončit. Pokud nebude, musíme dojít až po odmocninu, v proměnné MIN zůstane 0.

```
read(C);
MIN:=0;D:=2;
While (D*D<=C) and (MIN=0) do
 begin
  if C mod D=0 then MIN:=D;
  D:=D+1
 end;
write(MIN)
```

Sestavte program, který určí největšího vlastního dělitele zadaného čísla.

Vstupem bude opět číslo C, výstupem proměnná MAX – nejmenší vlastní dělitel a opět pokud nebude žádný dělitel existovat, nastavíme MAX na hodnotu nula. Mohli bychom zkoušet dělitelnost všemi čísly od C-1 do odmocniny z C (včetně). Jestliže bude číslo C některým z těchto čísel dělitelné nastavíme MAX na toto číslo a můžeme cyklus ukončit, pokud nebude, musíme dojít až po odmocninu, v MAX zůstane 0.

Efektivnější řešení bude, když si uvědomíme, že MAX=C div MIN. Na nejmenšího dělitele (pokud existuje) narazíme mnohem dříve než na největšího odzadu.

Proto si najdeme MIN jako v předchozím příkladě (MIN je pomocná proměnná) a pak jen spočítáme MAX:=C div MIN (samozřejmě za předpokladu, že MIN existuje).

```
read(C);
MIN:=0;
D:=2;
while (D*D<=C) and (MIN=0) do
 begin
  if C mod D=0 then MIN:=D;
  D:=D+1
 end;
if MIN>0 then MAX:=C div MIN else MAX:=0;
write(MAX)
```

20 Dělitelnost 3 (VY_32_INOVACE_280)

Sestavte program, který vypíše všechny společné dělitele zadaných čísel.

Vstupem budou čísla A,B, výstupem dělitelé D. Jako v předchozích výpisech budeme testovat dělitelnost obou čísel zároveň čísly od 1 do menšího z čísel A,B. Testování ukončíme v A, pokud bude A>B, prohodíme na začátku hodnoty A a B před cyklem.

```
read(A,B);
if A>B then begin D:=A; A:=B; B:=D end;
D:=1;
repeat
 if (A mod D=0) and (B mod D=0) then write(D);
 D:=D+1;
until D>A;
```

Sestavte program, který určí počet všech společných dělitelů dvou zadaných čísel.

Vstupem budou čísla A,B, výstupem proměnná P.

Upravíme předchozí příklad na počítání dělitelů.

```
read(A,B);
if A>B then begin D:=A; A:=B; B:=D end;
P:=0;
D:=1;
repeat
 if (A mod D=0) and (B mod D=0) then P:=P+1;
 D:=D+1;
until D>A;
write(P)
```

Sestavte program, který určí největšího společného dělitele dvou zadaných čísel.

Vstupem budou čísla A,B, výstupem proměnná NSD. Budeme hledat podobně jako jsme hledali nejmenšího vlastního dělitele. Dělitelnost budeme testovat čísly od menšího z čísel A,B po 1 (tedy odzadu). Je dobré si uvědomit, že NSD vždy existuje. Testování zastavíme, když na nějakého společného dělitele narazíme. Existují i mnohem efektivnější algoritmy.

```
read(A,B);
if A>B then begin D:=B else D:=A;
NSD:=0;
repeat
 if (A mod D=0) and (B mod D=0) then NSD:=D;
 D:=D-1;
until NSD>0;
write(NSD)
```

Sestavte program, který určí nejmenší společný násobek dvou zadaných čísel.

Vstupem budou čísla A,B, výstupem proměnná NSN. Pro určení NSN využijeme vztah A*B=NSD*NSN, takže NSN=A*B div NSD. Stačí tedy určit NSD jako v předchozím příkladu a pak jen spočítat NSN.

```
read(A,B);
if A>B then begin D:=B else D:=A;
NSD:=0;
repeat
 if (A mod D=0) and (B mod D=0) then NSD:=D;
 D:=D-1;
until NSD>0;
NSN:=A*B div NSD;
write(NSN)
```

Eukleidovým algoritmem (odčítání) určete největšího společného dělitele čísel A,B.

Eukleidovým algoritmem (dělení) určete největšího společného dělitele čísel A,B.

Sestavte program pro určení nejmenšího a největšího vlastního kladného dělitele čísla C.

Sestavte program, který vypíše na obrazovku prvních n prvočísel.

Sestavte program, který určí n-té prvočíslo.

21 Literatura

TÖPFEROVÁ, Dana, TÖPFER, Pavel. *Sbírka úloh z programování*. 1. vyd. Praha: Grada 1992. ISBN 80-85424-99-1.

PŠENČÍKOVÁ, Jana. *Algoritmizace*. 1. vyd. Kralice na Hané: Computer Media, 2007. ISBN 80-86686-80-9.

PŠENČÍKOVÁ, Jana. *Programování v Pascalu*. 1. vyd. Kralice na Hané: Computer Media, 2008. ISBN 978-80-86686-96-7.