Jeff Kent

C++

bez předchozích znalostí

průvodce pro samouky

Základní příkazy, vývojové prostředí, první programy

Srozumitelnou řečí, bez složité teorie

Kontrolní otázky a závěrečný test

Začněte i vy programovat v C++

CPRESS

McGraw Hill Osborne

Jeff Kent

C++
bez předchozích znalostí

Computer Press
Brno
2013

C++
bez předchozích znalostí

Jeff Kent

Překlad: Tomáš Znamenáček
Odborná korektura: Matěj Dusík
Obálka: Lance Lekander, Martin Sodomka
Odpovědný redaktor: Martin Domes
Technický redaktor: Jiří Matoušek

Original edition copyright © 2004 by The McGraw-Hill Companies. All rights reserved.
Czech edition copyright 2009 by Computer Press, a. s. All rights reserved.

Autorizovaný překlad z originálního anglického vydání C++ Demystified.
Originální copyright: © The McGraw-Hill Companies, 2004.
Překlad: © Computer Press, a. s., 2009.

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-2411-6

Vydalo nakladatelství Computer Press v Brně roku 2013 ve společnosti Albatros Media a. s.
se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16 903.
© Albatros Media a. s. Všechna práva vyhrazena. Žádná část této publikace nesmí být kopírována a rozmnožována za účelem rozšiřování v jakékoli formě či jakýmkoli způsobem bez písemného souhlasu vydavatele.

Dotisk 1. vydání

ALBATROS MEDIA a.s.

Obsah

Úvod 9

Kapitola 1

Jak funguje program napsaný v C++ 13

Co je počítačový program 13
Co je programovací jazyk 14
Anatomie programu v C++ 14
Překlad zdrojového kódu 17
Práce v integrovaném vývojovém prostředí 19
Test 28

Kapitola 2

Paměť a datové typy 31

Paměť 31
Datové typy 35
Projekt: Jak zjistit velikost datových typů 41
Shrnutí 46
Test 46

Kapitola 3

Proměnné 47

Deklarování proměnných 47
Přiřazování hodnot 51
Shrnutí 59
Test 59

Kapitola 4

Aritmetické operátory 61

Aritmetické operátory 61
Projekt: Automat na drobné 69
Shrnutí 71
Test 71

Kapitola 5

Rozhodování: příkazy if a switch 73

Relační operátory 73
Vývojové diagramy 75
Příkaz if 76
Příkaz if/else 80
Vícenásobné větvení 83
Příkaz switch 85
Shrnutí 90
Test 90

Kapitola 6

Vnořené podmínky a logické operátory 91

Vnořené podmínky 91
Logické operátory 95
Logické operátory a příkaz switch 100
Shrnutí 101
Test 102

Kapitola 7

Cyklus for 103

Operátory ++ a -- 103
Cyklus for 106
Shrnutí 114
Test 114

Kapitola 8

Cykly while a do while 115

Cyklus while 115
Cyklus do while 122
Shrnutí 125
Test 126

Kapitola 9

Funkce 127

Definování a volání funkce 127
Životnost proměnných a rozsah platnosti 130
Předávání parametrů 135
Návratová hodnota funkce 142
Shrnutí 143
Test 144

Kapitola 10

Pole 145

Deklarování pole 145
Inicializace 149
Nastavení a zobrazení polí 152
Předávání polí v parametrech funkcí 158
Shrnutí 159
Test 160

Kapitola 11

Abyste nebloudili aneb Ukazatele 161

Deklarace ukazatele 161
Přiřazování ukazatelům 163
Dereferencování ukazatelů 164
Ukazatele jako proměnné a konstanty 166
Ukazatelová aritmetika 167
Ukazatele jako parametry funkce 170
Dynamická alokace paměti 174
Ukazatel jako návratová hodnota funkce 176
Shrnutí 178
Test 179

Kapitola 12

Znaky, céčkové řetězce a třída string 181

Čtení znaků 181
Užitečné funkce pro práci se znaky 189
Užitečné funkce pro práci s řetězci 192
Shrnutí 197
Test 198

Kapitola 13

Trvalé uložení dat aneb Soubory 199

Přehled 199
Otevření souboru pro zápis 201
Otevření souboru pro čtení 203
Otevření souboru pro čtení i zápis 204
Kontrola chyb 204
Uzavření souboru 205
Zápis do souboru 206
Čtení ze souboru 207
Souborové proudy v parametrech funkcí 211
Shrnutí 212
Test 213

Kapitola 14

Vyhlídky do budoucna: Struktury a třídy 215

Proč jste si vybrali tuto knihu? 215
Objektově orientované programování 216
Struktury 216
Třídy 227
Shrnutí 230
Test 231

Závěrečný test 233

Správné odpovědi 237

Rejstřík 249

O autorovi

Jeff Kent vyučuje informatiku na Los Angeles Valley College v kalifornském Valley Glen. Přednáší více programovacích jazyků, například Visual Basic, C++, Javu nebo – když má zrovna masochistickou chvilku – assembler, ale většinou učí C++. Kromě toho spravuje síť jedné losangeleské právnické firmy, jejíž zaměstnanci slouží jako pokusní králíci pro jeho aplikace, a jako advokát udílí rady mladším advokátům (ať se jim to líbí, nebo ne). Je autorem několika knih o programování, jedním z jeho posledních titulů je *Visual Basic.NET: A Beginner's Guide* pro nakladatelství McGraw-Hill/Osborne.

Jeffova profesní dráha je pestrá – přesněji řečeno, jeho profesní dráhy jsou pestré. Promoval na UCLA jako bakalář ekonomie, pak vystudoval právo na losangeleské Loyola School of Law a pustil se do právnické praxe. Během této doby (kdy se o počítačích tak nanejvýš zdálo panu Gatesovi) se Jeff živil také jako šachista; získal třetí místo na mistrovství Spojených států do 21 let a později i mezinárodní titul.

I tak si najde čas pro svou ženu Devvie, což není zas tak těžké, protože i ona přednáší informatiku na Valley College. Zároveň Jeff plní úlohu osobního řidiče své mladší dcery Emily (ta starší, Elise, už má svůj vlastní řidičský průkaz) a ve zbývajícím volném čase si užívá přenosy mezinárodních šachových turnajů na Internetu. Jeho životním cílem je začít zase běhat maratony, protože jinak jeho příští kniha – vzhledem k jeho neúspěšné bitvě s nadváhou – pravděpodobně ponese titul *Sumo bez záhad*.

Tuto knihu bych chtěl věnovat své ženě Devvie Schneider Kentové. V osobním i profesním životě mi dala tolik, že by to vydalo na samostatnou knihu (jejíž malou část najdete v Poděkováních). Kromě jiného je i má učitelka programování – bez ní bych tuto ani žádnou jinou knihu o programování nikdy nenapsal.

—Jeff Kent

Poděkování

Poděkování autora vydavateli vypadá jako povinnost (zvlášť pokud chce autor pro vydavatele ještě někdy napsat nějakou knihu), ale já ho myslím upřímně. Toto je má čtvrtá kniha pro nakladatelství McGraw-Hill/Osborne a já doufám, že ještě přijde mnoho dalších. Opravdu mne těší spolupracovat s profesionály, kteří jsou zároveň příjemní lidé a zároveň velice dobří v tom, co dělají – i kdyby to zrovna mělo být pečlivé sledování všech termínů, které se mi podařilo nedodržet.

Jako první bych chtěl poděkovat Wendy Rinaldiové, která mě do vydavatelství McGraw-Hill/Osborne kdysi v roce 1998 uvedla (vážně už je to tak dávno?). I tato kniha ostatně začala telefonickým hovorem s Wendy. Zrovna nám se ženou končila dovolená a Devvie, která byla na doslech od telefonu, se mě nevěřícím tónem vyhrazeným potenciálním šílencům zeptala, jestli *skutečně hodlám psát další knihu*.

Také musím poděkovat své akviziční koordinátorce Atheně Honoreové a projektové redaktorce Lise Wolters-Broderové. Obě byly nekonečně trpělivé, vždy ochotné mi pomoci a zároveň dbaly, abych se neodchýlil od stanovených termínů. (Které se v oboru hodně drží. „Je nám doopravdy líto, že jste si zlomil obě ruce a nohy; ale do pátku nám prosím odevzdejte další kapitolu, ano?“)

Redaktorské práce provedl Mike McGee společně s Lisou. Oba projevili velkou shovívavost k výpadkům, které zjevně musely doprovázet má školní léta. Text vylepšili, aniž by se odchýlili od jeho původního významu, takže jakékoliv chyby padají na mou hlavu. Mike navíc naznačil svou slabost pro některé z mých otřepaných vtipů, čímž si mě navěky získal.

Technickým redaktorem byl Jim Keogh. Mezi mnou a Jimem panoval vyvážený vztah vzájemného ohrožení – zatímco on byl technickým redaktorem mé knihy, já jsem byl technickým redaktorem dvou jeho knih *Data Structures Demystified* a *OOP Demystified*. Ale vážně: Jimovy poznámky pomohly mně i knize.

Při vydání knihy asistovala řada dalších talentovaných lidí, ale jak se říká během přebírání Oskarů, všechny je tu vyjmenovat nemůžu. To neznamená, že bych si nevážil jejich práce – vážím.

Upřímně děkuji své ženě, která je i můj nejlepší (ne-li jediný) kamarád a parťák. (Jako nejlepší milenku ji uvést nemůžu, protože programátoři se přeci o takové věci nezajímají.) Kromě jiného byla i můj technický redaktor. Má pro takovou práci kvalifikaci, protože už patnáct let učí informatiku, a navíc je pedant na jazyky (ano, já vím, slovo *optimální* se nestupňuje). Moc této knize pomohla.

A konečně si můj dík zaslouží mé dcery Elise a Emily a moje maminka Bea Kentová za svou shovívavost, když jsem se omlouval z rodinných setkání a něco si mumlal o nepřiměřených termínech a ukrutných redaktorkách (pardon, Atheno a Liso). Celé své rodině bych chtěl předem poděkovat za to, že mě nezbaví svéprávnosti, až začnu uvažovat o další knize.

Úvod

C++ byl můj první programovací jazyk. I když jsem se od té doby naučil další, C++ jsem vždycky považoval za „nejlepší"; nejspíš kvůli tomu, jakou moc programátorům dává. Tato moc je samozřejmě dvojsečná zbraň, kterou si při troše neopatrnosti můžete uříznout vlastní větev. Přesto mám C++ ze všech jazyků nejraději.

Nejsem sám, kdo má C++ rád. Kromě komerční sféry, které C++ nabízí svou velkou sílu, je oblíbené i v akademickém světě. Navíc z něj vychází mnoho dalších jazyků, včetně Javy a C#. (Java byla ostatně v C++ napsána.) Znalost C++ tím pádem usnadňuje studium dalších jazyků.

Proč jsem knihu napsal

Nikoliv kvůli bohatství, slávě nebo krásným ženám. (Možná jsem trochu sešel z cesty, ale trocha zdravého rozumu mi ještě zbyla.) Není pochyb o tom, že úvodů do C++ existuje hodně. Přesto jsem tuto knihu napsal, protože si myslím, že mohu nabídnout jinou, a doufám že užitečnou perspektivu.

Jak možná víte z mého životopisu, učím informatiku na Los Angeles Valley College – vyšší odborné škole ze San Fernando Valley v Los Angeles, kde jsem vyrostl a prožil většinu svého života. Dělám i programátora, ale výuka programování mě o programování naučila věci, které bych se v praxi jinak nedozvěděl. Nejde jen o zodpovídání studentských dotazů během lekcí. Každý týden strávím hodiny v naší počítačové učebně a pomáhám studentům s jejich programy; každý týden strávím hodiny opravováním a známkováním jejich úkolů. Postupem času se ukázalo, jaký přístup doopravdy funguje, v jakém pořadí se mají probírat jednotlivá témata, na jaké úrovni složitosti se k nim poprvé dostat a podobně. Z legrace svým studentům říkám, že jsou beta testeři mých věčných pokusů o lepší výuku, ale v tom vtipu je velký kus pravdy.

Moji bet – studenti si navíc věčně stěžují na učebnici, ať už vyberu jakoukoliv. Hodně z nich se ptá, proč nějakou nenapíšu sám. Možná se mi jen snaží lichotit (neříkám, že to nefunguje), možná by mě kromě mizerné výuky rádi popotahovali i za mizernou učebnici. Protože jsem už ale několik knih napsal, jejich otázky mi do hlavy nasadily nápad na knihu, která by kromě široké veřejnosti posloužila i jako doplněk k učebnici.

Komu je kniha určena

Komukoliv, kdo zaplatí. Dělám si legraci. (Ale žádnému kupci neřeknu ne!)

Vydavatelé i autoři chtějí pro každou knihu získat co nejširší publikum, to není žádná novinka. Tato část knihy proto většinou vysvětluje, že kniha je přesně pro vás, ať už jste kdokoliv a děláte cokoliv. Žádná programátorská kniha ale není pro každého. Pokud například programujete výhradně hry v Javě, tato kniha vám nejspíš moc nepomůže. (I když jako učitel bych mohl být váš další zákazník, neměli byste zájem o titul *Učitelé versus vetřelci z vesmíru*?)

Takže i když tato kniha samozřejmě není pro každého, pro vás by docela dobře být mohla. C++ se chce nebo musí naučit hodně lidí, ať už v rámci univerzitního studia, pracovního školení nebo

třeba i z vlastního zájmu. C++ není úplně nejjednodušší téma. Řada autorů ho navíc moc neusnadňuje, protože před vás položí komplikovaný telefonní seznam plný cizích výrazů. Tato kniha se vám naproti tomu snaží vysvětlit C++ bez záhad, jak už napovídá její titul. Jde rovnou k základním pojmům a vysvětluje je pěkně po pořádku, hezky česky.

Co v knize najdete

Jsem pevným zastáncem myšlenky, že programovat se člověk naučí nejlépe programováním. Proto jsou myšlenky jednotlivých kapitol ilustrované jasně a podrobně vysvětleným kódem. Můžete si ho sami spustit a můžete nad ním postavit programy, na kterých si popisované myšlenky vyzkoušíte podrobněji.

První kapitola vás dostane do tempa – například vám vysvětlí, co je počítačový program a co programovací jazyk. Pak popisuje anatomii základního programu v C++, a to včetně „dění za oponou", tedy jak preprocesor společně s překladačem a linkerem přeloží váš kód do podoby srozumitelné počítači. Nakonec vám první kapitola ukáže, jak vytvořit a spustit projekt v integrovaném vývojovém prostředí (IDE).

Schopnost napsat a spustit program, který na obrazovku vypíše `Hello World`, je dobrý začátek. Většina programů ale potřebuje pracovat s daty, například čísly a textem. Druhá kapitola proto popisuje různé typy počítačové paměti, včetně paměti s náhodným přístupem neboli RAM. Následuje diskuse o adresách, které popisují umístění dat v RAM, a bajtech neboli jednotkách potřebných pro uložení informací. A protože informace mívají různou podobu, kapitola se dále věnuje různým datovým typům pro celá čísla, desetinná čísla a text.

Hlavní hvězdou třetí kapitoly je proměnná, která nejenže rezervuje místo v paměti pro uložení informací, ale navíc vám poskytne jméno, pod kterým můžete s těmito informacemi pracovat. Úkolem proměnných je ukládat informace, takže proměnná bez přiřazené hodnoty je užitečná zhruba stejně jako banka bez peněz. Proto kapitola dále vysvětluje, jak se proměnným přiřazují hodnoty – ať už během překladu pomocí operátoru přiřazení, nebo za běhu pomocí objektu `cin` a operátoru čtení z proudů (>> a <<).

Jako bývalého profesionálního šachistu mě vždycky ohromoval fakt, že šachový počítač může sehrát vyrovnanou partii se světovým šampiónem. Počítače těží ze své schopnosti provádět výpočty mnohem rychleji a přesněji než člověk. Aritmetické operátory, díky kterým můžeme jejich výpočetní kapacitu využít i my, popisuje čtvrtá kapitola.

Aby mohl program zvládat i složitější úkoly, musí mít možnost měnit průběh výpočtu podle pravdivosti určité podmínky. Kdybyste například využili aritmetické operátory ze čtvrté kapitoly a naprogramovali kalkulačku, konkrétní aritmetická operace počítaná vaším programem by se lišila podle toho, jestli uživatel vybral sčítání, odčítání, násobení, nebo dělení. V páté a šesté kapitole se proto podíváme na relační a logické operátory, které se hodí při rozhodování, a příkazy `if` a `switch`, kterými se dá na základě tohoto rozhodování měnit průběh výpočtu.

Malé děti své rodiče někdy trápí tím, že něco opakují neustále dokola. Občas je potřeba opakovat i kus kódu. Když například uživatel aplikaci zadá chybná data, můžete ho dokola prosit o nové zadání, dokud vám data nezadá správně nebo dokud program neukončí. Hlavním předmětem sedmé a osmé kapitoly jsou takzvané smyčky, které slouží k opakovanému provádění kódu po dobu platnosti nějaké podmínky. Sedmá kapitola se zabývá smyčkou `for` a zároveň vám ukáže operátory zvýšení a snížení hodnoty proměnné, které se ve smyčkách často používají. Osmá kapitola doplní diskusi smyčkami `while` a `do while`.

Devátá kapitola je o funkcích. Funkce je blok jednoho nebo více příkazů. Většina kódu, který v C++ napíšete, bude součástí nějaké funkce. Tato kapitola vám vysvětlí, proč se kód dělí do funkcí a jak se funkce dělají. Ukáže vám, jak se píše prototyp funkce, jak se funkce definuje a jak se volá. Také se dozvíte, jak pomocí parametrů předat funkci informace a jak informace z volané funkce vrátit. Vysvětlený bude i rozdíl v předávání parametrů hodnotou a odkazem. Nakonec si vysvětlíme rozsah platnosti proměnných, životnost proměnných a rozdíl mezi lokálními, statickými a globálními proměnnými.

Desátá kapitola je věnovaná polím. Od proměnných popisovaných v předchozí části knihy se pole liší v tom, že mohou najednou obsahovat větší počet hodnot. Často se používají ve spojení se smyčkami, o kterých se mluví v kapitolách sedm a osm. Mimo jiné se podíváme na rozdíly mezi polem znaků a polem jiných datových typů. Nakonec probereme konstanty, které se v mnohém podobají proměnným, ale jejichž hodnota se po dobu běhu programu nemění.

Jedenáctá kapitola je o ukazatelích. Při zaslechnutí slova ukazatel se mnohým adeptům C++ ježí chlupy na zádech, ale zbytečně. Jak už jsme si říkali v souvislosti s druhou a třetí kapitolou, informace se v paměti ukládají na nějakou adresu. Ukazatele jsou jednoduše efektivní nástroj pro práci s těmito adresami. V jedenácté kapitole se také dozvíte o operátoru hvězdička (`*`), dereferencování a ukazatelové aritmetice.

Hodně informací bývá uložených v podobě znaků, céčkových řetězců nebo řetězcových tříd C++. Kapitola dvanáct se proto věnuje užitečným funkcím pro práci s těmito datovými typy, například členským funkcím třídy `cin`.

Aby informace zůstaly k dispozici i po skončení programu, často se ukládají do souborů. Kapitola třináct vám vysvětlí práci se souborovými proudy, tedy třídami `fstream`, `ifstream` a `ofstream` a jejich členskými funkcemi `open`, `read`, `write` a `close`.

A konečně abyste měli solidní základy i pro další studium po absolvování této úvodní knihy, uvede vás čtrnáctá kapitola do objektově orientovaného programování (OOP) a dvou konceptů, které se v něm často používají: struktur a tříd.

Za každou kapitolou najdete test, kterým si můžete ověřit, jestli jste základní pojmy kapitoly bezpečně zvládli. (Na rozdíl od školních testů budete mít v druhém dodatku i odpovědi.) V prvním dodatku na vás čeká velká závěrečná prověrka; i její řešení najdete v druhém dodatku.

Jak knihu číst

Psal jsem knihu tak, aby se dala číst od začátku do konce. Na první pohled se asi zdá, že to jinak ani nejde. Od svých studentů ale často slýchám oprávněné stížnosti na to, že se učitel nebo kniha během vysvětlování myšlenky opírají o pojmy, které jsou vysvětlené až o několik kapitol dál (nebo v horším případě vůbec ne). Proto jsem se důsledně snažil postupovat lineárně, logicky. Jednak se vyhnete frustracím z textu, který je třeba číst napřeskáčku, a jednak můžete v každé kapitole stavět na výsledcích té předchozí.

Speciality

Každou kapitolu doprovází poznámky, tipy, upozornění na problematická místa a výpisy kódu. Abyste měli lepší zpětnou vazbu, najdete za každou kapitolou malý test a za celou knihou pak v prvním dodatku závěrečnou zkoušku. Odpovědi k oběma jsou v druhém dodatku. Hlavním cílem knihy je, abyste se rychle dostali do tempa, bez zbytečné suché teorie a nadbytečných podrobností. Tak pojďme na to. Programování v C++ není těžké a je to zábava.

Kontakt na autora

K čemu? (No dobře.) Nadšené ovace a lichotky nikdy neublíží, ale uvítám i komentáře, rady a dokonce snad i kritiky. Nejlepší bude, když mi napíšete e-mail na adresu *jkent@genghiskhent.com* (doména je nazvaná podle přezdívky, kterou jsem dostal od svých studentů). Případně se můžete podívat na mé webové stránky *www.genghiskhent.com*. Nenechte se zmást titulní stránkou – web slouží především jako zdroj informací pro mé přednášky, ale je na něm i odkaz týkající se této knihy.

Doufám, že si knihu užijete stejně, jako jsem si já užil její psaní.

Poznámka redakce českého vydání

Nakladatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
redakce PC literatury
Holandská 8
639 00 Brno

nebo *knihy@cpress.cz*

Další informace a případné opravy českého vydání knihy najdete na internetové adrese *http://knihy.cpress.cz/k1370*. Prostřednictvím uvedené adresy můžete též naší redakci zaslat komentář nebo dotaz týkající se knihy. Na vaše reakce se srdečně těšíme.

Jak funguje program napsaný v C++

1

S počítačovými programy se během průměrného dne potkáte mnohokrát. Když dorazíte do práce a zjistíte, že vám nefunguje počítač, zavoláte technickou podporu. Na druhém konci telefonní linky je počítačový program, který vás donutí prokličkovat bludištěm hlasového systému a během nekonečného čekání vás mučí neupřímnými vzdechy o tom, jak si vašeho hovoru váží a jak už vás dozajista brzy přepojí.

Když pak skoncujete s technickou podporou, rozhodnete se přihlásit k opravenému počítači a rozdat si to s gigantickými hmyzáky z planety Megazoid. Správce sítě vás bohužel nachytá na švestkách, a to díky dalšímu počítačovému programu pro sledování počítačů. Vaši výplatní pásku, pokud kdy ještě nějakou uvidíte, sestaví opět program.

Na cestě domů si uvědomíte, že potřebujete nějakou hotovost. Zastavíte se tedy u bankomatu, kde vám počítačový program – doufejme – potvrdí, že máte dost peněz na účtu, a nakáže bankomatu vyplatit hotovost, načež – bohužel – tutéž částku strhne z vašeho účtu.

Většina lidí se během tohoto každodenního setkávání s počítačovými programy vůbec nemusí zajímat, jak vlastně počítačový program vypadá nebo funguje. Počítačový programátor by to ale vědět měl – stejně jako řadu dalších souvisejících věcí, například co je to programovací jazyk nebo jak vlastně funguje program napsaný v C++. Až dočtete tuto kapitolu, budete odpovědi na podobné otázky znát a budete umět napsat a spustit svůj vlastní počítačový program.

Co je počítačový program

Počítače jsou v naší společnosti tolik rozšířené, protože mají oproti nám lidem tři výhody. Za prvé zvládají uložit ohromné množství informací, za druhé si tyto informace umí rychle a přesně vybavit, a za třetí umí rychlostí blesku provádět přesné výpočty.

Tyto výhody se přenáší i do intelektuálních sportů, například do šachu. V roce 1997 porazil počítač Deep Blue světového šachového šampióna Garry Kasparova. V roce 2003 se chtěl Kasparov revanšovat jinému počítači nazvanému Deep Junior, ale pouze remízoval. Přestože je Kasparov pravděpodobně vůbec nejlepší šachista všech dob, je pouze člověk, a tím pádem se nemůže měřit s rychlostí a kapacitou počítačů.

My ale máme oproti počítačům jednu zásadní výhodu, alespoň zatím: umíme samostatně přemýšlet. Počítač není chytrý, je jen rychlý. Cokoliv provádí, dělá jen na základě podrobného návodu,

počítačového programu. Autorem tohoto programu je samozřejmě člověk, programátor. Počítačové programy nám umožňují využít obrovskou výpočetní kapacitu počítače.

Co je programovací jazyk

Když vejdete do tmavé místnosti a chcete se rozhlédnout kolem, zapnete světlo. Když odcházíte, světlo vypnete.

První počítače se podobaly právě takovému vypínači. Obsahovaly řadu přepínačů a drátů a elektrický proud jimi procházel po různých cestách podle toho, které vypínače byly právě zapnuté a vypnuté. Sám jsem si takový jednoduchý počítač postavil, když jsem byl malý (což v představách mých dětí bývalo ještě v dobách, kdy se po Zemi proháněli dinosauři).

Stav každého z vypínačů se dá vyjádřit číslem: jednička je vypínač zapnutý, nula vypínač vypnutý. Pokyny pro první počítače, tedy stav jejich jednotlivých vypínačů, byly proto v podstatě řady nul a jedniček. Dnešní počítače jsou samozřejmě mnohem výkonnější a složitější. Takzvaný strojový jazyk (tedy jazyk, kterému počítače rozumí) se ale nezměnil – pořád jde o nuly a jedničky.

Zatímco počítače přemýšlí v nulách a jedničkách, lidé obyčejně ne. Složitý program se navíc může skládat z milionů strojových příkazů, takže by jeho psaní zabralo neúnosně dlouhou dobu. To je důležitý faktor, protože díky tlaku konkurence se programy musí psát rychleji a rychleji.

My naštěstí programy ve strojových příkazech psát nemusíme. Můžeme místo nich použít příkazy *programovacího* jazyka, které jsou nám mnohem srozumitelnější, protože jsou bližší lidskému vyjadřování než nulám a jedničkám. V programovacím jazyce se navíc programuje mnohem rychleji, protože má vyšší vyjadřovací schopnost – jeden příkaz programovacího jazyka vydá za řadu příkazů jazyka strojového.

C++ je jedním z mnoha programovacích jazyků. Mezi další oblíbené programovací jazyky patří Java, C# nebo VisualBasic. Existuje řada dalších a neustále vznikají nové. Všechny programovací jazyky ale mají v podstatě tentýž účel – umožnit lidem vysvětlit počítači, co přesně má dělat.

Proč byste měli dát C++ přednost před jinými jazyky? Za prvé je hojně rozšířené, mezi firmami i na školách. Za druhé z něj vychází hodně dalších jazyků (například Java nebo C#); Java v něm byla přímo napsána. Když budete umět C++, bude pro vás snazší zvládnout další programovací jazyky.

Anatomie programu v C++

Knihy o programování v C++ většinou dodržují nepsanou tradici a jako první ukázku kódu svým čtenářům předloží program, který na obrazovku vypíše zprávu „Hello, world!“ (viz obrázek 1.1).

Poznámka: Text `Press any key to continue` vypsaný těsně za `Hello, world!` není součástí programu. Jen vám napovídá, jak zavřít příkazový řádek.

Za tímto prvním příkladem většinou bohužel rychle následují další, aniž by se kniha nebo učitel na chvilku zastavili a vysvětlili, jak vlastně *Hello world* funguje. Výsledkem je dezorientovaný student nebo čtenář, který je připravený se s krutým světem C++ zase hodně rychle rozloučit.

Hello world sice vypadá jednoduše, ale za oponou se toho děje hodně. Proto si teď program projdeme řádku po řádce, byť napřeskáčku.

Obrázek 1.1: Program v C++ vypisuje na obrazovku zprávu „Hello, world!“

```
#include <iostream>
using namespace std;
int main(void)
{
    cout << "Hello, world!";
    return 0;
}
```

Poznámka: Kód, který píše programátor, se většinou označuje jako *zdrojový kód*. Ukládá se do souboru, který má v případě C++ příponu .cpp (cé plus plus).

Funkce main

Na začátku kapitoly jsme si řekli, že v C++ i všech ostatních programovacích jazycích zadává programátor příkazy počítači. Úloha, kterou má počítač provést, se většinou nevejde do jednoho jediného příkazu; bývá potřeba několik souvisejících příkazů. *Funkce* je skupina právě takových souvisejících příkazů, které společně slouží k provedení nějakého úkolu.

Každá funkce má jméno, kterým se na její příkazy můžete odvolávat. Výraz `main` v našem zdrojovém kódu *Hello world* je název funkce. Programy mívají funkcí víc (psaním funkcí se bude zabývat devátá kapitola), ale každý program v C++ má jen jednu a právě jednu hlavní funkci jménem `main`. Tato funkce je vstupním bodem programu – kdyby ji program neměl, počítač by nevěděl, kde program začít provádět. A kdyby naopak program obsahoval více než jednu funkci `main`, počítač by nevěděl, kterou z nich si má vybrat.

Poznámka: Významem slova `int` před názvem funkce `main` a závorkami se slovem `void` se budeme zabývat v deváté kapitole.

Tělo funkce

Všechny příkazy funkce `main` jsou uvedené v takzvaném *těle* funkce, které začíná levou složenou závorkou `{` a končí pravou složenou závorkou `}`.

Většina příkazů končí středníkem. V naší funkci `main` máme příkazy dva:

```
cout << "Hello, world!";
return 0;
```

Příkazy se provádí popořadě, odshora dolů. Prvním z nich je:

```
cout << "Hello, world!";
```

`cout` je objekt, který slouží pro výstup dat (proto *out* jako *ven*). Patří mezi takzvané *datové proudy*, které se dělí na dvě skupiny. První skupinu tvoří výstupní datové proudy, které slouží k výstupu dat z programu, například na monitor, na tiskárnu nebo do souboru. Druhou skupinu tvoří vstupní datové proudy, které řeší načítání dat do programu, například ze souboru nebo klávesnice.

`cout` je výstupní datový proud, který posílá data na standardní výstupní zařízení. V běžných případech je standardním výstupním zařízením monitor, ale můžete ho přesměrovat i na tiskárnu nebo soubor na pevném disku.

Konstrukce `<<` za `cout` je operátor. S nějakými operátory už jste se asi setkali – například aritmetickými operátory `+`, `-`, `*` a `/`, které slouží ke sčítání, odčítání, násobení a dělení. Operátor `<<` slouží ke vkládání dat do proudu. Vezme informace ze své pravé strany (v tomto případě text `Hello, world!`) a vloží je do proudu na své levé straně, což je v tomto případě standardní výstupní zařízení a tedy nejčastěji monitor.

Poznámka: Ve třetí kapitole se dozvíte o protějšku k objektu `cout`. Jmenuje `cin` a obsluhuje standardní vstup; používá se ve spojení s operátorem `>>`.

Druhý a poslední příkaz vrací operačnímu systému (ať už Windows, Unixu nebo čemukoliv jinému) chybový kód nula. Podle něj operační systém pozná, že program skončil úspěšně. Kdyby program skončil neúspěšně, například kdyby vyčerpal veškerou volnou paměť, vrátil by nenulový chybový kód a operační systém by věděl, že při běhu programu došlo k chybě.

Direktiva include

Váš program „ví“, že má začít funkcí `main`, protože toto chování předepisuje jazyk C++. Program funkcí `main` začne, aniž byste kvůli tomu museli psát jediný řádek kódu. Podobně program ví, že objekt `cout` ve spojení s operátorem `<<` vypisuje informace na monitor. Rozhodně jsme kvůli tomu nemuseli psát žádný kód navíc. Objekt `cout` ale na rozdíl od funkce `main` není součástí samotného jazyka C++, patří do takzvané *standardní knihovny*. Standardní knihovna C++ obsahuje řadu souborů, z nichž každý definuje některé běžně používané objekty. Výpis na obrazovku rozhodně mezi běžné činnosti patří – mohli byste si ho sice napsat sami, ale jeho implementace ze standardní knihovny vám šetří starosti s „objevováním Ameriky“.

Objekt `cout` už tedy máte k dispozici ve standardní knihovně, ale abyste ho mohli použít, musíte do svého programu vložit příslušný soubor ze standardní knihovny. K tomu slouží direktiva `#include`, za kterou následuje název knihovního souboru. Pokud soubor patří do standardní knihovny (tedy pokud jste si ho nepsali sami, což také můžete), jeho název se uzavírá do špičatých závorek `<` a `>`.

Objekt `cout` je definovaný ve standardním knihovním souboru `iostream`. Písmena „io“ v názvu souboru jsou zkratka od slov *input* a *output*, vstup a výstup, a *stream* znamená proud. Pokud chceme používat objekt `cout` ve svém programu, musíme sáhnout do standardní knihovny a vložit z ní do našeho programu soubor `iostream`. K tomu slouží příkaz:

```
#include <iostream>
```

Direktiva `#include` je instrukcí pro *preprocesor* – jeden z programů, které budou zdrojový kód zpracovávat. Po preprocesoru se vašemu kódu bude věnovat ještě překladač a linker, o všech třech programech se podrobně pobavíme v následující části kapitoly. Direktivy preprocesoru se na rozdíl od běžných příkazů neukončují středníkem.

Jmenné prostory

Poslední příkaz, který nám zbývá, je:

```
using namespace std;
```

Každý program obsahuje řadu jmen. Například jméno `main` označuje hlavní funkci, jméno `cout` označuje objekt pro práci se standardním výstupem a podobně. Asi není potřeba zdůrazňovat, že jména musí být do velké míry jedinečná: Kdybyste měli dva objekty jménem `cout`, počítač by při zmínce o objektu `cout` nevěděl, o kterém z nich mluvíte.

Problém je v tom, že větší programy se obvykle skládají z mnoha různých kusů napsaných různými programátory. Když tyto části skládáte dohromady, snadno se může stát, že jméno použité v některé z nich už je zabrané jinou. K oddělení jmen z různých částí programu slouží právě *namespaces* neboli *jmenné prostory*. Ty fungují podobně jako naše příjmení, jen se místo za jméno dávají před něj a oddělují se dvěma dvojtečkami (`::`). Například objekt `cout` patří do jmenného prostoru standardní knihovny, která se jmenuje `std`, a tak se `cout` plným jménem jmenuje `std::cout`.

Když je v běžné řeči zjevné, koho myslíme, obejdeme se i bez příjmení. Stejně tak ve zdrojovém kódu je praktičtější uvádět plná jména jen tehdy, když hrozí nějaká záměna. Proto se můžete s počítačem dohodnout, že kdykoliv najde jméno bez jmenného prostoru, doplní si automaticky nějaký výchozí jmenný prostor. Příkaz `using namespace std` říká, že před každé jméno bez jmenného prostoru si má počítač doplnit `std::`.

Překlad zdrojového kódu

Vy už teď zdrojovému kódu *Hello world* rozumíte, ale počítač ještě ne. Počítač sám o sobě nerozumí C++ ani žádnému jinému programovacímu jazyku. Rozumí pouze svému strojovému jazyku. O překlad zdrojového kódu do spustitelné podoby neboli *binárky*, která už je srozumitelná počítači, se starají tři programy. V pořadí, ve kterém se dostanou k vašemu zdrojovému kódu, jsou to:

1. Preprocesor
2. Překladač
3. Linker

Preprocesor

Preprocesor je program, který se stará o předzpracování (*pre–processing*) zdrojového kódu. Když například narazí na direktivu `#include`, nahradí ji obsahem odkazovaného souboru. V našem případě se odkazujeme na soubor `iostream` ze standardní knihovny, a tak preprocesor na místo původní direktivy vloží obsah tohoto souboru (který mimo jiné obsahuje definici objektu `cout`).

Překladač

Překladač je další program, který vezme předzpracovaný zdrojový kód (tedy kód, ve kterém už preprocesor provedl všechny své náhrady) a přeloží ho do odpovídajících příkazů strojového jazyka. Výsledek uloží do samostatného *objektového souboru* s příponou `.obj` nebo `.o`. Každý jazyk má svůj vlastní překladač, ale základní úkol všech překladačů je v podstatě tentýž: Přeložit příkazy z programovacího jazyka do strojového.

Překladač váš zdrojový kód pochopí a přeloží pouze v případě, že se budete držet syntaxe daného programovacího jazyka. C++ má pevná pravidla pro zápis jednotlivých slov a jejich skládání do větších celků – v tom se nijak neliší od všech ostatních, nejen programovacích jazyků. Když budete mít v zápisu syntaktickou chybu, překladač váš program do strojového jazyka nepřeloží a bude si na chybu stěžovat. Z tohoto pohledu tedy překladač zároveň funguje jako kontrola překlepů a gramatiky.

Linker

V objektovém souboru už jsou sice příkazy strojového jazyka, ale spustit se ještě nedá. Musí se k němu ještě přidat kód z knihoven, které jste použili při psaní programu, například kód pro načítání vstupu z klávesnice nebo výpis na monitor. To za vás udělá linker. Jeho výstupem je spustitelný soubor, který v našem případě vypíše zprávu `Hello, world!`.

Proč tak složitě?

Možná vám teď překlad programu připadne zbytečně složitý. K čemu ukládat objektové soubory, když je stejně používá jen linker? Nebylo by jednodušší mít jeden program, který by ze zdrojového kódu bez větších okolků rovnou vytvořil binárku? Odpověď zní, že by to jednodušší bylo, ale jen pro menší programy.

Zdrojový kód každého alespoň trochu většího programu je hodně dlouhý, od stovek řádků až po desítky milionů. Kdybychom takový zdrojový kód nechali v jednom souboru, měli bychom s ním potíže my i počítač – my bychom se v něm nevyznali a počítači by jeho překlad do strojové podoby trval neúnosně dlouho.

Potřebujeme tedy zdrojový kód rozdělit na víc souborů. Je jasné, že nemá smysl zdrojový kód dělit řekněme po dvou stech řádcích na soubor, to by nám v orientaci moc nepomohlo. Zdrojový kód se dělí podle logických celků – co zdrojový soubor, to nějaká poměrně samostatná součástka programu. Takových zdrojových souborů čili součástek bývají u větších projektů desítky až stovky.

Jak bude probíhat překlad takto rozděleného kódu? Samozřejmě by šlo jednotlivé soubory pospojovat do jednoho velkého a ten pak přeložit. Tím bychom si ale moc nepomohli: překlad by trval stejně dlouho jako prve. Klíčový postřeh je v tom, že kdykoliv měníte velký program, měníte jen jeho malou část. Změníte jednu součástku, jeden soubor, a program přeložíte. Nabízí se nám tak možnost překládat jednotlivé zdrojové soubory samostatně a překládat vždy jen ty, které se od posledního překladu změnily.

Překladač tedy dostane ke zpracování seznam souborů, které se změnily, a každý z nich přeloží do objektového kódu. Pak přijde na řadu linker, který vezme všechny objektové soubory vašeho programu a vytvoří z nich spustitelný soubor. Když se změní jen jeden zdrojový soubor, překladač změní jen jeden objektový soubor a ušetří si většinu práce. Linker má vždycky práce stejně, ale to nevadí, protože linker je nesrovnatelně rychlejší než překladač.

A k čemu potřebujeme preprocesor? Musíte si uvědomit, že překladač nemůže s jednotlivými soubory pracovat úplně samostatně. Když chcete používat například objekt `cout` definovaný v jiném souboru, překladač musí mít přístup k definici tohoto objektu. Preprocesor se stará, aby měl překladač k dispozici definice všech objektů potřebných pro překlad aktuálního souboru.

Přijde vám to složité? Nebojte, ve skutečnosti je to ještě složitější. Ale tím se teď nemusíte trápit, přesné principy překladu nejsou prozatím podstatné. Jednak se k nim můžete vrátit později a jednak existují integrovaná vývojová prostředí, která vás od překladu do velké míry odstíní.

Práce v integrovaném vývojovém prostředí

Zdrojový kód můžete psát v libovolném textovém editoru, klidně i v Poznámkovém bloku. Také překladač si můžete stáhnout zdarma, většinou s ním dostanete i preprocesor a linker. Program si pak přeložíte ručně, například v Příkazovém řádku Windows.

Na textovém editoru a překladu z příkazové řádky sice není vůbec nic špatného, ale hodně programátorů – včetně mě – dává přednost integrovaným vývojovým prostředím neboli IDE. Slovo *integrovaný* znamená, že dostanete textový editor, preprocesor, překladač a linker pod jednou softwarovou střechou. IDE se o překlad kódu postará za vás, takže se nemusíte jednotlivými programy zabývat ručně. Většina IDE má navíc grafické rozhraní, které je pro hodně lidí schůdnější než příkazová řádka; a konečně většina IDE nabízí další funkce, které vám pomůžou s hledáním a opravou chyb.

Hlavní nevýhoda je v tom, že za většinu IDE musíte platit (i když existují i některá zdarma). IDE také zabere víc místa na disku a v paměti. Já bych vám IDE doporučil, protože se díky němu můžete soustředit výhradně na programování v C++ a nemusíte řešit, který přepínač máte použít na příkazové řádce.

Na trhu je mnoho dobrých IDE pro různé operační systémy a platformy. Dobrá zpráva je, že hodně výrobců nabízí limitované verze svých produktů pro nekomerční účely zadarmo. Microsoft například nabízí své komplexní řešení pro vývoj zvané Microsoft Visual Studio 2008 Express Edition ke stažení zdarma. Nás bude zajímat především nástroj Microsoft Visual C++ 2008. Borland nabízí svůj C++ Builder pro vývoj pro OS Windows. Pokud používáte operační systém Linux (nebo Unix) bude zřejmě nejlepší volbou používat GNU C/C++ překladač. Jedná se většinou o integrální součást každé linuxové distribuce, takže ho není třeba nějak instalovat. Používá se z příkazové řádky, ale existuje mnoho grafických nástaveb, které práci s ním udělají příjemnější. Například součástí populární linuxové distribuce Fedora je i vývojové prostředí KDevelop C/C++.

V této knize budu používat Microsoft Visual C++ 2008 Express Edition, protože je to prostředí, které mám nainstalované a často ho používám. Nicméně většina IDE pracuje na stejném principu a náš kód bude kompilovatelný a spustitelný bez ohledu na to, které IDE používáte, pokud nebudete používat knihovní soubory specifické pro dané IDE. Standardní knihovní soubory, které budeme používat, jako např. iostream a string, jsou stejné pro všechna IDE pro C++.

Instalace Microsoft Visual C++ 2008 Express Edition

Předtím než se pustíme do vytváření prvního projektu, je nutné mít nainstalované nějaké IDE. Popíšu zde, kde stáhnout a jak nainstalovat Microsoft Visual C++ 2008 Express Edition, protože se

na něj budu odkazovat v příkladech této knihy. Nic vám ale nebrání nainstalovat si vaše oblíbené IDE a pracovat na příkladech v něm, vše bude fungovat.

Nejprve si ukážeme odkud stáhnout naše IDE. Nejlépe přímo ze stránek Microsoftu:

http://www.microsoft.com/Express/

Klepněte na odkaz download a na stránce s produkty ke stažení zvolte Microsoft Visual C++2008 Express Edition (anglická jazyková verze). Na váš počítač se stáhne malý instalační program. Po jeho spuštění si instalátor zjistí softwarovou konfiguraci vašeho PC a stáhne si všechny potřebné komponenty z Internetu. Je zde také volba pro stažení ISO obrazu celé množiny produktů Microsoft Visual Studio, jejž lze následně vypálit na DVD.

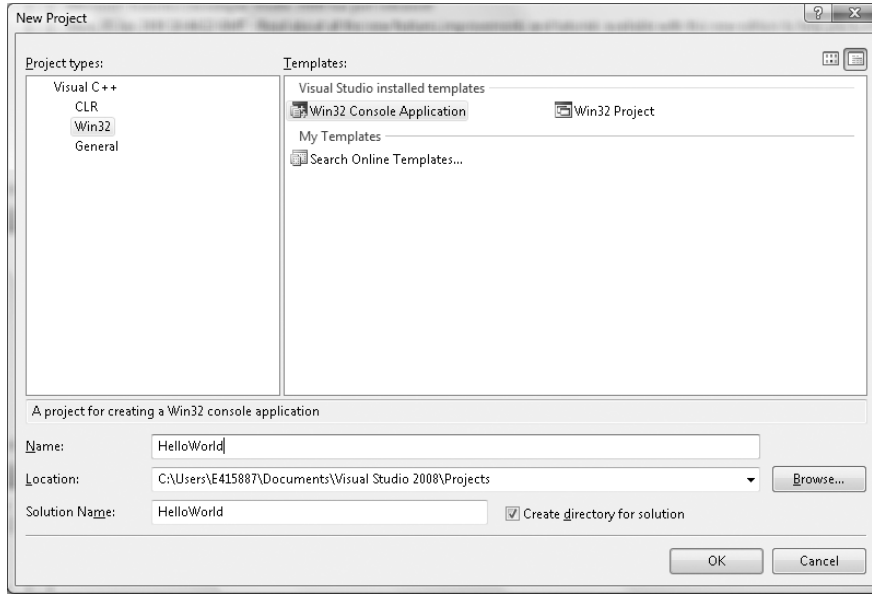
Zvolte ještě možnost nainstalovat si i dokumentaci k Microsoft Visual Studiu s názvem MSDN Express Library (Microsoft Developer Netowork). Může se hodit pro vaše další pokusy s C++.

Po úspěšné instalaci by se ve vaší nabídce Start měla objevit složka „Microsoft Visual C++ 2008 Express Edition“ a v ní stejnojmenná ikona pro spuštění IDE.

Vytváříme projekt „Hello World!“

Pokud již na vašem počítači sídlí Microsoft Visual C++ 2008, pak jste připravení pro start vašeho prvního projektu, jehož cílem je vytvořit a spustit aplikaci „Hello World!“.

1. Spusťte Microsoft Visual C++ 2008.
2. V menu vyberte příkaz File → New → Project. Tento příkaz aktivuje dialog pro vytvoření nového projektu (New Project dialog), viz obrázek 1.2 (hodnoty položek „Name“, „Solution Name“ a „Location“ budou nastaveny v krocích 5 a 6).
3. V levém panelu dialogu pro vytvoření projektu vyberte položku Win32, jak je ukázáno na obrázku 1.2.



Obrázek 1.2: Vytváření nového projektu

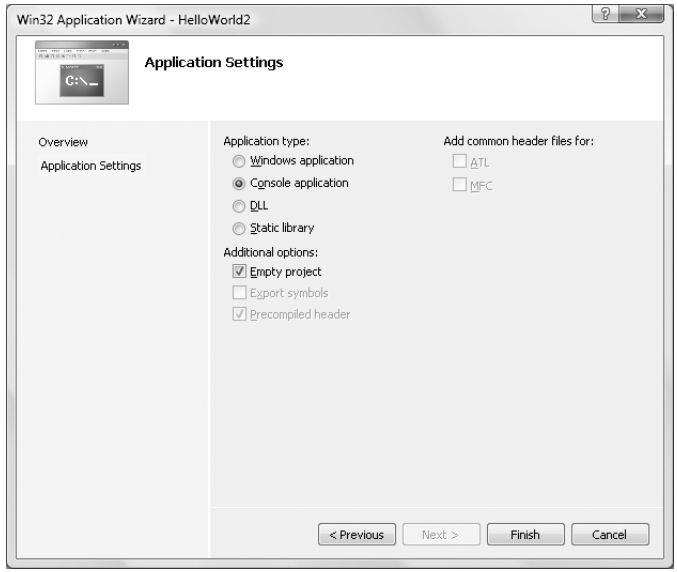
4. Vyberte „Win32 Console Application“ napravo v kontextovém panelu dialogu pro vytvoření nového projektu. Slovo konzola (console) značí, že vytváříme aplikaci, která bude běžet k okně konzoly. Označení Win32 značí, že aplikace bude určena pro 32bitový operační systém Windows, jako např. Windows 2000, XP, Vista atd.
5. V poli „Location“ za použití tlačítka „Browse“ vyberte existující adresář, ve kterém bude vytvořen projekt.
6. V poli „Name“ zadejte jméno, které jste vybrali pro projekt. Jméno projektu se automaticky vyplní i jako jméno řešení (Solution) a vytvoří se adresář s tímto jménem, do něhož se uloží všechny projektové soubory.
7. Klepněte na tlačítko OK. Zobrazí se průvodce pro vytvoření Win32 konzolové aplikace (Win32 Application Wizard), viz obrázek 1.3.

Obrázek 1.3: Průvodce pro vytvoření Win32 konzolové aplikace

8. Klepněte na tlačítko „Next“ vpravo dole. Průvodce se přepne na nastavení parametrů aplikace, jak je vidět na obrázku 1.4.
9. Pokud to bude nutné, zvolte konzolovou aplikaci (Console application) jako aplikační typ (Application Type). Zaškrtněte checkbox v „Empty project“ (prázdný projekt) v sekci Additional Options (Dodatečná nastavení). Po zaškrtnutí volby „Empty project“ se zablokují ostatní volby ze sekce „Additional options“.

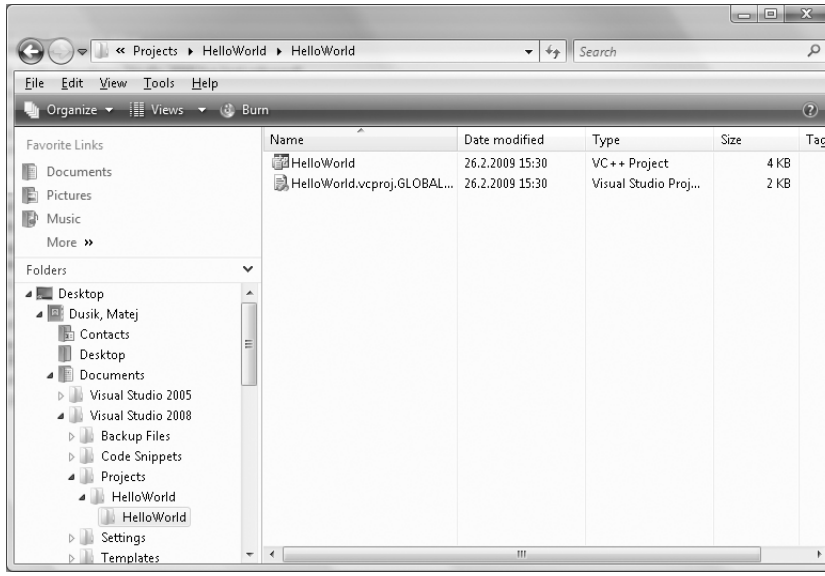
Upozornění: Věnujte tomuto kroku velkou pozornost, obzvláště nezapomeňte zaškrtnout volbu „Empty project“. Nesprávná konfigurace aplikačních nastavení je obvyklou chybou a může zapříčinit to, že budete muset začít znovu.

10. Klepněte na tlačítko „Finish“. Obrázek 1.5 zobrazuje nově vytvořené adresáře. Jména adresářů odpovídají zadaným jménům pro řešení (Solution) a projekt, tedy HelloWorld. Tato jména byla zadána v krocích 5 a 6.



Obrázek 1.4: Průvodce vytvořením Win32 konzolové aplikace – aplikační nastavení

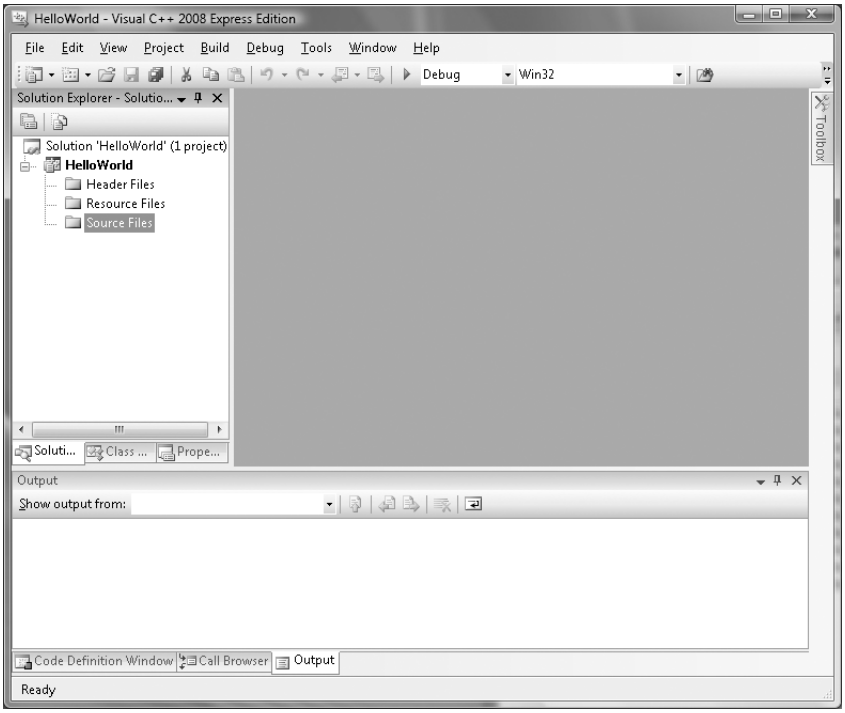
Nyní jste vytvořili projekt pro vaši aplikaci. Projekt je skořápkou pro vaši aplikaci, obsahuje soubory pro podporu vytváření a běhu aplikace. Nicméně teď je projekt prázdný, neobsahuje žádný kód, takže nedělá nic. Dalším krokem je začít se psaním kódu.



Obrázek 1.5: Průzkumník ukazující nově vytvořené projektové soubory a adresáře

Psaní zdrojového kódu

Visual C++ obsahuje panel podobný tomu z Průzkumníka. Jmenuje se „Solution Explorer“ (Průzkumník řešení) a je zobrazen na obrázku 1.6. Pokud není „Solution Explorer“ vidět, lze ho aktivovat v menu View → Solution Explorer.



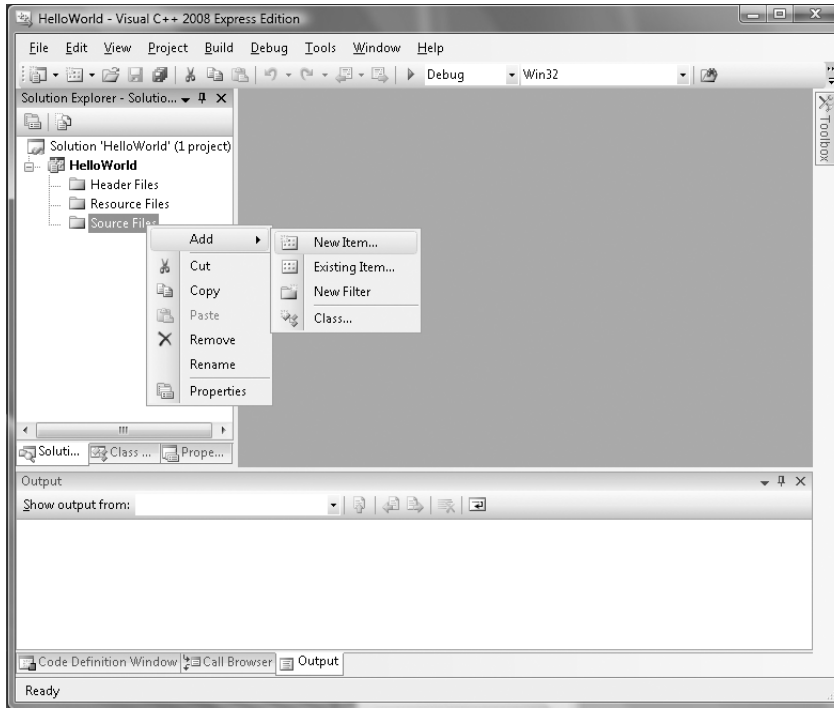
Obrázek 1.6: Průzkumník ukazující nově vytvořené projektové soubory a adresáře

Průzkumník řešení má složku pro zdrojové i hlavičkové soubory. Soubor, který se chystáme vytvořit a do kterého chceme umístit kód aplikace „Hello World!“, je soubor zdrojový. Zdrojové soubory mají příponu .cpp pro programu napsané v C++. Oproti tomu soubor iostream, který je vložený direktivou include, je hlavičkový soubor. Hlavičkové soubory mají příponu .h (h jako header – hlavička).

My použijeme průzkumníka řešení (Solution explorer) k přidání nového zdrojového souboru do projektu, poté začneme s psaním kódu do tohoto souboru.

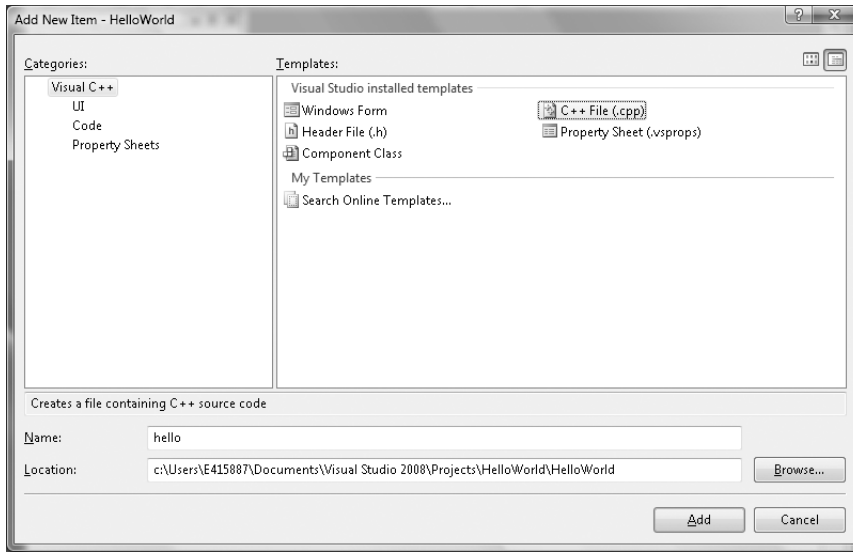
Následující kroky slouží k přidání nového zdrojového souboru do projektu:

1. Klepněte pravým tlačítkem myši na složku „Source Files“. Zobrazí se kontextové menu zobrazené na obrázku 1.7.
2. Z menu vyberte položku Add → Add New, zobrazí se dialog pro přidání nové položky, viz obrázek 1.8.



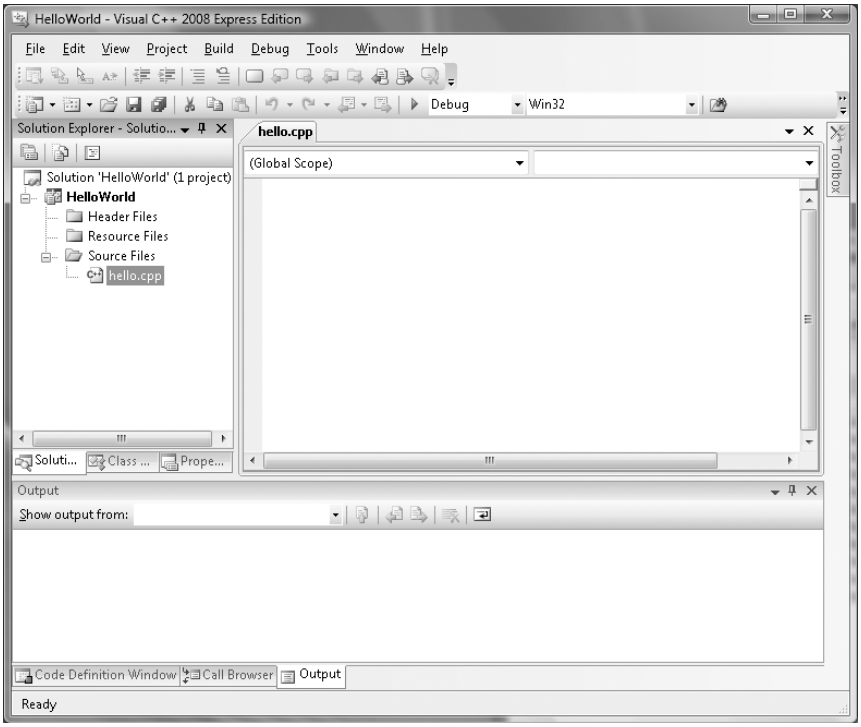
Obrázek 1.7: Kontextové menu složky „Source Files“

Poznámka: Pokud chcete vložit již existující zdrojový soubor, můžete to udělat pomocí položky Add → Add Existing Item z kontextového menu složky „Source Files“.



Obrázek 1.8: Přidávání nového souboru do projektu

3. Obvykle není nutné měnit umístění nově vytvářeného souboru v poli „Location“. Přednastavená složka je složkou, kde jsou umístěny projektové soubory projektu. Zadejte název nového zdrojového souboru do pole „Name“. Není třeba zadávat příponu .cpp; bude přidána automaticky, protože přece vytváříme zdrojový soubor. Zadáme-li jako název „hello“, viz obrázek 1.8, pak se nový soubor bude jmenovat hello.cpp.
4. Po zadání názvu souboru klepněte na tlačítko „Open“. Na obrázku 1.9 si můžete prohlédnout nově vytvořený soubor v průzkumníku řešení (Solution Explorer).

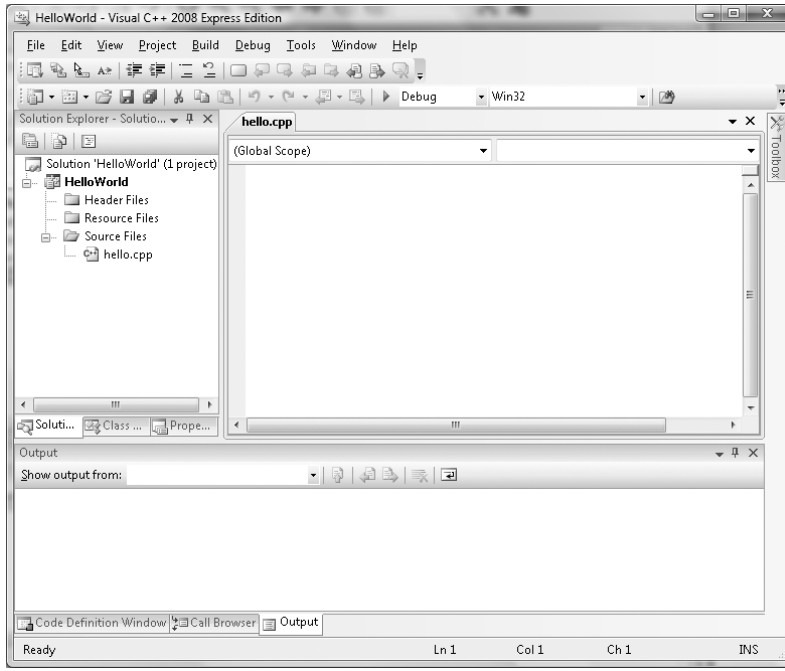


Obrázek 1.9: Průzkumník řešení (Solution Explorer) zobrazující nový .cpp soubor

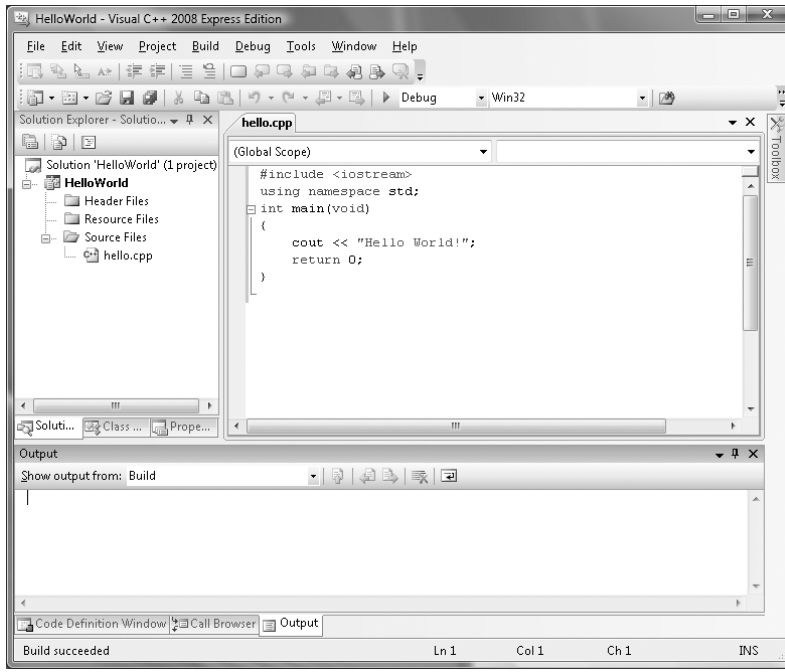
Psaní kódu není nic složitého. Stačí poklepání na soubor hello.cpp v průzkumníku řešení (Solution Explorer) a otevře se editor zdrojového souboru, který je zatím ještě prázdný (viz obrázek 1.10). Nyní jednoduše napište kód programu Hello World! Po dokončení by měl program vypadat tak jako na obrázku 1.11.

Upozornění: K psaní kódu můžete klidně použít Notepad nebo kterýkoliv jiný textový editor. Nicméně nepoužívejte Microsoft Word či jakýkoliv jiný textový procesor. Textové procesory umožnují pokročilé formátování textu, kterého je dosaženo použitím spousty skrytých formátovacích znaků, kterým překladač nerozumí a vyhodnotí je jako syntaktické chyby.

Uložte výsledky své práce klepnutím na tlačítko „Save“ na nástrojové liště. Nyní jsme připraveni ke kompilaci programu.



Obrázek 1.10: Zdrojový soubor před psaním kódu



Obrázek 1.11: Zdrojový soubor po zadání kódu

Sestavení (build) projektu

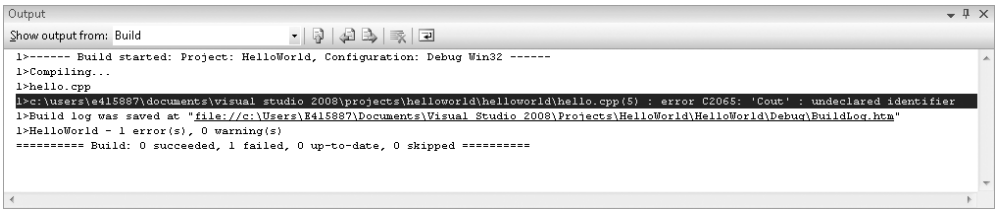
Kompilaci kódu provedeme příkazy z menu „Build“. Ke zkompilování vedou všechny z následujících voleb:

- Build → Solution
- Rebuild → Solution
- Build → HelloWorld
- Rebuild → HelloWorld

HelloWorld je název našeho projektu. Řešení (neboli Solution) může obsahovat více než jeden projekt. Naše řešení ovšem obsahuje pouze jeden projekt, takže není žádný praktický rozdíl mezi projektem a řešením.

Sestavením (build) se myslí kompilace změn od poslední kompilace (pokud nějaká proběhla). Opětovné sestavení (rebuild) znamená kompilaci veškerých zdrojů od začátku, proto je sestavení většinou rychlejší než opětovné sestavení. Opětovné sestavení je dobré použít, pokud došlo v kódu od poslední kompilace k velkým změnám. Z praktického hlediska je mezi nimi mizivý rozdíl.

Před kompilací ještě uděláme jednu změnu ve kódu. Zaneseme do něj chybu, a to tak, že změníme písmeno malé c na velké C ve slovu cout (cout → Cout). Pak vybereme jednu ze čtyř kompilačních možností. V okně „Output“ – výstup by se měla objevit chybová hláška překladače, viz obrázek 1.12. Chybová hláška „c2060“ nám říká, že identifikátor Cout není deklarovaný.



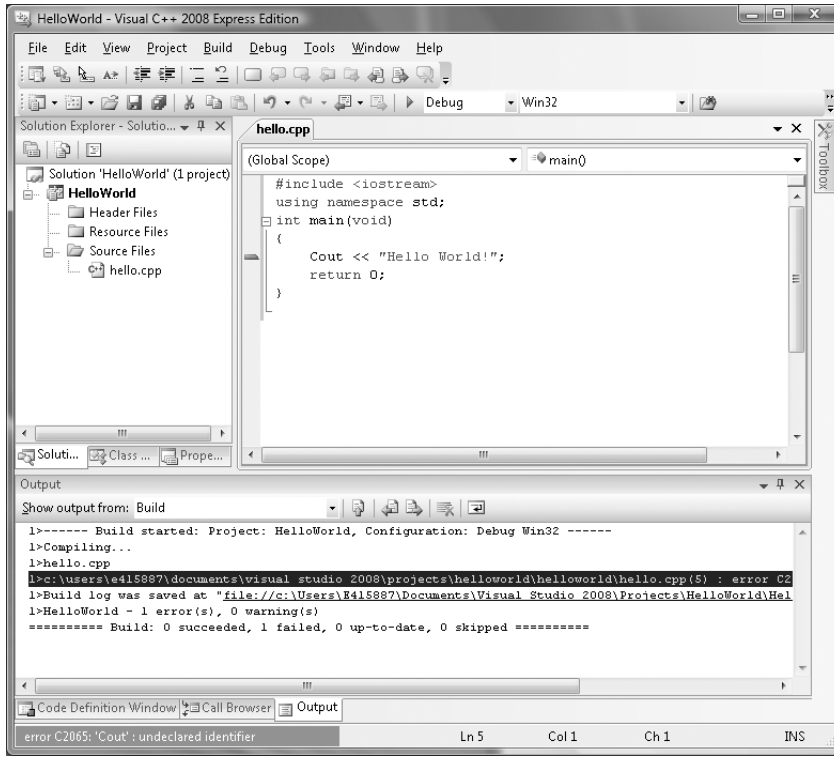
Obrázek 1.12: Okno „Output“ s chybovou hláškou kompilátoru

Tip: Pokud v okně „Output“ není chyba zobrazena celá, můžete si okno roztáhnout či uvolnit a rozbalit si ho na celou obrazovku.

Jak bylo vysvětleno v předchozích kapitolkách, kompilátor porozumí vašemu kódu a přeloží ho do strojového jazyka, pouze pokud váš kód odpovídá syntaktickým pravidlům daného programovacího jazyka. Jak již bylo vysvětleno, C++ má řadu pravidel pro zápis slov a pro gramatiku příkazů a konstrukcí. Pokud tato pravidla porušíte, jedná se o syntaktickou chybu. Následkem toho kompilátor nemůže přeložit váš kód do instrukcí strojového jazyka a odmění vás chybovou hláškou o syntaktické chybě.

Kód v C++ rozlišuje malá a velká písmena. To znamená, že stejné slovo napsané velkými a malými písmeny není považováno překladačem za jedno a totéž. Správný zápis je cout, Cout je špatně. Protože C++ neví, co Cout znamená, překladač vypíše chybovou hlášku, že se jedná o nedefinovaný identifikátor.

V našem projektu je pár řádek kódu, ale jakmile se kód rozroste, je nesnadné nalézt místo, kde se chyba nachází. Pokud dvakrát klepnete na chybovou hlášku v okně „Output“, kurzor v editoru kódu se přesune na místo chyby a řádek, na kterém se nachází, je označen ikonou (viz obrázek 1.13).



Obrázek 1.13: Chyba označena v editoru kódu

Nyní opravte Cout za cout a znovu zkompilujte váš kód. Tentokrát by měla být kompilace úspěšná. Pokud se nyní pomocí průzkumníku z Windows podíváte do adresáře projektu HelloWorld, naleznete tam soubory hello.obj a hello.exe. Jedná se o objektový a spustitelný soubor, o kterém jsme se bavili v kapitole „Překlad zdrojového kódu“. Jak bylo řečeno, při sestavení byl použit preprocesor, kompilátor a linker.

Spuštění kódu

Finálním krokem je spuštění kódu. K tomuto úkolu slouží menu „Debug“ – ladit. Můžeme zvolit položku Debug → Start of Debug nebo Debug → Start Without Debugging. Rozdíl mezi nimi je použití debuggeru (funkce či program IDE pro usnadnění hledání chyb v programu), tato problematika bude probrána v následujících kapitolách. Protože tentokrát debugger nepoužijeme, vyberte Debug → Start Without Debugging, jenž je trochu rychlejší. Výsledkem je konzolové okno zobrazující nápis „Hello World!“ (ukázáno na obrázku 1.1).

Test

1. Co je počítačový program?
2. Vyjmenujte několik věcí, které počítač při zpracování informací zvládá lépe než člověk.

3. Co je programovací jazyk?
4. Proč se vyplatí naučit se C++?
5. Co je funkce?
6. Kolik funkcí `main` najdete v běžném programu napsaném v C++?
7. Co je to standardní knihovna?
8. K čemu slouží direktiva `#include`?
9. Co dělá preprocesor?
10. Co dělá překladač?
11. Co dělá linker?

Paměť a datové typy

2

Když jsem dopsal svou první knihu, každý den jsem netrpělivě kontroloval poštovní schránku, jestli už začaly chodit prosby o autogram. Výsledek se záhy dostavil, schránka přetékala. Člověk by měl být opatrnější v tom, co si přeje – mezi hlavní zájemce o můj podpis patřila hypoteční banka, účet za kreditní karty, pojištění, telefon, elektřinu a další fanoušci, které si jistě dokážete představit.

Všechny firmy, které s takovou láskou posílají účty, by své zákazníky jistě nezvládly evidovat pomocí tužky a papíru. Místo nich používají počítačové programy, těží z jejich schopnosti uložit obrovské množství informací a bleskem si je vybavit.

My lidé si informace ukládáme do paměti. Totéž dělají počítače, ale jejich paměť se od té naší výrazně liší. Fungování počítačové paměti se bude věnovat tato kapitola.

Informace neboli data mají různou podobu. Některá data jsou číselná, například můj účet za plyn. Jiná data jsou textová, například jméno na mém účtu za plyn. Typ dat, například číslo nebo text nebo cokoliv dalšího, se celkem logicky označuje jako *datový typ*. Datový typ informací se promítá do způsobu jejich uložení i do množství paměti, kterou zaberou. Různými datovými typy se bude zabývat následující text.

Paměť

Počítačové programy se skládají z příkazů a dat. Jak bylo řečeno v předchozí kapitole, příkazy jsou napsané v nějakém programovacím jazyce (například C++) a pomocí překladače se převádí do strojového jazyka. Krok za krokem počítači říkají, co má dělat. Naproti tomu data jsou to, s čím program pracuje. Pokud například uživatel vašeho programu potřebuje seznam všech studentů se studijním průměrem pod tři, jeho data by mohl tvořit seznam studentů a jejich studijních průměrů. Program na základě příkazů seznam projde a vypíše všechny studenty s průměrem pod tři.

Příkazy programu i jeho data musí být v paměti počítače, jinak by program nefungoval. Následující text rozebere jednotlivé typy počítačové paměti a ukáže vám, kam se v paměti ukládají příkazy a kam data.

Druhy paměti

Váš počítač obsahuje tři základní paměťová centra:

- Procesor neboli CPU
- Operační paměť neboli RAM
- Trvalou paměť

Vyrovnávací paměť procesoru

Procesor je mozek počítače. Možná jste nad ním přemýšleli, když jste naposledy kupovali počítač, protože rychlost procesoru hraje při koupi počítače výraznou roli. Čím rychlejší procesor, tím rychlejší počítač.

Poznámka: Rychlost procesorů se udává v hertzech. Hertz, pojmenovaný po Heinrichovi Hertzovi, který poprvé dokázal existenci elektromagnetických vln, představuje jednu změnu stavu za sekundu. Rychlost procesorů se většinou udává v megahertzech (MHz), tedy milionech hodinových cyklů za sekundu, nebo gigahertzech (GHz), tedy miliardách hodinových cyklů za sekundu. Procesor s rychlostí 800 MHz provede za jednu vteřinu 800 milionů hodinových cyklů. Každá instrukce strojového jazyka zabere určitý počet cyklů, takže rychlost procesoru určuje, kolik příkazů zvládne procesor provést za jednu sekundu.

Kromě toho, že procesor řídí provoz počítače, má přímo u sebe i malé množství paměti. Této paměti se říká *vyrovnávací paměť* neboli *cache*. Používá se pro uložení nejčastěji používaných příkazů a dat. Jelikož je tak blízko procesoru, přístup do ní je velice rychlý. Na druhou stranu se toho do ní moc nevejde, stačí jen na nejčastěji používané příkazy a data. Mimo *cache* se v procesoru ještě nachází tzv. paměťové registry. Ty slouží k uložení dat, se kterými procesor přímo provádí operace. Zbytek příkazů a dat se musí ukládat jinam.

Operační paměť

„Jinam“ znamená ve většině případů do operační paměti neboli RAM. I nad tou jste nejspíš při posledním nákupu počítače uvažovali, protože čím víc operační paměti počítač má, tím je rychlejší a tím víc programů na něm může najednou běžet.

Procesor se do operační paměti dostane skoro stejně rychle jako do cache, a navíc má operační paměť ve srovnání s vyrovnávací pamětí nesrovnatelně vyšší kapacitu. Obě paměti jsou ale dočasné, jejich obsah se ztratí v okamžiku vypnutí počítače. Na tento nepříjemný fakt už jste nejspíš narazili, když se vám počítač vypnul kvůli výpadku elektřiny nebo se musel restartovat a vy jste přišli o neuložená data.

My bychom byli rádi, kdyby data v pořádku přečkala vypnutí programu i vypnutí počítače. Na počítači navíc určitě máte další programy (například e-mailového klienta, webový prohlížeč nebo textový procesor), které momentálně neběží, ale které na počítači potřebujete. Podobně jistě máte řadu souborů, například ročníkových prací, dopisů, daňových tabulek, e-mailů a podobně, které zrovna nepoužíváte, ale které budete potřebovat časem. Proto potřebujeme ještě jeden typ paměti, který by byl na rozdíl od cache a RAM trvalý, tedy přežil vypnutí počítače.

Trvalá paměť

Trvalou pamětí je nejčastěji pevný disk, CD, DVD nebo flash disk. Všechny trvalé paměti mají společné to, že na nich příkazy i data zůstanou i po vypnutí počítače. Počítač můžete mít vypnutý třeba měsíce, ale když ho zase zapnete, všechny soubory jsou stále na svém místě.

Kromě toho, že je trvalá paměť nezávislá na zapnutém počítači, má i mnohem větší kapacitu než paměť operační – řekněme stokrát až tisíckrát. Jelikož je trvalá a má vysokou kapacitu, ukládají se do ní v počítači programy i data. Když například na počítač nainstalujete program Microsoft Word, jeho soubory se uloží na pevný disk. Stejně tak se na pevný disk ukládají všechny dokumenty, které ve Wordu napíšete.

Trvalá paměť sice vydrží a má velkou kapacitu, ale počítač neumí provádět příkazy na ní uložené. Pokud se mají nějaké příkazy provést, musí se načíst do operační paměti. S daty je to podobné – pokud má počítač změnit nějaká data, musí je načíst z trvalé paměti do operační.

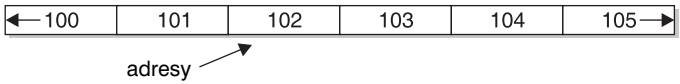
Poznámka: Zvláštním případem trvalé paměti je ***paměť virtuální*** neboli ***swap space***. To je trvalá paměť, která slouží jako dočasné odkladiště pro data z operační paměti. Její podrobnější popis je bohužel nad rámec této kapitoly.

Obecně vzato ukládají programy většinu svých příkazů a dat v operační paměti, takže ta nás bude zajímat nejvíc. Velká část naší diskuze bude platit i pro paměť trvalou. Vyrovnávací paměť procesoru a registry jsou úplně jiné téma a většinou přichází na pořad pouze u programovacích jazyků, které jsou blíže strojovému jazyku (například u assembleru).

Adresy

Když se vás někdo zeptá, kde bydlíte, odpovíte třeba že v Mockingbird Lane 1313. To je vaše adresa, podle které vás kdokoliv může najít. Adresy většinou dodržují určitý logický pořádek. Čísla v rámci jednoho bloku mohou jít od 1300 po 1399, v dalším bloku budou čísla 1400–1499 a tak dále.

Podobně jsou adresovaná i místa v paměti počítače. Vypadají úplně jinak než naše čísla ulic, na která jsme zvyklí, protože většinou jde o šestnáctková čísla – například 0x8fc1. Ať už je ale samotné číslo zapsané jak chce, adresy dodržují tentýž logický pořádek a jejich čísla jdou hezky za sebou (viz obrázek 2.1).



Obrázek 2.1: Posloupnost paměťových adres

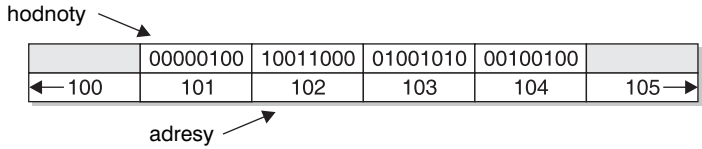
Poznámka: Co se šestnáctkových čísel týká – my jsme zvyklí na čísla v desítkové soustavě, ve které se používají číslice od nuly do devíti. Naproti tomu adresy v paměti se většinou udávají v soustavě šestnáctkové. Pro zápis číslic v šestnáctkové soustavě bychom potřebovali šestnáct různých číslic, ale jelikož máme číslic jen deset (0–9), místo zbývajících pěti „číslic" se používají písmena od A po F. Například desítkové číslo 16 se v šestnáctkové soustavě zapíše jako 10.

Šestnáctková soustava má výhodu v tom, že je úspornější. Paměťové adresy bývají poměrně veliká čísla, která jsou v šestnáctkovém zápisu kratší než v desítkovém – například 1 000 000 desítkově je f4240 šestnáctkově. Převody mezi desítkovým a šestnáctkovým zápisem si vysvětlíme za chvilku.

Bity a bajty

Zatímco na číslech ulic žijí lidé, na paměťových adresách jsou uložené *bajty*. V předchozí kapitole jsme mluvili o tom, že první počítače byly v podstatě hromada přepínačů, které mohly být buď zapnuté (1), nebo vypnuté (0). Máme tu tedy určitou nejmenší možnou užitečnou jednotku informace, která může mít hodnotu 1 nebo 0. Této jednotce se v počítačové terminologii říká *bit*. Počítač přemýšlí v bitech, ale zpracování dat po jednotlivých bitech by nebylo praktické. Proto se bity sdružují do skupin po osmi, a skupina osmi bitů se označuje jako *bajt*.

Na každé adrese v paměti je uložený právě jeden bajt informací, tedy osm jedniček nebo nul. A stejně jako můžeme podle čísla ulice najít člověka, podle paměťové adresy najdeme bajt informací. Posloupnost adres a hodnot na nich uložených ilustruje obrázek 2.2.



Obrázek 2.2: Posloupnost bajtů a jejich adresy v paměti

Dvojková číselná soustava

Informace jsou v paměti uložené jako posloupnost jedniček a nul, které toho většině z nás moc neřeknou. Zato počítač se ve šrůtcích jedniček a nul vyzná výborně.

Například já jsem se z pohledu svého počítače narodil v roce 11110100000. Ještě než začnete protestovat, klidně vám prozradím, že jsem se narodil v roce 1952. Jak jsem se mohl narodil v roce 11110100000 a zároveň v roce 1952?

My jsme zvyklí s čísly pracovat v desítkové soustavě, ve které je každé číslo zapsané posloupností číslic od nuly do devítky. Když jsem mluvil o roce 1952, uvedl jsem letopočet v desítkové soustavě.

Každá posloupnost jedniček a nul je také číslo, i když možná trochu jiné, než na jaké jste zvyklí. Když jsem mluvil o roce 11110100000, uvedl jsem letopočet v dvojkové neboli binární soustavě. Ve dvojkové soustavě se každé číslo zapisuje posloupností nul a jedniček.

Zatímco lidé většinou uvažují v desítkové soustavě, počítač „přemýšlí" v soustavě dvojkové. Proto se při programování setkáte s čísly v obou těchto soustavách.

Převody mezi číselnými soustavami

Bez schopnosti převádět mezi dvojkovým a desítkovým zápisem čísel se při programování docela dobře obejdete. Převod ale není těžký a občas vám pomůže pochopit dění za oponou. Pokud máte zájem, čtěte dál.

Převod z dvojkové do desítkové soustavy je prostý. Směrem zprava doleva se první dvojková číslice vynásobí číslem 2^0 (tedy jedničkou), druhá číslice se vynásobí číslem 2^1 (dvojkou), třetí číslice se vynásobí číslem 2^2 (čtyřkou) a tak dále. Výsledek všech násobení se sečte a dostanete desítkový zápis původního dvojkového čísla. Příklad výpočtu pro čísla od nuly do pěti najdete v tabulce 2.1.

Tabulka 2.1: Převod čísel od nuly do pěti z dvojkové do desítkové soustavy

Dvojkový zápis	Výpočet	Desítkový zápis
0	$0\times2^0 = 0\times1$	0
1	$1\times2^0 = 1\times1$	1
10	$1\times2^1+0\times2^0 = 2+0$	2
11	$1\times2^1+1\times2^0 = 2+1$	3
100	$1\times2^2+0\times2^1+0\times2^0 = 4+0+0$	4
101	$1\times2^2+0\times2^1+1\times2^0 = 4+0+1$	5

A co převod opačným směrem, z desítkové soustavy do dvojkové? Důležité je si uvědomit, co přesně zápis čísla v dané soustavě znamená. Například číslo 145 v desítkové soustavě znamená $1\times100 + 4\times10 + 5\times1$. Tento rozklad běžně děláme v řeči: „Jedno sto čtyřicet pět." Není na něm

nic objevného – zajímavější bude, když si ho přepíšeme do tvaru $1\times10^2 + 4\times10^1 + 5\times10^0$. Tady už je vidět, že každá číslice ze zápisu čísla v desítkové soustavě říká, kolikrát číslo obsahuje určitou mocninu desítky.

Tento postřeh se dá zobecnit na libovolnou číselnou soustavu: Každá číslice ze zápisu čísla o základu *n* říká, kolikrát číslo obsahuje určitou mocninu *n*. Praktickým zvykem je psát nejnižší mocniny vpravo, takže například číslo 110 ve dvojkové soustavě obsahuje nulakrát dvojku na nultou, jedenkrát dvojku na první a jedenkrát dvojku na druhou. Jde tedy o šestku, která se v desítkové soustavě zapisuje jako šestkrát deset na nultou: 6. Stejně tak bychom ji mohli převést do šestnáctkové soustavy, kde dopadne jako šestkrát šestnáct na nultou: 6.

Když tedy chcete převést číslo z desítkového zápisu na dvojkový, stačí se podívat, kolikrát jsou v něm zastoupené jednotlivé mocniny dvojky. K tomu nám dobře poslouží zbytek po dělení, zkusme to například pro číslo 6:

- 6/2 = 3, zbytek 0
- 3/2 = 1, zbytek 1
- 1/2 = 0, zbytek 1

Všimněte si, že zbytky tvoří přesně dvojkový zápis čísla šest, 110. Když tedy chcete převést číslo z desítkové soustavy do dvojkové, dělte jej dvojkou, zbytky po dělení si zapisujte odzadu a výsledek dělení dělte dál, dokud není nulový.

Stejný postup se dá použít pro převod mezi desítkovou a šestnáctkovou soustavou. Když převádíte ze šestnáctkové soustavy do desítkové, násobíte každou číslici odpovídající mocninou šestnáctky (přičemž šestnáctková „číslice“ A odpovídá desítce, B jedenáctce a tak dále). Například číslo šestnáctkové číslo 5c je $5\times16^1 + 12\times16^0 = 80+12 = 92$.

A naopak když převádíte z šestnáctkové do desítkové soustavy, vydělíte 92 šestnácti, dostanete pětku a zbytek 12. Pětku opět vydělíte šestnácti, dostanete nulu a zbytek 5. Teď už byl výsledek dělení nulový, takže hledané šestnáctkové číslice jsou zbytky po dělení: pět a dvanáct. Číslice dvanáct se šestnáctkově zapisuje jako C, takže hledaný šestnáctkový zápis je 5c.

Datové typy

Jedničky a nuly uložené v paměti mohou reprezentovat text (například mé jméno, Jeff Kent), celá čísla (například mou výšku v palcích, 72) nebo desetinná čísla (například můj průměrný studijní prospěch, na který už si nevzpomenu). Stejně dobře ale mohou reprezentovat logickou hodnotu pravda/nepravda (true/false), například jestli jsem občan Spojených států.

Data mají zkrátka celou řadu podob, které se dají obecně rozdělit na textové a číselné. Čísla se dají rozdělit ještě na celá (například 6, 0 nebo –7) a desetinná (například ⅔, 7,3 nebo –6,1). Typ informací se kromě jiného projeví i na velikosti paměti, kterou informace zaberou.

Celočíselné datové typy

S celými čísly pracujeme neustále. Stačí se zamyslet nad tím, kolik aut stojí na parkovišti, na kolik přednášek chodíte nebo kolik máte sourozenců. Každá z odpovědí je nějaké číslo, u kterého není potřeba uvažovat žádná desetinná místa. (Pokud tedy nemáte 3,71 bratrů.)

Některá celá čísla bývají menší, například který nešťastník by chodil na 754 361 přednášek. Jiná celá čísla musí být větší – například studenti astronomie budou vědět, že Měsíc je od Země

asi 240 tisíc mil daleko. A některá celá čísla bývají hodně, hodně veliká; kupříkladu vzdálenost ze Země na Pluto je v nejlepším případě 2,7 miliardy mil.

Často se stává, že číslo nemůže být záporné. I kdybyste na zkoušce úplně vyhořeli, záporný počet bodů nedostanete. Jiná celá čísla bývají záporná běžně, například teplota na severním pólu.

S ohledem na tyto různorodé požadavky existuje v počítači několik různých datových typů pro celá čísla, viz tabulku 2.2. Uvedené velikosti a rozsahy hodnot jsou jen orientační, liší se podle použitého překladače a operačního systému. Hodnoty platné pro váš překladač a systém se naučíte zjistit později v této kapitole, až se dostaneme k operátoru `sizeof`.

Tabulka 2.2: Rozsah a velikost běžných celočíselných datových typů

Datový typ	Velikost v bajtech	Rozsah
`short`	2	–32 768 až 32 767
`unsigned short`	2	0–65 535
`int`	4	–2 147 483 648 až 2 147 483 647
`unsigned int`	4	0–4 294 967 295
`long`	4	–2 147 483 648 až 2 147 483 647
`unsigned long`	4	0–4 294 967 295

Poznámka: Asi vás napadne, k čemu je datový typ `long`, když je podle tabulky 2.2 úplně stejný jako `int`. Jak už bylo řečeno před tabulkou, skutečná velikost a tedy i rozsah datových typů závisí na operačním systému a překladači. U některých kombinací operačních systémů a překladačů může například `short` vycházet na jeden bajt, `int` na dva a `long` na čtyři.

Začínající programátoři propadají při pohledu na tabulku 2.2 skepsi, jelikož se bojí, že se čísla nezvládnou naučit nazpaměť. Dobrá zpráva je, že vás do toho nikdo nenutí. Dá rozum, že nějaké to memorování je potřeba téměř u všeho. Ale protože programování vyžaduje obrovské množství informací, programátoři v podstatě neustále hledí do nápovědy nebo nějaké referenční příručky. Věřte mi, nejsem výjimka.

Mnohem důležitější než mechanické učení je schopnost pochopit, proč a jak program funguje. Proto se v následujícím textu podrobně podíváme, jak fungují datové typy. Úplně bez výpočtů se samozřejmě neobejdeme, ale výpočty nebudou nijak složité, a když je pochopíte, ulehčíte si programování v dalších kapitolách.

Se znaménkem, nebo bez?

Tabulka 2.2 uvádí tři datové typy: `short`, `int` a `long`. Každý z těchto typů před sebou ještě může mít slovo `unsigned`, například `unsigned short`. Výraz *unsigned* znamená „bez znaménka“. Datové typy bez znaménka mají pouze nezáporné hodnoty, tedy nulu a kladná čísla. Datové typy se znaménkem mohou mít hodnoty záporné, nulové i kladné. Když neuvedete modifikátor `unsigned`, datový typ se automaticky bere jako `signed`. (Například `short` a `signed short` je tím pádem totéž.)

Datové typy bez znaménka nemohou obsahovat záporné hodnoty, a tak je například u datového typu `unsigned short` v tabulce 2.2 jako nejmenší možná hodnota uvedena nula. Naproti tomu nejmenší možná hodnota typu `short` je –32 768, protože datový typ se znaménkem může obsahovat i záporná čísla.

Velikost datového typu

Každý z celočíselných datových typů z tabulky 2.2 má určitou velikost, ostatně stejně jako všechny datové typy C++. Velikost datových typů se ale na rozdíl od lidí neudává v palcích nebo librách (které jsou pro mě citlivé téma), nýbrž v bajtech.

Všechny datové typy zabírají alespoň jeden bajt, protože s menšími datovými typy by se pracovalo velice neefektivně. Většina datových typů zabírá bajtů hned několik, viz všechny typy z tabulky 2.2. Velikost datového typu je vždy celé číslo – neexistuje datový typ o velikosti dejme tomu 3,5 bajtu, protože se čtyřmi bity neboli polovinou bajtu by se špatně pracovalo.

Jelikož počítač pracuje ve dvojkové soustavě, velikost datových typů bývá mocnina dvojky. Mezi běžné velikosti datových typů tak patří jeden bajt (2^0), dva bajty (2^1), čtyři bajty (2^2) a osm bajtů (2^3). Z velikosti datového typu plynou dvě podstatné věci: rozsah datového typu a velikost paměti, která je potřeba pro jeho uložení.

Rozsah datového typu

Rozsahem datového typu se myslí jeho nejvyšší a nejnižší hodnota. Například rozsah typu `unsigned short` je 0–65 535. Rozsah není náhodně vybraný, plyne z velikosti datového typu a způsobu jeho uložení.

Datový typ o velikosti *n* bitů může celkem uložit 2^n různých hodnot. `Short` je dlouhý dva bajty neboli 16 bitů. Z toho plyne, že celkem zvládne uložit 2^{16} = 65 536 různých celých čísel.

Nejvyšší hodnota, kterou typ `unsigned short` zvládne uložit, ovšem není 65 536. Začíná totiž nulou, a proto zvládne uložit nanejvýš číslo 65 535. Nejvyšší možná hodnota bezznaménkového datového typu o velikosti *n* bitů je tedy 2^n–1.

U datových typů se znaménkem je situace o něco složitější. Jelikož se v nich mohou objevit i záporná čísla, musíme někam uložit informaci o tom, že je číslo záporné. Nám lidem se stačí podívat, jestli je před číslem minus. Ale bit v počítači je zkrátka jednička nebo nula, žádné minus pro záporná čísla nenabízí.

Informatika nabízí několik řešení tohoto problému, například samostatný znaménkový bit, jedničkový doplněk nebo dvojkový doplněk, ale my se do složitějšího vysvětlování pouštět nemusíme.

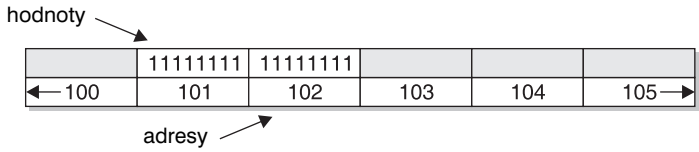
Například typ `signed short` opět uloží nanejvýš 2^0 neboli 65 536 různých kombinací. Jelikož ale musí pamatovat na záporná čísla, rozdělí tyto kombinace rovnoměrně mezi kladná a záporná čísla. To souhlasí s tabulkou 2.2, kde je rozsah typu `signed short` uvedený od –32 768 do 32 767.

K horní a spodní hranici rozsahu datového typu se znaménkem se dá dojít ještě jinak. Jelikož musí jeden bit nutně posloužit pro uložení znaménka, zůstane jen 15 bitů pro uložení čísel. Nejvyšší možná hodnota, která se dá uložit na patnácti bitech, je 2^{15}–1 (nesmíme zapomenout odečíst jedničku, protože začínáme od nuly). Nejnižší možná hodnota je –(2^{15}) neboli –32 768. (A tentokrát žádnou jedničku neodečítáme, protože začínáme od –1, nikoliv od nuly.)

Způsob uložení v paměti

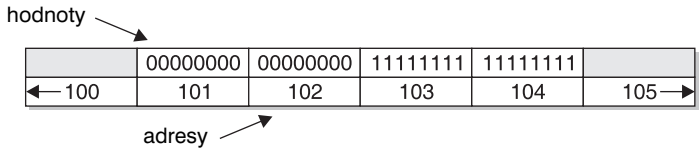
Ve dvojkové soustavě a formátu `unsigned short` má číslo 65 535 v paměti podobu šestnácti jedniček: 1 111 111 111 111 111. Šestnáct jedniček se nám na jednu adresu nevejde – na jednu adresu se vejde jen jeden bajt, osm bitů. Jak tedy hodnotu v paměti uložit?

Odpověď zní, že číslo 65 535 se musí uložit na dvě adresy. Dvě adresy znamenají dva bajty, a ty už nám pro uložení takového čísla stačí. Proto datový typ `short` zabírá dva bajty. Na obrázku je vidět, jak číslo 65 535 ve formátu `unsigned short` vypadá v paměti.



Obrázek 2.3: Číslo 65 535 jako unsigned short v paměti

Datový typ `int` zabírá čtyři bajty. Na obrázku 2.4 je vidět, jak by číslo 65 535 v paměti vypadalo, kdyby bylo uložené jako `int`.



Obrázek 2.4: Číslo 65 535 v paměti, tentokrát jako int

Oprávněně by vás mohlo zajímat, proč zabírá totéž číslo ve formátu `unsigned int` o dva bajty víc než jako `unsigned short`. Jinými slovy, proč se v případě typu `int` vyhradí čtyři bajty, když by stačily dva. Odpověď zní, že v okamžiku vyhrazení paměti ještě není jasné, jaká hodnota bude v paměti uložena. Navíc se hodnota může časem změnit. Proto se vždy vyhradí tolik bajtů, kolik je potřeba pro uložení největší možné hodnoty daného typu.

K čemu jsou menší datové typy

Také by vás mohlo zajímat, proč vlastně vůbec používat typ `short`, když nám `int` nabízí mnohem větší rozsah. Odpověď zní, že za větší rozsah platí `int` svou cenu: zabere dvojnásobek paměti, čtyři bajty namísto dvou.

Mohli byste namítnout, že dnešní počítače mají stovky megabajtů operační paměti a každý z megabajtů má 1 048 576 bajtů. Pár bajtů sem, pár bajtů tam – komu na tom záleží? Kdyby šlo skutečně jen o dva bajty, nezajímaly by nikoho. Kdybyste ale psali program pro pojišťovnu, která má milion klientů, byly by ze dvou bajtů rázem dva megabajty. Proto byste neměli po větších datových typech sahat automaticky.

Přesto platí, že `int` budete ze zmíněných datových typů používat nejčastěji. Jen je dobré o ostatních typech vědět, možnost volby se neztratí.

Datové typy pro desetinná čísla

Byl jsem odmalička krátkozraký, dokud jsem si nenechal oči opravit laserem. Při tomto zákroku zadá chirurg do programu informace, podle kterých laser odpaří tenoučkou vrstvu některých částí rohovky, zatímco jiné části – vzdálené na pouhé setiny milimetru – nechá netknuté.

Dovedete si představit můj výraz, kdyby mi chirurg řekl, že není žádný troškař a pracuje výhradně v celočíselné aritmetice? Že nějaká desetinná čísla jsou pod jeho úroveň? V ordinaci by po mně zůstala jen díra ve stěně a v dálce na obzoru moje silueta. (Jelikož ale ke svému očaři – který si mimochodem na vysokou školu vydělával jako programátor – ještě pořád chodím a nerad bych si to u něj rozházel, musím rychle dodat, že byl velice přesný a zákrok byl úspěšný.)

Celá čísla se hodí v mnoha případech, kdy zlomky vůbec nedávají smysl. Kdo by například tvrdil, že má 2¾ dítěte? V jiných případech desetinná místa přichází v úvahu, ale jsou zbytečná. Například u vzdáleností často stačí mluvit o devadesáti osmi mílích, nikoliv o 98,177 mílích.

Jindy na desetinných místech záleží hodně. Laserová operace očí je extrémní příklad, ale existuje i řada obyčejnějších. Kdybyste měli průměrný prospěch 3,9, asi by vás překvapilo, kdyby škola devět desetin zahodila a nechala průměr na třech. Podobně by dopadla banka, která by sledovala pohyb peněz jen v celých dolarech. Při milionech transakcí denně by záhy měla jen velice nepřesnou představu o tom, kolik peněz má k dispozici a kolik peněz má kdo na účtu.

Pro případy, ve kterých záleží na číslech napravo od desetinné čárky, jsou určené datové typy s plovoucí desetinnou čárkou. Výraz *plovoucí* znamená, že desetinná čárka tentokrát není na rozdíl od celých čísel umístěná pomyslně na konci čísla, ale může „plavat" kdekoliv uvnitř jeho zápisu. Čísla s plovoucí desetinnou čárkou se někdy označují jako *reálná*. (Ovšem reálným číslům v matematickém slova smyslu neodpovídají.)

Datové typy s plovoucí desetinnou čárkou shrnuje tabulka 2.3. Stejně jako u celočíselných datových typů platí, že uvedené velikosti a rozsahy se mohou v závislosti na operačním systému a překladači lišit.

Tabulka 2.3: Datové typy s plovoucí desetinnou čárkou

Datový typ	Velikost v bajtech	Rozsah
float	4	±3,4E-38 až ±3,4E38
double	8	±1,7E-308 až ±1,7E308
long double	10	±3,4E-4932 ±3,4E4932 (Poznámka českého vydavatele: Rozsah reálných čísel je trochu složitější. Záleží na tom, zda je číslo normalizované, či denormalizované. Nicméně rozsah je od +- absolutně nejmenších čísel až po +- absolutně největší čísla.)

Poznámka: Na mnoha kombinacích překladačů a operačních systémů má `long double` velikost 8 bajtů, nikoliv 10.

Vědecký zápis čísel

Zápis rozsahů v tabulce 2.3 vám možná připadne cizí. Čísla v něm totiž nejsou zapsaná tradičním způsobem, ale takzvanou *vědeckou notací*.

Datové typy s plovoucí desetinnou čárkou zvládnou uložit obrovská čísla, například jedničku se sedmatřiceti nulami, což by mohla být například vzdálenost přes celý vesmír. Stejně tak dobře uloží i čísla nevýslovně malá – například číslo, ve kterém je za desetinnou čárkou sedmatřicet nul a teprve pak jednička, což by mohl být například průměr subatomární částice. Aby se taková čísla nemusela roztahovat přes celou stránku, zapisují se zvláštním způsobem, viz tabulku 2.4.

Tabulka 2.4: Vědecký zápis čísel

Běžný zápis	Vědecký zápis	Zjednodušený vědecký zápis
123,45	$1{,}2345 \times 10^{2}$	1,2345E2
0,0051	$5{,}1 \times 10^{-3}$	5,1E–3
1 200 000 000	$1{,}2 \times 10^{9}$	1,2E9

Ve vědecké notaci se číslo zapisuje jako součin dvou čísel. První z nich, takzvaná *mantisa*, má vždy právě jednu číslici před desetinnou čárkou a libovolný počet číslic za ní. Druhé z čísel je mocnina desítky, která je u velkých čísel kladná a u čísel blízkých nule záporná. Vynásobením těchto dvou čísel dostaneme původní číslo. Zkrácená vědecká notace je totéž co běžná vědecká notace, jen znaménko součinu a desítka se nahradí písmenem E (za kterým zůstane původní exponent).

Způsob uložení v paměti

Jelikož v paměti mohou být jen nuly a jedničky, pro uložení čísel s plovoucí desetinnou čárkou se musí používat složité kódování, jehož popis už je za hranicí této knihy. I přes složité kódování ale počítač mnoho desetinných čísel dokáže uložit jen přibližně. Proto si programátoři u některých programů musí dávat pozor, aby se drobné nepřesnosti nesčítaly a nevedly k vážnějším chybám ve výpočtech.

Poznámka: Výpočty v plovoucí desetinné čárce jsou náročné na výpočetní výkon, a tak se často řeší specializovaným obvodem, takzvanou FPU (*floating point unit*). Ta dříve mívala podobu samostatného procesoru, takzvaného *matematického koprocesoru*; dnes už bývá integrovaná do hlavního procesoru.

Textové datové typy

Textové datové typy jsou dva. První se jmenuje *char* neboli znak. Většinou zabírá jeden bajt a můžete do něj uložit libovolný jeden znak – písmeno, číslici, interpunkci, mezeru a podobně. Druhým textovým datovým typem je řetězec neboli *string*. Do toho se vejde větší počet znaků, například tato věta, odstavec nebo celá stránka. Bajtů zabere podle toho, kolik je v něm uložených znaků.

Poznámka: Na rozdíl od datového typu char a dalších typů, které jsme probírali, není `string` přímou součástí C++. Je součástí standardní knihovny a definovaný je v souboru `string`. Pokud ho chcete používat, musíte tento soubor příkazem `#include <string>` vložit. Podrobné informace o direktivě `include` najdete v první kapitole u příkladu Hello world.

Způsob uložení v paměti

Typ `char` nezabírá jeden bajt náhodou. Ještě v dobách dálnopisu vznikl kdysi v Americe standard ASCII, který popisuje kódování znaků pomocí čísel (například A=65, Z=90 a podobně). Obsahuje celkem 128 znaků: 94 tisknutelných, jednu mezeru a 33 řídicích znaků, které mění způsob zpracování textu. K uložení libovolného ze 128 znaků je potřeba celkem 7 bitů, 2^7=128.

Když vznikaly první počítače, sáhly po tomto sedmibitovém kódování, které se dobře vešlo do jednoho bajtu. ASCII kódy některých běžně používaných znaků najdete v tabulce 2.5. Například písmeno J má ASCII kód 74, binárně 1001010. Když na nějaké adrese v paměti najdete hodnotu 1001010, mohlo by to být písmeno J.

Poznámka: Binární hodnota 1001010 by ale stejně dobře mohla znamenat číslo 74. Záleží na tom, jaký datový typ je s daným kusem paměti spojený. V následující kapitole se budeme věnovat proměnným, které slouží právě ke spojení datového typu s určitou adresou v paměti.

Bajt má osm bitů, a tak zůstane po uložení ASCII kódu nějakého znaku jeden bit nevyužitý. Proto hodně lidí napadlo, že by se dala tabulka ASCII rozšířit na 256 znaků, aby se využila plná kapacita jednoho bajtu. A že bylo proč rozšiřovat – původní tabulka ASCII totiž neobsahuje žádné znaky s diakritikou, počítá jen se základními anglickými písmeny. Problém je v tom, že rozšíření ASCII

vznikala v různých částech světa živelně a do velké míry nezávisle, takže nejsou vzájemně kompatibilní. Zatímco v jednom kódování je číslo 130 vyhrazené písmenu é, jinde tento kód znamená hebrejský znak gimel (ג). Nejčastěji používané kódování znaků pro jazyky střední Evropy jsou ISO-8859-2 a Windows-1250.

Problémy s kódováním národních znaků nějakou dobu přetrvávaly, až je nakonec vyřešil nový standard Unicode. Ten počítá se všemi znaky světa, a dokonce i některými znaky cizích světů. Tolik znaků už ovšem neuloží do jednoho bajtu, a tak jsou v C++ pro Unicode nové dva typy: `wchar_t` pro uložení Unicode znaku a `wstring` pro Unicode řetězce. My se v knize budeme pro jednoduchost držet obyčejných typů `char` a `string`, ale až budete psát opravdový program, měli byste se na typy `wchar_t` a `wstring` podívat.

Tabulka 2.5: ASCII kódy běžně používaných znaků

Znaky	ASCII kódy
0–9	48–57
A–Z	65–90
a–z	97–122

Datový typ bool

Kromě popsaných základních datových typů existuje ještě typ `bool`. Ten má jen dvě možné hodnoty, pravdu (`true`) a nepravdu (`false`), a v paměti většinou zabírá jeden bajt. Jeho název je zkratka od příjmení britského matematika George Boolea a souvisí s takzvanou booleovskou algebrou.

Zmiňujeme ho samostatně, protože do škatulky číselných ani textových datových typů příliš nezapadá. Za číselný datový typ by se dal považovat, kdybychom nulu brali jako nepravdu a jedničku (nebo libovolné další číslo) jako pravdu. Možná vám nepřijde intuitivní, že by zrovna nula měla být nepravda a jednička pravda – stačí si uvědomit, že počítače informace v podstatě ukládají pomocí přepínačů, které jsou buď zapnuté (1), nebo vypnuté (0).

Projekt: Jak zjistit velikost datových typů

Jak bylo řečeno v předchozím textu, velikost datových typů závisí na vašem překladači a operačním systému. V rámci tohoto projektu se naučíte zjistit velikost datových typů na vašem počítači pomocí operátoru `sizeof`.

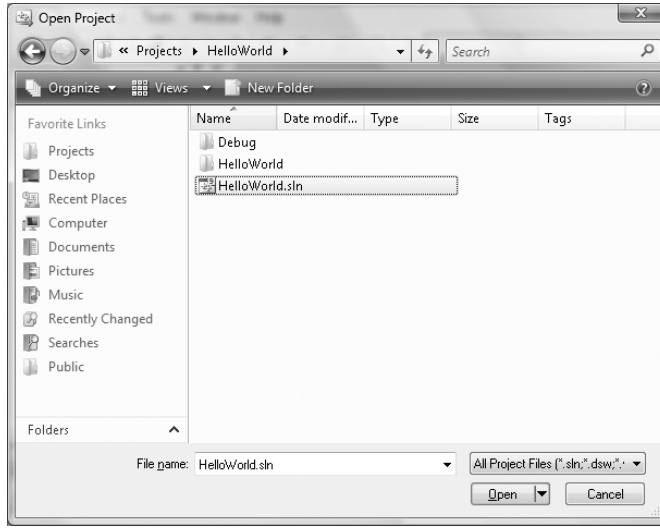
Operátor sizeof

Operátor `sizeof` vrací velikost datového typu v bajtech, stačí napsat `sizeof` a název datového typu v závorkách. Například na mém počítači má výraz `sizeof(int)` hodnotu 4. Znamená to, že při překladu tímto překladačem má typ `int` na mém počítači velikost čtyři bajty.

Změna zdrojového souboru našeho projektu

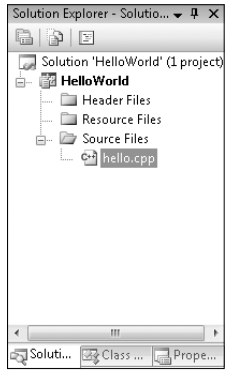
Zkusme si vytvořit a spustit následující program za pomocí postupu z první kapitoly, kde jsme vytvářeli program „Hello World!“. Mohli bychom vytvořit nový projekt, ale zkusíme znovu použít projekt z kapitoly 1. Je dobré znát obě varianty; jak vytvořit nový projekt a jak znovupoužít stávající projekt.

1. Spusťte Visual C++ 2008 Express Edition
2. Použijte příkaz z menu File → Open → Project/Solution k zobrazení dialogu pro otevření řešení (viz obrázek 2.5).



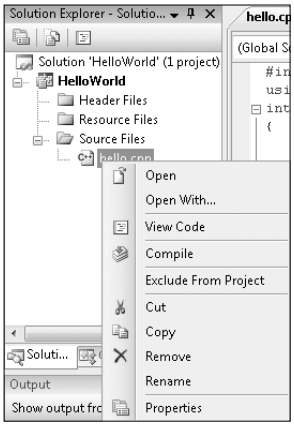
Obrázek 2.5: Otevření existujícího řešení

3. Přepněte se do adresáře, kde jste uložili svůj projekt, a najděte soubor řešení (solution file). Má příponu .sln, jako „solution". Soubor řešení je helloworld.sln, tak jak je ukázáno na obrázku 2.5.
4. Otevřete soubor s řešením. Mělo by dojít k otevření vašeho projektu.
5. Pokud je průzkumník řešení (Solution Explorer) skrytý, zobrazte ho pomocí příkazu v menu View → Solution Explorer. Rozbalte složku „Source Files" (zdrojové soubory) tak, abyste viděli soubor hello.cpp (viz obrázek 2.6).



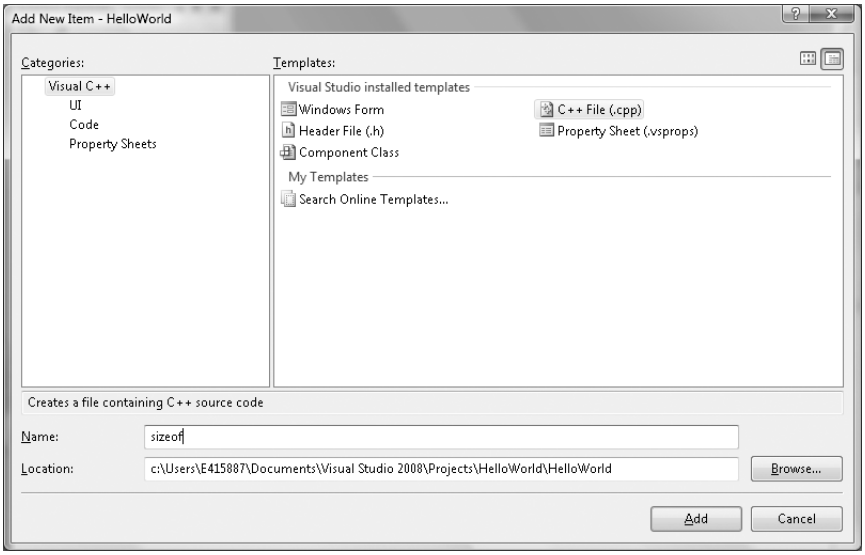
Obrázek 2.6: Existující zdrojový soubor v průzkumníku řešení

6. Klepněte pravým tlačítkem na soubor hello.cpp a z kontextového menu vyberte položku „Remove“ – odstranit (viz obrázek 2.7). Nebojte se, nedojde ke smazání souboru z disku. IDE se vás zeptá, zda chcete soubor pouze odstranit z projektu či vymazat. Vyberte volbu „Remove“ – odstranit. Vybraný soubor se pouze odstraní z projektu. Pokud si to budete přát, stále jej budete moct kdykoliv později použít.



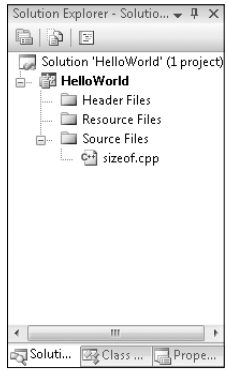
Obrázek 2.7: Položka „Remove“ k kontextovém menu

7. Klepněte pravým tlačítkem myši na složku „Source Files“ – zdrojové soubory a z kontextového menu vyberte příkaz „Add New Item“ – přídání nové položky. Objeví se námi známý dialog, jenž je zobrazen na obrázku 2.8.



Obrázek 2.8: Přidání nového zdrojového souboru do projektu

8. Neměňte hodnotu pole „Location“ – umístění. Nový soubor bude umístěn do složky projektu. Zadejte jméno nového zdrojového souboru do pole „Name“, nazvěte ho sizeof.cpp.
9. Jakmile budete hotovi, klepněte na tlačítko „Open“. Obrázek 2.9 ukazuje nový soubor sizeof.cpp v průzkumníku řešení.



Obrázek 2.9: Průzkumník řešení a nový soubor sizeoff.cpp

Poklepejte na soubor sizeoff.cpp v průzkumníku řešení (Solution Explorer). Soubor se otevře v editoru kódu. Nyní je sizeoff.cpp prázdný. V následujících sekcích do něj přidáme kód.

Zdrojový kód a výstup

Do zdrojového souboru, který jste vytvořili, zadejte následující kód. Jak funguje, si vysvětlíme v následujícím textu.

```
#include <iostream>
using namespace std;
int main(void)
{
    cout << "Velikost typu short: " << sizeof(short) << "\n";
    cout << "Velikost typu int: " << sizeof(int) << "\n";
    cout << "Velikost typu long: " << sizeof(long) << "\n";
    cout << "Velikost typu float: " << sizeof(float) << "\n";
    cout << "Velikost typu double: " << sizeof(double) << "\n";
    cout << "Velikost typu long double: " << sizeof(long double) << "\n";
    cout << "Velikost typu char: " << sizeof(char) << "\n";
    cout << "Velikost typu bool: " << sizeof(bool) << "\n";
    return 0;
}
```

Teď projekt přeložte a spusťte, stejně jako jsme to udělali s *Hello world* v první kapitole. Na mém počítači je výstup následující:

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

Poznámka: Čísla zobrazená na vašem počítači se mohou lišit, protože velikost datového typu závisí na operačním systému a překladači. Je docela pravděpodobné, že se alespoň v jednom z nich lišíme.

Výrazy

Řádek zdrojového kódu:

```
cout << "Velikost typu int: " << sizeof(int) << "\n";
```

se na výstupu programu projeví takto:

```
Velikost typu int: 4
```

V principu se tedy kód `sizeof(int)` na výstupu nahradí čtyřkou. Kódu typu `sizeof(int)` se říká *výraz*. Výraz je kus kódu, který má nějakou hodnotu – obvykle hodnotu, která se musí zjistit za běhu programu. Jednoduchým výrazem je například kód `4+4`, který se za běhu vyhodnotí na osmičku. V našem případě se výraz `sizeof(int)` za běhu programu vyhodnotí na čtyřku, která se pak vypíše na obrazovku.

Naproti tomu část příkazu uzavřená v uvozovkách (`"Velikost typu int:"`) se na výstup dostane beze změn. Není co vyhodnocovat. Této části se říká *řetězcový literál* – výraz *řetězcový* označuje datový typ a *literál* znamená, že se má brát doslova, bez dalšího vyhodnocování. Příkladem řetězcového literálu byl i řetězec `"Hello, world!"` v první kapitole.

Jak vypsat výraz

Výraz `sizeof(int)` je od literálu `"Velikost typu int:"` oddělený operátorem vložení do proudu (`<<`). Co kdyby kód vypadal takto?

```
cout << "Velikost typu int: sizeof(int)\n";
```

Výstup by dopadl úplně jinak:

```
Velikost typu int: sizeof(int)
```

Protože jakmile výraz `sizeof(int)` uzavřete do uvozovek, přestane se brát jako výraz. Bere se jako obyčejný literál, a tudíž se vypíše bez vyhodnocování. Jelikož je `"Velikost typu int:"` literál a `sizeof(int)` výraz, musí se před vložením do výstupního proudu oddělit operátorem `<<`.

Poznámka: V řetězci `"Velikost typu int: "` je za dvojtečkou ještě mezera. Kdybyste ji vypustili, na výstupu byste dostali `Velikost typu int:4`. Na mezery musíte dávat pozor vy jako programátoři, C++ to za vás neudělá.

Jak vypsat speciální znaky

Řetězec `"\n"` za výrazem `sizeof(int)` je také literál, takže je od výrazu oddělený operátorem `<<` a nebude se nijak vyhodnocovat. Přeci jen to ale není úplně obyčejný řetězec – obsahuje totiž zpětné lomítko (`\`), které má zvláštní význam. Když překladač v řetězci narazí na zpětné lomítko, pozná, že další znak řetězce nemá brát doslova. Když například za zpětným lomítkem najde malé písmeno `n`, vloží do řetězce znak nového řádku. Několik nejpoužívanějších zvláštních znaků najdete v tabulce 2.6.

Tabulka 2.6: Běžně používané speciální znaky

Znaky	Popis
`\a`	při výpisu pípne
`\n`	konec řádky
`\t`	tabulátor
`\\`	zpětné lomítko
`\'`	apostrof
`\"`	uvozovky

Shrnutí

Aby program fungoval, jeho příkazy a data musí být v operační paměti. Kromě operační paměti neboli RAM má počítač ještě další dvě zásadní centra paměti: vyrovnávací paměť u procesoru (+ registry procesoru) a trvalou paměť, například pevný disk. Z našeho pohledu je nejdůležitější operační paměť, ve které se uchovávají příkazy a data běžících programů.

Příkazy a data jsou v paměti uložené v podobě jedniček a nul na číselných adresách. Nejmenší jednotce informace, která může být jen jednička nebo nula, se říká *bit*. Tak malá jednotka je ale nepraktická na práci, a tak se většinou jako nejmenší jednotka používá *bajt* neboli osmice bitů. Na každé adrese v paměti je uložený jeden bajt.

Některé informace mají podobu číselnou, jiné textovou. Podrobněji by se datové typy daly rozdělit na celá čísla, desetinná čísla (neboli čísla s plovoucí desetinnou čárkou) a text. Všechny datové typy mají nějakou velikost, zabírají v paměti určitý počet bajtů. Velikost číselných typů určuje i jejich *rozsah*, tedy nejmenší a největší hodnotu, kterou typ zvládne uložit.

Velikost datových typů závisí na překladači a operačním systému. Velikost typů na vašem systému můžete zjistit pomocí operátoru `sizeof`.

Test

1. Do jaké z těchto pamětí má procesor nejblíže – vyrovnávací, operační, nebo trvalé?
2. Která z pamětí není dočasná – cache, RAM, nebo pevný disk?
3. Kolik informací se dá uložit na jedné paměťové adrese?
4. Je velikost datových typů C++ nezávislá na počítači?
5. Co je rozsah datového typu?
6. Jaký je rozdíl mezi datovými typy se znaménkem a bez?
7. Jaké číslo se skrývá pod zápisem 5,1E–3?
8. Co je to ASCII kód?
9. Co dělá operátor `sizeof`?
10. Co je to řetězcový literál?
11. Co je to výraz?

Proměnné

3

Když na mě nedávno kdosi v plné místnosti zavolal *hej*, otočilo se kromě mě i hezkých pár dalších lidí. Nikdo z nás totiž netušil, na koho ten pán mluví. Kdyby takhle rovnou zakřičel *hej, Jeffe*, hned bych věděl, že mluví na mě. (Ledaže by samozřejmě v místnosti bylo Jeffů víc.)

My lidé říkáme jeden druhému jménem. S programováním je to podobné – když se potřebujete odkázat na jeden konkrétní údaj třeba i z tisíců dalších, můžete ho pojmenovat.

K pojmenování informací slouží proměnné. Proměnná vám jednak umožní odkazovat se na nějakou informaci jménem, a jednak pro ni vyhradí potřebné místo v paměti. V této kapitole si ukážeme, jak se proměnné vytváří, jak se do nich ukládají informace a jak se z nich informace zase získávají nazpět.

Deklarování proměnných

V předchozí kapitole jste se dozvěděli, že pokud má program pracovat s nějakými informacemi, musí je mít uložené v operační paměti. A než můžete do paměti něco uložit, musíte si vyhradit potřebné místo. Právě k tomu slouží takzvané *deklarování proměnných*.

Deklarováním proměnné si vyhradíte místo v paměti a získáte pohodlný způsob, jak se na vyhrazenou paměť v programu odkazovat. V předchozí kapitole jsme se bavili o tom, že adresy v paměti jsou čísla, nadto zapisovaná většinou v šestnáctkové soustavě (například 0012fed4). Takové hodnoty se dost špatně pamatují. Pokud se daná informace týká řekněme výsledku testu, mnohem lépe by se nám pamatovala například pod názvem `pocetBodu`. Po deklarování proměnné se můžete na paměť odkazovat jménem proměnné, což je mnohem jednodušší, než si pamatovat šestnáctkovou adresu.

Deklarovat proměnnou není nijak těžké, stačí řádek kódu. Za oponou se toho ale děje poměrně hodně. Program na konci této části vám ukáže, jak se dá zjistit adresa a velikost paměti vyhrazené deklarací proměnné.

Jak deklarovat proměnnou

Proměnnou musíte deklarovat ještě před tím, než ji začnete používat. Deklarace proměnné vypadá následovně:

```
[datový typ] [jméno proměnné];
```

Jako datový typ můžete uvést například libovolný z datových typů popisovaných v předchozí kapitole, tedy `int`, `float`, `bool`, `char` nebo `string`. Podle datového typu počítač pozná, kolik paměti má na proměnnou vyhradit. V předchozí kapitole jsme se bavili o tom, že různé proměnné mají různé paměťové nároky. Když deklarujete proměnnou, která má při překladu vaším překladačem na vašem operačním systému velikost čtyři bajty, počítač pro ni vyhradí čtyři bajty paměti.

Jméno proměnné je na vás; pojmenování proměnných se budeme věnovat později v této kapitole. Jméno proměnné bude ve zdrojovém kódu sloužit jako zástupce pro vyhrazenou paměť. Když tedy deklarujete proměnnou `pocetBodu`, do které budete chtít ukládat výsledky testu, můžete se v kódu na vyhrazenou paměť odkazovat pod jménem `pocetBodu` a nemusíte se zajímat o žádné šestnáctkové adresy.

Deklarace proměnné končí středníkem, podle kterého překladač pozná, že příkaz skončil. Proměnnou můžete deklarovat uvnitř funkce (například uvnitř funkce `main`) nebo též nad všemi funkcemi (hierarchicky myšleno), těsně pod případnými direktivami `#include`. Naše programy mají prozatím jen jednu funkci (`main`), a tak budeme všechny proměnné deklarovat v ní. Až se nám počet funkcí rozroste, k deklarování proměnných se vrátíme.

Následující program deklaruje ve funkci `main` celočíselnou proměnnou jménem `pocetBodu`:

```
int main(void)
{
    int pocetBodu;
    return 0;
}
```

Poznámka: Všimněte si, že na rozdíl od prvních dvou kapitol nepotřebujeme žádnou direktivu `#include`, protože nepoužíváme `cout` ani žádné další objekty ze standardní knihovny.

Když se pokusíte o proměnné zmínit ještě před její deklarací, překladač ohlásí chybu. V následujícím kódu si první zmínka o proměnné `pocetBodu` vyslouží chybové hlášení o nedeklarovaném identifikátoru:

```
int main(void)
{
    pocetBodu;
    int pocetBodu;
    return 0;
}
```

Překladač ohlásí chybu, přestože proměnnou deklarujete hned na následujícím řádku. Zdrojový kód totiž překladač čte odshora dolů, takže když dorazí na první zmínku o identifikátoru `pocetBodu`, deklaraci proměnné ještě neviděl.

Chyba s nedeklarovanou proměnnou se podobá chybě, kterou jsme dostali u *Hello world* z první kapitoly po záměrném přejmenování `cout` na `Cout`. Jméno proměnné `pocetBodu` nepatří na rozdíl od `main`, `int` a podobných jmen přímo do jazyka C++, a tak ho překladač sám od sebe nepozná. Teprve když proměnnou deklarujete, překladač pozná všechny další výskyty jejího jména jako odkaz na tuto proměnnou.

Deklarování více proměnných najednou

Když potřebujete několik proměnných téhož typu, můžete je deklarovat jednotlivě:

```
int pocetBodu;
int mojeVyska;
int mojeVaha;
```

Pokud jsou ale stejného typu, nemusíte mít pro každou samostatný příkaz. Můžete je všechny deklarovat jedním příkazem, v rámci kterého je oddělíte čárkou:

```
int pocetBodu, mojeVyska, mojeVaha;
```

V tomto případě se datový typ uvádí jen jednou, i když deklarujete tři proměnné. Jelikož jsou všechny tři proměnné deklarované jedním příkazem, datový typ bude platit pro všechny společně. Takto společně se dají deklarovat pouze proměnné stejného typu. Dvě proměnné typu `int` a `float` už jedním příkazem deklarovat nejde, ty musíte deklarovat zvlášť:

```
int pocetBodu;
float prumer;
```

Pojmenování proměnných

Proměnné mají stejně jako lidé jména, kterými se na ně můžete odkazovat v kódu. Pro pojmenování proměnných platí několik pravidel:

- Název proměnné nesmí začínat jinak než písmenem abecedy (A–Z, a–z) nebo podtržítkem (_). Tajný agent se 007 jmenovat může, proměnná nikoliv. Počínaje druhým znakem už může jméno obsahovat i číslice.
- Název proměnné nesmí obsahovat mezery (`moje promenna`) ani interpunkci; jedinou výjimkou je podtržítko (_).
- Názvy proměnných se musí vyhýbat vyhrazeným slovům jazyka samotného, například `main` nebo `int`.
- Proměnná se nesmí jmenovat stejně jako jiná proměnná ve stejném rozsahu platnosti. (Rozsahu platnosti se budeme podrobně věnovat v osmé kapitole.) Stručně řečeno to prozatím znamená, že ve funkci `main` nemůžete deklarovat dvě shodně pojmenované proměnné.

S ohledem na tato pravidla už můžete proměnné pojmenovat téměř libovolně. Celkem dobrý nápad je vybírat jména, která vám něco řeknou. Když své proměnné budete označovat `prom1`, `prom2`, `prom3` a tak dál až do `prom17`, asi budete mít časem problém vzpomenout si na rozdíl mezi `prom8` a `prom9`. (A když s tím budete mít problém už vy sami, co potom asi bude říkat váš kolega, který kód vůbec nepsal.)

Pokud si chcete zachovat příčetnost (a případně i zdraví, pokud máte nervóznější kolegy), nejlepší pro vás bude volit názvy proměnných podle jejich určení. Například `pocetBodu` je slušný popisný název proměnné, která reprezentuje výsledek testu.

Název `pocetBodu` vznikl spojením dvou slov, *počet* a *bodů*. Mezera se nesmí objevit v názvu proměnné, takže `pocet bodu` se proměnná jmenovat nemůže. (Poznámka českého vydavatele: Ani diakritika se v názvech proměnných nepoužívá. Teoreticky tomu nic nebrání, v praxi byste narazili na problémy.) Proto se slova musí spojit dohromady a odliší se velkým písmenem na začátku druhého slova. Názvy proměnných bývá zvykem psát s malým písmenem na začátku.

Jmenné konvence

V praxi se programátoři při pojmenování proměnných drží nějakého jednotného systému, takzvané jmenné konvence. Jednu jmennou konvenci jsme právě popsali. Další možnost je přidat před každý název proměnné třípísmennou předponu, která označuje typ proměnné:

- `intBody`, celočíselný počet bodů, například z písemky.
- `strJmeno`, řetězec se jménem, například jméno osoby.
- `blnMistni`, booleovská proměnná, podle které se pozná, jestli je člověk místní.

Je celkem jedno, kterou jmennou konvenci se rozhodnete používat. Důležité je nějakou vybrat a držet se jí.

Operátor adresy

Při deklaraci proměnné dojde k vyhrazení paměti. Adresu této rezervované paměti dostanete takzvaným *operátorem adresy* (`&`). Jeho zápis vypadá následovně:

```
&[název proměnné]
```

Například následující kód vypíše na mém počítači hodnotu `0012fed4`. U vás bude vypsaná adresa proměnné `pocetBodu` nejspíš jiná – dokonce i na mém počítači se může při pozdějším spuštění programu lišit.

```
#include <iostream>
using namespace std;
int main(void)
{
    int pocetBodu;
    cout << &pocetBodu;
    return 0;
}
```

Adresa `0012fed4` je v šestnáctkovém zápisu, jak už bývá u adres zvykem (viz předchozí kapitola). Adresu proměnné vybírá operační systém, nikoliv programátor. Vybraná adresa závisí na typu proměnné, aktuální velikosti obsazené paměti a dalších vlivech.

Vy se o adresy starat nemusíte, protože se budete na proměnnou odkazovat jménem, nikoliv adresou. Jak ale zjistíte v jedenácté kapitole, operátor adresy je přesto velice užitečný.

Použití operátorů & a sizeof na proměnné

Velikost paměti, která se na proměnnou vyhradí, závisí na typu proměnné. Už v předchozí kapitole jsme si říkali, že různé typy mají různou velikost. Zároveň jsme si zkoušeli operátorem `sizeof` zjistit velikost různých datových typů na vašem operačním systému a překladači. Stejným způsobem se pomocí operátoru `sizeof` dá zjistit velikost různých proměnných (která opět závisí na operačním systému a překladači).

Velikost proměnné se operátorem `sizeof` určuje skoro stejně jako velikost datového typu, jen do závorek za `sizeof` napíšete místo názvu datového typu název proměnné. Následující kód vypíše adresu a velikost dvou proměnných:

```
#include <iostream>
using namespace std;
int main(void)
{
    short pocetBodu;
    float prumer;
    cout << "Adresa promenne pocetBodu: " << &pocetBodu << "\n";
    cout << "Velikost promenne pocetBodu: " << sizeof(pocetBodu) << "\n";
    cout << "Adresa promenne prumer: " << &prumer << "\n";
    cout << "Velikost promenne prumer: " << sizeof(prumer) << "\n";
    return 0;
}
```

Toto je pouze náhled elektronické knihy. Zakoupení její plné verze je možné v elektronickém obchodě společnosti eReading.