Aritmetické operátory

Když jsem chodil na základní školu (což bylo podle mých dětí někdy v dobách, kdy Zemi ještě ovládali dinosauři), musel jsem všechny výpočty dělat v duchu nebo na papíře. Žádné kalkulačky ještě nebyly, jen logaritmická pravítka. (Pozor! Jakmile byť jen připustíte, že vám výraz logaritmické pravítko něco říká, už budete za starého páprdu.)

Když začaly do školy chodit mé děti a nechal jsem je nad domácím úkolem něco spočítat, sáhly po kalkulačce. A když jsem jim řekl, aby to zkusily v duchu nebo na papíře, dostalo se mi pohledu někde na půli mezi úžasem a soucitem: „Takhle už to dneska nikdo nedělá!"

Možná mají pravdu. Když píšu program, ani já už to dneska takhle nedělám. Spolehnu se na tu nejrychlejší a nejpřesnější kalkulačku, kterou mám: na svůj počítač. Aritmetické výpočty se v programech objevují zcela běžně. Kromě toho, že počítače zvládnou uložit ohromné množství informací, umí také počítat mnohem rychleji a přesněji než my. Aritmetickým operátorům, které vám obrovský výpočetní výkon počítač zpřístupní, se budeme věnovat právě v této kapitole.

Aritmetické operátory

Operátor je symbol, který zastupuje nějakou akci. My už jsme se s operátory setkali v prvních kapitolách, v té předchozí to například byl operátor přiřazení (=). C++ obsahuje řadu aritmetických operátorů, konkrétně operátory pro sčítání, odčítání, násobení a dělení (viz tabulku 4.1).

Tabulka 4.1: Aritmetické operátory

Operátor	Význam	Příklad	Výsledek
`+`	sčítání	`5+2`	`7`
`-`	odčítání	`5-2`	`3`
`*`	násobení	`5*2`	`10`
`/`	dělení	`5/2`	`2`
`%`	zbytek po dělení	`5%2`	`1`

Operátor `%`, takzvané *modulo*, pro vás možná bude novinka. Počítá zbytek po dělení a podrobněji se mu budeme věnovat později v této kapitole.

Aritmetické operátory jsou většinou *binární*, což znamená, že vyžadují dva parametry neboli *operandy*. Ve výrazu `5+2` jsou pětka a dvojka operandy a znak `+` je operátor.

Poznámka: Ne všechny aritmetické operátory jsou binární. Například minus ve výrazu `-3` je unární operátor, protože vyžaduje jen jeden parametr (v našem případě trojku). Existují i operátory ternární, které vyžadují operandy tři.

Aritmetické operátory pracují s kladnými i zápornými čísly a (s výjimkou modula) s čísly celými i desetinnými. Operátor sčítání funguje kromě čísel i na řetězcích. Následující text vám každý z operátorů ukáže na skutečném programu pro evidenci studentů. Program vychází z mých praktických zkušeností získaných během výuky informatiky na Los Angeles Valley College.

Operátor sčítání

Na škole, kde učím informatiku, se většina studentů do přednášek registruje ještě před začátkem semestru. Někteří si ale přednášku přidají až v průběhu semestru. Následující program obsahuje dvě celočíselné proměnné, `registrovano` a `pridano`. Do proměnné `registrovano` uloží hodnotu, kterou uživatel zadá jako počet předregistrovaných studentů. Do proměnné `pridano` uloží počet studentů, kteří si přednášku zapsali v průběhu semestru, a pak obě proměnné pomocí operátoru `+` sečte. Součet uloží do proměnné `registrovano`, která bude na konci programu obsahovat celkový počet studentů zapsaných na danou přednášku – těch, co se registrovali předem, i těch přidaných dodatečně. Úplně na závěr program součet vypíše.

```
#include <iostream>
using namespace std;
int main(void)
{
    int registrovano, pridano;
    cout << "Zadejte pocet predregistrovanych studentu: ";
    cin >> registrovano;
    cout << "Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: ";
    cin >> pridano;
    registrovano = registrovano + pridano;
    cout << "Celkovy pocet studentu: " << registrovano << "\n";
    return 0;
}
```

Vstup a výstup programu vypadají například následovně:

```
Zadejte pocet predregistrovanych studentu: 30
Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: 3
Celkovy pocet studentu: 33
```

Spojení s operátorem přiřazení

Začínajícím programátorům někdy dělají potíže výrazy typu `registrovano = registrovano + pridano`, protože matematicky vzato se proměnná jen zřídka rovná součtu sebe sama s jiným číslem. My jsme ale v C++, nikoliv v matematice. Operátor `=` zde neznamená rovnost, ale přiřazení.

Výraz `registrovano = registrovano + pridano` se dá v C++ vyjádřit stručněji:

```
registrovano += pridano;
```

Z pohledu překladače jsou obě varianty stejné, většina programátorů dává přednost zápisu `registrovano += pridano`. Pro někoho je elegantnější, pro někoho čitelnější a někdo je rád, že si ušetří psaní.

Tato zkrácená varianta není jen doménou operátoru sčítání. Jak je patrné z tabulky 4.2, dá se použít i s ostatními aritmetickými operátory. (Deklaraci celočíselné proměnné `a` si domyslete.)

Tabulka 4.2: Spojení aritmetických operátorů s přiřazením

Příkaz	Zkrácená podoba
`a = a+2;`	`a += 2;`
`a = a-2;`	`a -= 2;`
`a = a*2;`	`a *= 2;`
`a = a/2;`	`a /= 2;`
`a = a%2;`	`a %= 2;`

Priorita aritmetických operátorů a přiřazení

Výraz `registrovano = registrovano + pridano` obsahuje dva operátory, přiřazení a sčítání. Sčítání má jakožto aritmetický operátor *vyšší prioritu* než přiřazení – dostane při vyhodnocování výrazu přednost. Už intuitivně vzato to dává větší smysl než opačná možnost. Jak ale vysvětlím později v této kapitole, priorita hraje roli i mezi různými aritmetickými operátory. Tam už správné pořadí tak zjevné není.

Přetečení a podtečení

Při sčítání může dojít k přetečení a podtečení. Datový typ `int` má na mém operačním systému a překladači rozsah od –2 147 483 648 po 2 147 483 647. Takhle by dopadlo, kdyby má přednáška začala registrací 2 147 483 647 studentů a následně se v semestru přidal jeden navíc:

```
Zadejte pocet predregistrovanych studentu: 2147483647
Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: 1
Celkovy pocet studentu: -2147483648
```

Záporný vstup u tohoto programu nedává smysl, protože záporný počet studentů se na přednášku nepřihlásí. Jiné programy ale záporná čísla používají, například pro teplotu pod nulou. Takhle dopadne, když pomocí záporných čísel způsobíme podtečení:

```
Zadejte pocet predregistrovanych studentu: -2147483648
Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: -1
Celkovy pocet studentu: 2147483647
```

Sčítání řetězců

Při zmínce o sčítání se nám sice automaticky vybaví čísla, ale operátor sčítání se dá použít i na řetězce. Následující program vypíše `Jmenujete se JeffKent`:

```
#include <iostream>
#include <string>
using namespace std;
int main(void)
{
    string jmeno = "Jeff";
    string prijmeni = "Kent";
    cout << "Jmenujete se " << jmeno + prijmeni << "\n";
    return 0;
}
```

Sečíst dva řetězce tedy znamená připojit druhý na konec prvního. Sčítat můžete jen čísla s čísly a řetězce s řetězci. Pokud se pokusíte sečíst číslo s řetězcem, překladač ohlásí chybu. Operátor sčítání umí sčítat čísla s čísly a řetězce s řetězci; číslo k řetězci přičíst nedokáže.

Operátor odčítání

Na naší škole studenti přednášek nejen přibývají, ale i ubývají. Některé studenty vyškrtnu, protože na přednášky vůbec nezačnou chodit; jiní si přednášku vyzkouší a pak ji zruší.

Následující program vychází z předchozí ukázky sčítání. Přidává celočíselnou proměnnou `zruseno` pro počet studentů, kteří si přednášku zrušili nebo vůbec nezačali chodit. Hodnotu proměnné `zruseno` zadá uživatel, program ji odečte od proměnné `registrovano`.

```
#include <iostream>
using namespace std;
int main(void)
{
    int registrovano, pridano, zruseno;
    cout << "Zadejte pocet predregistrovanych studentu: ";
    cin >> registrovano;
    cout << "Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: ";
    cin >> pridano;
    registrovano += pridano;
    cout << "Zadejte pocet studentu, kteri si prednasku zrusili: ";
    cin >> zruseno;
    registrovano -= zruseno;
    cout << "Celkovy pocet studentu: " << registrovano << "\n";
    return 0;
}
```

Vstup a výstup programu může vypadat například následovně:

```
Zadejte pocet predregistrovanych studentu: 30
Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: 3
Zadejte pocet studentu, kteri si prednasku zrusili: 5
Celkovy pocet studentu: 28
```

Ve výrazu `registrovano -= zruseno` jsme použili spojení aritmetického operátoru `-` s operátorem přiřazení, abychom nemuseli psát delší `registrovano = registrovano - zruseno`. Jak už bylo popsáno u operátoru sčítání, obě varianty dělají v praxi totéž.

Při odčítání může stejně jako u sčítání dojít k podtečení a přetečení. Na rozdíl od sčítání se ale odčítání nedá použít na řetězce.

Operátor násobení

Když se vrátím ke svému školnímu příkladu, každý student musí za přednášku zaplatit školné 72 dolarů. Školné platí i studenti, kteří na registrované přednášky nechodí nebo si přednášku zruší. Následující program opět rozšiřuje naši předchozí verzi, do které přidává nový příkaz:

```
cout << "Skolne celkem: " << (registrovano + zruseno) * 72 << " dolaru.\n";
```

Celý program vypadá takto:

```
#include <iostream>
using namespace std;
```

```
int main(void)
{
    int registrovano, pridano, zruseno;
    cout << "Zadejte pocet predregistrovanych studentu: ";
    cin >> registrovano;
    cout << "Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: ";
    cin >> pridano;
    registrovano += pridano;
    cout << "Zadejte pocet studentu, kteri si prednasku zrusili: ";
    cin >> zruseno;
    registrovano -= zruseno;
    cout << "Celkovy pocet studentu: " << registrovano << "\n";
    cout << "Skolne celkem: " << (registrovano + zruseno) * 72 << " dolaru.\n";
    return 0;
}
```

Vstup a výstup programu by mohl vypadat takto:

```
Zadejte pocet predregistrovanych studentu: 30
Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: 3
Zadejte pocet studentu, kteri si prednasku zrusili: 5
Celkovy pocet studentu: 28
Skolne celkem: 2376 dolaru.
```

Proměnné `registrovano` a `zruseno` se sčítají, protože školné musí zaplatit i studenti, kteří si přednášku nakonec zrušili.

Přetečení a podtečení fungují u násobení stejně jako u sčítání a odčítání. Na rozdíl od sčítání ale násobení nepracuje s řetězci (podobně jako odčítání).

Priorita aritmetických operátorů

Příkaz, který jsme do programu právě přidali, obsahuje dva aritmetické operátory: sčítání a násobení. Hodně záleží na pořadí, ve kterém se tyto operátory provedou. Pokud se první provede sčítání (28+5=33) a teprve výsledek tohoto výpočtu se vynásobí 72, dostaneme číslo 2376. Kdybychom ale první provedli 5×72=360 a výsledek přičetli k číslu 28, dostali bychom číslo 388.

Pořadí výpočtů se řídí takzvanou *prioritou operátorů*. Už jsme na ni v této kapitole jednou narazili – zajímalo nás, jestli se jako první provede operátor přiřazení, nebo operátor sčítání. Tentokrát nás bude zajímat vzájemná priorita jednotlivých aritmetických operátorů, viz tabulku 4.3.

Tabulka 4.3: Priorita aritmetických operátorů

Priorita	Operátory
nejvyšší	- (unární minus)
střední	* / %
nízká	+ -

Operátory, které jsou v tabulce 4.3 na stejném řádku, mají prioritu stejnou. Například násobení má stejnou prioritu jako dělení a podobně sčítání má stejnou prioritu jako odčítání.

V tabulce 4.4 je vidět, jak se priorita projeví na několika aritmetických výrazech. Operátorem dělení jsme se ještě nezabývali, ale v tabulce 4.4 se chová přesně jako v matematice.

Tabulka 4.4: Priorita operátorů v akci

Výraz	Výsledek
2+3×4	14 (nikoliv 20)
8/2–1	3 (nikoliv 8)

Když mají operátory stejnou prioritu, pořadí se určí podle jejich *asociativity*, viz tabulku 4.5.

Tabulka 4.5: Asociativita aritmetických operátorů

Operátor	Asociativita
- (unární minus)	pravá
* / %	levá
+ -	levá

Vezměte si například výraz 8/2×4. Bez závorek by nemuselo být úplně jasné, jak by se měl vyhodnocovat – jako (8/2)×4, nebo jako 8/(2×4)? Priorita operátorů nám tu nepomůže, protože obě operace mají prioritu shodnou. Pomůže nám asociativita. Operátory násobení a dělení asociují *zleva*, takže prostřední operand (v našem případě dvojka) patří *levému* z nich a výraz bude vyhodnocen jako (8/2)×4=16.

Běžně se stává, že potřebujete operace vyhodnotit v jiném pořadí, než v jakém by se vyhodnocovaly podle své asociativity a priority. Například v našem ukázkovém programu by se podle standardních pravidel nejprve vynásobila proměnná `zruseno` číslem 72 a výsledek by se přičetl k proměnné `registrovano`.

Pořadí vyhodnocování výrazů se dá změnit pomocí závorek:

```
cout << "Skolne celkem: " << (registrovano + zruseno) * 72 << " dolaru.\n";
```

Výrazy v závorkách se vyhodnotí jako první. Díky tomu se nejprve sečtou proměnné `registrovano` a `zruseno`, a teprve poté se výsledek vynásobí číslem 72.

Operátory dělení

O každou z operací sčítání, odčítání a násobení se stará jeden operátor. Pro dělení existují operátory dva: `/` vrací podíl a `%` zbytek po dělení.

Podíl a zbytek po dělení jsou – podobně jako dělenec a dělitel – výrazy, které jsem se naučil kdysi na základní škole a následně řadu let vůbec nepoužíval. Pokud si touto terminologií nejste jisti (ani já jsem nebyl), pomůže vám příklad. Když dělíme 7/2, sedmička je dělenec, dvojka dělitel, podíl je 3 a zbytek po dělení 1.

Podíl

Operátor `/` vrací podíl dvou čísel. Výsledek závisí na tom, jestli je alespoň jeden z operandů desetinné číslo. Například 10/4 je 2,5, ale v C++ by se odpovídající výraz vyhodnotil jako 2. Pokud jsou totiž oba operandy celá čísla, operátor `/` provede dělení celočíselné. Platí to i v případech, kdy výsledek ukládáme do proměnné typu `float`; výstupem následujícího programu je číslo 2:

```
#include <iostream>
using namespace std;
int main(void)
{
```

```
    int a=10, b=4;
    float vysledek = a/b;
    cout << a << "/" << b << "=" << vysledek << "\n";
    return 0;
}
```

Když ale výraz změníme na `10.0/4`, dostaneme výsledek 2,5. Kdykoliv je totiž alespoň jeden z operandů desetinné číslo (a výraz `10.0` by se určitě vyhodnotil jako desetinné číslo), operátor `/` provede klasické dělení. Když v předchozím programu změníme typ prvního operandu na `float`, výsledek dělení bude 2,5:

```
#include <iostream>
using namespace std;
int main(void)
{
    float a=10, vysledek;
    int b=4;
    cout << a << "/" << b << "=" << vysledek << "\n";
    return 0;
}
```

Vraťme se ale k předchozímu příkladu. Pokud chcete získat desetinný podíl dvou celočíselných proměnných, musíte jednu z těchto proměnných *přetypovat* na `float`. Přetypování nezmění typ proměnné, změní pouze datový typ hodnoty použité pro výpočet. Když chcete proměnnou přetypovat na nějaký datový typ, napíšete název tohoto datového typu před proměnnou a uzavřete název typu nebo proměnnou do závorek. Takto bychom v prvním příkladu mohli přetypováním dostat desetinný podíl obou čísel:

```
#include <iostream>
using namespace std;
int main(void)
{
    int a=10, b=4;
    float vysledek = (float) a/b;
    cout << a << "/" << b << "=" << vysledek << "\n";
    return 0;
}
```

Stejně dobře by fungoval i libovolný z následujících výrazů:

```
float vysledek = float(a) / b;
float vysledek = a / (float) b;
float vysledek = a / float(b);
```

V některých programech vám ale bude vyhovovat původní celočíselné dělení, viz například automat na drobné na konci této kapitoly.

Zkusme teď dělení využít v příkladu s registrací studentů. V rámci poslední změny jsme vypočetli školné za všechny studenty, včetně těch, kteří už na přednášky nechodí. Teď program upravíme tak, aby spočítal a vypsal průměrné školné na každého doposud zapsaného studenta. Zavedeme celočíselnou proměnnou `skolne`, do které se uloží celkové vybrané školné spočítané jako `(registrovano + zruseno) * 72`. Průměrné školné na zapsaného studenta se pak vypočítá následujícím příkazem:

```
cout << "Prumerne skolne na zapsaneho studenta: "
     << (float) skolne / registrovano << " dolaru.\n";
```

Celý kód programu teď vypadá takto:

```
#include <iostream>
using namespace std;
int main(void)
{
    int registrovano, pridano, zruseno;
    cout << "Zadejte pocet predregistrovanych studentu: ";
    cin >> registrovano;
    cout << "Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: ";
    cin >> pridano;
    registrovano += pridano;
    cout << "Zadejte pocet studentu, kteri si prednasku zrusili: ";
    cin >> zruseno;
    registrovano -= zruseno;
    cout << "Celkovy pocet studentu: " << registrovano << "\n";
    skolne = (registrovano + zruseno) * 72;
    cout << "Skolne celkem: " << skolne << " dolaru.\n";
    cout << "Prumerne skolne na zapsaneho studenta: "
         << (float) skolne / registrovano << " dolaru.\n";
    return 0;
}
```

Vstup a výstup programu by mohl vypadat například takto:

```
Zadejte pocet predregistrovanych studentu: 30
Zadejte pocet studentu, kteri si prednasku registrovali dodatecne: 3
Zadejte pocet studentu, kteri si prednasku zrusili: 5
Celkovy pocet studentu: 28
Skolne celkem: 2376 dolaru.
Prumerne skolne na zapsaneho studenta: 84.8571 dolaru.
```

Přetypování ve výpočtu průměru je nutné, bez něj by průměrné školné vyšlo na 84 dolarů.

Modulo

Operátor `%` neboli modulo vrací zbytek po celočíselném dělení. Například `7%2` není podíl 3, ale zbytek 1. Operátor `%` funguje pouze na celých číslech, na desetinných číslech je jeho výsledek nedefinovaný. Většinou překladač nahlásí chybu, ale záleží na překladači. Praktický příklad použití `%` si ukážeme za okamžik.

Upozornění: Ani jeden z operátorů `/` a `%` se nedá použít pro dělení nulou. Když se o něj pokusíte, dojde k chybě.

Mocniny

Na rozdíl od mnoha jiných jazyků nemá C++ operátor pro umocňování. Místo něj obsahuje vestavěnou funkci `pow` definovanou v souboru `cmath`. Funkce `pow` má dva parametry. Prvním z nich je základ, druhým exponent. Například výraz `pow(2, 3)` umocní dvě na třetí, výsledkem je osm. Hodnota výrazu `pow(2, 3)` je sice celé číslo, ale typu `double`. Umocňovat můžete i desetinná čísla, výsledkem je opět desetinné číslo.

Funkce `pow` se hodí při řešení matematických problémů. Obsah kruhu se dá vypočítat podle vzorce πr^2, kde r je poloměr kruhu. Povrch kruhu o daném poloměru typu `double` tedy můžeme spočítat takto:

```
double povrch = 3.14159 * pow(polomer, 2);
```

Následující program spočítá povrch kruhu o poloměru zadaném uživatelem:

```
#include <iostream>
# include <cmath>
using namespace std;
int main(void)
{
    double polomer, povrch;
    cout << "Zadejte polomer kruhu: ";
    cin >> polomer;
    povrch = 3.14159 * pow(polomer, 2);
    cout << "Povrch kruhu je " << povrch << "\n";
    return 0;
}
```

Vstup a výstup programu vypadá takto:

```
Zadejte polomer kruhu: 6
Povrch kruhu je 113.097
```

Projekt: Automat na drobné

Když moje matka potřebovala zabavit otrávená nebo rozparáděná vnoučata, neštítila se ani takových triků, jakým byl automat na drobné. Děti nakrmily automat stovkami centů a užasle sledovaly, jak je automat třídí na větší částky, které už se dají odnést do banky a vyměnit za čtvrťáky, dolary a větší kořist. Děti se naštěstí dají zabavit snadno. I když v tomhle případě to nebyla úplně férová hra – asi byste uhádli, komu nakonec zůstaly čtvrťáky.

Popis programu

Náš program se uživatele zeptá na počet centů (řekněme, že uživatel vždy zadá kladné celé číslo). Pak vypíše počet dolarů, čtvrťáků, desetníků, pětníků a centů, na které se zadaný počet centů dá směnit:

```
Zadejte pocet centu: 387
Dolaru: 3
Ctvrtaku: 3
Desetniku: 1
Petniku: 0
Centu: 2
```

Nejdřív se podíváme na zdrojový kód, pak si ho vysvětlíme. Zkuste si zdrojový kód napsat předem sami. Pokud to zvládnete, dobře pro vás. Pokud ne, nevadí – i tak vám následující zdrojový kód a jeho popis řeknou víc, když program zkusíte napsat sami předem.

Napovím vám první tři řádky funkce `main` (kdo nechcete, nedívejte se):

```
int celkem, dolary, ctvrtaky, desetniky, petniky, zbytek;
cout << "Zadejte pocet centu: ";
cin >> celkem;
```

Do proměnné `celkem` se uloží celkový počet centů zadaný uživatelem. Do proměnné `dolary` přijde počet celých dolarů v zadaném počtu centů – například do 387 centů z předchozího příkladu se dolary vejdou tři. Do proměnných `ctvrtaky`, `desetniky` a `petniky` přijde počet celých

čtvrťáků, desetníků a pětníků; podle našeho předchozího příkladu tedy hodnoty tři, jedna a nula. Do proměnné `zbytek` se nakonec uloží počet centů, které zbudou (v našem předchozím příkladu dvojka). V průběhu výpočtu proměnná `zbytek` poslouží ještě k uložení mezivýsledků. Program by se samozřejmě dal napsat i s větším nebo menším počtem proměnných.

Zdrojový kód

Program se dá napsat více způsoby, takhle jsem ho napsal já:

```
#include <iostream>
using namespace std;
int main(void)
{
    int celkem, dolary, ctvrtaky, desetniky, petniky, zbytek;
    cout << "Zadejte pocet centu: ";
    cin >> celkem;
    dolary = celkem / 100;
    zbytek = celkem % 100;
    ctvrtaky = zbytek / 25;
    zbytek %= 25;
    desetniky = zbytek / 10;
    zbytek %= 10;
    petniky = zbytek / 5;
    zbytek %= 5;
    cout << "Dolaru: "    << dolary    << "\n";
    cout << "Ctvrtaku: "  << ctvrtaky  << "\n";
    cout << "Desetniku: " << desetniky << "\n";
    cout << "Petniku: "   << petniky   << "\n";
    cout << "Centu: "     << zbytek    << "\n";
    return 0;
}
```

Algoritmy

V první kapitole jsme si řekli, že počítačový program je seznam příkazů, které programátor dává počítači. Tyto příkazy jsou v nějakém programovacím jazyce, například C++. Ještě než ale něco přikážete počítači, musíte si příkazy rozmyslet v přirozeném jazyce.

Algoritmus je přesný návod, podle kterého lze krok za krokem vyřešit nějaký problém. Programátor je ten, kdo vymýšlí a implementuje algoritmy. Zápisu algoritmů v programovacím jazyce se říká programování. Tvorba algoritmů se dá naučit. Dobrým základem jsou pro ni veškeré obory, které vyžadují analytické myšlení, například matematika.

Řekněme, že byste dostali nějaký počet centů, dejme tomu 387, a měli je v hlavě směnit na větší mince – pětníky, desetníky, čtvrťáky a dolary. Jak byste to udělali?

Celkem logické je začít od dolarů. Jeden dolar je sto centů. Když vydělíme 387 stem, dostaneme počet dolarů v zadaných centech: tři. Jak to ale dopadne v C++, nevyhodnotí se výraz `387/100` jako 3,87? V předchozí části této kapitoly věnované operátoru dělení jsme si říkali, že pokud dělíme dva celočíselné operandy, dostaneme celočíselný výsledek. Kdyby se nám to nehodilo, mohli bychom jeden z operandů přetypovat na `float`. Ale nám se celočíselný výsledek hodí, a tak nic přetypovávat nebudeme. Celkový počet centů (387) je uložený v celočíselné proměnné `celkem`, takže když tuto proměnnou vydělíme celým číslem `100`, dostaneme celočíselný podíl: 3.

```
dolary = celkem / 100;
```

Když od 387 centů odečtete tři dolary, zůstane vám 87 centů neboli zbytek po dělení `celkem / 100`. Tento zbytek se dá získat operátorem modulo, uložíme ho do celočíselné proměnné `zbytek`:

```
zbytek = celkem % 100;
```

Stejným postupem zjistíte počet čtvrťáků ve zbývajících 87 centech. Jediný rozdíl je v tom, že namísto stovkou dělíte číslem 25 a místo proměnné `celkem` dělíte proměnnou `zbytek`:

```
ctvrtaky = zbytek / 25;
zbytek %= 25;
```

Analogicky se zjistí i počet desetníků a pětníků:

```
desetniky = zbytek / 10;
zbytek %= 10;
petniky = zbytek / 5;
zbytek %= 5;
```

Počet centů, které zůstanou po závěrečném dělení pětkou, už se na žádnou vyšší minci směnit nedá. Z toho plyne, že nemáme co dál dělit a program je hotový. Všimněte si, že nepotřebujeme samostatnou proměnnou pro počet zbývajících centů, protože ten už máme v proměnné `zbytek`. Zbývá jen vypsat počet dolarů, čtvrťáků, desetníků a centů.

Postup, který jsme si právě popsali, je algoritmus – první z mnoha, kterým se v knize budeme věnovat.

Shrnutí

Aritmetické výpočty jsou běžnou součástí počítačových programů. Kromě toho, že počítače dokážou uložit ohromné množství dat, umí ve srovnání s lidmi i mnohem rychleji a přesněji počítat. Tuto výpočetní sílu můžete využít prostřednictvím aritmetických operátorů.

C++ nabízí aritmetické operátory pro sčítání, odčítání, násobení a dělení. Pro dělení existují operátory hned dva: podíl `/` a zbytek po dělení neboli modulo, `%`.

Všechny aritmetické operátory umí pracovat s celými čísly a všechny kromě modula umí pracovat i s čísly desetinnými. Sčítání jako jediné pracuje i s řetězci, sečíst dva řetězce znamená spojit je dohromady.

C++ nemá na rozdíl od jiných jazyků operátor umocňování. Jeho funkci zastupuje vestavěná funkce `pow` definovaná v souboru `cmath`.

V následující kapitole se dozvíte o relačních a logických operátorech a řídicích strukturách, které umožňují měnit průběh výpočtu podle nějaké podmínky.

Test

1. Pro kterou ze čtyř základních aritmetických operací nabízí C++ více než jeden operátor?
2. Který z aritmetických operátorů funguje kromě čísel i na řetězcích?
3. Který z aritmetických operátorů nepracuje s desetinnými čísly?
4. Který z aritmetických operátorů nemůže mít jako druhý operand nulu?
5. Jak jinak byste mohli zapsat výraz `celkem = celkem + 2`?

6. Jaká je hodnota výrazu `2+3*4`?
7. Jaká je hodnota výrazu `8/2*4`?
8. Jaká je hodnota výrazu `10/4`?
9. Jakým operátorem nebo funkcí se v C++ umocňuje?
10. Co je to algoritmus?