Kapitola 2

Některé základní úlohy algoritmizace

Příklad 2.1.: Na vstupu máme tři celá vzájemně různá čísla A, B a C.
Definujte obsahy výstupních proměnných MAX, STR a MIN tak, aby
v proměnné MAX byla největší hodnota z A, B, C
v proměnné STR byla druhá největší (nejmenší) hodnota z A, B, C a
v proměnné MIN byla nejmenší hodnota z A, B, C.

Jednu variantu řešení vycházejících z postupného porovnávání obsahu dvou vstupních proměnných a stanovení určitého závěru v okamžiku, kdy lze tak učinit, nabízíme následujícím zápisem:

```
begin readln(A,B,C);
      if A > B then if   A > C   then begin   MAX:=A;
                                              if B > C   then   begin   STR:=B;
                                                                       MIN:=C
                                                                end
                                                         else   begin   STR:=C;
                                                                       MIN:=B
                                                                end
                                            end
                                       else begin   MAX:=C;
                                                    STR:=A;
                                                    MIN:=B;
                                            end
                    else if   B > C   then begin   MAX:=B;
                                                   if A > C   then   begin   STR:=A;
                                                                             MIN:=C
                                                                      end
                                                              else   begin   STR:=C;
                                                                             MIN:=A
                                                                      end
                                            end
                                       else begin   MAX:=C;
                                                    STR:=B;
                                                    MIN:=A;
                                            end;
          writeln(MAX, STR, MIN)
end.
```

Je zřejmé, že na uvedeném principu existuje velké množství různých algoritmů (můžeme začít např. dotazem **if** C > B $\Longrightarrow$ jiný postup řešení, nebo dotazem **if** B > C $\Longrightarrow$ další postup řešení problému atd.). Další algoritmy řešeného problému můžeme sestavovat na základě jiných strategií k přístupu řešení.

Tak např. zvolme strategii, která je o něco více logicky náročná než výše uvedený postup. Vychází ze setřídění dvojice hodnot a v zatřídění hodnoty třetí:

```
begin readln(A,B,C);
      if A > B      then   begin   MAX:=A; STR:=B
                           end
                    else   begin   MAX:=B; STR:=A
                           end;
      if C>MAX      then   begin   MIN:=STR; STR:=MAX; MAX:=C
                           end
                    else   if C >  STR then begin MIN:=STR; STR:=C
                                              end
                                   else MIN:=C;
    writeln(MAX, STR, MIN)
end.
```

Myšlenka a způsob realizace setřídění dvou hodnot je snad zřejmá každému. Maximální hodnotou MAX se stává buď hodnota první A, nebo hodnota druhá B a střední hodnotou STR buď hodnota druhá B, nebo hodnota první A v závislosti na výsledku relace A > B. Dokud třetí hodnota C nevstoupí „do hry“, nelze definovat třetí výstupní hodotu MIN.

Větší problém dělá zatřídění hodnoty třetí, která je umístěná v proměnné C: nelze ji porovnávat ani s hodnotou první umístěnou v proměnné A, ani s hodnotou druhou umístěnou v proměnné B; výsledek takovéhoto porovnání nám nic neříká. Musíme ji porovnat s hodnotou uloženou v nějaké výstupní proměnné, např. s hodnotou v proměnné MAX. Výsledkem relace C > MAX je poznání, zda C je největší, nebo zda C není největší. Pakliže je v C hodnota největší, tak patří do proměnné MAX, která je však obsazena. Jaká je hodnota v MAX? No přece větší hodnota z A a B. Co s ní? Zřejmě se tato hodnota stává hodnotou druhou největší a patří do proměnné STR. Obdobně dosud druhá největší hodnota se stává hodnotou nejmenší a je třeba ji z proměnné STR přestěhovat do proměnné MIN. Řešení v podobě po sobě jdoucích příkazů

MAX := C; STR := MAX; MIN := STR

je chybné a vede k uložení hodnoty C do proměnné MAX, poté do STR a nakonec i do MIN. Řešením je sekvence příkazů

MIN := STR; STR := MAX; MAX := C

pomocí kterých nejprve uvolníme proměnnou STR pro dosud největší hodnotu z MAX, poté můžeme uvolnit proměnnou MAX pro novou největší hodnotu C a tu tam nakonec zapsat.
Není-li hodnota v C největší, tj. podmínka C > MAX není splněna, je třeba se dále tázat, zda je C alespoň druhé největší (**if** C > STR) a postupujeme obdobně, jak jsme si právě ukázali.

V úvodní kapitole jsme uvedli názor na pojem „optimální algoritmus“. Nuže, nabízí se otázka: který algoritmus z výše dvou uvedených algoritmů je optimální? Algoritmus 1. nebo algoritmus 2.? Snad většina čtenářů cítí, že ten druhý. Bezesporu je elegantnější a je v něm myšlenka. Avšak podívejme se na oba algoritmy blíže. Oba se skládají pouze z podmíněných příkazů a přiřazovacích příkazů:

T1	**Překlad**		**Výpočet**			
			vhodné A,B,C		nevhodné A,B,C	
	podmínka	přiřazení	podmínka	přiřazení	podmínka	přiřazení
algor. 1.	5	16	2	3	3	3
algor. 2.	3	10	3(2)	3(5)	3	4

Již na prvý pohled je zřejmé, že druhý algoritmus je kratší. I z tabulky vyplývá, že obsahuje pouze 3 podmíněné příkazy a 10 přiřazovacích příkazů oproti 5 podmíněným a 16 přiřazovacím příkazům prvého algoritmu. Je tedy zřejmé, že doba potřebná k překladu druhého, kratšího, algoritmu do strojové řeči počítače bude kratší, než doba potřebná k překladu algoritmu prvého, delšího. Vzhledem k prováděným výpočtům však bude situace jiná, opačná. V prvním algoritmu je zapotřebí ke splnění požadovaného úkolu provést maximálně 3 podmíněné a 3 přiřazovací příkazy. Výpočet prováděný dle prvého algoritmu pro nejhorší kombinaci vstupních hodnot A, B a C je rychlejší, než výpočet dle 2. algoritmu pro optimální kombinaci vstupních hodnot A, B a C. Zobecnění na rozsáhlé algoritmy, ve kterých skutečně jde o čas, nechť si čtenář laskavě odvodí sám.

konec Příkladu 2.1 konec Příkladu 2.1

Příklad 2.2: Na vstupu jsou hodnoty představující platy zaměstnanců podniku. Požadujeme výstup průměrného platu v podniku.

Řešení spočívá v opakovaně prováděném

- získání vstupní hodnoty, tj. načtení informace o platu jednoho zaměstnance (readln(PLAT))
- zpracování načtené informace, tj.
  - přičtení platu k sumě dříve přečtených platů – matematický zápis

$$SUMA = \sum_{i=1}^{N} PLAT_i,$$

    je v PASCALu vyjádřen opakovaně prováděným příkazem

    SUMA := SUMA + PLAT

    vždy s jiným obsahem proměnné PLAT
  - zvýšení počítadla dosud zpracovaných platů (POCET := POCET + 1)

U řešení každého problému se vyskytují určité těžkosti na začátku algoritmu a na jeho konci. Na začátku algoritmu jsou pochopitelně všechny deklarované proměnné nedefinovány, tj. jejich obsahem není *známá hodnota*. Je zřejmé, že v našem algoritmu musí být na začátku, před zpracováváním prvního platu, uvedeny příkazy, kterými je vynulován obsah proměnných SUMA a POCET, tj.

SUMA := 0; POCET := 0;

Problém při ukončování algoritmu je problémem stanovení koncových podmínek. Kdy přestaneme se čtením a zpracováním platu, tj. s vykonáváním těla cyklu? Zřejmě poté, co jsme již všechny platy počítači odevzdali. To ovšem musí počítač nějak poznat. Řešení tohoto problému je v podstatě dvojí:

- Dohodneme se s počítačem na poslední vstupní hodnotě. Vzhledem k potřebě obecnosti algoritmu není vhodné, aby tato hodnota byla tvořena poslední konkrétní hodnotou určité aplikace, ale aby byla tvořena nějakou smluvenou, vzhledem k možným aplikacím nesmyslnou hodnotou.
- Na začátku, před čtením prvé zpracovávané hodnoty, poskytneme počítači informaci o počtu zpracovávaných hodnot.

Uvažujme dále případ, že počet vstupních čísel není na počátku znám, tj. musíme je počítat. Jádro algoritmu (tělo cyklu) budou tedy tvořit příkazy

```
write('Zadej plat:');
readln(PLAT);
SUMA:= SUMA + PLAT;
POCET:= POCET + 1;
```

Tyto příkazy budeme chtít provádět opakovaně - *v cyklu* - tak dlouho, dokud nezpracujeme všechny vstupní hodnoty.

Pro formulaci příkazu cyklu je nezbytné – neznáme-li na začátku kolik čísel je ke čtení (zpracovávání) – znát, jaké číslo je na konci. To je zpravidla zjistitelné, ovšem konkrétní hodnota je na újmu obecnosti algoritmu, neboť by se musel vždy pro tu kterou konkrétní hodnotu přizpůsobovat. Daleko častější je řešení algoritmu za předpokladu ukončení vstupních hodnot dohodnutou, fiktivní a zpravidla nesmyslnou hodnotou. V našem případě to může být hodnota 0 (nula). Pak řešení cyklu může mít tvar:

```
repeat   write('Zadej plat':');
         readln(PLAT);
         SUMA:=SUMA + PLAT;
         POCET:=POCET + 1
until    PLAT = 0;
```

Již víme, že při zpracování přiřazovacího příkazu musí být v jeho pravé části, výrazu, definovány obsahy všech použitých proměnných. Otázkou je, co chceme mít v proměnných SUMA a POCET při prvém zpracování příkazů SUMA:=SUMA + PLAT a POCET:=POCET+1. Odpověď je nasnadě: v těchto proměnných chceme mít 0. Další problém nastává po ukončení výše uvedeného příkazu cyklu **repeat - until** v okamžiku, kdy hodláme definovat obsah proměnné AP příkazem

AP:= SUMA/POCET;

Co se nachází v proměnných SUMA a POCET? V proměnné SUMA je zřejmě *součet všech* vstupních hodnot (tedy i fiktivní hodnoty 0) a v proměnné POCET je hodnota udávající *počet všech* vstupních hodnot (je tedy započítána i fiktivní hodnota 0). Zatímco příkaz

SUMA:= SUMA – PLAT

za příkazem cyklu nemá smysl (v proměnné PLAT je 0, ovšem při jakékoliv jiné koncové hodnotě (např. libovolném záporném čísle) by bylo nezbytné tento příkaz provést), příkaz

POCET:= POCET – 1

je nezbytný, neboť poslední číslo nebylo platem. Pak celý algoritmus *může* mít jeden ze tvarů:

```
a)
  begin SUMA:=0;
        POCET:=0;
        repeat write('Zadej plat:');
               readln(PLAT);
               if PLAT<>0 then
               begin SUMA:=SUMA + PLAT;
                     POCET:= POCET + 1
               end;
        until PLAT = 0;
        if POCET=0 then AP:=0
                   else AP:=SUMA/POCET;
        writeln('Průměrný plat je ',AP:8:2)
  end.
```

Algoritmus za předpokladu existence alespoň jedné hodnoty na vstupu (ostatně bez ní nemá význam algoritmus vůbec použít) a určitých logických korekcí může být uveden rovněž ve tvarech:

```
b)
begin SUMA:=0;
      POCET:=0;
      repeat write('Zadej plat:');
             readln(PLAT);
             SUMA:=SUMA + PLAT;
             POCET:= POCET + 1
      until PLAT = 0;
      POCET:= POCET - 1;
      AP:=SUMA/POCET;
      writeln('Průměrný plat je ',AP:8:2)
end.
```

```
c)
begin SUMA:=0;
      POCET:=-1;
      repeat write('Zadej plat:');
             readln(PLAT);
             SUMA:=SUMA + PLAT;
             POCET:=POCET + 1
      until PLAT = 0;
      AP:=SUMA/POCET;
      writeln('Průměrný plat je ',AP:8:2)
end.
```

V průběhu vytváření algoritmu je třeba naše jistě dobře míněné myšlenky ověřovat. Ověření správnosti algoritmu provádíme např. pomocí *sledovací tabulky*, do které zapíšeme všechny v algoritmu použité proměnné a na vzorku vstupních hodnot si hrajeme na počítač a vyplňujeme tabulku. Hodnoty získané simulací činnosti počítače nad programem porovnáme s očekávanými hodnotami získanými zkušebním (ručním) výpočtem. Shodují-li se, pak můžeme stanovit závěr, že algoritmus *pro konkrétní vzorek vstupních dat* pracuje podle očekávání.

Vzorek vstupních dat může mít např. tvar
2800 3600 2500 4600 2500 0

Postupně vytvářená sledovací tabulka je ve tvaru:

SUMA	POCET	PLAT	AP
0			
	0		
		2800	
2800	1	3600	
6400	2	2500	
8900	3	4600	
13500	4	2500	
16000	5	0	
	6		
	5		3200.0

Při tisku paměťového místa označovaného identifikátorem AP je v proměnných SUMA hodnota 16000, POCET hodnota 5, PLAT poslední přečtená hodnota 0 a AP hodnota 3200.0

Algoritmus je možno také zapsat s použitím příkazu **while - do**. Jedná se z hlediska algoritmického o správně navržený algoritmus problému zpracování posloupnosti informací, u které je známa koncová hodnota, která však již není předmětem věcného zpracování. Algoritmus může mít tvary:

```
d)
begin SUMA:=0;
      write('Zadej plat:');
      readln(PLAT);
      POCET:=1;
      while PLAT > 0 do
            begin SUMA:=SUMA + PLAT;
                  write('Zadej plat: ');
                  readln(PLAT);
                  POCET:=POCET + 1;
            end;
      AP:=SUMA/(POCET - 1);
      writeln('Průměrný plat je ',AP:8:2)
end.
```

nebo také tvar:

```
e)
begin SUMA:=0;
      POCET:=0;
      write('Zadej plat:');
      readln(PLAT);
      while PLAT > 0 do
            begin SUMA:= SUMA + PLAT;
                  POCET:=POCET + 1;
                  write('Zadej plat:');
                  readln(PLAT);
            end;
      AP:=SUMA/POCET;
      writeln('Průměrný plat je ',AP:8:2)
end.
```

Uvažujme nyní druhý případ, že počet vstupních čísel je znám na počátku tj. první vstupní hodnotou je informace o počtu N dále následujících platů. Pak možné řešení s využitím příkazu cyklu **for-to-do** je:

```
begin SUMA := 0;
      write('Zadej počet platů na vstupu: ');
      readln(POCET);
      for I:=1 to POCET do begin write('Zadej plat: ');
                                 readln(PLAT);
                                 SUMA:=SUMA + PLAT
                           end;
      AP:=SUMA/POCET;
      write('Průměrný plat je ',AP:8:2)
end.
```

Pochopitelně můžeme napsat také řešení s využitím příkazů **while-do** resp. **repeat-until**:

```
begin SUMA := 0;
      write('Zadej počet platů na vstupu: ');
      readln(POCET);
      I := 1;
      while I <= POCET do
            begin write('Zadej plat: ');
                  readln(PLAT);
                  SUMA:=SUMA + PLAT;
                  I:=I + 1;
            end;
      AP:= AP/POCET;
      writeln('Průměrný plat je ',AP:8:2)
end.
```

resp.

```
begin SUMA := 0;
      write('Zadej počet platů: ');
      readln(POCET);
      I := 0;
      repeat write('Zadej plat: ');
             readln(PLAT);
             SUMA:=SUMA + PLAT;
             I:=I + 1;
      until I = POCET;
      AP:= AP/POCET;
      writeln('Průměrný plat je',AP:8:2)
end.
```

Předchází-li všem výše uvedeným algoritmům hlavička programu a část definicí a deklarací např. ve tvaru:

```
Program  PRUMER;
    var  PLAT, POCET, I : word;
         SUMA : longint;
         AP : real;
```

pak hovoříme o programu v programovacím jazyku Pascal–Turbo.

Předchází-li všem výše uvedeným algoritmům hlavička programu a část definicí a deklarací např. ve tvaru:

```
Program  PRUMER(input,output);
    var  PLAT, POCET, I : integer;
         AP, SUMA : real;
```

pak hovoříme o programu ve standardním PASCALu, popř. o implementaci standardnímu PASCALu blízké (např. HP - Pascalu).

konec Příkladu 2.2 konec Příkladu 2.2

Příklad 2.3.: Sestavte algoritmus pro nalezení nejmenší hodnoty z řady čísel na vstupu. Řada čísel je ukončena dohodnutým číslem (např. číslem -13), které již do řady nepatří.

Řešení : obecný tvar vstupů je

$$a_1\ a_2 \cdots a_i\ \cdots\ -13$$

a prakticky může vypadat takto:

```
5  11  -7   8  -3  -13
```

přičemž hledanou hodnotou je hodnota -7 (minimální).

Jádrem algoritmu bude opět čtení hodnoty a dále zjišťování, zda přečtené číslo je nejmenší :

```
.
.
read(CISLO);
if CISLO < MIN then MIN := CISLO ;
.
.
```

Pokud je opravdu z dosud zpracovaných (přečtených a vyhodnocených) čísel řady právě zpracovávané číslo nejmenší, pamatujeme si je v paměti v místě označovaném identifikátorem MIN.

Výše uvedená část tvoří jádro celého algoritmu. Toto jádro se bude cyklicky opakovat tak dlouho, dokud ze vstupního souboru budeme číst hodnoty různé od -13.

```
.
.
read(CISLO);
while CISLO <> -13 do  begin  if CISLO < MIN then MIN := CISLO ;
                              read(CISLO)
                       end;
write(MIN);
.
.
```

Stejně jako u příkladu 2.2 je i nyní jádro algoritmu jasné a vyplývá ze zadání. Obtíže vznikají při ukončování algoritmu (vyřešili jsme testem na předem dohodnutou hodnotu koncového čísla) a při začátku algoritmu, lépe řečeno při zpracování první načtené hodnoty. Pozorný čtenář z uvedené části algoritmu již ví, že při prvém průchodu cyklem je hodnota v proměnné MIN nedefinována. Použijme obdobnou myšlenku jako u předchozího příkladu. Tam jsme hledali takovou počáteční hodnotu SUMA, pro kterou by platilo, že její součet s první načtenou hodnotou (v proměnné CISLO) vytvoří hodnotu, reprezentující součet jednoprvkové řady, tedy SUMA = 0 + CISLO. Nyní hledáme takovou hodnotu MIN, pro kterou by platilo, že jakékoliv číslo umístěné na začátku vstupního souboru a z něho načtené do proměnné CISLO bude chápáno jako minimální, tzn. požadujeme, aby platilo

$$\text{CISLO} < \text{MIN}$$

Otázka tedy zní : jaké číslo je určitě větší (obsah MIN) než kterékoliv libovolné číslo (obsah CISLO).

Odpověď : Číslo $+\infty$ (nekonečno) je určitě větší, než kterékoliv jiné číslo.

Jelikož ani člověk nemá představu o čísle reprezentující nekonečno, nemůže o tuto představu žádat počítač. Je zapotřebí zvolit konkrétní hodnotu, která by v daném případě měla charakter maximálně

nepříznivého čísla. Bude-li řada čísel tvořena hodnotami z intervalu $\langle -99, +99\rangle$, pak hodnotu +100 můžeme považovat za jakési $+\infty$. Budeme-li pracovat pouze se zápornými hodnotami, pak může nekonečno reprezentovat hodnota 0. Místo + resp. $-\infty$, hovoříme u počítače o maximálním resp. minimálním zobrazitelném čísle (např. u typu integer je to konstanta MAXINT).

Pak konečný tvar algoritmu pro hledání minimální hodnoty je

```
var  MIN, CISLO : integer;
begin  MIN:= 9999;
       write('Zadej celé číslo: ');
       readln(CISLO);
       while  CISLO <> -13 do begin if  CISLO <MIN then MIN:=CISLO;
                                    write('Zadej další číslo (konec je -13): ');
                                    readln(CISLO)
                              end;
       writeln('Minimální hodnota ze vstupních čísel je ',MIN)
end.
```

Je zřejmé, že námi uvedený algoritmus selže v případě, že v řadě vstupních hodnot nebude ani jedna hodnota menší jak 9999.

Jiná strategie konstrukce algoritmu řešícího zadání vychází z úvahy, že prvou hodnotu vstupního souboru chápeme jako počáteční hodnotu MIN.

Pak algoritmus

```
var  MIN, CISLO : integer;
begin  write('Zadej celé číslo: ');readln(MIN);
       write('Zadej další celé číslo: '); readln(CISLO);
       while  CISLO <> -13 do begin if  CISLO <MIN then MIN:=CISLO;
                                    write('Zadej další číslo (konec je -13): ');
                                    readln(CISLO)
                              end;
       writeln('Minimální hodnota ze vstupních čísel je ',MIN)
end.
```

je jedním z dalších možných řešení daného problému. Druhá možnost způsobu definování počátečního obsahu proměnné MIN je často používána, i když je méně obecná (předpokládá při řešení pomocí **while-do** v řadě alespoň jednu hodnotu, při řešení s využitím příkazu **repeat-until** alespoň dvě hodnoty v řadě - v obou případech kromě hodnoty koncové).

Opatříme-li poslední dva uvedené zápisy algoritmů a deklarací hlavičkou např. ve tvaru

Program HLAVICKA(input,output);

pak hovoříme o programech ve standardním PASCALu, popř. o programech v implementacích Pascalu blízkých standardnímu PASCALu (např. v HP Pascalu). Hlavička programu v Pascalu-Turbo nemusí být uvedena, bývá však zvykem kvůli přehlednosti ji ve zkráceném tvaru uvádět, např.

Program HLAVICKA;

konec Příkladu 2.3 konec Příkladu 2.3

Příklad 2.4.: Sestavte algoritmus pro nalezení dvou nejmenších hodnot z řady čísel na vstupu. Řada čísel je ukončena dohodnutým číslem (např. číslem -13), které již do řady nepatří.

Jelikož kromě nejmenší hodnoty, jak tomu bylo u příkladu 2.3, budeme hledat ještě i druhou nejmenší hodnotu z řady čísel na vstupu, rozšíříme oproti předchozímu příkladu deklarace o další proměnnou:

var MIN1,MIN2,CISLO :integer;

Jádrem algoritmu bude opět zpracování načtené hodnoty. Bude-li další načtenou hodnotou hodnota z dosud načtených a zpracovaných hodnot nejmenší, pak zřejmě patří do proměnné MIN1. Nepostačuje si ji ovšem zapamatovat, neboť příkazem

if CISLO<MIN1 **then** MIN1:=CISLO

zrušíme (přepíšeme) dosavadní obsah proměnné MIN1, který nahradíme novou nejmenší hodnotou. Ovšem původní nejmenší hodnota v MIN1 není na vyhození. Najdeme–li novou nejmenší hodnotu, pak dosud nejmenší hodnota by se měla stát druhou nejmenší hodnotou a patří do MIN2. Tuto myšlenku musíme do algoritmu začlenit a výše uvedený příkaz modifikovat na příkaz

if CISLO<MIN1 **then** **begin** MIN2:=MIN1;
MIN1:=CISLO
end

Co když právě zpracovávaná hodnota (CISLO) není nejmenší? Zřejmě může být ještě druhou nejmenší. Ostatní eventuality nás již nezajímají:

if CISLO<MIN1 **then** **begin** MIN2:=MIN1;
MIN1:=CISLO
end
else **if** CISLO<MIN2 **then** MIN2:=CISLO

Uvedený příkaz tedy zcela řeší zpracování každého přečteného čísla. Čtení vstupních hodnot i způsob algoritmizace opakování výše uvedeného příkazu je zcela stejné, jak bylo uvedeno u příkladu 2.3, tedy:

```
var    MIN1,MIN2,CISLO: integer;
begin  MIN1:=maxint;
       write('Zadej vstupní hodnotu:'); readln(CISLO);
       while CISLO<>9999 do begin if CISLO<MIN1 then begin MIN2:=MIN1;
                                                            MIN1:=CISLO
                                                      end
                                               else if CISLO<MIN2 then MIN2:=CISLO;
                                  write('Zadej další vstupní hodnotu (konec=9999):');
                                  readln(CISLO)
                            end;
       writeln('Nejmenší hodnotou ze vstupních čísel byla hodnota:'MIN1);
       writeln('Druhou nejmenší hodnotou ze vstupních čísel byla hodnota:'MIN2);
end.
```

Pozorný čtenář si jistě povšiml, že jsme na začátku algoritmu neuvedli další inicializační příkaz MIN2:=maxint. Jeho uvedení nic nepokazí, ovšem je zbytečné. Konstanta maxint (počítačové ∞ v oblasti hodnot typu integer, tj. maximální zobrazitelná hodnota typu integer) se do proměnné MIN2 dostane v okamžiku zpracovávání první načtené hodnoty. Jaká je to hodnota? Je určitě menší než maxint (∞ v oboru integer), tudíž se provádějí příkazy

MIN2:=MIN1; a MIN1:=CISLO.

Tím se dostane maxint do MIN2 a hodnota prvého načteného čísla do MIN1.

konec Příkladu 2.4 konec Příkladu 2.4

Příklad 2.5.: Na vstupu jsou dvojice reálných hodnot. Poslední dvojici tvoří hodnoty 0 0. Určete pořadová čísla těch dvojic, které jsou tvořeny jedním kladným a jedním záporným číslem.

Čtenou a zpracovávanou informací v daném okamžiku bude zřejmě dvojice čísel X a Y. Budeme číst a zpracovávat nejprve 1. dvojici, potom 2. dvojici ... I.dvojici.. až načteme poslední dvojici tvořenou dvěma nulami. Poněvadž potřebujeme znát, kolikátou dvojici zpracováváme, zavedeme si počítadlo I. Hodnotu počítadla vytiskneme vždy, když součin X $*$ Y bude číslo záporné (kladné číslo vynásobené záporným je záporné, v ostatních kombinacích je výsledkem číslo kladné). Můžeme tedy sepsat deklarace a algoritmus, které budou mít např. tvar:

```
var X,Y  :real;     I:byte;
begin write('Zadej první dvojici: '); readln(X,Y);
      I:=1;
      while(X<>0.0) or (Y<>0.0) do
           begin if X*Y<0.0 then writeln(' ':10,I,'. dvojice');
```

```
                I:=I + 1;
                write('Zadej další dvojici (konec nastává při 0 0):');
                readln(X,Y)
          end
end.
```

Za povšimnutí ještě stojí sestavení podmínky cyklu **while**. Ze zadání by snad někdo mohl zformulovat podmínku ve tvaru

(X<>0) **and** (Y<>0),

která by vedla k ukončení cyklu (a v tomto případě i celého programu) v okamžiku, kdy i pouze jedna z načtených dvou hodnot je nulová. To ovšem zadání odporuje. Správnou podmínkou je také podmínka ve tvaru

not ((X=0.0) **and** (Y=0.0)).

K algorimizaci řešeného problému můžeme použít i příkaz **repeat**–**until**:

```
var    X,Y:real; I:byte;
begin  I:=0;
       repeat write('Zadej dvojici čísel (konec - 0 0):'); readln(X,Y);
              I:=I + 1;
              if X*Y<0.0 then writeln(I,'. dvojice');
       until  (X=0) and (Y=0);
end.
```

Takto zformulovaný postup řešení je možný, poněvadž v tomto případě i zpracovávání poslední dvojice (fiktivních, koncových hodnot) nemá vliv na požadovaný výsledek.

konec Příkladu 2.5 konec Příkladu 2.5

Příklad 2.6.: Na vstupu jsou trojice celočíselných hodnot. Poslední trojici tvoří hodnoty 0 0 0. Určete aritmetický průměr každé trojice, počet lichých čísel mezi prvními čísly každé trojice a maximální hodnotu ze všech třetích čísel trojice.

Na vstupu je ke čtení vždy trojice hodnot A, B a C. Takovýchto trojic je blíže neurčený počet, na konci jsou tři nuly.

Požadované úkoly zřejmě řeší příkazy

AP:=(A + B + C)/3;

if odd(A) **then** PL:=PL + 1;

if C>MAX **then** MAX:=C.

První i třetí příkazy jsou již v této fázi stadia čtenáři zřejmě zcela jasné. Druhým příkazem zvyšujeme počítadlo lichých čísel (PL) ze všech hodnot postupně načítaných do proměnné A. K testu, zda obsahem proměnné A je liché číslo, využíváme standardní logickou funkci PASCALu. Stejný efekt, jako podmínkou odd(A) můžeme docílit ovšem i podmínkou (A **mod** 2) = 1
(tj. testem, zda zbytek po celočíselném dělení hodnoty A dvěma je jedna – tuto vlastnost mají lichá čísla) nebo podmínkou (A **div** 2) ∗ 2<>A, neboť tuto podmínku opět splňují pouze lichá čísla (operátor **div** je operátor celočíselného dělení pro dva celočíselné operandy). Požadované řešení zadaného příkladu může být ve tvaru:

```
var    A,B,C,PL,MAX,I: integer;
       AP            : real;
begin  PL:=0; MAX:= -maxint; I:=1;
       write('Zadej 1. trojici: '); readln(A,B,C);
       while(A<>0) or (B<>0) or (C<>0) do
            begin AP:=(A + B + C)/3;
                  writeln('Průměr ',I,'. trojice je:',AP:8:2);
                  if (A mod 2) = 1 then PL:=PL + 1;
                  if C>MAX then MAX:=C;
```

```
                write('Zadej další trojici (konec je 0 0 0):');
                readln(A,B,C); I:=I + 1
            end.
        writeln('Počet lichých ze všech A je:', PL);
        writeln('Hledaná maximální hodnota je:', MAX)
end.
```

konec Příkladu 2.6 konec Příkladu 2.6

Vzhledem k potřebě dodržet omezený rozsah tohoto učebního textu, není možné provádět verifikaci (ověření) *všech* zde navrhovaných řešení. V praxi ovšem toto ověřování především začátečníci provádí *po každém* novém návrhu postupu řešení a tak se dostávají postupně k funkčnímu algoritmu.

Uvedeme ještě sledovací (trasovací) tabulku algoritmu z Příkladu 2.6 za předpokladu, že vstupní hodnoty jsou ve tvaru:

```
3  -3  2
0   4  1
5   1  6
0   0 -3
8  -2  0
0   0  0
```

pak postupně vyplňovaná sledovací tabulka má tvar:

A	B	C	PL	MAX	I	AP
			0			
				-32767		
					1	
3	-3	2				
						0.6667
			1			
				2		
0	4	1				
					2	
						1.6667
5	1	6				
					3	
						4.0000
			2			
				6		
0	0	-3				
					4	
						−1.0000
8	-2	0				
					5	
						2.0000
0	0	0				
					6	
			2	**6**		