EDICE
myslíme v ...

knihovna programátora

- Přehledná a praktická učebnice C++ pro začátečníky i uživatele ostatních programovacích jazyků
- Základní programovací konstrukce, šablony, vestavěné datové typy, příkazy jazyka C++, pole a ukazatele
- Uživatelem definované neobjektové typy, výrazy a deklarace, funkce, práce s preprocesorem jazyka C++, objektové typy, výjimky, vstupní a výstupní operace
- Více než 170 příkladů použití v jednoduchých programech
- Podle mezinárodního standardu jazyka C++ z roku 2017

Programování v C++

od základů k profesionálnímu použití

MIROSLAV VIRIUS

soubory ke stažení na WWW.GRADA.CZ

GRADA

EDICE
myslíme v ...

knihovna programátora

Programování v C++

od základů k profesionálnímu použití

Miroslav Virius

GRADA
Publishing

Miroslav Virius

Programování v C++

od základů k profesionálnímu použití

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
obchod@grada.cz, www.grada.cz
tel.: +420 234 264 401, fax: +420 234 264 400
jako svou 6791. publikaci

Odpovědný redaktor Petr Somogyi
Sazba Petr Somogyi
Počet stran 416
První vydání, Praha 2018
Vytiskly Tiskárny Havlíčkův Brod, a. s.

ISBN 978-80-271-0956-2 (pdf)
ISBN 978-80-271-0502-1 (print)

Obsah

Předmluva

Kniha, kterou držíte v ruce, vás seznámí se základy programování v jazyce C++. Je určena jak čtenářům, kteří ještě vůbec neprogramovali, tak i těm, kteří již v nějakém jazyce programovali a chtějí se seznámit s jazykem C++. Předpokládám, že čtenář je sice začátečníkem v programování, ale umí s počítačem zacházet uživatelsky, tedy dokáže spustit program, zkopírovat nebo smazat soubor atd. V průběhu dvaceti kapitol této knihy se jej pokusím dovést na úroveň lehce pokročilého programátora.

Co v této knize najdete

Jediným způsobem, jak se naučit nějaký programovací jazyk, je psát v něm programy, a proto už ve druhé kapitole začneme programovat. Cílem druhé, třetí a čtvrté kapitoly je uvést na scénu vybrané konstrukce jazyka C++ a umožnit čtenářům zkoušet si vše, o čem si v následujících kapitolách povíme, na vlastních programech – jednoduchých a možná nešikovných, ale fungujících.

Než se ovšem pustíme do programování, musíme si ujasnit některé pojmy a sjednotit terminologii. Proto se v první kapitole seznámíme s pojmy, které budeme používat. Povíme si, co je třeba vědět o počítači, o programovacích jazycích, algoritmech, objektově orientovaném programování a některých dalších věcech, bez nichž se při programování neobejdeme.

Ve druhé kapitole napíšeme svůj první program, ukážeme si, jak ho přeložit (neboť jazyk, v němž ho píšeme, je jiný než jazyk, v němž „myslí" počítač) a jak ho spustit. Seznámíme se s integrovaným vývojovým prostředím Code::Blocs a s překladačem g++, které budeme v této knize používat.

Ve třetí kapitole napíšeme několik verzí jednoduchých programů, které přečtou číslo, vypočtou z něj jistou hodnotu a výsledek vypíší. To nám umožní seznámit se alespoň povrchně s některými základními programovacími konstrukcemi, s rozdělením programu do několika souborů, s pojmem projektu apod. Se znalostmi z této kapitoly byste měli být schopni zkoušet si vše, co se v dalších kapitolách naučíte, na vlastních programech.

Ve čtvrté kapitole si ukážeme jednoduchý objektový program, seznámíme se velmi povrchně se šablonami a předvedeme si, jak ladit program – tedy jak zjistit, proč dělá něco jiného, než co jsme chtěli naprogramovat.

Od páté kapitoly začíná výklad „naostro". Po seznámení se základními pojmy postupně poznáme vestavěné datové typy, příkazy jazyka C++, naučíme se používat tzv. pole a ukazatele. V deváté kapitole napíšeme první větší program – jednoduchou hru, v níž počítač klade otázky a podle uživatelových odpovědí hádá zvíře.

V následujících kapitolách se seznámíme s uživatelem definovanými neobjektovými typy, s výrazy a deklaracemi, s funkcemi, s preprocesorem jazyka C++, s objektovými typy a s tzv. výjimkami (nástroj pro ošetřování chyb za běhu). V předposlední kapitole probereme nástroje pro vstupní a výstupní operace (tedy pro čtení údajů z klávesnice, ze souboru apod.) a pro zápis údajů na obrazovku, do souboru apod.

V poslední kapitole postupně vyvineme několik verzí programu pro seřazení obsahu textového souboru. Jde o analogii filtru *sort*, který je standardní součástí běžných operačních systémů; naše verze se ovšem bude od standardního programu v mnoha ohledech lišit. To nám umožní seznámit se s řadou nástrojů ze standardní knihovny jazyka C++ a s některými užitečnými programátorskými technikami.

V poslední kapitole si krátce povíme o věcech, o nichž jsme v této knize nehovořili nebo jsme si řekli jen nejnutnější informace, ale které by měl pokročilý programátor v C++ znát.

Nástroje

Výklad v této knize je založen na mezinárodním standardu jazyka C++ [3] z roku 2017. Pro psaní, překlad, sestavení a ladění programů používáme vývojové prostředí Code::Blocs, které lze zdarma získat na stránkách [15] spolu s překladačem g++. Vedle toho zde najdeme poznámky pro čtenáře, kteří se rozhodnou pro Visual C++, jehož základní verzi lze také získat zdarma. Poznamenejme, že verze 4.9.2 překladače g++, která je implicitně dodávána s prostředím Code::Blocs, ve skutečnosti implementuje jazyk C++ na úrovni standardu z roku 2011. Není však problém instalovat si a používat nejnovější verzi tohoto překladače (v době dokončování této knihy to byla verze 6.4), která plně implementuje standard z roku 2014 a většinu novinek standardu z roku 2017. Místo překladače g++ si také můžete instalovat jakýkoli jiný překladač C++. Je však třeba poznamenat, že rozšíření, která přinesly pozdější verze standardu, v této knize nepoužívám, neboť výklad o nich přesahuje možnosti knihy určené začátečníkům.

Terminologie

V celé knize používám důsledně českou terminologii. Vím, že má mnoho odpůrců, jsem však přesvědčen, že použití vhodných českých názvů výrazně usnadní pochopení, oč jde. Anglické termíny samozřejmě uvádím alespoň při prvním výskytu také.

Příklady

Výklad v této knize doprovází velké množství příkladů. Jejich zdrojové texty si můžete stáhnout z webových stránek nakladatelství Grada Publishing (www.grada.cz, vyhledejte příslušnou sekci této knihy), nebo z mých osobních stránek, jejichž adresu najdete dále. Tyto příklady jsem odladil pod překladačem g++ verze 4.9.2 a pod Visual C++ 2015.

Poděkování

Na závěr bych chtěl poděkovat všem, kteří svými radami a připomínkami napomohli zdárnému dokončeni tohoto díla, především pak Ing. Rudolfu Pecinovskému, CSc., z VŠE v Praze a Ing. Vladimiru Jarému, Ph.D., z KSI FJFI ČVUT, kteří četli vybrané části této knihy a měli k nim řadu připomínek. I přes veškerou péči, kterou jsem této knize věnoval, se do ní ovšem mohly vloudit chyby; za ty nesu pochopitelně veškerou zodpovědnost já. Jestliže v této knize nějakou chybu najdete, pošlete mi prosím zprávu na níže uvedenou adresu. Bude-li to možné, uveřejním na svých webových stránkách opravu.
Ženeva, CERN, 21. srpna 2017

Miroslav Virius
miroslav.virius@fjfi.cvut.cz
http://people.fjfi.cvut.cz/virius

1 Než začneme

Než si začneme povídat o programovacím jazyce C++, musíme se alespoň povrchně seznámit s některými pojmy, které budeme používat. Začneme od počítače jako takového, pak si něco povíme o programovacích jazycích, algoritmech a objektově orientovaném programování a skončíme u C++ a nástrojů, které budeme potřebovat. Některé z uvedených pojmů nepochybně znáte; přesto vás prosím, abyste si toto povídání alespoň zběžně přečetli, abychom se mohli shodnout na terminologii.

1.1 Počítač

S osobními počítači se v současné době setkáváme denně – najdeme je nejen v každé kanceláři, ale i v téměř každé domácnosti. Dovolím si tedy předpokládat, že s počítačem umíte zacházet jako uživatelé – umíte ho zapnout, vytvořit na něm textový dokument apod. Ovšem programátor (a to i začínající) o něm musí přece jen vědět víc než běžný uživatel. I když bude povídání v této kapitole silně zjednodušené, pro naše účely bude stačit.

Obvykle se říká, že počítač obsahuje čtyři základní části:

- **Procesor** je součást, která opravdu „počítá", lépe řečeno která zpracovává informace. Zároveň také řídí činnost všech ostatních částí počítače.
- **Operační paměť** slouží k ukládání dat (informací), která počítač zpracovává, a programů, tedy příkazů, které určují, co má dělat. Vše, co je v této paměti, se při vypnutí počítače ztratí, „zapomene". Používá se pro ni označení *RAM*, což je zkratka anglických slov *Random Access Memory*, tedy **paměť s náhodným přístupem**.
- **Vstupní a výstupní zařízení** slouží k výměně informací s okolím. (Počítač by nám nebyl nic platný, kdyby nám nemohl předat výsledky své práce.) Typickými příklady vstupních zařízení jsou klávesnice, myš, skener atd. Typickým příkladem výstupních zařízení může být obrazovka monitoru nebo tiskárna. Pro vstupní a výstupní zařízení se používá zkratka V/V nebo I/O (z anglického *input/output*).
- **Trvalá paměť** slouží k trvalému ukládání dat a programů.[1] Data v ní se při vypnutí počítače neztrácejí, má zpravidla mnohonásobně větší kapacitu než operační paměť, ale práce s ní je mnohonásobně pomalejší než práce s operační pamětí. Jako trvalá paměť se používají převážně magnetické disky (pevný disk, disk SSD – jeho název pochází z anglických slov *solid state drive* a je to paměťové zařízení podobné jako flash disk neboli „fleška", ovšem s větší kapacitou) a samozřejmě také disky CD nebo DVD.

1 Setkáme se také s označením **vnější paměť**, neboť v dobách počítačové prehistorie byly magnetické disky opravdu uloženy mimo vlastní počítač. I dnes lze ovšem připojit k počítači vnější (externí) paměťové médium.

1.1.1 Operační paměť

Základem operační paměti jsou elektronické obvody, které mohou mít dva stavy – například vypnuto nebo zapnuto. Jeden z těchto stavů obvykle odpovídá číslici 0, druhý číslici 1. Údaje v paměti jsou proto vyjádřeny jen pomocí nul a jedniček, v tzv. dvojkové soustavě. Místo, na které můžeme uložit číslici 0 nebo 1, označujeme jako **bit**. Operační paměť je tedy dlouhá řada bitů.

Ovšem práce s jednotlivými bity je nepohodlná, a proto se bity sdružují do větších celků. V dnešních počítačích se téměř bez výjimky používají skupiny velikosti 8 bitů, které se nazývají **bajty** (anglicky *byte*, tedy slabika). Poznamenejme, že pro bity se používá značka b a pro bajty značka B.

Jednotlivé bajty, tvořící operační paměť, jsou očíslovány. Počáteční bajt má číslo 0, následující má číslo 1 atd. Toto pořadové číslo se nazývá **adresa** bajtu. (Ve skutečnosti může být záležitost s adresami trochu složitější, ale to nás v souvislosti s C++ vůbec nebude zajímat.)

1.1.2 Soustava SI, bity a bajty

Vnitřní paměť běžného počítače má dnes velikost několik miliard bajtů, kapacita disků a podobných zařízení se vyjadřuje v bilionech bajtů. Podobně jako v běžném životě nevyjadřujeme například vzdálenosti měst v metrech, ale v kilometrech, i v informatice se používají k vyjádření kapacity paměti (nebo obecně množství informace) větší jednotky. Běžně se hovoří o kilobajtech (kB), megabajtech (MB), gigabajtech (GB) atd.

Vzhledem k tomu, že počítače jsou založeny na dvojkové soustavě, je však přirozenější používat jednotky založené na mocninách dvou, nikoli na mocninách deseti, jak je tomu v soustavě SI. Proto by se měly používat jednotky kibibajt (1 KiB = 2^{10} B = 1024 B), mebibajt (1 MiB = 1024 KiB = 2^{20} B = 1048576 B), gibibajt (1 GiB = 2^{30} B = 1073741824 B) atd. Předpony těchto jednotek jsou složeny z první slabiky odpovídající předpony soustavy SI a slabiky **bi** ze slova *binary*.[2]

1.2 Datové typy a proměnné

Snadno zjistíme, že nejmenší číslo, které může jeden bajt obsahovat, se skládá z osmi nul a představuje i v desítkové soustavě nulu. Největší takové číslo se bude skládat z osmi jedniček a v desítkové soustavě představuje 255. To je samozřejmě málo – s počítačem, který by znal jen čísla od 0 do 255, bychom si nedokázali ani přepočítat výplatu. Proto se pro ukládání dat obvykle používají různě velké skupiny za sebou následujících bajtů.

Ani to ovšem nestačí. Snadno se přesvědčíme, že kdybychom vzali například skupinu dvou za sebou následujících bajtů, mohli bychom do ní uložit celá čísla v rozmezí od 0 do 65 535. Ale co když budeme potřebovat záporná čísla? Co když budeme potřebovat reálná čísla? Co když budeme chtít vyjádřit znaky nebo logickou hodnotu (nějaké tvrzení platí nebo neplatí)?

Musíme tedy zajistit způsob, který nám umožní reprezentovat data různých „druhů" v paměti počítače. Jinými slovy, musíme najít způsob, jakým určité skupině bitů přiřadíme hodnotu, kterou tato skupina představuje – jak ji v počítači **zakódovat**.

2 Tyto jednotky byly zavedeny Mezinárodní elektrotechnickou komisí r. 1998, přijaty hlavními standardizačními organizacemi (jsou například součástí standardu [5]) a tvoří součást mezinárodního systému měrných jednotek.

Můžeme se například dohodnout, že bajt s hodnotou `01000001` bude představovat znak `'A'`. Táž skupina bitů může za jiných okolností ovšem také představovat celé číslo, které má hodnotu 65. Stejný bajt ale může být i součástí většího celku s úplně jiným významem.

Předchozí příklady ukazují, že pracuje-li počítač s nějakým kouskem paměti, s nějakou skupinou za sebou následujících bajtů na určité adrese, musí vědět, jak je tato skupina velká a jak má její obsah interpretovat. Jinými slovy, musí znát **datový typ** hodnoty, která je tam uložena, musí vědět, zda jde o celé číslo, znak, logickou hodnotu atd.

Podle datového typu se také budou lišit operace, které lze s danou hodnotou provádět. Celá čísla lze například sčítat a odečítat, znaky lze spojovat do řetězců, tedy do souvislého textu.

Nyní se na celý problém podívejme trochu jinak. Už víme, že když budeme chtít pracovat s nějakou hodnotou, musíme si ji uložit do paměti. To ale znamená, že si tam pro ni musíme vyhradit místo a říci, jakého typu budou údaje, které bude obsahovat. Takové místo pro ukládání hodnoty budeme nazývat **proměnná**.

Abychom s proměnnou mohli zacházet, musíme ji pojmenovat, musíme jí přidělit **identifikátor**. Tomu se v programování říká **deklarace** proměnné.[3]

1.3 Programy a programovací jazyky

Aby mohl počítač nějak zpracovávat data, která mu předložíme, musíme mu také říci, co má vlastně dělat – musíme mu dát **program**.

Procesor umí s daty řadu operací, ovšem velice jednoduchých. Lze mu například říci „vezmi celé číslo, které je na adrese 6548, a přičti k němu celé číslo z adresy 7895". Protože do jeho paměti nelze uložit nic jiného než čísla, musí být tyto příkazy vyjádřeny – zakódovány – také čísly. Toto číselné vyjádření instrukcí (příkazů) pro procesor se nazývá **strojový kód** a je to jediná věc, které procesor rozumí. Jedním z problémů je, že různé druhy počítačů používají různé strojové kódy, takže programy ve strojovém kódu nejsou přenositelné mezi počítači s různými procesory.

Jiným – a možná horším – problémem je, že programování ve strojovém kódu je velice namáhavé a nepřehledné. Také to téměř nikdo nedělá; místo toho se používají tzv. **vyšší programovací jazyky** – PHP, Java, C, Basic, a také C++.

Zápis programu ve vyšším programovacím jazyce se zpravidla skládá z vybraných anglických slov a z výrazů zapsaných podobně jako v matematice. Programování ve vyšším jazyce je pochopitelně daleko jednodušší než programování ve strojovém kódu. Je tu ovšem jeden háček: takovýto program nelze přímo spustit, neboť počítač mu nerozumí. Program ve vyšším programovacím jazyce se proto musí buď **přeložit** do strojového kódu, nebo **interpretovat**. V obou případech k tomu potřebujeme další program (nebo skupinu programů), které to za nás udělají.

Zápis programu ve vyšším programovacím jazyce se zpravidla označuje jako **zdrojový kód** nebo **zdrojový program**, v „programátorštině" **zdroják**. Textový soubor, který tento zápis obsahuje, označujeme jako **zdrojový soubor**. O jeho **překlad** do strojového kódu (**kompilaci**) se stará program zvaný **překladač** neboli **kompilátor** (anglicky nazývaný *compiler*). V mnoha případech s ním spolupracuje ještě **sestavovací program** neboli *linker*, který může spojit několik nezávisle přeložených částí programu do jednoho celku. Sestavovací program také připojí knihovny – části programu, které už někdo naprogramoval předem a které můžeme už jen používat.

3 Později uvidíme, že identifikátor – tedy jméno – mohou mít i jiné části programu, nejen proměnné.

Překladem a sestavením programu vznikne soubor obsahující strojový kód, který lze na cílovém počítači rovnou spustit. Mezi často používané překládané programovací jazyky patří například C nebo C++.

Můžeme však také použít speciální program, jenž bude číst zdrojový text a provádět příkazy, které v něm najde – interpretovat ho. Typickým interpretovaným jazykem je klasický Basic.[4]

Interpretované programy zpravidla běží výrazně pomaleji než překládané programy. Navíc musíme na cílový počítač spolu s naším programem dodat také interpretační program.

Doplňme, že překladač libovolného programovacího jazyka (i C++) zároveň kontroluje syntaktickou správnost programu – tedy zda program je napsán podle jistých formálních pravidel, které zaručují, že mu počítač porozumí. (Syntaktická správnost programu bohužel nezaručuje věcnou správnost programu, tedy nezaručuje, že program bude dělat to, co si přejeme. Zaručuje pouze, že ho překladač dokáže přeložit.)

1.3.1 Jazyk C++

Už víme, že C++ patří mezi překládané programovací jazyky. Jeho zdrojový kód se může skládat z několika souborů. Překlad takového programu probíhá ve třech základních fázích:

1. Nejprve preprocesor upraví zdrojové texty – odstraní z nich komentáře (v následující kapitole se dozvíme, co to znamená) a provede některé další úpravy. Výsledkem jsou zdrojové soubory připravené k vlastnímu překladu.
2. Pak překladač přeloží samostatně jednotlivé zdrojové soubory. Překladem z nich vzniknou tzv. relativní soubory (anglicky nazývané *object file*). Relativní soubor se typicky jmenuje stejně jako zdrojový soubor, má však jinou příponu – pod Windows to je zpravidla `.obj`, v prostředí Linuxu to je `.o`.
3. Nakonec sestavovací program (linker) spojí relativní soubory v jeden celek, připojí k nim potřebné knihovny (už hotové součásti, které má překladač k dispozici) a vytvoří z nich spustitelný soubor; ten má pod Windows příponu `.exe`. V jiných operačních systémech nemusí mít žádnou zvláštní příponu.

Preprocesor, překladač a linker mohou být – a dnes zpravidla jsou – spojeny do jednoho programu, který provede postupně všechny potřebné kroky. Přepínači zadanými při spuštění tohoto programu lze ovšem zajistit, že proběhne například jen zpracování preprocesorem, jen zpracování preprocesorem a překlad nebo jen sestavení relativních souborů do spustitelného programu. Podrobněji se s tím seznámíme v příští kapitole.

1.3.2 Asembler

Mezistupněm mezi strojovým kódem a vyššími programovacími jazyky je **asembler** (*assembler, assembly language*). Je to jazyk složený především z mnemotechnických zkratek jednotlivých instrukcí strojového kódu. Například instrukce:

```
MOV EAX, 0
```

znamená v jednom z asemblerů procesorů Intel používaných na PC „ulož 0 do registru EAX". (Registry jsou součásti procesoru, v nichž se provádějí operace s daty.) Asembler také zavádí proměnné jako pojmenovaná místa v paměti, a proto se pro ně dříve používalo označení **ja-**

4 Nehovoříme o Visual Basicu. Poznamenejme, že ve skutečnosti může být kterýkoli jazyk překládán nebo interpretován (existuje například interpret jazyka C++), takže by bylo přesnější hovořit o jazycích *zpravidla* překládaných a *zpravidla* interpretovaných.