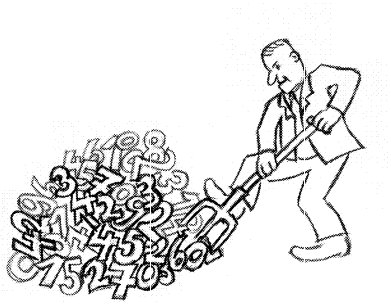
INFORMATIKA

Číselné operace s velkými celými čísly

STANISLAV TRÁVNÍČEK

Přírodovědecká fakulta UP, Olomouc



Celá čísla typu Integer mají rozsah od $-32\,768$ do $32\,767$, celá čísla typu Longint mají rozsah od $-2\,147\,483\,648$ do $2\,147\,483\,647$. S těmito hodnotami ve většině aplikací bohatě vystačíme. Přesto nás může zajímat, jak by bylo možno pracovat i s čísly vícecifernými, třeba sto- nebo i vícecifernými. A tu se již dostáváme k úlohám, které jsou vděčné při výuce programování a kterým se nyní věnujeme.

Sčítání

Úloha: Vytvořte program, který z textového souboru přečte dvě kladná celá čísla maximálně stociferná, vytvoří jejich součet a oba sčítance i se součtem zobrazí na displeji.

Jaké činnosti programu musíme zajistit? Nejprve se musíme postarat o přečtení a uložení zadaných čísel. (Pro jednodušší vyjadřování nebudeme dále rozlišovat pojmy „číslice“ a „jednociferné číslo“. Dále poznamenejme, že horní hranice počtu cifer nemusí být 100, my ji dále budeme označovat *MaxCif*.) U obou čísel budeme číst postupně jednotlivé číslice a budeme je ukládat do vhodně deklarovaného číselného pole (je třeba dát pozor na to, aby počet čtených cifer nebyl větší než deklarovaná veli-

kost pole). Pak u obou čísel najdeme číslice nultého řádu (vyhledání číslic nultého řádu může být jednoduché, když při čtení číslic ze souboru aktivujeme dvě počítadla počtu číslic) a postupně s rostoucím řádem je podle známého algoritmu sčítání sečítáme, přičemž nulté řády těchto součtů postupně „odzadu“ ukládáme do pole *Soucet* a případnou jedničku prvního řádu (při přechodu přes desítku) přičítáme k součtům číslic následujícího vyššího řádu. Musíme ovšem vyřešit, jak postupovat, když oba sčítanci mají různý počet cifer, tj. u kratšího sčítance už cifry dojdou, ale u delšího ještě máme pokračovat. Poslední, už radostnou prací je zajištění výstupu výsledků na obrazovku.

Tak nějak by se mohly odvíjet myšlenkové pochody žáků při programové analýze úlohy; realizace vlastního sčítání však může být různá. V následujícím programu *SoucetCisel* byl volen tento postup: program přečte ze zvoleného souboru oba sčítance do polí Clen[1, I], Clen[2, I] s počty cifer D[1], D[2]. Vynuluje se pole Soucet a přesune se do něho 1. sčítanec tak, že nultý řád sčítance je uložen v posledním prvku pole. Pak se k němu připočítává 2. sčítanec a to tak, že počínaje nultým řádem se sečtou vždy číslice téhož řádu, do příslušného řádu pole Soucet se vloží poslední číslice tohoto součtu a v případě, že součet je větší než 9, tj. při přechodu přes desítku, se přičte 1 k vyššímu řádu pole Soucet. Při sčítání se kontroluje počet zbývajících číslic 2. sčítance a po jejich vyčerpání se už jen reprodukují číslice 1. sčítance (případně nuly).

Jaký postup navrhnou vaši žáci?

```
program SoucetCisel;
{secita dve velka prirozena cisla}
const
  MaxCif = 100;
var
  I, J, Pomoc: Integer;
  JmenoS: string;
  F: Text;
  C: Char;
  D: array [1..2] of Integer;
  Clen: array [1..2, 1..MaxCif] of Integer;
  Soucet: array [0.. MaxCif] of Integer;

begin  {program}
```

```
  Write('Soucet cisel ulozenych v souboru: ');
  ReadLn(JmenoS);
  Assign(F, JmenoS);
  Reset(F);
{ Nacteni cisel }
  for I := 1 to 2 do
  begin
    D[I] := 0;
    repeat
      Read(F, C);
      Inc(D[I]);
      if D[I] > MaxCif then
      begin
        WriteLn(I, '. cislo mimo rozsah..');
        ReadLn;
        Close(F);
        Halt
      end;
      Clen[I, D[I]] := Ord(C) - Ord('0');
    until Eoln(F);
    ReadLn(F)
  end;
  Close(F);
{ Operace secitani }
  for I := 0 to MaxCif do
    Soucet[I] := 0;
  for I := 1 to D[1] do
    Soucet[MaxCif - D[1] + I] := Clen[1, I];
  for I := MaxCif downto 1 do
  begin
    J := D[2] - MaxCif + I;
    if J > 0 then
      Pomoc := Soucet[I] + Clen[2, J]
    else
      Pomoc := Soucet[I];
    Soucet[I] := Pomoc mod 10;
    Soucet[I - 1] := Soucet[I - 1] + Pomoc div 10
  end;
```

```
{ Vystup vysledku }
  WriteLn;
  for I := 1 to 2 do
  begin
    for J := 1 to D[I] do
      Write(Chr(Clen[I, J] + Ord('0')));
    WriteLn;
    case I of
      1: WriteLn('+');
      2: WriteLn('=')
    end
  end;
  J := 0;
  repeat
    Inc(J)
  until (Soucet[J] <> 0) or (J = MaxCif);
  for I := J to MaxCif do
    Write(Chr(Soucet[I] + Ord('0')));
  WriteLn;
  ReadLn
end.  {program}
```

Odčítání

Úloha: Vytvořte program, který z textového souboru přečte menšence a menšitele, tedy dvě kladná celá čísla maximálně stociferná (není známo, které z nich je větší), vytvoří jejich rozdíl a menšence, menšitele i rozdíl zobrazí na displeji.

Načtení čísel je stejné jako u součtu. Při odčítání bychom chtěli odčítat od většího čísla číslo menší a v případě, že menší je menšenec, přidáme k výsledku znaménko minus. Porovnáme tedy nejprve počet cifer (nepředpokládáme zadání s nulami na začátku, jako je např. 00012). Při rovnosti počtu cifer pak porovnáváme od nejvyššího řádu jednotlivé číslice. Při programování známého algoritmu odčítání opět postupujeme od nultých řádů a výsledky postupně ukládáme „odzadu“ do pole *Rozdil*; v případě, že číslice menšitele je větší než číslice menšence, zvětšíme tuto o 10, provedeme odečtení, nultý řád uložíme do pole *Rozdil* a následující vyšší řád menšence zmenšíme o 1. Řešení situace, kdy menšenec je menší než menšitel, může

být různé. V následujícím programu *RozdilCisel* byla vytvořena funkce VI (výměna indexů), která při výpočtu simuluje výměnu indexů obou čísel, tedy záměnu menšence a menšitele. Výsledky však již vystupují v zadaném pořadí.

```
program RozdilCisel;
{odcita dve velka prirozena cisla}
const
  MaxCif = 100;
var
  I, J : Integer;
  JmenoS: string;
  F: Text;
  C: Char;
  Zn: Boolean;
  D: array [1..2] of Integer;
  Clen: array [1..2, 1.. MaxCif] of Integer;
  Rozdil: array [0.. MaxCif] of Integer;

function VI(K: Integer):
Integer;   {vymena indexu}
begin {VI}
  if ZN then VI := 3 - K
else VI := K
end;  {VI}

begin {program}
  Write('Rozdil cisel ulozenych v souboru: ');
  ReadLn(JmenoS);
  Assign(F, JmenoS);
  Reset(F);
{ Nacteni cisel }
```

Tato část je shodná s předchozím programem.

```
{ Operace odcitani }
  Zn := (D[2] > D[1]);
  if D[1] = D[2] then
```

```
  begin
    I := 0;
    repeat
      Inc(I)
    until (Clen[1, I] <> Clen[2, I]) or (I = D[1]);
    Zn := Clen[2, I] > Clen[1, I]
  end;
  for I := 0 to MaxCif do
    Rozdil[I] := 0;
  for I := 1 to D[VI(1)] do
    Rozdil[MaxCif - D[VI(1)] + I] := Clen[VI(1), I];
  for I := MaxCif downto 1 do
  begin
    J := D[VI(2)] - MaxCif + I;
    if J > 0 then
      Rozdil[I] := Rozdil[I] - Clen[VI(2), J];
    if Rozdil[I] < 0 then
    begin
      Inc(Rozdil[I], 10);
      Dec(Rozdil[I - 1])
    end
  end;
{ Vystup vysledku }
  WriteLn;
  for I := 1 to 2 do
  begin
    for J := 1 to D[I] do
      Write(Chr(Clen[I, J] + Ord('0')));
    WriteLn;
    case I of
      1: WriteLn('-');
      2: WriteLn('=')
    end
  end;
  J := 0;
  repeat
    Inc(J)
  until (Rozdil[J] <> 0) or (J = MaxCif);
```

```
  if Zn then
    Write('-');
  for I := J to MaxCif do
    Write(Chr(Rozdil[I] + Ord('0')));
  WriteLn;
  ReadLn
end.  {program}
```

Výpočet rozdílu, kdy menšenec je menší než menšitel lze řešit i jinak. Např. tak, že nezjišťujeme, které z čísel je větší a „normálně“ provedeme předchozí algoritmus odčítání. Pokud by byl menšitel větší než menšenec, dostali bychom v `Rozdil[0]` hodnotu -1. V tom případě bychom všechny číslice výsledku museli vhodně „doplnit“ (přesvědčte se, jak).

Násobení

Úloha: Vytvořte program, který z textového souboru přečte dvě kladná celá čísla maximálně stociferná, vytvoří jejich součin a oba činitele i se součinem zobrazí na displeji.

Zde je třeba již v deklaracích uvážit, že maximální počet cifer součinu může být i dvojnásobný než je u jednotlivých činitelů. Přístup k řešení je tu stejný jako u předchozích programů, takže další komentáře už neuvádíme. Snad jen to, že v tomto případě je na počátku výpočtu pole *Soucin* v potřebném rozsahu vynulováno a postupně se do něj přičítají součiny 2. činitele jednotlivými číslicemi 1. činitele počínaje nultým řádem.

```
program SoucinCisel;
{nasobi dve velka prirozena cisla}
const
  MaxCif = 100;
var
  I, J, Pomoc: Integer;
  JmenoS: string;
  F: Text;
  C: Char;
  D: array [1..2] of Integer;
  Cinitel: array [1..2, 1.. MaxCif] of Integer;
  Soucin: array [1..2 * MaxCif] of Integer;
```

```
begin  {program}
  Write('Soucin cisel ulozenych v souboru: ');
  ReadLn(JmenoS);
  Assign(F, JmenoS);
  Reset(F);
{ Nacteni cisel }
```

Tato část je shodná s předchozími programy.

```
{ Operace nasobeni }
  for I := 1 to D[1] + D[2] do
    Soucin[I] := 0;
  for I := D[2] downto 1 do
    for J := D[1] downto 1 do
    begin
      Pomoc := Soucin[J + I] + Cinitel[1, J] * Cinitel[2, I];
      Soucin[J + I - 1] := Soucin[J + I - 1] + Pomoc div 10;
      Soucin[J + I] := Pomoc mod 10
    end;
{ Vystup vysledku }
  WriteLn;
  for I := 1 to 2 do
  begin
    for J := 1 to D[I] do
      Write(Chr(Cinitel[I, J] + Ord('0')));
    WriteLn;
    case I of
      1: WriteLn('x');
      2: WriteLn('=')
    end
  end;
  J := 0;
  repeat
    Inc(J)
  until (Soucin[J] <> 0) or (J = 2 * MaxCif) ;
  for I := J to D[1] + D[2] do
    Write(Chr(Soucin[I] + Ord('0')));
  WriteLn;
```

```
  ReadLn
end.  {program}
```

Předložené tři úlohy jsou nejen zajímavým, vhodným a nijak zvlášť obtížným cvičením v programování, ale navíc nutí programátora zamýšlet se více i nad matematickou stránkou známých algoritmů operací sčítání, odčítání a násobení. Umožňují také vymýšlet různé úpravy a rozšíření; např.

- sečítání a násobení více víceciferných čísel (vyřešit počet míst výsledku),
- výpočet 2. a 3. mocniny zadaného čísla.

Dělení

Úloha: V textovém souboru jsou uložena dvě čísla. Program má vypočítat a vytisknout na obrazovku jejich celočíselný podíl a zbytek dělení.

Vytvoření programu na dělení je již o něco obtížnější, neboť algoritmus používaný ve škole je tu nepoužitelný. Vhodné je použití postupného odčítání dělitele „od předu“, jak to dříve názorně předváděly mechanické kancelářské kalkulačky. Ukažme si příklad:

```
15674:483          1567⌋4          0
                   1084            1
                    601            2
                    118            3
                    1184⌋          30
                     701           31
                     218           32
                   zbytek          podíl
```

Tedy nastaví se zarážka ⌋ a od části dělence před zarážkou se odčítá dělitel, dokud to jde; počet těchto odečtení tvoří první cifru podílu. Pak se zarážka posune o jedno místo doprava a vytváří se druhá cifra, atd. Je více možností, jak upravit program. V předloženém *PodilCis* je volen tento postup. Načtení čísel je stejné jako u předchozích programů. Protože v algoritmu dělení se často rozhoduje, zda lze (dělitele) ještě odčítat, byla vytvořena booleovská funkce *LzeOdecist*, která to řeší metodou použitou u programu na odčítání. Také samotné odčítání dělitele je zpracováno touž metodou, a to zvlášť jako procedura *Odecti*. Ukazuje se účelné hned při přípravě dělení přesunout dělence do pole *Zbytek*, protože po realizaci celého algoritmu dělení zde skutečně zůstane zbytek dělení. Algoritmus

funguje tak, jak bylo ukázáno na příkladu. Při výpisu výsledků se zbavujeme nul na počátku čísel stejně jako u předchozích programů, jen je to zde soustředěno do funkce *PocatecniIndex* pro dvojnásobné použití.

```
program PodilCisel;
{deli dve velka prirozena cisla}
const
  MaxCif = 100;
var
  I, J, DP, Zar : Integer;
  JmenoS: string;
  F: Text;
  C: Char;
  D: array [1..2] of Integer;
  Clen: array [1..2, 1.. MaxCif] of Integer;
  Podil, Zbytek: array [0.. MaxCif] of Integer;

function LzeOdecist(I2: Integer): Boolean;
var Pom: Boolean;
    IL: Integer;
begin  {LzeOdecist}
  Pom := (Zbytek[I2 - D[2]] > 0);
  if not Pom then
  begin
    IL := 0;
    repeat
      Inc(IL)
    until (Zbytek[I2 - D[2] + IL] <> Clen[2, IL]) or
         (IL = D[2]);
    Pom := Zbytek[I2 - D[2] + IL] > Clen[2, IL]
  end;
  LzeOdecist := Pom
end;  {LzeOdecist}

procedure Odecti(IK: Integer);
var IA, IB: Integer;
begin  {Odecti}
  for IA := D[2] downto 1 do
```

```
  begin
    IB := IK + IA - D[2];
    Zbytek[IB] := Zbytek[IB] - Clen[2, IA];
    if Zbytek[IB] < 0 then
    begin
      Inc(Zbytek[IB], 10);
      Dec(Zbytek[IB - 1])
    end
  end
end;   {Odecti}

function PocatecniIndex(DD: Integer; XX: array of Integer):
Integer;
var PI: Integer;
begin  {PocatecniIndex}
  PI := 0;
  repeat
    Inc(PI)
  until (XX[PI] > 0) or (PI = DD);
  PocatecniIndex := PI
end;  {PocatecniIndex}

begin  {program}
  Write('Podil cisel ulozenych v souboru: ');
  ReadLn(JmenoS);
  Assign(F, JmenoS);
  Reset(F);
{ Nacteni cisel }
```

Tato část je shodná s předchozími programy.

```
{ Operace deleni}
  Zbytek[0] := 0;
  for I := 1 to D[1] do
    Zbytek[I] := Clen[1, I];
  Podil[1] := 0;
  DP := 1;
  if D[1] >= D[2] then
```

```
  begin
    Zar := D[2];
    repeat
      while (not LzeOdecist(Zar)) and (Zar < D[1]) do
      begin
        Inc(Zar);
        Inc(DP);
        Podil[DP] := 0
      end;
      while LzeOdecist(Zar) do
      begin
        Odecti(Zar);
        Inc(Podil[DP])
      end
    until Zar = D[1]
  end;
{ Vystup vysledku }
  WriteLn;
  for I := 1 to 2 do
  begin
    for J := 1 to D[I] do
      Write(Chr(Clen[I, J] + Ord('0')));
    WriteLn;
    case I of
      1: WriteLn('/');
      2: WriteLn('=')
    end
  end;
  for I := PocatecniIndex(DP, Podil) to DP do
    Write(Chr(Podil[I] + Ord('0')));
  WriteLn;
  WriteLn('Zbytek =');
  for I := PocatecniIndex(D[1], Zbytek) to D[1] do
    Write(Chr(Zbytek[I] + Ord('0')));
  WriteLn;
  ReadLn
end.  {program}
```

I zde mohou studenti přemýšlet o různých zlepšeních a také mohou vytvořit program, který zahrne všechny tyto operace.

(Autorkou úvodní ilustrace je Mgr. Jaroslava Čermáková z Hlinska v Čechách.)

Zpracování naměřených teplot

(Úlohy z MO – kategorie P, 16. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

(*Dokončení*)

3. Časově optimální řešení

Ze vstupu budeme postupně číst jednotlivé naměřené teploty a budeme si je ukládat do seznamu realizovaného v poli. Podobně jako u výše popsaného jednoduchého řešení budeme pole využívat cyklicky, abychom v něm uložené hodnoty nemuseli posouvat. S každou načtenou hodnotou si uložíme i její pořadové číslo. V seznamu si budeme uchovávat vždy jenom ty hodnoty, z nichž je třeba vybrat minimum. Ty nejstarší proto budeme průběžně odstraňovat. Na rozdíl od úvodního primitivního řešení úlohy však budeme ze seznamu průběžně odstraňovat také ty údaje o naměřených teplotách, které sice nejsou moc staré, ale které již nemohou nijak ovlivnit výsledek.

Když přečteme ze vstupu novou hodnotu a chceme ji vložit na konec seznamu, porovnáme ji nejdříve s číslem, které momentálně na konci seznamu je. Je-li číslo na konci seznamu větší než nově načtená teplota nebo je-li stejné, můžeme toto číslo z konce seznamu rovnou vyřadit. Je totiž starší a zároveň není menší než nově ukládaná hodnota, takže ve výsledku se již rozhodně neprojeví. Tímto způsobem můžeme odstranit z konce seznamu i několik uložených čísel najednou (v krajním případě třeba i všechna), než tam vložíme novou teplotu. Jak jsme ukázali, ta musí

být vždy větší než její předchůdce v seznamu. Seznam uložených hodnot tedy bude stále rostoucí.

Ze seznamu musíme odstranit také hodnotu, která je o K kroků starší než nově uložená teplota, pokud ovšem tato hodnota v seznamu ještě vůbec je. Jestliže tam skutečně je, je to nejstarší prvek seznamu a je tedy umístěn na samém začátku seznamu. Mohlo se ale stát, že tato hodnota již byla někdy dříve odstraněna při operaci popsané v předchozím odstavci. Abychom to poznali, ukládáme si do seznamu s každou naměřenou teplotou i její pořadové číslo, ze kterého snadno spočítáme „stáří“ této hodnoty.

Nyní nám už jen zbývá vypsat na výstup nejmenší teplotu obsaženou momentálně v seznamu. Protože je ale seznam vzestupně uspořádaný, je to snadné. Nemusíme seznam vůbec procházet a jeho hodnoty porovnávat, minimální číslo je v seznamu uloženo vždy na jeho začátku.

Paměťová složitost popsaného řešení je opět $O(K)$, neboť jako jedinou datovou strukturu používá seznam délky K uložený v jednorozměrném poli. Časová složitost je poněkud překvapivě pouze $O(N)$, tedy asymptoticky optimální. Lepšího času rozhodně nelze dosáhnout, neboť už jenom samotné vytvoření výstupu délky N nutně vyžaduje provést řádově N operací. Zpracování každého jednoho čísla ze vstupu má v nejhorším případě časovou složitost $O(K)$, jelikož může vyvolat postupné odstraňování až K čísel ze seznamu. To by nás mohlo vést k hrubému odhadu časové složitosti $O(N.K)$ na zpracování všech N čísel. Při postupném načítání N čísel ze vstupu je ale každé z těchto čísel právě jednou do seznamu vloženo a právě jednou ze seznamu odstraněno (až na posledních K čísel, která se již ze seznamu nemusejí odstraňovat vůbec) a každá z těchto operací má konstantní časovou složitost. Rovněž určení minima z uložených hodnot má v každém kroku výpočtu konstantní složitost. Časová složitost celého algoritmu je proto skutečně pouze $O(N)$.

Na závěr si ukážeme programovou realizaci výsledného řešení úlohy. Oproti předchozímu řešení je nejen asymptoticky rychlejší, ale program je také o hodně kratší a jednodušší.

```
program Teploty2;
{časově optimální řešení úlohy}

const MaxK = 1000;   {maximální přípustná hodnota K}
      Konec = -1000; {ukončovací hodnota na vstupu}
```

```
var  K: word;         {délka sledovaného úseku teplot}
     T: real;         {teplota}
     I: word;         {pořadové číslo aktuální teploty}
     A: array[1..MaxK] of  {seznam posledních teplot}
          record
            T: real;  {teplota}
            I: word   {její pořadové číslo}
          end;
     Zac, Kon: word;  {začátek a konec seznamu v poli A}
     Poc: word;       {počet prvků v seznamu A}

begin
write('Délka  sledovaného úseku K: ');
read(K);
writeln;
writeln('Posloupnost naměřených teplot ukončená -1000:');

Poc:=0;       {prázdný seznam}
Zac:=1;
Kon:=0;

read(T);      {první teplota}
I:=1;

while T <> Konec do
  begin
  {vynechat poslední větší teploty:}
  while (Poc > 0) and (A[Kon].T >= T) do
    begin
    dec(Poc);
    dec(Kon);
       if Kon = 0 then Kon:=MaxK
    end;

  {vložit novou teplotu do seznamu:}
  inc(Poc);
  inc(Kon);
    if  Kon  > MaxK then Kon:=1;
```

```
  A[Kon].T:=T;
  A[Kon].I:=I;

  {vynechat příliš starou teplotu:}
  if I-A[Zac].I = K then
    begin
    dec(Poc);
    inc(Zac);
    if Zac > MaxK then Zac:=1
    end;

  {vypsat minimum z teplot v seznamu:}
  writeln(A[Zac].T:8:2);

  {načíst novou teplotu:}
  read(T);
  inc(I)
  end;

writeln
end.
```

Literatura

[1] *Töpfer, P.:* Algoritmy a programovací techniky, Prometheus, Praha, 1995.

[2] *Kára, J. – Král', D. – Mareš, M.:* Recepty z programátorské kuchařky Korespondenčního semináře z programování – 1. část. RMF, r. 80 (2005) č. 1.

[3] *Töpfer, P.:* Optimalizace cesty (Úlohy z MO – kategorie P, 15. část). MFI r.16 (2006/07), č. 3.

[4] *Horák, K. a kol.:* 53. ročník matematické olympiády na středních školách, JČMF, Praha, 2006.