den 3

Proměnné a konstanty

Programy potřebují nějakým způsobem uchovávat data, která používají. Různé způsoby reprezentace, uchování a manipulace s daty nabízí proměnné a konstanty.

Dnes se naučíte a dozvíte:

- Jak se deklarují a definují proměnné a konstanty.
- Jak se proměnným přiřazují hodnoty a jak se s těmito hodnotami zachází.
- Jak se na obrazovku vypíše hodnota proměnné.

Co je to proměnná?

Proměnná je v programu C++ místo, kde se uchovává informace. Rozumíme tím místo v paměti počítače, na které můžete hodnotu uložit a ze kterého ji můžete později znovu získat.

Proměnné poskytují pouze dočasné uchování informací. Když počítač vypnete, tyto hodnoty se ztratí. Trvalé uchování hodnot je jiná záleži-

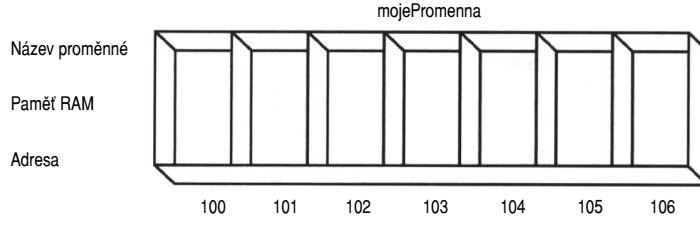
tost a obvykle se realizuje v databázi nebo v souboru na disku. Ukládání hodnot do souborů na disku se budeme věnovat v den 16. – „Pokročilá dědičnost“.

Reprezentace dat v paměti

Paměť počítače si můžeme představit jako řadu přihrádek, které jsou seřazeny vedle sebe. Každé přihrádce – neboli umístění v paměti – je postupně přiřazeno číslo. Těmto číslům říkáme adresy v paměti. Proměnná si vyhradí jednu nebo více přihrádek, do kterých může uložit hodnotu.

Názvem proměnné (například `mojePromenna`) rozumíme jmenovku na přihrádce, díky které ji můžete snadno najít, aniž byste museli znát skutečnou adresu v paměti. Na obrázku 3.1 si můžete prohlédnout schematické znázornění paměti. Povšimněte si, že proměnná `mojePromenna` začíná na paměťové adrese s číslem 103. Podle velikosti pak může proměnná `mojePromenna` zabrat jednu nebo více paměťových adres.

Obrázek 3.1
Schematické znázornění paměti



POZNÁMKA Zkratka *RAM* pochází z angličtiny a znamená „Random Access Memory“, tedy paměť s přímým (náhodným) přístupem. Program se po svém spuštění načte ze souboru na disku do paměti RAM. Zde se také vytvoří všechny proměnné. Když programátoři hovoří o paměti, mají obvykle na mysli právě paměť RAM.

Vymezení paměti

Když definujete v C++ proměnnou, musíte kompilátoru oznámit, o jaký druh proměnné se jedná: zda je to celé číslo, znak a podobně. Podle této informace vyhradí kompilátor v paměti příslušné místo a zároveň ví, jaký druh hodnoty chcete v proměnné uchovávat.

Kompilátor také získává možnost varovat vás nebo zobrazit chybu, když se náhodou pokusíte uložit do takové proměnné hodnotu nesprávného typu. (Programovací jazyk s touto charakteristikou se označuje za „silně typový“.)

Každá přihrádka má velikost jeden bajt. Jestliže typ proměnné, kterou vytvoříte, bude mít velikost čtyři bajty, vyhradí se čtyři bajty paměti neboli čtyři přihrádky. Prostřednictvím typu proměnné (například celé číslo) kompilátoru sdělíte, kolik paměti (kolik přihrádek) si má pro tuto proměnnou vyhradit.

Kdysi musel programátor bitům a bajtům dobře rozumět, jedná se nakonec o základní jednotky uložení informací. Počítačové programy se zlepšily a dnes není nutné znát všechny detaily. Stále je však užitečné vědět, jak se data ukládají. Krátký přehled základů binární matematiky najdete v dodatku A „Binární a hexadecimální aritmetika“.

POZNÁMKA Jestliže se vám při slově matematika zachtělo knihu rychle zavřít, pak se s dodatkem A nezatěžujte, nebudete jej nezbytně potřebovat. Je skutečností, že v dnešní době už programátor nemusí být zároveň matematikem, ačkoli logické a racionální uvažování by mu přesto mělo být vlastní.

Velikost celých čísel

Na každém počítači zabere každý typ proměnné v paměti jednotnou a neměnnou velikost. To znamená, že celé číslo může mít na jednom počítači velikost dva bajty a na druhém čtyři, ale na každém z nich to bude za všech okolností stejně.

Proměnná typu `char` (používá se k ukládání znaků) má většinou velikost jednoho bajtu.

POZNÁMKA Vedou se nekonečné debaty o tom, jak by se mělo vyslovovat slovo `char`. Někteří jej vyslovují „kar“, jiní „char“, další zase „kér“. Jisté je, že výslovnost „kar“ je správná, protože ji používám *já*, ale vy si samozřejmě můžete vybrat podle svého.

Celé číslo typu `short` má na většině počítačů velikost dva bajty, celé číslo typu `long` má obvykle velikost čtyři bajty a celé číslo (bez klíčových slov `short` nebo `long`) může mít dva nebo čtyři bajty. Asi byste očekávali, že to jazyk bude specifikovat přesněji, ale není tomu tak. Pouze říká, že typ `short` musí mít velikost menší nebo rovnu velikosti typu `int`, která dále musí být menší nebo rovna velikosti typu `long`.

S největší pravděpodobností pracujete právě s počítačem, jehož celočíselné hodnoty typu `short` mají velikost dva bajty, u typu `int` jsou to čtyři bajty a v případě typu `long` také čtyři bajty.

Velikost celého čísla určuje procesor (16bitový nebo 32bitový) a kompilátor, který používáte. U moderních 32bitových počítačů (Pentium), které používají moderní kompilátory (například Visual C++ 4 nebo novější), mají celočíselné hodnoty velikost *čtyři* bajty.

Pozor Při vytváření programů nikdy nesmíte předpokládat, kolik paměti jednotlivé typy využívají.

Na svém počítači nyní zkompilujte a spusťte program z výpisu 3.1. Řekne vám, jaká je ve vašem případě přesná velikost jednotlivých datových typů.

Výpis 3.1

Určení velikosti typů proměnných na vašem počítači

```
0: #include <iostream>
1:
2: int main()
3: {
4:     using std::cout;
5:
6:     cout << "Velikost typu int je:\t\t"
7:          << sizeof(int) << " bajtu.\n";
8:     cout << "Velikost typu short je:\t\t"
9:          << sizeof(short) << " bajtu.\n";
```

```
10:     cout << "Velikost typu long je:\t\t"
11:          << sizeof(long) << " bajtu.\n";
12:     cout << "Velikost typu char je:\t\t"
13:          << sizeof(char) << " bajtu.\n";
14:     cout << "Velikost typu float je:\t\t"
15:          << sizeof(float) << " bajtu.\n";
16:     cout << "Velikost typu double je:\t"
17:          << sizeof(double) << " bajtu.\n";
18:     cout << "Velikost typu bool je:\t\t"
19:          << sizeof(bool) << " bajtu.\n";
20:
21:     return 0;
22: }
```

Výstup

```
Velikost typu int je:           4 bajtu.
Velikost typu short je:         2 bajtu.
Velikost typu long je:          4 bajtu.
Velikost typu char je:          1 bajtu.
Velikost typu float je:         4 bajtu.
Velikost typu double je:        8 bajtu.
Velikost typu bool je:          1 bajtu.
```

POZNÁMKA Na vašem počítači se uvedené počty bajtů mohou lišit.

Většina kódu, který najdete ve výpisu 3.1, by vám už neměla činit problémy. Některé řádky jsme rozdělili, aby odpovídaly velikosti stránky v knize. Například obsah řádků 6 a 7 by ve skutečnosti mohl být na jediném řádku. Kompilátor prázdné prostory ignoruje (mezery, tabulátory, znaky konce řádku), takže s řádky 6 a 7 bude zacházet jako s jedním řádkem.

Tento program uvádí novinku, kterou je na řádcích 6 až 19 operátor `sizeof()`. Je součástí kompilátoru a vrací velikost objektu, který mu předáte jako parametr. Například na řádku 7 se operátoru `sizeof()` předá klíčové slovo `int`. Dnes se ještě dozvíte, že se `int` používá k popisu standardní celočíselné proměnné. Když použijete `sizeof` na počítači s Pentiem 4 a systémem Windows XP, pak má `int` velikost čtyři bajty, což je náhodou stejný počet jako má typ `long int` na témže počítači.

Další řádky výpisu 3.1 ukazují velikosti jiných datových typů. Více se o možném obsahu těchto datových typů a rozdílech mezi nimi dozvíte za chvíli.

Celá čísla se znaménkem (`signed`) a bez znaménka (`unsigned`)

U všech celočíselných typů rozlišujeme dvě varianty: čísla se znaménkem (`signed`) a bez znaménka (`unsigned`). Podstata spočívá v tom, že za jistých okolností můžete potřebovat pracovat se zápornými čísly a jindy ne. Celá čísla (všech typů, i `short` a `long`), která nejsou explicitně označena jako „`unsigned`“, tedy bez znaménka, se považují za „`signed`“, tedy se znaménkem. Celá čísla se znaménkem mohou nabývat kladných i záporných hodnot. Celá čísla bez znaménka jsou vždy kladná.

Protože pro obě varianty celých čísel, `signed` i `unsigned`, je vyhrazen stejný počet bajtů, bude největší číslo, které můžete uložit v celočíselné proměnné bez znaménka, dvakrát větší než největší kladné číslo, které můžete uložit do celočíselné proměnné se znaménkem. Celá čísla typu `unsigned short` mohou tedy nabývat hodnot od 0 do 65 535. Poloviční počet

možných hodnot připadajících na typ celého čísla `signed short` je záporný. Celá čísla typu `signed short` mohou proto nabývat hodnot od –32 768 do 32 767. Další informace o přednostech operátorů najdete v příloze C.

Základní typy proměnných

Do jazyka C++ je zabudováno několik dalších typů proměnných. Je možné je rozdělit na celočíselné proměnné (těmi jsme se dosud zabývali), proměnné s pohyblivou řádovou (desetinnou) čárkou a znakové proměnné.

Proměnné s pohyblivou řádovou čárkou nabývají hodnot, které lze vyjádřit ve tvaru zlomku; jedná se tedy o reálná čísla. Znakové proměnné zabírají jeden bajt a používají se k uchování 256 znaků a symbolů ze sady znaků ASCII nebo rozšířené sady ASCII.

Poznámka Sadou znaků ASCII rozumíme sadu znaků standardizovaných pro používání na počítačích. Jedná se o zkratku názvu „American Standard Code for Information Interchange“ (americký standardní kód pro výměnu informací). Sada ASCII je podporována téměř všemi počítačovými operačními systémy, ačkoli mnohé z nich podporují také mezinárodní sady znaků.

V tabulce 3.1 najdete přehled typů proměnných, které se v programech C++ používají. U každého typu proměnné v tabulce najdete jeho předpokládanou velikost v paměti a druh hodnot, které lze do proměnné tohoto typu uložit. Mezní hodnoty jsou však určeny velikostí typu v paměti, proto raději zkonzultujte váš výstup z programu 3.1. Je pravděpodobné, že zaznamenáte shodné velikosti, pokud ovšem nepoužíváte počítač se 64bitovým procesorem.

Tabulka 3.1 Typy proměnných

Typ	Velikost	Hodnoty
`bool`	1 bajt	`true` nebo `false`
`unsigned short int`	2 bajty	0 až 65 535
`short int`	2 bajty	-32 768 až 32 767
`unsigned long int`	4 bajty	0 až 4 294 967 295
`long int`	4 bajty	-2 147 483 648 až 2 147 483 647
`int` (16 bitů)	2 bajty	-32 768 až 32 767
`int` (32 bitů)	4 bajty	-2 147 483 648 až 2 147 483 647
`unsigned int` (16 bitů)	2 bajty	0 až 65 535
`unsigned int` (32 bitů)	4 bajty	0 až 4 294 967 295
`char`	1 bajt	256 znakových hodnot
`float`	4 bajty	1,2e-38 až 3,4e38
`double`	8 bajtů	2,2e-308 až 1,8e308

POZNÁMKA Velikost proměnných se může od hodnot uvedených v tabulce 3.1 lišit, a to v závislosti na typu kompilátoru a počítače, který používáte. Jestliže jste dostali stejný výstup, jaký je uveden u výpisu 3.1, měla by tabulka 3.1 platit i pro váš kompilátor. Jestliže se výstup u výpisu 3.1 lišil, měli byste nahlédnout do příručky ke kompilátoru a zjistit hodnoty, kterých mohou proměnné nabývat.

Definice proměnné

Již jste viděli vytvoření a používání řady proměnných. Nyní se dozvíte, jak vytvářet své vlastní. Proměnná se zavádí uvedením jejího typu, za nímž následuje jedna nebo více mezer, název proměnné a středník. Název proměnné může tvořit prakticky jakákoli kombinace písmen a číslic, ale nesmí obsahovat žádné mezery. Povolené názvy proměnných jsou `x`, `J23qrsnf` a `mujVek`. Názvy proměnných obvykle zároveň informují, k čemu slouží. Vhodně zvolené názvy proměnných usnadní porozumění programu. Následující příkaz definuje celočíselnou proměnnou s názvem `mujVek`:

```
int mujVek;
```

POZNÁMKA Když deklarujete proměnnou, dojde k alokaci (vyhrazení) paměti pro tuto proměnnou. Hodnota proměnné v paměti bude v tomto okamžiku náhodná. Za chvíli uvedeme, jak se do této paměti přiřadí nová hodnota.

K dobrým zásadám programování patří, že se snažíme vyhýbat názvům proměnných, jako je `J23qrsnf`. V případě proměnných, které budeme používat jen krátce, se omezujeme na jednopísmenné názvy (například `x` nebo `i`). Snažte se také používat názvy jako `mujVek` nebo `mnozstvi`. Takové názvy vám mohou pomoci, když si o tři týdny později budete lámat hlavu, co jste jistým řádkem kódu vlastně mysleli.

Proveďte následující pokus: Zkuste u uvedených programů uhodnout, co dělají, na základě prvních několika řádků:

Příklad 1

```
int main()
{
       unsigned short x;
       unsigned short y;
       unsigned short z;
       z = x * y;
       return 0;
}
```

Příklad 2

```
int main()
{
       unsigned short Sirka;
       unsigned short Delka;
       unsigned short Plocha;
       Plocha = Sirka * Delka;
       return 0;
}
```

POZNÁMKA Jestliže tento program zkompilujete, dostanete od kompilátoru upozornění, že nedošlo k inicializaci hodnot. Za chvíli uvedeme, jak tento problém odstranit.

Je zřejmé, že v druhém případě asi uhodnete snáze, k čemu je program určen. Nepohodlné psaní delších názvů proměnných se více než vyplatí, protože takový program se mnohem lépe čte i udržuje.

Rozlišování velkých a malých písmen

Jazyk C++ rozlišuje mezi velkými a malými písmeny. Jinými slovy, proměnná s názvem `vek` není totožná s proměnnou, jejíž název je `Vek`, a to je zase jiná proměnná než `VEK`.

POZNÁMKA U některých kompilátorů je možné rozlišování mezi velkými a malými písmeny vypnout. Raději to však nezkoušejte; vaše programy pak přestanou pracovat v prostředí jiných kompilátorů a kód se stane pro ostatní programátory C++ zavádějící.

Názvy proměnných

Můžete se setkat s různými způsoby pojmenování proměnných. Příliš nezáleží na tom, který si zvolíte, je však důležité v celém programu tento způsob konzistentně dodržovat. V opačném případě se budou v kódu ostatní programátoři jen obtížně orientovat.

Mnozí programátoři v názvech proměnných raději používají malá písmena. Jestliže je vhodné v názvu proměnné použít dvě slova (například „moje auto“), jsou rozšířeny dva způsoby pojmenování: `moje_auto` nebo `mojeAuto`. Pro druhý případ se vžilo označení velbloudí zápis, protože velká písmena uvnitř názvu připomínají velbloudí hrby.

Některým se lépe čte název s podtržítkem (_), ale jiní se mu snaží vyhýbat, protože jeho psaní je poněkud nepohodlné. V této knize budeme používat ten druhý typ zápisu, ve kterém všechna následující slova v názvu proměnné začínají velkým písmenem: `mojeAuto`, `rychlaHnedaLiska` a podobně.

Mnozí zkušení programátoři používají styl zápisu, kterému se říká maďarský. U tohoto typu zápisu se před název proměnné přidá předpona, která se skládá ze sady znaků popisujících typ proměnné. Zápis názvu celočíselné proměnné by mohl začínat malým písmenem i, v případě typu `long` malým písmenem l a podobně. Dále by se podobným způsobem označily například konstanty, ukazatele, globální proměnné a tak dále. Většina z těchto úmluv má své opodstatnění při programování v C, v této knize je však používat nebudeme.

Poznámka Tento typ zápisu se označuje jako maďarský podle jeho autora. Jedná se o Charlese Simonyiho z Microsoftu, který pochází z Maďarska. Jeho původní monografii můžete najít na `http://www.strangecreations.com/library/c/naming.txt`**.**

Microsoft od tohoto způsobu zápisu nedávno upustil a v doporučeních pro syntaxi v C# se výslovně radí maďarský způsob zápisu *nepoužívat*. Jejich zdůvodnění pro C# platí i pro C++.

Klíčová slova

V jazyce C++ jsou některá slova vyhrazena a vy je nesmíte používat jako názvy proměnných. Jedná se o názvy příkazů, které kompilátor interpretuje jako pokyny pro řízení programu. Mezi klíčová slova patří `if`, `while`, `for` a `main`. V příručce ke kompilátoru najdete jejich úplný seznam, ale obecně se dá říci, že žádný vhodný název proměnné není klíčovým slovem. Seznam klíčových slov jazyka C++ najdete v dodatku B a také v tabulce 3.2.

Tabulka 3.2 Klíčová slova jazyka C++

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

Dále jsou rezervována následující slova:

and	bitor	not_eq	xor
and_eq	compl	or	xor_eq
bitand	not	or_eq	

ANO	NE
ANO, definujte proměnnou tak, že napíšete její typ a pak název.	**NE**, nepoužívejte klíčová slova jazyka C++ jako názvy proměnných.
ANO, používejte smysluplné názvy proměnných.	**NE**, nevytvářejte programy tak, aby závisely na počtu bajtů používaných k uložení nějaké proměnné.
ANO, pamatujte na to, že C++ rozlišuje malá a velká písmena	**NE**, nepoužívejte pro záporná čísla proměnné typu `unsigned`.
ANO, snažte se uvědomit si, jaký počet bajtů každá proměnná v paměti spotřebuje a jaké hodnoty lze do proměnných určitého typu uložit.	

Vytvoření více proměnných najednou

V jednom výrazu je možné definovat i více proměnných najednou. Napíšete nejdříve typ a pak názvy proměnných, které oddělíte čárkami. Příklad:

```
unsigned int mujVek, mojeVaha;  // dvě celočíselné proměnné bez znaménka
long int plocha, sirka, delka;  // tři proměnné typu long
```

Jak vidíte, proměnné `mujVek` i `mojeVaha` jsou obě deklarovány jako celočíselné proměnné typu `unsigned`, bez znaménka. Na druhém řádku je deklarace tří samostatných proměnných typu `long` s názvy `plocha`, `sirka` a `delka`. Typ `long` je přiřazen ke všem třem proměnným; v jednom příkazu pro definici není možné typy proměnných míchat.

Přiřazení hodnoty do proměnné

Hodnota se do proměnné přiřazuje s pomocí operátoru přiřazení (=). Proměnné `Sirka` přiřadíte hodnotu 5 takto:

```
unsigned short Sirka;
Sirka = 5;
```

POZNÁMKA `long` **je zkrácená verze pro** `long int` **a** `short` **je zkrácená verze pro** `short int`**.**

Oba tyto kroky můžete spojit a proměnnou `Sirka` inicializovat přímo v definici:

```
unsigned short Sirka = 5;
```

Inicializace se velmi podobá přiřazení a u celočíselných proměnných je rozdíl minimální. Později, když budeme probírat konstanty, uvidíte, že některé hodnoty je nutné inicializovat, protože není možné do nich později přiřadit hodnotu. Rozdíl spočívá především v tom, že k inicializaci dochází ve stejném okamžiku, kdy proměnná vznikne.

Podobně jako je možné definovat více proměnných najednou, je možné inicializovat více než jednu proměnnou zároveň. Například:

```
// vytvoření více proměnných typu long a jejich inicializace
long sirka = 5, delka = 7;
```

V tomto případě se inicializuje celočíselná proměnná `sirka` typu `long` na hodnotu `5` a celočíselná proměnná `delka` typu `long` na hodnotu `7`. Je dokonce možné kombinovat definice s inicializacemi:

```
int mujVek = 39, tvujVek, jehoVek = 40;
```

V tomto případě se vytvoří tři proměnné typu `int` a první a třetí se zároveň inicializují.

Výpis 3.2 ukazuje hotový program připravený ke kompilaci, který vypočítá plochu obdélníka a na obrazovku napíše výsledek.

Výpis 3.2

Ukázka použití proměnných

```
 0: // Ukázka použití proměnných
 1: #include <iostream>
 2:
 3: int main()
 4: {
 5:      using std::cout;
 6:      using std::endl;
 7:
 8:      unsigned short int Sirka = 5, Delka;
```

```
 9:     Delka = 10;
10:
11:     // vytvoření proměnné typu unsigned short a její inicializace
12:     // na výsledek násobení šířky délkou
13:     unsigned short int Plocha = (Sirka * Delka);
14:
15:     cout << "Sirka: " << Sirka << "\n";
16:     cout << "Delka: " << Delka << endl;
17:     cout << "Plocha: " << Plocha << endl;
18:     return 0;
19: }
```

Výstup

```
Sirka: 5
Delka: 10
Plocha: 50
```

Analýza Řádek 1 obsahuje příkaz `include` pro knihovnu `iostream` a objekt `cout` bude tedy fungovat. Na řádku 3 začíná program. Na řádcích 5 a 6 se definují objekty `cout` a `endl` jako součásti standardního oboru názvů (`std`).

Na řádku 8 se definuje proměnná `Sirka` jako celé číslo typu `unsigned short` a její hodnota se inicializuje na `5`. Další celé číslo typu `unsigned short`, `Delka`, se také definuje, ale neinicializuje. Na řádku 9 se proměnné `Delka` přiřadí hodnota `10`.

Na řádku 13 se definuje celočíselná proměnná typu `unsigned short`, `Plocha`, a inicializuje se na hodnotu, která se získá vynásobením proměnných `Sirka` a `Delka`. Na řádcích 15 až 17 se hodnoty všech proměnných vytisknou na obrazovku. Všimněte si, že speciální slovo `endl` vytvoří nový řádek.

Příkaz `typedef`

Opakované psaní `unsigned short int` může být únavné, a co je důležitější, náchylné k chybám. Jazyk C++ umožňuje pro celou tuto frázi vytvořit s pomocí klíčového slova `typedef` zkratku neboli alias. Slovo `typedef` pochází z anglického „type definition“ (definice typu).

Prakticky tak můžete vytvářet synonyma a je důležité, abyste tuto operaci rozlišovali od tvorby nových typů (což je téma šestého dne „Objektově orientované programování“). Za klíčové slovo `typedef` se píše stávající název typu a pak nový název. Výraz se ukončí středníkem. Příklad:

```
typedef unsigned short int USHORT;
```

Vzniká zde nový název typu `USHORT`, který budete moci použít všude tam, kde byste jinak museli psát `unsigned short int`. Výpis 3.3 je obdobou výpisu 3.2, ale místo `unsigned short int` se zde používá zkratka `USHORT`.

Výpis 3.3

Ukázka použití klíčového slova `typedef`

```
0: // ****************
1: // Ukázka použití klíčového slova typedef
2: #include <iostream>
```

```
 3:
 4: typedef unsigned short int USHORT;    // definice USHORT
 5:
 6: int main()
 7: {
 8:
 9:     using std::cout;
10:     using std::endl;
11:
12:     USHORT Sirka = 5;
13:     USHORT Delka;
14:     Delka = 10;
15:     USHORT Plocha = Sirka * Delka;
16:     cout << "Sirka: " << Sirka << "\n";
17:     cout << "Delka: " << Delka << endl;
18:     cout << "Plocha: " << Plocha << endl;
19:     return 0;
20: }
```


Výstup

```
Sirka: 5
Delka: 10
Plocha: 50
```

POZNÁMKA **Znak * označuje násobení.**

Analýza Na řádku 4 se definuje typ `USHORT` jako synonymum pro `unsigned short int`. Program i výstup je v podstatě stejný jako u výpisu 3.2.

Kdy používat `short` a kdy `long`

Někdy může být zdrojem nejasností nejistota, kdy proměnnou deklarovat jako typ `long` a kdy použít spíše typ `short`. Pravidlo, vykládá-li se správně, je poměrně přímočaré: Jestliže existuje sebemenší šance, že hodnota, kterou chcete do proměnné vložit, bude pro její typ příliš velká, použijte větší typ.

Jak ukazuje tabulka 3.1, proměnné typu `unsigned short` mohou nabývat celočíselných hodnot pouze do 65 535 (za předpokladu, že mají dva bajty). Proměnné typu `signed short` své hodnoty rozdělují mezi kladná a záporná čísla, a proto je zde maximální kladná hodnota, které mohou nabývat, pouze poloviční oproti typu `unsigned`.

Přestože celá čísla typu `unsigned long` mohou nabývat velmi vysokých hodnot (4 294 967 295), jedná se stále o konečné číslo. Budete-li potřebovat číslo větší, budete muset použít typ `float` nebo `double`, ale bude to vždy na úkor přesnosti. Typy `float` a `double` mohou nabývat extrémně vysokých hodnot, ale většina počítačů bere v úvahu pouze prvních 7 nebo 9 platných číslic. To znamená, že na tento počet číslic se hodnoty zaokrouhlují.

Kratší proměnné vyžadují méně paměti. V dnešní době je paměť levná a její životnost je krátká. Používejte bez obav typ `int`, který bude mít na vašem počítači pravděpodobně velikost čtyři bajty.

Přetečení celočíselného typu bez znaménka (unsigned)

U proměnných typu unsigned long existuje limit, kterého mohou maximálně dosáhnout jejich hodnoty. Jen zřídka se tento fakt stane problémem; k čemu však dojde v případě, že celé číslo tento limit překročí?

Když celé číslo typu unsigned dosáhne své maximální hodnoty, „přetočí" se zpět na nulu a počítání začíná znovu, podobně jako je tomu u tachometru v autě. Výpis 3.4 ilustruje, co se stane, když se pokusíte do celočíselné proměnné typu short vložit příliš velkou hodnotu.

Výpis 3.4

Ukázka vložení příliš velké hodnoty do celočíselné proměnné typu unsigned

```
 0: #include <iostream>
 1: int main()
 2: {
 3:
 4:      using std::cout;
 5:      using std::endl;
 6:
 7:      unsigned short int maleCislo;
 8:      maleCislo = 65535;
 9:      cout << "male cislo: " << maleCislo << endl;
10:      maleCislo++;
11:      cout << "male cislo: " << maleCislo << endl;
12:      maleCislo++;
13:      cout << "male cislo: " << maleCislo << endl;
14:      return 0;
15: }
```

Výstup

```
male cislo: 65535
male cislo: 0
male cislo: 1
```

Analýza Na řádku 7 se deklaruje maleCislo jako proměnná typu unsigned short int, což na typickém počítači znamená proměnnou o velikosti dvou bajtů, která může nabývat hodnot mezi 0 a 65 535. Na řádku 8 se do proměnné maleCislo přiřadí maximální možná hodnota a ta se vytiskne na řádku 9.

Na řádku 10 se hodnota proměnné maleCislo zvýší o jedničku. Symbol pro operaci, při které se hodnota proměnné zvýší o jedničku, je ++ (stejně jako C++ je vyšší verze jazyka než C). Hodnota v proměnné maleCislo by teď měla být 65 536. Celá čísla typu unsigned short však nemohou nabývat hodnot vyšších než 65 535. Hodnota se tedy vrátí zpět na 0, která se vytiskne na řádku 11.

Na řádku 12 se hodnota proměnné maleCislo znovu zvýší o jedničku a pak se tato nová hodnota 1 vytiskne.

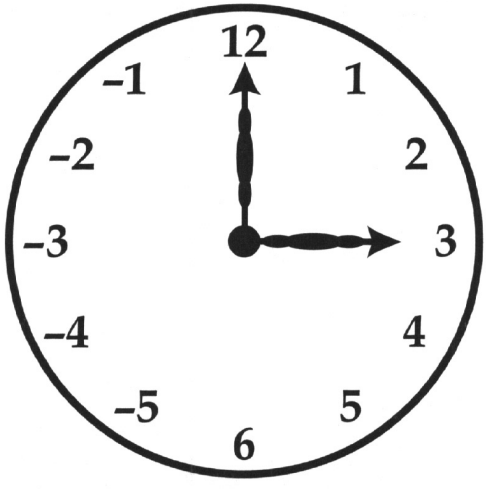
Přetečení celočíselného typu se znaménkem (signed)

U celých čísel typu signed se situace od celých čísel typu unsigned liší v tom, že polovina možných hodnot představuje záporná čísla. Místo tachometru si tentokrát raději představte hodiny, které se podobají těm z obrázku 3.2. Když se na nich pohybujeme ve směru hodi-

nových ručiček, čísla se zvyšují. Když jdeme proti směru hodinových ručiček, čísla se snižují. Čísla se pak střetnou ve spodní části hodin (v 6 hodin).

Obrázek 3.2

Kdyby se na hodinách používala čísla se znaménkem...



Čísla vedle 0 jsou buď 1 (ve směru hodinových ručiček), nebo –1 (proti směru hodinových ručiček). Když překročíme všechna kladná čísla, dostaneme se k největším záporným číslům a pak počítáme zpět k 0. Výpis 3.5 ukazuje, co se stane, když u proměnné typu `short integer` k maximálnímu kladnému číslu přičtete jedničku.

Výpis 3.5

Ukázka vložení příliš velké hodnoty do celočíselné proměnné typu `signed`

```
 0: #include <iostream>
 1: int main()
 2: {
 3:      using std::cout;
 4:      using std::endl;
 5:      short int maleCislo;
 6:      maleCislo = 32767;
 7:      cout << "male cislo: " << maleCislo << endl;
 8:      maleCislo++;
 9:      cout << "male cislo: " << maleCislo << endl;
10:      maleCislo++;
11:      cout << "male cislo: " << maleCislo << endl;
12:      return 0;
13: }
```

Výstup

```
male cislo: 32767
male cislo: -32768
male cislo: -32767
```

Analýza Na řádku 5 se deklaruje `maleCislo`, tentokrát jako proměnná typu `signed short` (jestliže výslovně neuvedete, že se jedná o `unsigned`, předpokládá se `signed`). Program pokračuje podobně jako předcházející, ale výstup se bude lišit. Chcete-li mu plně

porozumět, musíte být obeznámeni s tím, jak jsou čísla typu `signed` reprezentována jako bity v celém čísle o velikosti dvou bajtů.

Podstata však spočívá v tom, že stejně jako u typu celého čísla `unsigned`, i u celého čísla typu `signed` dochází k přetečení z nejvyšší možné kladné hodnoty na nejnižší možnou (tentokrát zápornou) hodnotu.

Znaky

Proměnná typu znak (typ `char`) má obvykle velikost 1 bajt, což je dost na to, aby pojala 256 hodnot (viz dodatek C). Proměnnou typu `char` můžeme interpretovat jako malé číslo (0 až 255) nebo jako prvek sady ASCII (American Standard Code for Information Interchange). Sada znaků ASCII a její obdoba ISO (International Standards Organization) nabízí způsob kódování všech písmen, číslic a interpunkčních znamének.

POZNÁMKA Počítače neznají písmena, věty nebo interpunkční znaménka. Jediné, čemu rozumí, jsou čísla. Ve skutečnosti to, co opravdu poznají, je, zda v konkrétním místě křížení spojů je dostatečné množství elektrické energie. Je-li tomu tak, bude tento stav symbolizovat 1, opačný stav pak symbolizuje 0. Podle seskupení jedniček a nul je počítač schopen generovat vzory, které je možné interpretovat jako čísla, a těm se pak mohou přiřadit písmena a interpunkční znaménka.

Podle kódu ASCII se hodnotě `97` přiřazuje malé písmeno „a“. Podobně jsou všechna malá a velká písmena, všechna čísla a interpunkční znaménka přiřazena hodnotám mezi 1 a 128. Zbývajících 128 značek a symbolů je vyhrazeno pro výrobce počítače, i když dnes se stala už téměř standardem rozšířená sada znaků od IBM.

POZNÁMKA Zkratka ASCII se obvykle vyslovuje „aski“.

Znaky a čísla

Když do proměnné typu `char` vložíte například hodnotu „a“, bude tam ve skutečnosti číslo mezi 0 a 255. Kompilátor však mezi sebou umí znaky (reprezentované apostrofem, pak písmenem, číslicí nebo interpunkčním znaménkem, následovaným opět apostrofem) a hodnoty sady ASCII tam i zpět překládat.

Funkce převádějící hodnoty na písmena a zpět může být libovolná. Neexistuje žádný zvláštní důvod, proč se malému písmenu „a“ přiřazuje právě hodnota 97. Pokud se na tom všichni (vaše klávesnice, kompilátor i obrazovka) shodnou, nedojde k žádným problémům. Je však nutné si uvědomit, že mezi hodnotou 5 a znakem „5“ existuje velký rozdíl. Skutečná hodnota znaku „5“ je 53, podobně jako písmeno „a“ je vyhodnoceno jako 97. Výpis 3.6 nám to ilustruje.

Výpis 3.6

Tisk znaků podle čísel

```
0: #include <iostream>
1: int main()
```

```
2: {
3:      for (int i = 32; i<128; i++)
4:          std::cout << (char) i;
5:      return 0;
6: }
```

Výstup

```
 !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv
wyz{|}~_
```

Tento jednoduchý program vytiskne hodnoty znaků pro celá čísla od 32 do 127. Výpis využívá k dosažení tohoto efektu na řádku 3 celočíselnou proměnnou i. Na řádku 4 je číslo v proměnné i přinuceno k zobrazení ve formě znaku.

Bylo by možné použít také znakovou proměnnou, a to způsobem zachyceným ve výpisu 3.7, jehož výstup je totožný.

Výpis 3.7

Tisk znaků podle čísel, druhá varianta

```
0: #include <iostream>
1: int main()
2: {
3:   for (unsigned char i = 32; i<128; i++)
4:     std::cout << i;
5:   return 0;
6: }
```

Jak vidíte, na řádku 3 je použit znak bez znaménka. Protože se místo číselné proměnné používá znaková proměnná, dokáže příkaz cout na řádku 4 přímo zobrazit znakovou hodnotu.

Speciální znaky

Kompilátor C++ rozeznává několik speciálních znaků určených k formátování. V tabulce 3.3 jsou uvedeny ty nejběžnější. Do kódu je zapíšete tak, že zadáte zpětné lomítko, za nímž bude následovat znak. Chcete-li například do kódu vložit znak tabulátoru, zadáte apostrof, zpětné lomítko, písmeno t a pak znovu apostrof:

```
char znakTab = '\t';
```

V tomto příkladě se deklaruje proměnná typu `char` (`znakTab`) a inicializuje se hodnotou znaku `\t`, který je interpretován jako tabulátor. Tyto speciální znaky pro formátování se používají při tisku na obrazovku, do souboru nebo na jiné výstupní zařízení.

Zpětné lomítko mění význam znaku, který za ním následuje. Například znak n normálně znamená písmeno n, ale když mu bude předcházet zpětné lomítko (\), bude se jednat o znak nového řádku.

Tabulka 3.3 Speciální znaky

Znak	Význam
`\a`	Pípnutí
`\b`	Smazání předchozího znaku (Backspace)

Znak	Význam
\f	Posun o stránku
\n	Nový řádek
\r	Návrat vozíku
\t	Tabulátor

Znak	Význam
\v	Vertikální tabulátor
\'	Apostrof
\"	Uvozovky
\?	Otazník
\\	Zpětné lomítko
\000	Oktalový (osmičkový) zápis
\xhhh	Hexadecimální (šestnáctkový) zápis

Konstanty

Podobně jako proměnné slouží i konstanty jako místa, kde se uchovávají data. Jak už sám název napovídá, konstanty na rozdíl od proměnných nemohou změnit hodnotu. Když definujete konstantu, musíte ji inicializovat a později jí už nemůžete přiřadit jinou hodnotu.

Literální konstanty

V jazyce C++ jsou dva typy konstant: literální a symbolické.

Literální konstantou rozumíme hodnotu, která se zadá přímo do programu. Například:

```
int mujVek = 39;
```

Proměnná `mujVek` je typu `int` a `39` je literální (doslovná) konstanta. Nemůžete 39 přiřadit hodnotu ani tuto hodnotu změnit.

Symbolická konstanta

Symbolickou konstantou rozumíme konstantu reprezentovanou názvem, podobně jako je názvem reprezentována proměnná. Na rozdíl od proměnné však nelze hodnotu konstanty po její inicializaci změnit.

Jestliže má program jednu celočíselnou proměnnou s názvem `studenti` a další s názvem `tridy`, můžete vypočítat, kolik máte při daném počtu tříd studentů; zde předpokládáme, že každá třída má 15 studentů.

```
studenti = tridy * 15;
```

V tomto případě je 15 literální konstantou. Váš kód by se snadněji četl a udržoval, kdybyste tuto hodnotu nahradili symbolickou konstantou:

```
studenti = tridy * studentiNaTridu;
```

Kdybyste se později rozhodli počet studentů v každé třídě změnit, stačilo by udělat to pouze v místě definice konstanty `studentiNaTridu`. Není nutné provádět jakékoli změny u žádného dalšího výskytu této konstanty. Symbolickou konstantu lze v C++ definovat dvěma

způsoby. Tradiční způsob, který je dnes už poněkud zastaralý, používá direktivu preprocesoru `#define`. Druhý a vhodnější způsob spočívá v aplikaci klíčového slova `const`.

Definice konstanty s pomocí `#define`

Protože direktivu preprocesoru `#define` používá mnoho již existujících programů, musíte dobře pochopit její skutečný způsob použití. Chcete-li konstantu definovat tradičním způsobem, zadejte:

```
#define studentiNaTridu 15
```

Všimněte si, že u konstanty `studentiNaTridu` není určen žádný typ (`int`, `char` a podobně). Příkaz `#define` znamená pouhou substituci textu. Pokaždé, když preprocesor narazí na slovo `studentiNaTridu`, vloží do textu místo něj hodnotu `15`.

Protože preprocesor běží před kompilátorem, kompilátor nikdy žádnou konstantu neuvidí, narazí pouze na číslo 15.

Pozor: I když se vám může zdát použití klíčového slova `#define` velmi jednoduché, měli byste se mu vyhýbat, protože je současný standard C++ považuje za již překonaný způsob deklarace.

Definice konstanty s pomocí `const`

Ačkoli konstanty definované s pomocí `#define` fungují dobře, existuje v C++ i lepší způsob:

```
const unsigned short int studentiNaTridu = 15;
```

V tomto příkladě se opět deklaruje symbolická konstanta s názvem `studentiNaTridu`, tentokrát se však jedná o konstantu typu `unsigned short int`. Výhodou této metody je snadnější údržba kódu a zlepšená prevence chyb. Největším rozdílem je však to, že konstanta má pevný typ a kompilátor si může vynutit, aby se používala v souladu se svým typem.

POZNÁMKA Konstanty není možné v době běhu programu měnit. Budete-li například chtít změnit konstantu `studentiNaTridu`, budete muset provést změnu v kódu a ten pak znovu zkompilovat.

ANO	NE
ANO, dávejte si pozor, aby velikost celých čísel nepřekročila dané meze a nevrátila se na nesprávnou hodnotu.	**NE**, nepoužívejte klíčová slova jako názvy proměnných.
ANO, dávejte svým proměnným smysluplné názvy, které vypovídají o jejich účelu.	**NE**, nepoužívejte direktivu preprocesoru `#define` k definování konstant. Pracujte raději s `const`.

Výčtové konstanty

Výčtové konstanty dovolují vytvořit nové typy a proměnné, jejichž hodnoty se omezují na množinu uvedených hodnot. Můžete například výčtem deklarovat typ `BARVA`, který bude nabývat následujících hodnot: `CERVENA`, `MODRA`, `ZELENA`, `BILA` a `CERNA`.

Pro syntaxi výčtových konstant platí, že začínají klíčovým slovem `enum`, za nímž následuje název typu, levá složená závorka, povolené hodnoty oddělené čárkou a nakonec pravá složená závorka se středníkem. Zde je příklad:

```
enum BARVA { CERVENA, MODRA, ZELENA, BILA, CERNA };
```

Tento příkaz způsobí:

1. Vytvoření nového výčtu `BARVA`, čímž dojde k založení nového typu.
2. Vytvoření symbolické konstanty `CERVENA` s hodnotou `0`, symbolické konstanty `BILA` s hodnotou `1`, symbolické konstanty `ZELENA` s hodnotou `2` a tak dále.

Každá výčtová konstanta má celočíselnou hodnotu. Jestliže neurčíte jinak, bude mít první konstanta hodnotu 0 a další vždy o jedničku větší. Každou z těchto konstant je však možné inicializovat na konkrétní hodnotu a těm, které inicializovány nejsou, se přiřadí hodnota podle předcházejících. Prohlédněte si tento příkaz:

```
enum BARVA { CERVENA=100, MODRA, ZELENA=500, BILA, CERNA=700 };
```

Konstanta `CERVENA` bude mít hodnotu `100`, `MODRA` bude mít hodnotu `101`, `ZELENA` bude mít hodnotu `500`, `BILA` hodnotu `501` a nakonec `CERNA 700`.

Nyní můžete definovat proměnné typu `BARVA`, ale bude možné jim přiřadit pouze jednu z hodnot výčtu (v tomto případě `CERVENA`, `MODRA`, `ZELENA`, `BILA` nebo `CERNA`, čili `100`, `101`, `500`, `501` nebo `700`). Proměnné typu `BARVA` tedy můžete přiřadit libovolnou hodnotu barvy. Ve skutečnosti jí můžete přiřadit libovolnou celočíselnou hodnotu, i když se nebude jednat o povolenou barvu, ačkoli dobrý kompilátor vás na to upozorní. Je důležité vědět, že výčtové proměnné mají ve skutečnosti typ `unsigned int` a že výčtové konstanty jsou vlastně celá čísla. Je však velmi praktické, když je možné při práci s barvami, dny v týdnu a dalšími typy jejich hodnoty pojmenovávat. Ve výpisu 3.8 najdete program, který používá výčtový typ.

Výpis 3.8

Ukázka výčtových konstant

```
 0: #include <iostream>
 1: int main()
 2: {
 3:     enum Dny { Pondeli, Utery, Streda,
 4:         Ctvrtek, Patek, Sobota, Nedele };
 5:
 6:     Dny dnes;
 7:     dnes = Pondeli;
 8:
 9:     if ( dnes == Sobota || dnes == Nedele )
10:         std::cout << "\nMiluji vikendove dny!\n";
11:     else
12:         std::cout << "\nVzhuru do prace.\n";
13:
14:     return 0;
15: }
```

Výstup

```
Vzhuru do prace.
```

Analýza Na řádcích 3 a 4 se s pomocí sedmi hodnot definuje výčtový typ `Dny`. Každá jeho konstanta má celočíselnou hodnotu, kterou počítáme od nuly nahoru; `Utery` má tedy hodnotu 1.

Vytvoříme nyní proměnnou typu `Dny`, která bude uchovávat jednu z platných hodnot ze seznamu uvedených konstant. Této proměnné pak přiřadíme na řádku 7 výčtovou hodnotu `Pondeli` a tuto hodnotu podrobíme na řádku 9 testu.

Výčtový typ z tohoto programu by mohl být nahrazen řadou celočíselných konstant, jak ilustruje výpis 3.9.

Výpis 3.9

Stejný program, který používá řadu celočíselných konstant

```
 0: #include <iostream>
 1: int main()
 2: {
 3:      const int Pondeli = 0;
 4:      const int Utery = 1;
 5:      const int Streda = 2;
 6:      const int Ctvrtek = 3;
 7:      const int Patek = 4;
 8:      const int Sobota = 5;
 9:      const int Nedele = 6;
10:
11:      int dnes;
12:      dnes = Pondeli;
13:
14:      if ( dnes == Sobota || dnes == Nedele )
15:           std::cout << "\nMiluji vikendove dny!\n";
16:      else
17:           std::cout << "\nVzhuru do prace.\n";
18:
19:      return 0;
20: }
```

Výstup

```
Vzhuru do prace.
```

Analýza Výstup bude u výpisu 3.9 identický s výstupem programu 3.8. V programu 3.9 se každá z konstant explicitně definuje (`Pondeli`, `Utery` atd.) a žádný výčtový typ `Dny` zde nenajdeme. Výčtové konstanty mají tu výhodu, že dokumentují samy sebe, čímž je záměr výčtového typu `Dny` okamžitě zřejmý.

Pozor **Řada proměnných deklarovaných v tomto programu se nepoužívá. Proto může při jeho překladu kompilátor zobrazit varování.**

Souhrn

V této kapitole jsme se seznámili s číselnými a znakovými proměnnými a konstantami, které se v C++ používají k uchovávání dat během provádění programu. Číselné proměnné jsou buď celočíselné (`char`, `short` a `long int`), nebo s plovoucí řádovou čárkou (`float` a `double`). U celočíselných proměnných dále rozeznáváme typy `signed` a `unsigned`, tedy se znaménkem a bez znaménka. Přestože se velikosti jednotlivých typů mohou od sebe na různých počítačích lišit, má typ u daného počítače přesně specifikovanou velikost.

Dříve než je možné proměnnou použít, je třeba ji deklarovat a pak do ní uložit správná data toho typu, se kterým byla tato proměnná deklarována. Jestliže do celočíselné proměnné vložíte příliš velké číslo, dojde k jejímu „přetečení“, takže její hodnota se vrátí na nejnižší možnou hodnotu a jako důsledek získáte nesprávné výsledky.

V této kapitole jsme také zavedli pojmy literální, symbolické a v neposlední řadě i výčtové konstanty. Ukázali jsme dva způsoby, kterými lze symbolickou konstantu deklarovat: s pomocí příkazu preprocesoru `#define` a klíčového slova `const`.

Otázky a odpovědi

Otázka: **Jestliže u proměnné typu `short int` může dojít v paměti k přetečení a následnému vrácení hodnoty zpět, proč se vždy nepoužívají celá čísla typu `long`?**

Odpověď: K přetečení v paměti dochází jak u proměnných typu `short`, tak u proměnných typu `long`, ale u posledně uvedených k tomu dochází až v případě mnohem větších čísel. Například proměnná typu `unsigned short int` se vrátí zpět na nulu po dosažení hodnoty 65 535, zatímco u proměnné typu `unsigned long int` k tomu dojde až při hodnotě 4 294 967 295. U většiny počítačů zabírá každá celočíselná proměnná typu `long` dvojnásobné množství paměti oproti typu `short` (čtyři bajty oproti dvěma bajtům). Program se stem takových proměnných spotřebuje o dvě stě bajtů paměti RAM navíc. Upřímně řečeno, dnes se jedná o mnohem menší problém než dříve, protože většina osobních počítačů má miliony (ne-li miliardy) bajtů paměti. Používání větších než nutných typů může rovněž vyžadovat další čas vašeho procesoru.

Otázka: **Co se stane, když desetinné číslo přiřadím do celočíselné proměnné místo do proměnné typu `float`? Představte si následující řádek kódu:**

```
int aCislo = 5.4;
```

Odpověď: Dobrý kompilátor vás upozorní, ale přiřazení je zcela v pořádku. Číslo, které jste přiřadili, bude zarovnáno na celočíselnou hodnotu. Jestliže celočíselné proměnné přiřadíte hodnotu 5,4, bude hodnota této proměnné nadále 5. Dojde však k nenávratné ztrátě informace, a jestliže se později pokusíte přiřadit hodnotu z této celočíselné proměnné do proměnné typu `float`, bude výsledkem pouze hodnota 5,0.

Otázka: **Proč se nepoužívají literální konstanty? Proč se mám zatěžovat s používáním symbolických konstant?**

Odpověď: Jestliže jistou hodnotu používáte na mnoha místech programu, umožní vám symbolická konstanta změnit všechny tyto výskyty najednou pouhou změnou definice konstanty. Symbolické konstanty také mluví samy za sebe. Může být jen obtížně pochopitelné, proč se číslo násobí hodnotou 360; mnohem srozumitelnější je, když na tomto místě figuruje konstanta `stupnuVKruznici`.

Otázka: **Co se stane, když do proměnné typu `unsigned` přiřadím zápornou hodnotu? Představte si následující řádek kódu:**

```
unsigned int aKladneCislo = -1;
```

Odpověď: Dobrý kompilátor vás upozorní, ale jedná se o povolené přiřazení. K zápornému číslu se bude přistupovat jako k jeho bitové reprezentaci a ta se uloží do proměnné. Hodnota této proměnné se bude interpretovat jako číslo typu `unsigned`. Číslu -1 odpovídá bitová reprezentace 11111111 11111111 (hexadecimálně 0xFF); takové číslo je u typu `unsigned` reprezentováno hodnotou 65 535.

Otázka: **Mohu pracovat s C++, aniž bych musel rozumět bitovým reprezentacím, binární a hexadecimální aritmetice?**

Odpověď: Samozřejmě ano, ale nevytěžíte z jazyka tolik, jako kdybyste těmto otázkám rozuměli. Jazyk C++ vás na rozdíl od jiných programovacích jazyků příliš neochrání před tím, co počítač doopravdy dělá. Ve skutečnosti je to výhoda, protože tím získáte obrovskou sílu, kterou jiné jazyky nenabízí. Ale jako u každého silného nástroje, chcete-li z něj dostat maximum, musíte rozumět tomu, jak funguje. Programátoři, kteří se snaží programovat v C++ bez pochopení základů binárního systému, jsou často zmateni výsledky, které dostávají.

Úkoly pro vás

Testovací otázky v této části vám pomohou upevnit znalosti, které jste v této kapitole získali. Cvičení slouží k tomu, aby vám nabídlo praktickou zkušenost s tím, co jste se naučili. Pokuste se na otázky v testu a cvičení odpovědět dříve, než se podíváte do klíče v dodatku D. Než přejdete k další kapitole, ujistěte se, že každé odpovědi rozumíte.

Test

1. Jaký je rozdíl mezi celočíselnou proměnnou a proměnnou s plovoucí (pohyblivou) řádovou čárkou?
2. Jaký je rozdíl mezi proměnnými typu `unsigned short int` a `unsigned long int`?
3. Jaké výhody má používání symbolické konstanty místo literální konstanty?
4. Jaké výhody má používání klíčového slova `const` místo `#define`?
5. Podle čeho posuzujeme, zda je název proměnné vhodný nebo nevhodný?
6. U daného výrazu `enum` rozhodněte, jaká je skutečná hodnota konstanty `MODRA`.

```
enum BARVA {BILA, CERNA=100, CERVENA, MODRA, ZELENA=300} ;
```

7. Které z následujících názvů proměnných považujete za vhodné, které za nevhodné a které jsou neplatné?
   a/ `Vek`
   b/ `!ex`
   c/ `R79J`
   d/ `CelkovyPrijem`
   e/ `_Neplatne`

Cvičení

1. Jaký je správný typ proměnné pro uložení následujících informací?
   a/ Váš věk.
   b/ Výměra vaší zahrady.
   c/ Počet hvězd v galaxii.
   d/ Průměrné srážky v měsíci lednu.
2. Vytvořte k výše uvedeným informacím vhodné názvy proměnných.
3. Deklarujte konstantu pro Ludolfovo číslo (pí) jako 3,14159 (pozor na desetinný oddělovač).
4. Deklarujte proměnnou typu `float` a inicializujte ji s pomocí vaší konstanty pí.