Microsoft

Ján Hanák

C++/CLI – Začínáme programovat

Ján Hanák

C++/CLI – Začínáme programovat

Artax
2009

Autor: Ing. Ján Hanák, MVP

C++/CLI – Začínáme programovat

Recenzenti:	doc. RNDr. Jozef Fecenko, CSc. Ing. Ľudovít Markus, CSc.
Jazyková korektura:	Ing. Ján Hanák, MVP
Vydání:	první
Rok prvního vydání:	2009
Náklad:	150 ks
Vydal:	Artax a.s., Žabovřeská 16, 616 00 Brno pro Microsoft s.r.o., Vyskočilova 1461/2a, 140 00 Praha 4
Tisk:	Artax a.s., Žabovřeská 16, 616 00 Brno
ISBN:	978-80-87017-04-3

Učební text *C++/CLI – Začínáme programovat* byl schválen v Edičním plánu Ekonomické univerzity v Bratislavě pro rok 2009 jako vysokoškolská učebnice pro studenty povinného předmětu *Automatizace programování* ve studijním programu *Hospodářská informatika*, 2. stupeň (inženýrské studium).

Obsah

Úvod ... 5

Obsahová struktura knihy ... 6

Pro koho je tato kniha určena ... 6

Software, jenž je v knize použit ... 7

Typografické konvence ... 8

1 První seznámení s jazykem C++/CLI, platformou Microsoft .NET Framework 3.5 a integrovaným vývojovým prostředím Visual C++ 2008 Express ... 11

1.1 Geneze vývoje: od jazyka C k jazyku C++/CLI ... 11

1.2 Platforma .NET Framework 3.5: prostředí pro vytváření řízených aplikací ... 23

1.3 Společný typový systém (CTS) a společná jazyková specifikace (CLS) ... 25

1.4 Řízená exekuce aplikací, mezijazyk MSIL a Just-In-Time kompilace ... 27

1.5 Prostředí CLR a služby, které nabízí řízeným aplikacím ... 33

1.6 Sestavení aplikace .NET a jeho architektura ... 40

1.7 Jaké aplikace lze na platformě .NET Framework 3.5 vyvíjet? ... 43

1.8 Visual C++ 2008 a Visual C++ 2008 Express: produkty, které umožňují psát aplikace pro platformu .NET Framework 3.5 v jazyce C++/CLI ... 44

1.9 Rozdíly mezi produkty Visual C++ 2008 a Visual C++ 2008 Express ... 46

1.10 Představení integrovaného vývojového prostředí Visual C++ 2008 Express ... 48

1.11 Vytváříme první aplikaci .NET v jazyce C++/CLI ... 52

1.12 Funkce main: vstupní bod konzolové aplikace .NET ... 67

1.13 Sestavení konzolové aplikace .NET ... 79

1.13.1 Co dělat, když se při kompilaci objeví chyba ... 81

1.13.2 Spuštění sestavené aplikace .NET ... 86

1.14 Seznámení se sestavovacími režimy aplikací .NET a ladícím programem (debuggerem) ... 87

1.15 Varianty přepínače /clr kompilátoru jazyka C++/CLI ... 94

1.16 Program PEVerify a kontrola sestavení aplikací .NET 101
1.17 Inspekce MSIL kódu a nástroj MSIL Disassembler (ILDASM) 106
1.18 Projektový management aneb důkladnější inspekce řešení a projektových souborů konzolové aplikace jazyka C++/CLI 112
1.19 Elektronická dokumentace, knihovna MSDN Express Library a prohlížeč Microsoft Document Explorer 129
1.20 Microsoft Document Explorer jako interní a externí prohlížeč elektronické dokumentace 141
1.21 Dynamická nápověda 143
1.21.1 Praktická ukázka práce Dynamické nápovědy 147
1.22 Management rozvržení oken v integrovaném vývojovém prostředí 151
1.23 Navigátor integrovaného vývojového prostředí (Navigátor IDE) 155
1.24 Exportování a importování konfiguračních nastavení integrovaného vývojového prostředí 156
1.25 Importování nastavení z konfiguračního souboru (.vssettings) 162
1.26 Integrované vývojové prostředí a automatické obnovení souborů 166
2 Základní výukový kurz algoritmizace a programování v jazyce C++/CLI 169
2.1 Program = data + algoritmy 169
2.2 Vlastnosti algoritmů 172
2.3 Výpočtová složitost algoritmů 177
2.3.1 Definice O-notace 181
2.3.2 Definice Ω-notace 182
2.3.3 Definice Θ-notace 183
2.4 Prostředky pro reprezentaci algoritmů 190
2.5 Datové typy a proměnné 199
2.6 Primitivní hodnotové datové typy jazyka C++/CLI 200
2.7 Definice proměnných 206
2.8 Přiřazovací příkaz 214

2.9 Celočíselné konstanty 223
2.10 Číselné soustavy v matematice a v programování 225
2.10.1 Oktálová číselná soustava 225
2.10.2 Hexadecimální číselná soustava 229
2.10.3 Binární číselná soustava 232
2.11 Reálné konstanty 236
2.12 Znakové konstanty 242
2.13 Globální proměnné 245
2.13.1 Zastínění globální proměnné 247
2.14 Konstantní proměnné 249
2.15 Typové konverze 250
2.16 Implicitní typové konverze 251
2.17 Explicitní typové konverze 256
2.17.1 Explicitní typové konverze ve stylu jazyka C 256
2.17.2 Explicitní typové konverze ve stylu jazyka C++ 263
2.17.3 Explicitní typové konverze realizované pomocí konverzních operátorů 265
2.17.4 Explicitní typové konverze uskutečňované prostřednictvím metod třídy Convert z jmenného prostoru System 267
2.18 Alokační kapacita proměnných 269
2.19 Operátory 273
2.19.1 Aritmetické operátory 275
2.19.2 Operátory pro inkrementaci a dekrementaci 280
2.19.3 Logické operátory 282
2.19.4 Relační operátory 284
2.19.5 Přiřazovací operátory 285
2.19.6 Bitové operátory 286
2.19.7 Operátory bitového posunu 288

2.20 Priorita a asociativita operátorů 291
2.21 Rozhodovací příkazy 295
2.21.1 Rozhodovací příkaz if 295
2.21.2 Rozhodovací příkaz if-else 298
2.21.3 Rozhodovací příkaz if-else if… 301
2.21.4 Rozhodovací příkaz if-else if-else 302
2.21.5 Rozhodovací příkazy: praktické cvičení 305
2.21.6 Rozhodovací příkaz switch 309
2.22 Programové cykly 312
2.22.1 Cyklus for 313
2.22.2 Cyklus while 319
2.22.3 Cyklus do-while 321
2.23 Funkce 323
2.24 Přetěžování funkcí 334
3 Základy objektově orientovaného programování v jazyce C++/CLI 340
3.1 Všeobecná teorie objektově orientovaného programování 340
3.2 Hybridní a objektově orientované programovací jazyky 347
3.3 Třída jako objektový uživatelsky deklarovaný odkazový datový typ 347
3.4 Deklarace třídy 348
3.5 Instanciace třídy 351
3.6 Přístupové metody třídy 357
3.7 Vlastnosti třídy 360
Závěr 367
O autorovi 368
Použitá literatura 370

Úvod

Vážení čtenáři,

tato vysokoškolská učebnice je jedinou knižní publikací v Česku a na Slovensku, která podává základní výukový kurz algoritmizace a programování v jazyce C++/CLI. Tento programovací jazyk společnosti Microsoft je nativně implementován v produktech Microsoft Visual Studio 2005, Microsoft Visual C++ 2005 Express, Microsoft Visual Studio 2008 a Microsoft Visual C++ 2008 Express. Samozřejmě, s jazykem C++/CLI se vývojáři setkají rovněž v produktech Microsoft Visual Studio 2010 a Microsoft Visual C++ 2010 Express, jež jsou v době vydání této knihy zatím pořád ještě ve vývoji.

Zatímco jiným .NET-kompatibilním programovacím jazykům (zejména C# a Visual Basicu) je věnována ve všech ohledech velká pozornost, jazyk C++/CLI stojí jaksi pořád v úzadí této „velké dvojky“. Možná právě proto se mnoho vývojářů, programátorů a softwarových expertů domnívá, že jazyk C++/CLI není plnohodnotným členem „rodiny .NET“. Toto tvrzení se ovšem nezakládá na pravdě, ostatně cílem této vysokoškolské učebnice je dokázat, že jazyk C++/CLI je produktivním prostředkem pro tvorbu moderního počítačového softwaru.

Předkládaná publikace se zaměřuje především na začínající programátory a studenty, kteří se chtějí naučit programovat v jazyce C++/CLI. Kniha byla psána tak, aby dovedla nabídnout kompletní servis, a to v tom smyslu, že čtenářům poskytuje veškeré potřebné rady, instrukce a informace. Takovýto přístup eliminuje nutnost paralelního studia jiných titulů, takže pokud máte tuto knihu a příslušný vývojový software, máte vše, co budete k úspěšnému zvládnutí jazyka C++/CLI potřebovat.

Na závěr by autor knihy rád vyjádřil své upřímné poděkování recenzentům, doc. RNDr. Jozefovi Fecenkovi, CSc., a Ing. Ľudovítovi Markusovi, CSc., za důkladné posouzení díla a hodnotné náměty na jeho další zkvalitnění.

Ján Hanák

Bratislava, květen 2009

Obsahová struktura knihy

Vysokoškolská učebnice *C++/CLI – Začínáme programovat* je tvořena následujícími tematickými celky:

1. **První seznámení s jazykem C++/CLI, platformou Microsoft .NET Framework 3.5 a integrovaným vývojovým prostředím Visual C++ 2008 Express.** V první části knihy se čtenáři seznámí s evolucí jazyka C++/CLI, vývojově-exekuční platformou Microsoft .NET Framework 3.5 a integrovaným vývojovým prostředím produktu Microsoft Visual C++ 2008 Express. Naučí se sestavit první funkční program v jazyce C++/CLI a pochopí, jak vytvořený program funguje.

2. **Základní výukový kurz algoritmizace a programování v jazyce C++/CLI.** Druhá část knihy podává teoreticko-praktický kurz programování v jazyce C++/CLI. Výklad je zahájen teorií algoritmů, která vysvětluje algoritmy, definuje vlastnosti algoritmů a prostředky pro jejich reprezentaci. Stranou ovšem nezůstává ani rozprava o výpočtové složitosti algoritmů. Dále jsou vysvětleny všechny důležité aspekty programování, k nimž patří datové typy, proměnné, typové konverze, operátory, rozhodovací příkazy, programové cykly a funkce. To vše doplněné praktickými ukázkami a příklady, na nichž lze lépe dokumentovat probíranou problematiku.

3. **Základy objektově orientovaného programování v jazyce C++/CLI.** Třetí část knihy pojednává o základních pilířích všeobecné teorie objektově orientovaného programování (VTOOP), přičemž předvádí praktickou aplikaci vybraných pilířů VTOOP v jazyce C++/CLI.

Po absolvování všech částí knihy by měli čtenáři získat kvalitní teoretické znalosti a hodnotné praktické zkušenosti, které budou jistě potřebovat při studiu pokročilých partií programování v jazyce C++/CLI.

Pro koho je tato kniha určena

Ovládnutí nového programovacího jazyka není nikdy snadné, obzvláště tehdy, když se čtenář s programováním ještě nikdy nesetkal. Prosím, mějte na paměti, že tato kniha nepředpokládá žádnou bázi znalostí u cílových příjemců. Jinými slovy, nepředpokládáme, že jste se někdy

s programováním setkali a dokonce ani nepožadujeme, abyste absolvovali kurzy teoretické informatiky. Postupně, jak budete procházet jednotlivými kapitolami této knihy, zjistíte, že všechno bude vysvětleno s důrazem na preciznost a přiměřené studijní zatížení.

Primárním cílovým segmentem knihy *C++/CLI – Začínáme programovat* jsou posluchači informaticky zaměřených vysokých škol univerzitního typu. Samozřejmě, kniha je vhodnou pomůckou také pro fanoušky programování a rovněž pro kohokoliv, kdo má zájem naučit se programovat v jazyce C++/CLI.

Software, jenž je v knize použit

Hlavní software, jejž v celé knize používáme, je Microsoft Visual C++ 2008 Express. Tento software společnost Microsoft poskytuje zdarma pro každého zájemce. To znamená, že pokud tento software ještě nemáte, můžete si jej bezplatně převzít z webových stránek Microsoftu (http://www.microsoft.com/exPress/download/). Tu a tam v knize ovšem poukážeme na pokročilou funkcionalitu, která není v expresní verzi produktu Visual C++ k dispozici. Pokročilé rysy jsou součástí vyšších (a placených) verzí Visual Studia 2008. Nicméně, partie výkladu, které se věnují pokročilým funkcím, jsou vždy zřetelně označeny a čtenáři jsou upozorněni, že popisovaný rys se nevyskytuje ve standardní výbavě produktu Microsoft Visual C++ 2008 Express.

Přestože jsme v knize dali přednost verzi 2008 produktu Microsoft Visual C++ Express před verzí 2005 téhož produktu, neznamená to, že byste nemohli použít i starší software, pokud ho už máte na počítači nainstalovaný. Zpravidla se ale snažíme pracovat s nejaktuálnějším dostupným softwarem, a proto logicky padla volba na Microsoft Visual C++ 2008 Express. V době, kdy se objeví Microsoft Visual C++ 2010 Express, budete moci tuto knihu dále používat také s tímto nejnovějším integrovaným vývojovým prostředím. Jistě, počítáme s tím, že se v novém Visual C++ něco změní, ovšem základy zůstanou bezesporu zachovány v původní podobě.

Kniha vám samozřejmě stejně dobře poslouží i tehdy, máte-li jednu z placených verzí (Standard, Professional, Team System) produktu Microsoft Visual Studio 2005 / 2008. Software od jiných počítačových společností ovšem použít nelze, poněvadž jazyk C++/CLI je začleněn pouze do zmíněných produktů společnosti Microsoft.

Typografické konvence

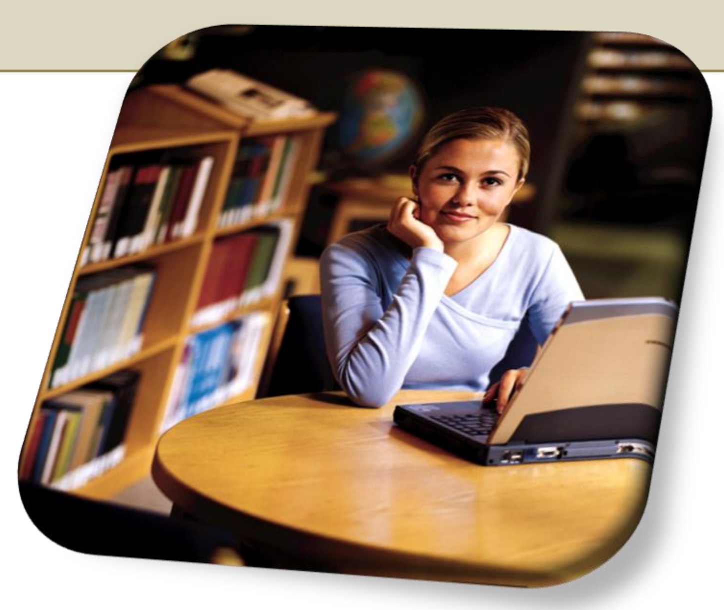
Abychom vám čtení této vysokoškolské učebnice zpříjemnili v co možná největší míře, byl přijat kodex typografických konvencí, jejichž pomocí došlo k standardizaci a unifikaci použitých textových stylů a grafických symbolů. Pevně věříme, že přijaté konvence zvýší přehlednost a uživatelskou přívětivost výkladu. Přehled typografických konvencí a informačních ikon uvádíme v tab. A a v tab. B.

<table>
<tr><th colspan="2">Tab. A: Přehled typografických konvencí</th></tr>
<tr><th>Typografická konvence</th><th>Ukázka použití
typografické konvence</th></tr>
<tr><td>Standardní text výkladu, který neoznačuje zdrojový kód, identifikátory, modifikátory a klíčová slova jazyka C++/CLI, ani názvy jiných syntaktických elementů a entit, je formátován tímto typem písma.</td><td>Vývojově-exekuční platforma Microsoft .NET Framework 3.5 vytváří společně s jazykem C++/CLI jednotnou technologickou bázi pro budování moderních řízených aplikací.</td></tr>
<tr><td>Názvy nabídek, položek nabídek, ovládacích prvků, komponent, dialogových oken, podpůrných softwarových nástrojů, typů projektů, jakožto i názvy dalších součástí grafického uživatelského rozhraní, jsou formátovány tučným písmem.</td><td>Pro založení nové konzolové aplikace v jazyce C++/CLI postupujte takto:

1. Na stránce Start Page klepněte na odkaz Create Project.
2. V dialogovém okně New Project klepněte ve stromové struktuře Project types na uzel Visual C++.
3. Ze seznamu projektových šablon Templates vyberte šablonu CLR Console Application.
4. Do textového pole Name zadejte název pro novou konzolovou aplikaci.
5. Nakonec klepněte na tlačítko OK.</td></tr>
<tr><td>Fragmenty zdrojového kódu jazyka C++/CLI jsou formátovány neproporcionálním písmem <code>Courier New</code>.</td><td><code>// Definice proměnné typu int.
int a;
// Inicializace definované proměnné.
a = 10;
// Vypsání hodnoty proměnné.
Console::WriteLine(a);</code></td></tr>
</table>

Tab. B: Přehled informačních ikon		
Informační ikona	**Text informační ikony**	**Charakteristika**
	Upozornění	Upozorňuje čtenáře na důležité skutečnosti, které by měli mít v každém případě na paměti, neboť na nich může záviset pochopení dalších souvislostí nebo úspěšné provedení postupu či algoritmu.
	Poznámka	Sděluje čtenářům další a podrobnější informace, které se pojí s vykládanou tematikou. Ačkoliv je míra důležitosti této informační ikony nižší než výše uvedené ikony, ve všeobecnosti se doporučuje, aby čtenáři věnovali doplňujícím informačním sdělením svoji pozornost. Mohou se tak dozvědět nová fakta, nebo najít skryté souvislosti mezi již známými poznatky.
	Tip	Poukazuje na lepší, efektivnější nebo rychlejší splnění programovacího úkolu či postupu. Uvidí-li čtenáři v textu publikace tuto informační ikonu, mohou si být jisti, že naleznou jedinečný a prověřený způsob, jak produktivněji dosáhnout kýženého cíle.

C++/CLI – Začínáme programovat



Část 1: První seznámení s jazykem C++/CLI,
platformou Microsoft .NET Framework 3.5
a integrovaným vývojovým prostředím
Visual C++ 2008 Express

1 První seznámení s jazykem C++/CLI, platformou Microsoft .NET Framework 3.5 a integrovaným vývojovým prostředím Visual C++ 2008 Express

1.1 Geneze vývoje: od jazyka C k jazyku C++/CLI

Počátkem sedmdesátých let dvacátého století vznikl programovací jazyk C, který byl navržen a implementován Dennisem Ritchiem a jeho spolupracovníky v Bellových laboratořích. C byl projektován jako kompilovaný jazyk střední úrovně, protože programátorům nabízel mnohem větší úroveň abstrakce od hardwarové infrastruktury než jazyk symbolických instrukcí. Ačkoliv se zpočátku jazyk C používal hlavně pro psaní vědecko-technických programů pro sálové počítače s terminály, společně s rozvojem mikropočítačů se dostal i mezi širokou odbornou veřejnost. „Céčko" se záhy vývojářům zalíbilo, a to tak moc, že ještě nyní, více než třicet let později, je tento jazyk vyučován na univerzitách a těší se značné oblibě. Důvodů pro výjimečnost jazyka C je více. Pokusme se shrnout alespoň ty nejmarkantnější:

- **C je přívětivý jazyk.** I když autor této publikace ze své pedagogické praxe ví, že zdaleka ne všichni studenti tuto ideu sdílejí, je nutno říci, že v době svého vzniku byl jazyk C mnohem lepším prostředkem pro psaní počítačového softwaru než cokoliv jiného. Díky jasně dané jazykové specifikaci byl vytvořený zdrojový kód přehledný a snadno pochopitelný. Navíc, ve srovnání s dalšími jazyky vycházejícími z jazyka C, jako je C++, C++/CLI či C#, je jazyk C jistě nejjednodušší. Céčko má rovněž svůj nepopíratelný půvab a jiskru, kterým když jednou propadnete, jenom stěží si budete zvykat na jazyk postavený na docela jiných syntakticko-sémantických pravidlech. Na druhou stranu musíme uznat, že ne každému může C vyhovovat. Mnozí programátoři říkají, že C nebude nikdy tak přímočaré jako Basic (respektive Visual Basic), s čímž souhlasíme. Ovšem pokud se hodláte věnovat vývoji softwaru přece jenom vážněji, s jazykem C (ostatně podobně jako i s dalšími uvedenými členy „rodiny C") se budete jistě často setkávat.

- **C je jazyk střední úrovně s přímým přístupem k nízkoúrovňovým službám počítače.** Ačkoliv C poskytuje vyšší míru hardwarové abstrakce, jsou v něm integrovány příkazy a syntaktické konstrukce, jejichž pomocí můžeme bez

jakýchkoliv potíží explicitně přistupovat k operační paměti počítače, videopaměti, portům a dalším počítačovým komponentám. Zejména možnost provádět rychlé operace s pamětí je vysvětlením toho, proč jsou kritické části mnoha složitých softwarových aplikací (kupříkladu operačních systémů) psány v starém dobrém céčku. O výjimečnosti jazyka C svědčí také fakt, že v tomto jazyce jsou napsány také výkonnostní knihovny optimalizovaných funkcí společností AMD a Intel. Oba výrobci mikroprocesorů poskytují vývojářům kolekce odladěných funkcí pro zpracování signálů, práci s audio a video daty či pokročilou aplikaci matematických operací (lineární algebra, vektorový a maticový počet). Pokud budete mít chuť, můžete se podívat na knihovny funkcí AMD Performance Library (APL) a Intel Integrated Performance Primitives (IPP).

- **C je kompilovaný jazyk.** Řečeno jinak, výsledkem práce kompilátoru a sestavovacího programu (známého též pod názvem linker) je fyzický soubor s kódem, jenž může být přímo podroben exekuci (takzvaný strojový kód). Kompilátor zabezpečí překlad zdrojového kódu jazyka C, přičemž provede několik druhů analýz (lexikální, syntaktická a sémantická analýza), v případě potřeby rovněž realizuje příslušná optimalizační opatření a vygeneruje odpovídající objektový kód. Jakmile je objektový kód spojen s knihovním kódem, dochází ke vzniku spustitelného souboru programu. Jak jsme si již pověděli, spustitelný soubor obsahuje strojový kód, který je vykonáván přímo procesorem počítače (CPU). Každý procesor obsahuje jistou množinu instrukcí, se kterými je schopen pracovat. Tato množina se označuje jako instrukční sada CPU. V současné době jsou ještě pořád nejrozšířenější 32bitové procesory s x86 instrukční sadou. Pokud je spuštěn program napsaný v jazyku C, instrukční sada procesoru začne zpracovávat všechny jeho instrukce. Díky své kompilační povaze jsou programy připravené v jazyku C zpravidla velmi rychlé, dokonce tak rychlé, že je jedině málokdy něco předčí.

  Samozřejmě, efektivnost algoritmu, jeho výpočtová složitost a další aspekty potenciální optimalizace by vydaly na samostatnou publikaci, no aniž bychom chtěli příliš zabíhat do čistě vědecké problematiky, můžeme prohlásit, že programy v C jsou vskutku rychlé. A jestliže nejsou některé partie kódu tak mrštné, jak by měly být, pak je můžeme optimalizovat v jazyce symbolických instrukcí. V každém případě je však kompilovaný program hbitější než interpretovaný program, což je dáno již samotným principem interpretace. Zatímco obsahem kompilovaného programu je strojový kód, uvnitř interpretovaného programu se nenacházejí

instrukce, které mohou být podrobeny přímé exekuci procesorem. Interpretovaný program pak potřebuje překladač, který dotyčný mezikód na požádání překládá do formy srozumitelné pro CPU.

- **Jazyk C je standardizován organizacemi ISO a ANSI.** Z hlediska vývojářů jsou podstatné dva standardy: první, známý jako C90, byl přijat v roku 1990 organizací ISO v dokumentu ISO 9899/ISO/IEC 9899:1990. O devět let později se jazyk C dočkal poměrně hlubokých inovací, které byly zakotveny ve standardu známém jako C99 (dokument ISO 9899/ISO/IEC 9899:1999). Přestože od přijetí standardu C99 již uplynula hezká řádka let, nemálo překladačů tento standard neimplementuje v celé šíři ani v současné době. Ve skutečnosti se pak setkáváme s překladači, které za základ berou standard C90, k němuž tu a tam přidávají některé novinky ze standardu C99. Jak se ke standardům stavějí překladače je jedna věc, ovšem mnohem důležitější je skutečnost, že tyto standardy vůbec existují. Hlavní předností je přenositelnost standardizovaného programového kódu na velký počet počítačových platforem. Pokud máte po ruce standardizované překladače a patřičné hardwarové zabezpečení, pak můžete programy napsané v jazyce C používat na různých počítačových platformách. Řečeno jednodušeji: můžete svůj program napsat na PC se systémem Windows a máte zaručeno, že tento program bude moci být spouštěn rovněž na jiných systémech, třeba Mac OS, Unix nebo Linux.

Jazyk C je typickým představitelem programovacího jazyka určeného pro strukturované programování. Podobně, jak se vyvíjejí programovací jazyky, tak změnou procházejí také obecné přístupy k tvorbě počítačových programů. Strukturované anebo též procedurální programování bylo založeno na myšlence přímé práce s daty. Program byl proto strukturován jako množina funkcí, které vykonávaly jisté činnosti a v případě potřeby mezi sebou komunikovaly. Přitom jedna z těchto funkcí měla výsadní postavení: jmenovala se hlavní funkce a reprezentovala vstupní bod programu. Když byl program spuštěn, byla automaticky zavolána jeho hlavní funkce a posléze zpracovány příkazy, které se nacházely v jejím těle. Funkcím se někdy říkalo i podprogramy, tudíž mnozí programátoři mluvili o své aplikaci jako o sbírce podprogramů. Aby se zabezpečila opětovná použitelnost jednou napsaného zdrojového kódu, funkce byly (obvykle podle příbuznosti) ukládány do knihoven, které se staly předmětem sdílení. Možná znáte dynamicky provázané knihovny (ano, to jsou ty známé knihovny DLL), jejichž služeb může využívat více klientů (aplikací). Knihovny DLL jsou ukázkou přístupu, jenž zabezpečuje maximalizaci opětovné použitelnosti zdrojového kódu.

Jak šel čas, vědečtí pracovníci, ale také programátoři z praxe, začali cítit, že pro jisté typy aplikací není strukturované programování tou pravou metodikou. Slabým místem strukturovaného programování se stala především jeho přílišná centralizace na samotná data. Časté výtky směrovaly k faktu, že práce s prostými daty bez návaznosti na operace, které lze s těmito daty uskutečnit, není optimální. Spíše bychom se měli snažit data a spřízněné operace zapouzdřit do logických jednotek, takzvaných objektů. Kromě principu zapouzdření (neboli enkapsulace) zavážil také návrh, že programy by měly modelovat procesy probíhající v reálném světě. A pakliže se v skutečném světě neustále potýkáme s objekty všeho druhu, nebylo by dobré tuto analogii přenést také do světa softwaru? Ano, samozřejmě, to je ono…

Možná to bude znít neuvěřitelně, no základní pilíře objektově orientovaného programování (OOP) byly postaveny před mnoha desítkami let. Objektová filosofie říká, že vytváření softwaru spočívá v kreaci virtuálních objektů, které jsou obdařeny vlastnostmi (atributy) a dovedou vykonávat jisté činnosti. Atributy tvoří datovou část objektu a činnosti, které objekt realizuje, jsou reprezentovány metodami. Tyto virtuální objekty jsou vytvářeny v procesu abstraktního modelování, v rámci něhož programátoři určí, jakými atributy a metodami budou finální objekty disponovat. Velice důležité je přitom zapouzdření: objekt je kompaktní jednotka, která spravuje svá data a používá je k provádění naprogramovaných akcí. Neméně důležité jsou další postuláty objektově orientované filosofie vývoje softwaru jako ukrývání dat a ukrývání implementace, dědičnost, polymorfizmus a opětovná použitelnost programového kódu.

Popojedeme-li o malinký kousíček dál, pak zákonitě zjistíme, že jazyk C není objektově orientovaný. O zavedení OOP principů do jazyka C se pokusil Bjarne Stroustrup, který začal tuto tematiku rozpracovávat ve své dizertační práci někdy koncem sedmdesátých let minulého století. Výsledkem bylo vynalezení nového programovacího jazyka, s pracovním názvem C with Classes (C s třídami). V boji o finální pojmenování však zvítězila proslulá formulka C++, která praví, že C++ je vlastně „vylepšené C" (anebo „C povýšené o jednu úroveň", jak bychom se mohli domnívat z užití postfixového inkrementačního operátoru). Jazyk C++ byl uveden v polovině 80. let a navzdory počáteční nejistotě se zanedlouho začal těšit bouřlivé popularitě. Z dnešního pohledu je úsměvné, že pro úspěšnou penetraci C++ se počítalo s tím, že s jazykem bude pracovat nejméně 5000 společností.

Jazyk C++ byl podobně jako jazyk C standardizován. Než však dospěly technické komise k finálnímu dokumentu, uběhlo dlouhých devět let. Standard ISO/IEC 14882:1998 Programming Languages - C++ spatřil světlo světa v roce 1998 a je někdy zkráceně

označován jako C++98. O pět let později byl standard minoritně aktualizován a vydán v dokumentu ISO/IEC 14882:2003 (tento standard se často označuje jako C++03). V současné době probíhají práce na novém, a nutno podotknout, že rázně inovovaném standardu, který je prozatím znám jako C++0x. Podle předpokladů by měl být standard dokončen v roce 2009.

Přestože na tomto místě mluvíme o jazyce C++, musíme jedním dechem dodat, že „Céčko se dvěma plusy" není čistě objektovým programovacím jazykem. Spíše bychom měli říci, že se jedná o hybridní jazyk, jenž umožňuje vyvíjet jak strukturovaně, tak i objektově. Po pravdě řečeno, nic vám nabrání v tom, abyste i v jazyce C++ programovali přesně tak jako v starším C. Zřejmě nemusíme připomínat, že to není optimální strategie.

C++ se řadí mezi nejoblíbenější a vůbec nejpoužívanější programovací jazyky na naší planetě. K dosažení této mety přispěly následující faktory:

- **C++ působí jako objektová nadstavba jazyka C**. Pokud umíte programovat v jazyce C, je přechod k C++ přirozený a plynulý. Jistě, je zapotřebí se naučit objektovému myšlení a novým syntaktickým konstrukcím, ovšem provází vás identický styl psaní kódu a známé příkazy. S osvojením jazyka C++ lze pochopitelně začít i v případě, když vývojář nemá dřívější zkušenost s jazykem C. Za popsaných okolností je nutno se veškerou problematiku naučit „na jeden šup". Nepopíráme, že se jedná o přeci jenom delší cestu, nicméně známe nemálo profesionálních vývojářů, kteří se k jazyku C++ dostali právě takto.

  Hlavní konkurenční výhodou jazyka C++ je důsledné zapracování objektově orientovaného programování se všemi těmi drahokamy jako abstrakce, zapouzdření, dědičnost a polymorfizmus. Není pochyb o tom, že C++ si s OOP rozumí více než dobře, no toto přátelství má i svou stinnou stránku. Ta se váže k ovládnutí jazyka jako takového. Křivka učení C++ není bohužel nijak strmá, což je zapříčiněno snad až přespříliš košatou jazykovou specifikací. Autor knihy vede na univerzitě kurzy programování v jazycích C, C++ a C#. Jak vyplývá z jeho dosavadních akademických zkušeností, jazyk C++ se studentům jeví jako docela složitý (a to i tehdy, nezabíháme-li do přílišných technických podrobností a implementačních detailů).

- **C++ přichází s novými programovými rysy, které nebyly dříve dostupné.** Říci o jazyku C++, že je pouhým rozšířením jazyka C, by byla asi taková troufalost, jako tvrdit, že Porsche Cayman je automobilem pro běžné lidi. Ano, můžeme prohlásit, že

C++ vychází z C stejně tak, jako můžeme konstatovat, že Porsche je také auto. Ovšem C++ vám umožní objevit daleko barvitější dimenze vývoje softwaru, zatímco Porsche vás naučí, co ve skutečnosti znamená pojem rychlost.

Pomineme-li syntaktické drobnůstky, pak C++ přináší vedle OOP také podporu pro práci s generickými (parametrizovanými) datovými typy, které jsou v tomto prostředí ztělesňovány šablonami. Díky šablonám lze uplatnit parametrický polymorfizmus jako doplněk k polymorfizmu inklusivnímu. Další novinkou jsou chybové výjimky, chytré objekty, s jejichž pomocí můžeme detekovat mimořádné (chybové) situace, v nichž se náš program může ocitnout. Práce s chybovými výjimkami splňuje kritéria dané objektovou filosofií a kromě toho je příjemnější než dřívější technika, jejíž smysl spočíval v kontrole návratových hodnot funkcí. Dlouze bychom mohli rozebírat další inovace jazyka C++, k nimž patří přetěžování funkcí a operátorů, vestavěný typ **string**, lépe navržené konstrukce pro dynamickou alokaci a dealokaci paměťových bloků či rozsáhlá standardní knihovna a standardní knihovna šablon (Standard Template Library, STL).

- **C++ se snaží zachovat kompatibilitu s jazykem C.** Zde bychom rádi poznamenali, že tato aktivita je smysluplná pouze tam, kde je zachování kompatibility ku prospěchu věci. Kupříkladu, zdrojový kód se v jazyce C++ píše podobně, jako v jazyce C. Příkazy jsou ukončovány středníkem, při definici proměnných se nejprve uvádí datový typ a pak identifikátor proměnné, rovněž v C++ máme rozhodovací příkazy **if** a **switch**, stejně jako cykly **for**, **while** a **do-while**. Ovšem někdy je nutno staré zbraně odložit a zapojit do hry nové „bouchačky“. Vezměme si například dynamickou alokaci a dealokaci paměti. Třebaže i v jazyce C++ smíme požádat o pomoc funkce **malloc** (respektive **calloc**) a **free**, daleko lepší alternativu přináší dvojice operátorů **new** a **delete**.

- **C++ je jazykem střední úrovně s přímým přístupem k operační paměti.** Tuto vlastnost si jazyk C++ půjčil od svého staršího sourozence a my jsme jenom rádi, že je tomu tak. Je hezké vidět, jak se z výšin někdy výsostně abstraktního objektového programování mohou vývojáři s vábivou lehkostí snést až k poslednímu paměťovému bajtu. Znáte-li z jazyka C ukazatele, pak v C++ si s nimi můžete pohrát do sytosti. Mimochodem, s ukazateli se můžete setkat rovněž v jazyce C#, ovšem pouze v blocích nebezpečného kódu, jenž je vyznačen klíčovým slovem **unsafe**. Ovšem pozor! Pokud nejste zkušenými programátory v C#, raději se těmto praktikám vyhněte. Míchání řízeného a nativního kódu nemusí totiž dělat dobrotu.

> Naopak, víte-li, jak na to, můžete svůj program do velké míry optimalizovat. Jednou jsme společně se studenty pracovali na aplikaci, která prováděla dvě barevné transformace s bitovými mapami. Šlo o inverzi obrázku a jeho převod do šedotónu. Když jsme volali vestavěné metody grafických objektů, zpracování trvalo velmi dlouho – při použití obrázků o větších rozměrech jsme se dostali k několika desítkám sekund. Poté jsme ale zapracovali explicitní přístup do paměti, v níž byla grafická data umístěna. Čas potřebný pro zpracování se rázem scvrknul na několik desetin vteřiny. A to už stojí za trochu nízkoúrovňového programování, není-liž pravda?

Na přelomu století začala společnost Microsoft prvýkrát přesazovat svou vizi počítačového softwaru nové generace. Hlavním motivem se přitom měli stát webové aplikace a XML webové služby pracující v distribuovaném ekosystému celosvětové sítě Internet. Vize byla opatřena strategií s přívlastkem .NET. Microsoft připravil novou sadu pracovního náčiní, která se k vývojářům dostala v roce 2002. A copak že tato sada obsahovala? Inu, všechny důležité ingredience, od Visual Studia .NET, přes vývojově-exekuční platformu .NET Framework až po zcela nový programovací jazyk C#, jenž byl projektován speciálně pro potřeby vývoje aplikací .NET. Implementace strategie .NET byla natolik hluboká, že s její realizací jsme začali dění v IT průmyslu dělit na „před .NET éru“ a „po .NET éru“.

Vyjma jazyka C# poskytl Microsoft vývojářům také další prostředky pro vývoj aplikací určených pro .NET Framework. Jedním z nich byl Visual Basic .NET, jehož jazyková specifikace byla v zásadní míře pozměněna. Inovace však Visual Basicu jasně prospěly, neboť se z něj stal opravdový programovací jazyk s plnou podporou objektově orientovaného programování. Připomeňme, že dřívější verze Visual Basicu (zejména oblíbená 6.0, často familiárně označována jako „šestka“) si s OOP až tak dobře nerozuměla. Změny, které zasáhly Visual Basic, způsobily u příznivců tohoto jazyka nemalé zemětřesení. Někteří se dokonce nechali slyšet, že Visual Basic .NET je tak jiný, že už to ani není Visual Basic. Na druhou stranu ti, kdo se s novým Visual Basicem lépe seznámili, již nechtěli pracovat nikde jinde, jenom v „.NET světě“.

Nás ovšem zajímají vývojářské produkty postavené na C a C++. V tomto směru se centrem našeho zkoumání stane software Visual C++ .NET, který byl součástí Visual Studia .NET. Visual C++.NET byl kouzelný software, poněvadž v sobě integroval až tři kompilátory. Jeden pro jazyk C (implementován dle standardu C90), další pro jazyk C++ (zde byl uplatněn standard C++98) a třetí… Nuže, třetí kompilátor byl připraven zpracovávat zdrojový kód

nově uváděného programovacího jazyka s názvem C++ s Managed Extensions (v originále název jazyka zní Managed Extensions for C++).

Jak vlastně tento nový jazyk vznikl a k čemu byl určen? To jsou dvě dobré otázky, na které se pokusíme ihned odpovědět. Tak za prvé, když společnost Microsoft uvedla na softwarový trh platformu .NET Framework, vznikl nový druh aplikací, které ke svému běhu tuto platformu vyžadovaly. Pro tyto aplikace se vžilo označení řízené aplikace nebo též aplikace .NET. Cílem tvůrců Visual Studia .NET bylo, aby mohli aplikace .NET vytvářet všichni programátoři, bez ohledu na použitý programovací jazyk. Zbrusu nový jazyk, „ostré céčko" čili C#, tento požadavek splňoval. Stejně se choval i Visual Basic .NET. Jenom vývojáři v jazyce C++ neměli nástroj, jehož prostřednictvím by dokázali vyvíjet řízené aplikace. Proto si mládenci v technických týmech pro Visual C++ umanuli, že rozšíří původní vzezření jazyka C++ tak, aby z něj vytesali plnohodnotný prostředek pro vytváření aplikací .NET. A jak pravili, tak také udělali. Jazyková specifikace doznala rázných úprav, přibylo množství nových klíčových slov a celková podoba jazyka byla přizpůsobena tak, aby vyhovovala koncepci platformy .NET Framework.

Když komerční programátoři pohlédli na hotové dílo redmonštích tvůrců, bylo jim ihned jasné, že přidaných rozšíření je ohromná spousta. Zapracované modifikace měnily původně nativní C++ v takovém měřítku, že tvůrci softwaru začali o řízeném C++ mluvit jako o zcela novém programovacím jazyce.

S vynalezením jazyka C++ s Managed Extensions se společnosti Microsoft povedlo účinně propojit dva softwarové světy. Na jedné straně stál svět nativních aplikací, které byly napsány v jazycích C a C++. Na straně druhé pak trůnil svět řízených aplikací (tedy aplikací pro .NET Framework), které bylo možné vytvářet za asistence nového jazyka C++ s Managed Extensions. O platformě .NET Framework budeme mluvit podrobněji dále v této části knihy, ale již nyní se můžeme podívat na analýzu silných a slabých stránek jazyka C++ s Managed Extensions.

Řízené C++ se mohlo pyšnit následujícími pozitivy:

- **C++ s Managed Extensions se pro vývojáře v jazycích C a C++ stalo hlavním jazykem pro vývoj aplikací .NET.** Tato zřejmě nejpatrnější výhoda byla brána v potaz již při projektování programovacího jazyka. V rámci své segmentační strategie se mělo řízené C++ stát logickým následníkem jazyka C++. V souvislosti s příchodem řízeného C++ se původní C++ začalo označovat jako nativní, v zájmu

zřetelného vymezení vývojářských prostředků pro zmíněné dva softwarové světy (nativní vs. řízený).

- **C++ s Managed Extensions byl jediný prostředek, jenž umožňoval míchání programového kódu.** Při programování v řízeném C++ jsme mohli mixovat zdrojový kód jazyků C++ a C++ s Managed Extensions. Bylo tedy možné programovat na rozhraní obou softwarových světů, což se osvědčilo zejména při velice úzké kooperaci nativních a řízených fragmentů kódu. Jazyk C++ s Managed Extensions tenhle zázrak zpřístupňoval díky technologiím C++ Interop a IJW (It Just Works). Vzpomenuté technologie byly k nezaplacení rovněž při transportu nativní aplikace napsané v jazyce C++ do prostředí .NET. Jako příklad uveďme přenesení hry Quake II od společnosti id Software na platformu .NET Framework. Mimochodem, znáte tuhle počítačovou hru? Domníváme se, že ano, vždyť svého času šlo o populární akční střílečku s působivou 3D akcelerovanou počítačovou grafikou, v níž hlavní hrdina bojuje proti vesmírným nestvůrám.

  Zvítězit nad tímto na první pohled nesnadným úkolem se podejmuli zaměstnanci Vertigo Software, americké softwarové společnosti. Jejich společné úsilí poháněla snaha zaměřená na demonstraci možností jazyka C++ s Managed Extensions. Celý proces konverze zabral takřka týden. Tento časový interval byl rozdělen na více etap, protože nejdřív bylo zapotřebí upravit původní kód hry, jenž byl napsán v jazyce C. Jakmile byla konverze z C do C++ dokončena, zahájili vývojáři konverzi z nativního C++ do řízeného C++. Přitom do hry doprogramovali nový radar, jenž zobrazoval rozvržení předmětů v úrovni. Práce byly úspěšně dokončeny a k radosti všech zúčastněných hra s názvem Quake II .NET běžela jako po másle.

  Konečně, na webové stránce http://www.vertigosoftware.com/Quake2.htm, se můžete přesvědčit sami.

- **C++ s Managed Extensions umožňovalo bezproblémovou interoperabilitu s dalšími .NET-kompatibilními jazyky.** Technologický rámec v pozadí .NET Frameworku dovoloval, aby programy napsané v jazycích Visual Basic .NET a C# vedly informační dialog s aplikací, jež byla vytvořena v C++ s Managed Extensions. Proces fungoval také opačně, čehož důkazem byl například tento scénář: Vývojář v řízeném C++ napsal třídu, která zapouzdřovala funkcionalitu pro posílání datových paketů přes síť. Tuto třídu mohl do svého projektu začlenit jak programátor v jazyce Visual Basic .NET, tak tvůrce softwaru v C#. Ba co víc, nejenomže mohli vývojáři

v jiných jazycích třídu napsanou v řízeném C++ použít, oni byli dokonce schopni odvodit od této třídy podtřídu a tu v případě potřeby dále rozšiřovat.

- **Jazyk C++ s Managed Extensions byl otevřený vůči COM komponentám a funkcím nativního rozhraní Win32 API.** Řízené C++ mělo více kamarádů, nebyl to jenom Visual Basic .NET a C#. Na opačném spektru postávaly řady COM komponent napsaných v jazyce C++. A opodál se zase utábořily regimenty funkcí rozhraní Win32 API, jež tvoří základní jádro operačního systému Microsoft Windows. Spolupráce se součástkami nativního kódu je velice důležitá, o to víc, když si uvědomíme, kolik milionů řádků zdrojového kódu jazyků C a C++ na světě existuje.

Když začnete mluvit s vývojáři, kteří v jazyce C++ s Managed Extensions aktivně programovali, je více než pravděpodobné, že se vám postěžují na kostrbatou syntaxi a nevábně vyhlížející klíčová slova. Jejich výtky jsou oprávněné: syntaktická výbava řízeného C++ opravdu nebyla takovým skvostem, jakým by mohla být. Mnoho klíčových slov se nevyhnulo dvojitým symbolům podtržení (typicky **__gc class**, **__gc new** apod.) a některé příkazy byly tak spletité, že jejich luštění zabralo pěknou porci času. Dále budou vývojáři patrně argumentovat poklesem výkonu, jímž aplikace .NET trpí ve srovnání s programy napsanými v „čistém C nebo C++“. Jak se dozvíte dále v textu, řízené aplikace nebudou nikdy stejně rychlé jako jejich nativní protějšky. To je prostý fakt, jenž plyne z charakteru práce virtuálního exekučního systému CLR platformy .NET Framework. Pokud se ovšem výkonnostní penalizace pohybuje v rozmezí 5 - 15 %, v praxi nepředstavuje nijak významné omezení. Vraťme se k příkladu s portováním hry Quake II do prostředí .NET. V reálných testech běžela konvertovaná aplikace asi o 15 % pomaleji, než původní hra napsaná v jazyce C. To znamená, že když nativní Quake II běžel rychlostí 50 snímků za sekundu, tak řízený ekvivalent dosahoval rychlost 42,5 snímků za sekundu.

Koncem roku 2005 uvedla společnost Microsoft novou bázi pro své vývojářské náčiní. Vlajkovou lodí se staly produkty Visual Studio 2005 a SQL Server 2005 s desítkami inovací, které v mnoha směrech těžily z nabušené platformy Microsoft .NET Framework 2.0. No snad největší novinkou bylo představení nového programovacího jazyka s názvem C++/CLI (akronym CLI znamená Common Language Infrastructure, tedy společnou jazykovou infrastrukturu). Jazyk C++/CLI se stal nástupcem dřívějšího C++ s Managed Extensions.

Softwaroví architekti jazyka C++/CLI chtěli odstranit slabá místa, s nimiž se potýkalo C++ s Managed Extensions.

Při návrhu nového řízeného C++ byly proto brány v potaz tyto požadavky:

- **C++/CLI musí disponovat přehlednější a intuitivnější syntaxí.** Konečně došlo k likvidaci nekonformních klíčových slov, sjednotilo se názvosloví a pročistily se příkazy. Díky omlazovacímu procesu byl zdrojový kód jazyka C++/CLI lépe čitelný, a tudíž snáze srozumitelný. To rozjařilo tváře C++ programátorů, kteří se chystali přejít do řízeného světa platformy Microsoft .NET Framework 2.0.

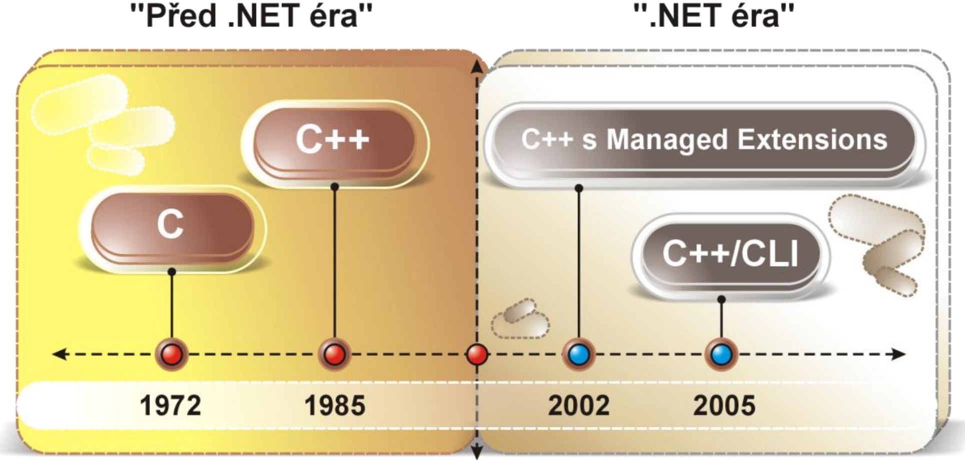
- **C++/CLI přinese prvotřídní podporu programových rysů infrastruktury CLI.** CLI je společná jazykový infrastruktura, která přesně charakterizuje všechny součástí řízeného prostředí, v němž probíhá vývoj a běh aplikací .NET. Jak se dozvíte později, CLI uvádí do světa vývoje softwaru vskutku revoluční prvky, jako je kupříkladu společný typový systém CTS (Common Type System), společnou jazykovou specifikaci CLS (Common Language Specification) a virtuální exekuční systém CLR (Common Language Runtime). Pro vývojáře je podstatné to, že programovací jazyk C++/CLI je s infrastrukturou CLI úzce propojen, takže dovede využívat všechny její přednosti.

- **C++/CLI převezme všechno dobré z nativního C++.** Za více než dvacet let svého praktického působení prokázal jazyk C++ své kvality a do povědomí vývojářů se zapsal jako produktivní programovací jazyk. A ačkoliv se konstrukční práce na jazyce C++/CLI neobešly bez úprav stávající specifikace C++, tento jazyk pořád disponuje mnoha zbraněmi těžkého kalibru, se kterými se programátoři jenom neradi loučí. Zkrátka a jasně, C++ umí přehršle zajímavých a užitečných věcí, a jak usoudili architekti v Microsoftu, jejich absence v novém C++/CLI by mohla být v odborných kruzích považována za šlápnutí vedle. Do jazyka C++/CLI si tak našly cestu šablony, jež v jazyce C++ představovaly modlu generického programování. Kromě toho se pozměnil vztah mezi destruktory a finalizéry, takže nyní lze zdroje asociované s objekty uvolňovat deterministicky.

- **C++/CLI umožní rychlou portaci nativních programů do řízeného prostředí platformy .NET Framework.** Vývojářům se čas od času postaví do cesty program, který by bylo záhodno přenést do prostředí .NET. Potíž je v tom, že dotyčný program je napsaný v jazyce C nebo C++. V závislosti na množství kódu a jeho složitosti se pak vytyčují scénáře transportu aplikace. Jazyk C++/CLI umožňuje míchání nativního a řízeného kódu, což je jistě pozoruhodná vlastnost, ovšem někdy bychom rádi aplikaci převedli, aniž bychom se pouštěli do nějakých velkých konverzních akcí.

Dobrou zprávou je, že kompilátor řízeného C++ disponuje přepínačem **/clr**, jehož zapnutí způsobí aktivaci všech řízených rozšíření se zachováním zpětné kompatibility s jazyky C a C++. Ještě lepší zprávou je, že přepínač **/clr** je konfigurovatelný, a tudíž vývojáři mohou sami určit, jak velmi „řízenou" chtějí svoji aplikaci mít.

Obr. 1.1 zachycuje vývojovou genezi jazyků C, C++, C++ s Managed Extensions a C++/CLI.



Obr. 1.1: Od nativního C až k řízenému C++/CLI

Koncem roku 2007 uvedla společnost Microsoft doposavad nejaktuálnější verzi svého integrovaného vývojového prostředí s názvem Visual Studio 2008, a to společně s platformou .NET Framework 3.5. Vedle jazyků Visual Basic 2008 a C# 3.0 se v této kolekci nachází také jazyk C++/CLI, jenž je implementován v produktech Visual C++ 2008 a Visual C++ 2008 Express.

A co nás čeká v budoucnosti? Zřejmě v roce 2009 nebo 2010 bude uvedeno Visual Studio 2010 s Visual C++ 2010 uvnitř. Podpora jazyka C++/CLI trvá i nadále, což znamená, že všechny své nabyté znalosti můžete okamžitě uplatnit i v tomto vývojovém prostředí.

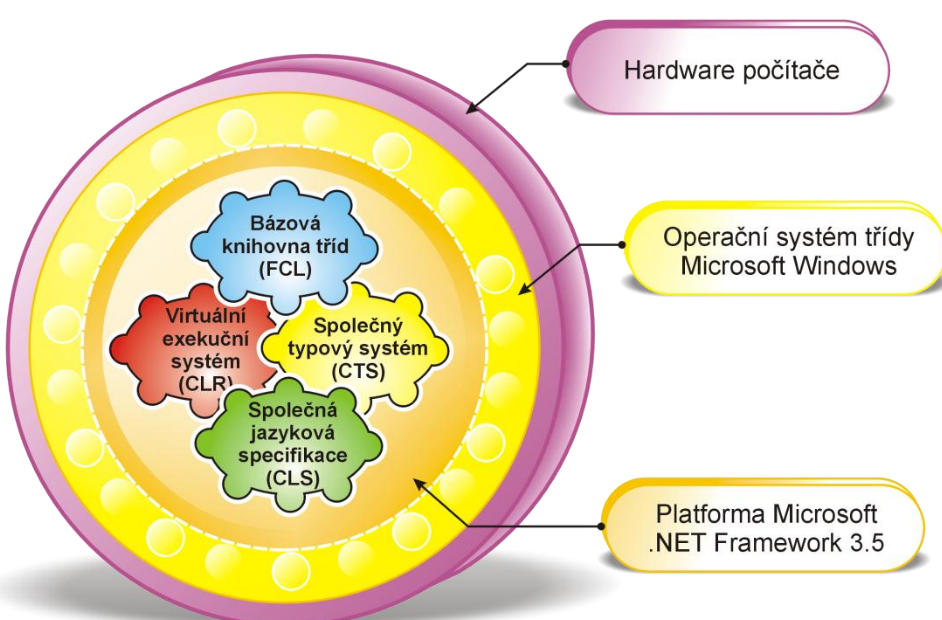
Jelikož vás nechceme připravit o pochopení důležitých souvislostí, které pro vývojáře v jazycích C a C++ mělo uvedení prostředí .NET, uděláme na chvíli malou odbočku a podrobněji si přiblížíme platformu .NET Framework 3.5 a běh aplikací .NET.

1.2 Platforma .NET Framework 3.5: prostředí pro vytváření řízených aplikací

Microsoft .NET Framework 3.5 je název pro platformu, která vymezuje infrastrukturu definující nástroje pro vývoj a běh aplikací .NET. Pro další rozpravu přijměme dohovor, že pod pojmem „aplikace .NET" budeme chápat počítačový program, jenž bude vyhovovat standardům daných prostředím Microsoft .NET Framework 3.5.

Základní stavební bloky architektury platformy .NET Framework 3.5 tvoří čtyři komponenty:

1. Bázová knihovna tříd FCL (Framework Class Library).
2. Virtuální exekuční systém CLR (Common Language Runtime).
3. Společný typový systém CTS (Common Type System).
4. Společná jazyková specifikace CLS (Common Language Specification).



Obr. 1.2: Soukolí reprezentující základní pilíře platformy .NET Framework 3.5

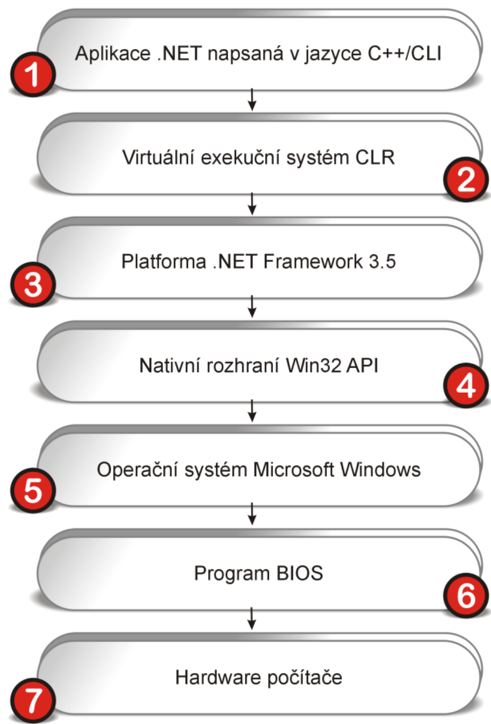
Bázová knihovna tříd je studnicí několika tisíc předem připravených tříd, struktur, delegátů, rozhraní a dalších softwarových entit. Výborné je, že FCL integruje ohromnou porci

intelektuálního vlastnictví inženýrů společnosti Microsoft, které je vám kdykoliv k dispozici. Zabudované know-how má přitom nebývale širokou působnost, poněvadž se věnuje snad všem myslitelným oblastem vývoje softwaru. Na dosah ruky tak máte třídy zapouzdřující funkcionalitu pro práci s datovými typy a strukturami, kolekcemi, databázemi, bitovými mapami, šifrovacími algoritmy atd. Bázová knihovna tříd je vítaným společníkem, jenž vás oprošťuje od nutnosti vyvíjet to, co již bylo dávno vytvořeno. Když se tak nad tím zamyslíme, budování aplikací se podobá hře s kostkami stavebnice Lego. Jednoduše uchopíme součástku, připojíme ji k jiné a tak pokračujeme dál, až dokud nemáme před sebou líbivý model závodního auta nebo propracovaný náklaďák.

Virtuální exekuční systém CLR je zodpovědný za správné spouštění a běh aplikací .NET. Vzájemná relace mezi systémem CLR a aplikací .NET je natolik silná, že aplikace nemůže bez CLR vůbec existovat. Jistě se nyní ptáte, proč je tomu tak. Odpovědí na tuto otázku je skutečnost, že prostředí CLR řídí běh aplikací .NET. Jelikož jsou všechny aplikace .NET řízeny systémem CLR, označují se adjektivem řízené. Naproti tomu, jako neřízené aplikace se ponímají programy, které ke svému běhu prostředí CLR nevyžadují. Mezi typické zástupce neřízených aplikací patří programy, jež byly napsány v nativních jazycích, k nimž patří C, C++, Visual Basic 6.0, Visual J++ 6.0 a jiné.

V této knize se naučíte, jak psát v jazyce C++/CLI programy pro platformu .NET Framework 3.5. Když vytvoříte svou vlastní aplikaci, jistě vás bude zajímat, zda ji můžete spustit i na jiných počítačích (například u kolegy v práci, nebo na školním PC). Jelikož jsou řízené aplikace úzce propojeny s platformou .NET Framework 3.5, je nutné, aby byla tato platforma na cílové počítačové stanici nainstalována. Úplný instalační balíček platformy .NET Framework 3.5 má přibližně 200 megabajtů. Je však docela dobře možné, že .NET Framework 3.5 nebudete muset sami instalovat, neboť na cílovém PC již bude přítomen.

Ještě než přejdeme k ozřejmění procesu řízené exekuce aplikací .NET, chtěli bychom vás seznámit se vztahem, jenž panuje mezi aplikací .NET, prostředím CLR, platformou .NET Framework 3.5 a operačními systémy Microsoft Windows. Tato relace je zachycena na obr. 1.3.



Obr. 1.3: Ukázka komunikačního modelu mezi aplikací .NET, prostředím CLR, operačním systémem Windows a hardwarem počítače

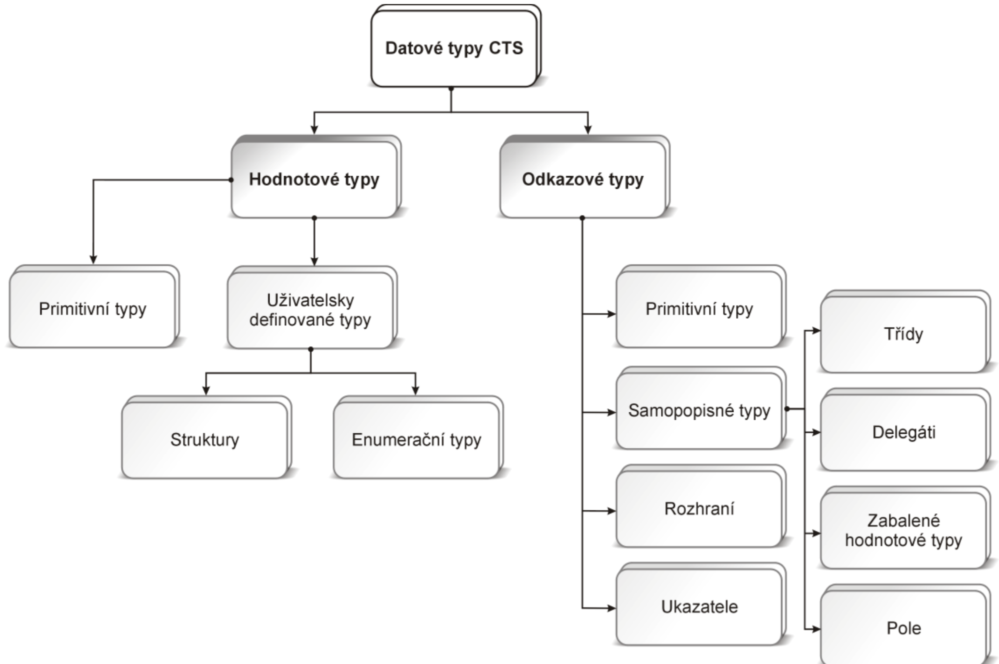
1.3 Společný typový systém (CTS) a společná jazyková specifikace (CLS)

Bezproblémové interoperability mezi softwarovými fragmenty různých jazyků by nebylo dosaženo bez společného typového systému (CTS). Společný typový systém zabezpečuje, že datové typy budou mít v jazyce MSIL stejnou interní reprezentaci, bez ohledu na použitý programovací jazyk vyšší úrovně. To je velký krok kupředu, který nám konečně dovoluje zapomenout na svízelné otázky typu: „Proč v jazyce Visual Basic alokuje proměnná celočíselného typu **Integer** 2 bajty, když stejná proměnná typu **int** jazyka C zabírá 4 bajty?“, anebo „Jak vyjádřit typ **String** Visual Basicu v jazyce C++? Jako **char***, **std::string**, nebo snad **wchar_t***?“.

Společný typový systém definuje požadavky, jež musejí splnit všechny datové typy, s nimiž programátoři na platformě .NET Framework 3.5 pracují. CTS dělí všechny typy na dvě základní skupiny:

- Hodnotové datové typy.
- Odkazové datové typy.

Každá z uvedených skupin datových typů je tvořena jednak primitivními typy (tedy typy, které jsou již zabudovány do CTS a kompilátory .NET-kompatibilních programovacích jazyků jsou schopny s těmito typy ihned pracovat) a rovněž tak i uživatelsky definovanými typy (to jsou nové typy, které navrhuje samotný programátor). O datových typech společného typového systému si více povíme v druhé části této knihy, v níž společně absolvujeme základní kurz programování v jazyce C++/CLI. Již nyní si však můžeme dopřát malou ochutnávku v podobě dekompozice společného typového systému (obr. 1.4).



Obr. 1.4: Klasifikace datových typů společného typového systému CTS

Společný typový systém CTS je zárukou toho, že aplikace napsané v různých jazycích platformy .NET Framework 3.5 mohou mezi sebou vést informační dialog. Jenomže vedle datových typů je nutno standardizovat rovněž programovací jazyky a jejich kompilátory. Odborníci společnosti Microsoft proto zpracovali kolekci základních požadavků, které musí splňovat jakýkoliv programovací jazyk, jehož pomocí bude možné vytvářet aplikace pro .NET Framework 3.5. Kolekce dostala název společná jazyková specifikace CLS. Jestliže jistý programovací jazyk vyhovuje nárokům charakterizovaných v CLS, pak může být opatřen pečetí „Toto je .NET-kompatibilní programovací jazyk“.

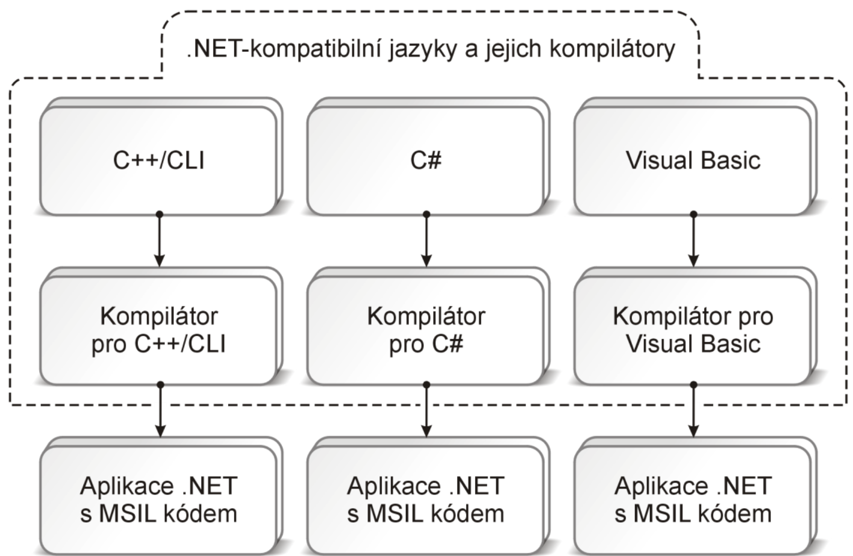
Programátoři pracující v takovémto jazyce mají zaručeno, že specifikace jazyka a jeho syntakticko-sémantická struktura jim umožní využívat všech výhod řízeného prostředí. Rovněž se mohou spolehnout, že kompilátor dotyčného jazyka bude schopen převést zdrojový kód jazyka do instrukcí mezijazyka MSIL. Společné jazykové specifikaci vyhovují jazyky Visual Basic 2008, C# 3.0, C++/CLI, J# a více než čtyřicet dalších jazyků, které byly přeneseny na platformu .NET.

1.4 Řízená exekuce aplikací, mezijazyk MSIL a Just-In-Time kompilace

Virtuální exekuční systém je aktivován pokaždé, když dojde ke spuštění aplikace .NET (iniciátorem spuštění může být buď uživatel, nebo jiná aplikace). Po svém startu a inicializaci začne prostředí CLR zpracovávat kód aplikace .NET. Na rozdíl od nativních aplikací, řízené protějšky neobsahují instrukce strojového kódu, které by mohly být přímo zpracovány instrukční sadou procesoru. Místo toho leží uvnitř aplikace .NET fragmenty mezijazyka, jenž je znám jako Microsoft Intermediate Language (zkráceně MSIL nebo jen IL, případně CIL – Common Intermediate Language). MSIL je speciální jazyk, který byl navržen kvůli potřebám infrastruktury CLI. Máme-li před sebou řízenou aplikaci, můžeme se spolehnout na to, že obsahuje kód mezijazyka MSIL. Řečeno jinak, zdrojový kód aplikace .NET, který zapíšete v jazyce C++/CLI (anebo také v jazycích C# a Visual Basic 2008), bude kompilátorem převeden do formy MSIL instrukcí. Tyto instrukce pak budou zality do řízeného modulu s MSIL kódem. Řízený modul působí jako jedna ze součástek sestavení aplikace .NET. Výklad na téma „Sestavení řízených aplikací a jeho interní kompozice“ odložíme na jindy, ovšem nemusíte mít strach, dozvíte se vše podstatné.

Nuže, prozatím jsme se dopracovali k řízenému modulu aplikace .NET s kódem jazyka MSIL. Rovněž jsme si řekli, že kompilátory .NET-kompatibilních programovacích jazyků generují ze zdrojového kódu příslušného jazyka ekvivalentní MSIL kód. K čemu je to vůbec dobré?

Důvodů je samozřejmě více. Začněme však garancí typové nezávislosti a softwarové interoperability. Poněvadž MSIL působí jako univerzální jazyk platformy .NET Framework 3.5, můžeme veškerý řízený kód (bez ohledu na jazyk, jenž byl použit k jeho napsání) vyjádřit sérií instrukcí tohoto mezijazyka. Ve skutečnosti je MSIL objektově orientovaným jazykem nízké úrovně a pokud byste měli chuť, mohli byste aplikace .NET psát přímo v něm. MSIL je často označován jako jazyk symbolických instrukcí platformy .NET, čímž je poukázáno na velice přátelský vztah mezi ním a jazyky vyšší úrovně (C++/CLI, C#, Visual Basic).



Obr. 1.5: Proces vytváření aplikací .NET v jazycích C++/CLI, C# a Visual Basic

Ačkoliv veškerý řízený kód jazyků vyšší úrovně může být bez potíží zapsán v MSIL, naopak to neplatí. MSIL totiž obsahuje též takové syntaktické konstrukce, které nejsou dostupné v jazycích vyšší úrovně.

Jazyk MSIL vytváří účinnou vrstvu, díky které je možné, aby mezi sebou mohli promlouvat aplikace napsané v různých .NET-kompatibilních programovacích jazycích. Spolupráce aplikací neboli interoperabilita je vysoce ceněná, s čím budou souhlasit zejména vývojáři, jež měli tu čest pracovat s technologiemi předešlých generací. Jak byste vyřešili kooperaci dynamické knihovny s kolekcí exportovaných funkcí a aplikace napsané ve Visual Basicu verze 5.0? Pokud říkáte, že se snažíme přenést o více než deset let zpátky, pak máte samozřejmě pravdu, no takovéto situace bývaly skutečně docela běžné.

Poznámka: Ano, s takovými potížemi jsme se potýkali docela často, zejména proto, že realizace některých výpočetních operací byla v dřívějším Visual Basicu docela pomalá. Kritické rutiny se proto psaly jako funkce v jazycích C/C++ a posléze se exportovaly do knihoven DLL. Bohužel, v dávných dobách nebyl žádný společný mezijazyk jako MSIL. Také jsme neměli společný typový systém, a tak jsme se věčně prali s chybami, jež vyplývaly z nesouladu rozsahů datových typů.

Dobrá, předpokládejme nyní, že jsme spustili aplikaci .NET. Již víte, že někde na pozadí musí běžet prostředí CLR, které řídí životní cyklus aplikace .NET. Stejně tak je vám známo, že aplikace .NET obsahuje řízený kód jazyka MSIL. Když se zamyslíme nad dalším postupem, rychle dojdeme k poznání, že potřebujeme zpracovat instrukce jazyka MSIL. Bohužel, žádný ze současných procesorů není opatřen takovou instrukční sadou, která by si dovedla poradit s instrukcemi zapsanými v jazyce MSIL. Vesměs všechny procesory, až už jedno- nebo vícejádrové, jsou uzpůsobeny pouze pro exekuci strojového kódu. A tak se pomalu dostáváme k zajímavé situaci. Na jedné straně máme aplikační kód vyjádřený v jazyce MSIL, zatímco na straně druhé si tiše mumlá 32 nebo 64bitový procesor, jenž čeká na dodávku vydatné porce strojových instrukcí. Co s tím? Nemusíme být zrovna kouzelníci, abychom tuto zápletku rozřešili. Jednoduše potřebujeme nějaký mechanismus, který nám na požádání dovede přeložit kód jazyka MSIL do formy strojových instrukcí. Ano, to je ono! Tento mechanismus pohání stroj, jemuž se říká Just-In-Time (JIT) kompilátor.

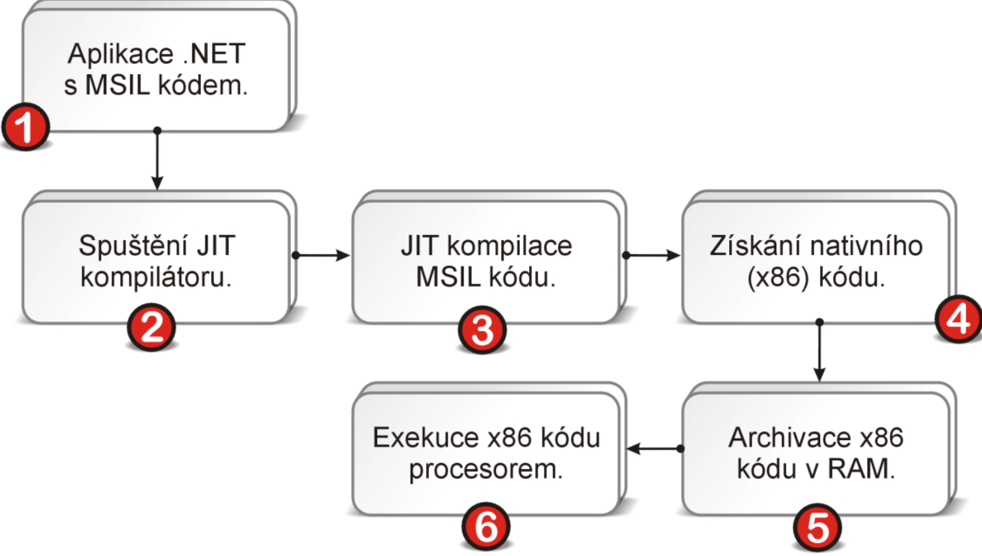
Poznámka: Koncepce Just-In-Time systému je známá z prostředí logistiky, kde jde o jednu z metod optimalizovaného zásobování jistého podniku (zpravidla strojírenského). JIT zásobování se dennodenně využívá třeba v závodech vyrábějících automobily. Cílem této metody je minimalizovat zásoby komponent, dílů a dalších součástek na skladu. Pracovníci při montážních linkách se spíše spoléhají na to, že manažeři logistiky zabezpečí potřebné dodávky vždy, když to bude zapotřebí. Filosofie „dodání všeho potřebného na požádání“ se uplatňuje s úspěchem i v dalších oborech lidské činnosti – a jak zjišťujeme, tak informatika není výjimkou.

JIT kompilátor má jedinou, o to však důležitější úlohu: vždy včas a v nejvyšší možné kvalitě převést MSIL kód na ekvivalentní strojový kód. Když se spustí aplikace .NET, nejprve je načten její kód do paměti, a poté je vyhledán její vstupní bod. Vstupním bodem je funkce, která je zpracována jako první. V jazycích C/C++ je takovou funkcí **main**, v C# **Main** a ve Visual Basicu **Sub Main**. Veškerý kód vstupní funkce je uložen jako MSIL, takže přichází čas pro JIT kompilátor. Když je JITter (jak je volán JIT kompilátor v kruhu přátel) povolán, zabezpečí transformaci kódu. MSIL instrukce se rázem mění na ryzí strojový kód, který je v další etapě postoupen procesoru ke zpracování. Pokud jsou volány další funkce nebo

metody, JITter na požádání kompiluje také jejich kód. V této souvislosti vás musíme obeznámit se dvěma důležitými aspekty práce JIT kompilátoru.

Za prvé, JITter překládá pouze ty metody, které jsou opravdu volány. Jak uživatel pracuje s aplikací .NET, dochází k volání metod, jež reagují na uživatelovi pokyny. Dnešní aplikace bývají nezřídka velice rozsáhlé, čili sdružují stovky ne-li tisícky rozmanitých metod. Zacházení s aplikací se může napříč uživateli lišit, dokonce „průchod" aplikací se různí i mezi dvěma relacemi, jež byly iniciovány jedním a tímtéž uživatelem. Přitom platí prosté pravidlo: kdykoliv je to potřebné, zasáhne JIT kompilátor a uskuteční překlad kódu.

Za druhé, ve chvíli, kdy JIT kompilátor dokončí překlad jisté metody, její přeloženou verzi (tedy tu se strojovým kódem) uloží v operační paměti pro budoucí použití. To se může hodit, vždyť co když uživatel vyvolá tutéž metodu dva nebo i vícekrát po sobě? Povězme například, že uživatel bude chtít vytisknout jednu kopii dokumentu, no později zjistí, že potřebuje další kopii. Pokud má vytištění dokumentu ve své kompetenci metoda s názvem **TisknoutDokument**, pak její MSIL kód přeloží JITter pouze jednou, při realizaci první tiskové operace. Podruhé již není kód metody znovu překládán, nýbrž je okamžitě použita uschovaná přeložená verze této metody. Tím JIT kompilátor maximalizuje svou pracovní produktivitu a minimalizuje náklady spojené s generováním postranní režie.



Obr. 1.6: Práce Just-In-Time (JIT) kompilátoru

Když autor této publikace přednáší studentům o JIT kompilátoru, zpravidla se někdo z přítomných zeptá zajímavou otázku: „A co když nebude možné jednou přeloženou verzi metody uložit?“, případně „A co když nebude možné přeloženou a uloženou verzi metody znovu použít?“. Obě míří do černého, podle čehož je zřejmé, že studenti o problematice přemýšlejí (takoví si obvykle vedou lépe než jejich méně aktivní kolegové). V praxi se nestává, že by nebylo kde uložit nativní kód přeložené metody. Pro uskladnění nativních obrazů původně MSIL metod alokuje prostředí CLR dynamickou paměť. CLR přitom ví, kolik paměti je nutno přidělit, takže zde problémy nenastávají. Obdobně bychom se mohli vyjádřit také k druhé otázce, máme-li k dispozici přeloženou verzi metody, pak ji lze bez okolků aktivovat. Pro úplnost ovšem musíme dodat, že mohou nastat situace, a to obzvláště při kritickém využití paměťových zdrojů, kdy CLR usoudí, že uložené nativní obrazy metod uvolní. Je-li tomu tak, pak samozřejmě přicházíme o strojový kód přeložených metod, a pokud budou tyto metody v budoucnu volány, nezbývá nám nic jiného, než požádat o pomoc JIT kompilátor.

Programátory vždycky interesovala rychlost čehokoliv, a totéž platí i pro JIT kompilaci MSIL kódu. Již jsme zmínili, že překlad MSIL instrukcí na požádání není úplně zadarmo. Jisté výkonnostní penalizace se pojí nejenom se samotným procesem překladu MSIL kódu, nýbrž také s přítomností JIT kompilátoru. Někteří vývojáři si JITter pletou s interpretem, což je ale mylný úsudek. Pokud problém zjednodušíme, pak můžeme tvrdit, že ztráta výkonu je způsobena jednak nastartováním a inicializací JIT kompilátoru a jednak časovou prodlevou, která vzniká v důsledku překladu MSIL kódu za běhu aplikace .NET. Jestliže byste rádi omezili tato slabá místa, máme pro vás řešení. Jmenuje se Native Image Generator, nebo zkráceně NGen. Native Image Generator je program, který převede řízený (MSIL) kód aplikace .NET do podoby nativního obrazu, jenž je poskládán pouze z instrukcí strojového kódu. Záleží jenom na vás, kdy se rozhodnete vaši aplikaci přeložit. První variantou je udělat tak poté, co jste aplikaci dokončili a odladili (tedy předtím, než ji budete distribuovat). To se může jevit jako nejschůdnější řešení. Máte pravdu, je to vskutku jednoduché, no má to jeden háček.

JIT kompilátor dovede generovaný strojový kód optimalizovat podle cílové platformy. Jinými slovy, pokud bude JITter konvertovat aplikaci .NET na počítači s procesorem Intel Pentium 4, je více než pravděpodobné, že do finálního přeloženého kódu implementuje optimalizační techniky, které zabezpečí rychlejší exekuci vyprodukovaného strojového kódu. Jestliže použijete utilitu NGen pro vytvoření nativního obrazu před distribucí vaší aplikace, bude JIT kompilátor aktivován na vašem PC a nikoliv na počítači uživatele. Z dosud vyřčeného je evidentní, že vygenerovaný strojový kód bude optimalizován pro instrukční sadu vašeho

CPU. V praxi se proto častěji preferuje zavolání NGenu těsně poté, co byla kýžená aplikace .NET nainstalována na PC cílového uživatele. Je totiž rozumné, když vznikne korektní nativní obraz podle adekvátní hardwarové platformy uživatele.

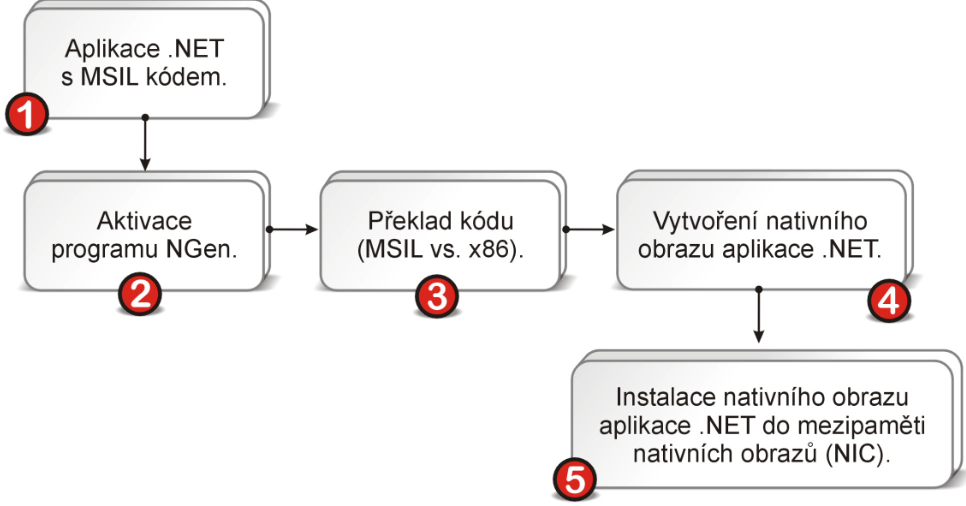
Výhody, které s sebou Native Image Generator přináší, se vypoukleji objevují v následujících dvou oblastech:

1. Maximalizace rychlosti startu aplikace .NET.
2. Eliminace práce, kterou musí JIT kompilátor vynaložit při překladu MSIL kódu.

U přeložené aplikace .NET byste měli zaznamenat citelné zvýšení rychlosti při startu. S větší razancí bude samozřejmě zpracováván rovněž samotný kód, který již není řízený, nýbrž nativní. Odpadá tak nutnost aktivovat JITter.

Poznámka: Najde-li Native Image Generator během své práce s aplikací .NET metody, jež nemohou být přeloženy do strojového kódu, ponechá je v původním stavu (v podobě kódu jazyka MSIL). Bude-li na příkaz uživatele volána právě nepřeložená metoda, virtuální exekuční systém CLR bude nucen aktivovat JITter, aby dotyčnou metodu rychle přeložil.

Nativní obrazy přeložených aplikací .NET jsou samočinně instalovány do mezipaměti nativních obrazů (Native Image Cache, NIC). Tuto skutečnost demonstruje obr. 1.7.



Obr. 1.7: Vysvětlení práce programu NGen (Native Image Generator)

Je-li součástí aplikace .NET soubor s nativním obrazem, prostředí CLR se bude snažit použít nejprve jej. To znamená, že pokud bude k překladu řízené aplikace použit program NGen, bude vytvořen nativní obraz aplikace, přičemž tato nativní forma bude spouštěna jako první vždy, pokud nedojde k nějakým potížím. Někdy se totiž může stát, že nativní obraz se poškodí, anebo se stane neplatným. V těchto případech se CLR vrátí k standardnímu režimu, jenž využívá JIT kompilaci MSIL kódu na požádání.

1.5 Prostředí CLR a služby, které nabízí řízeným aplikacím

Prostředí CLR poskytuje aplikacím .NET několik nízkoúrovňových služeb. Pojďme je nyní společně prozkoumat.

Když dojde ke spuštění aplikace .NET, operační systém pro ni v paměti vyčlení fyzický proces. Fyzický proces je rezervovaná oblast RAM, do které jsou načteny všechny součásti spouštěné aplikace. Soudobé operační systémy pracují v režimu preemptivního multitaskingu, takže umožňují pseudoparalelní nebo paralelní běh více aplikací najednou. Aby mezi jednotlivými běžícími programy nedocházelo k nežádoucím interferencím, systém každou aplikaci umístí do vlastního fyzického procesu. Fyzické procesy můžeme proto považovat za prostředek pro izolaci jedné aplikace od aplikací ostatních. Fyzický proces determinuje logický adresový prostor, jenž může aplikace využívat.

Do fyzického procesu je dále načten virtuální exekuční systém CLR, jenž řídí běh aplikace .NET. V rámci fyzického procesu vytvoří CLR další logický kontejner, do něhož bude načtena kýžená aplikace .NET. Tomuto logickému kontejneru se říká aplikační doména, anebo přesněji primární aplikační doména. Aplikační doména nabízí vyšší úroveň zapouzdření aplikace .NET a někdy se označuje jako logický proces. V rámci jednoho fyzického procesu operačního systému může existovat více logických procesů. To tedy znamená, že pomocí aplikačních domén může v jednom fyzickém procesu běžet více řízených aplikací. Do primární aplikační domény je posléze uloženo primární programové vlákno, na němž bude zpracováván aplikační kód. Programové vlákno představuje jeden exekuční kanál aplikace .NET.

Teorie programování pozná jedno- a vícevláknové aplikace. Pokud není stanoveno jinak, každá aplikace .NET je implicitně jednovláknová, takže existuje pouze jedna cesta pro zpracování instrukcí aplikačního kódu. Model aplikace s jedním vláknem není vždy vyhovující, takže v případě potřeby se přidávají další (takzvaná pracovní nebo také vedlejší)

vlákna, čímž se počet exekučních cest aplikace zvyšuje. Aplikační kód je poté vykonáván na všech vláknech – tím se programu jakoby přidají další ruce, s nimiž může provádět větší počet činností.

Abychom byli upřímní, s jednovláknovými aplikacemi se v běžné programátorské praxi setkáte spíše výjimečně. I průměrný program vykonává několik časově náročných operací, které je v zájmu zvýšení objektivní rychlosti aplikace lepší implementovat na samostatných vláknech. Více vláken se hodí zejména tehdy, pokud požadujeme, aby mohli uživatelé s programem pracovat i během doby, kdy je výpočetní jádro zatíženo realizací jisté náročné operace. Uveďme si malý příklad. Jistě často stahujete soubory z Internetu, nebo posloucháte na svém počítači hudbu. Co byste řekli tomu, kdyby se s download managerem nedalo v průběhu transportu datových paketů souborů vůbec pracovat? Nebo jinak: jaký byste měli pocit, kdyby se váš multimediální přehrávač při interpretaci vaší oblíbené písničky tvářil jako kamenný sloup a nebyl by schopen reagovat na vaše pokyny? Inu, s takovými programy byste pravděpodobně nebyli spokojeni, a my také ne.

Řešením je doplnění jednoho nebo i dalších pracovních vláken, na něž se proporčně přenese celková tíha, s níž se program musí vypořádat.

Uvažujeme-li o vícevláknové aplikaci, tak vás může napadnout otázka, jakže funguje zpracování kódu na jednotlivých vláknech. Jak vůbec s vlákny žonglovat tak, abychom se nezamotali a nezpůsobili zahlcení aplikace i procesoru? Nuže, zde je stručné vysvětlení. Každé vlákno v množině vláken disponuje jistou prioritou – prostředí CLR zná několik prioritních stupňů: od nejnižší, přes standardní, až po vysokou prioritu. Vláknům je na základě stanoveného stupně priority přidělena určitá časová dávka (takzvané časové kvantum), během které procesor realizuje kód dotyčného vlákna. Jestliže vyprší dávka pro jedno vlákno, procesor se přesouvá na další vlákno a zahájí zpracování kódu na tomto vlákně. Popsaný koloběh se opakuje do chvíle, kdy nejsou uspokojeny potřeby všech programových vláken. Poté se procesor vrací zase k prvnímu vláknu a celá situace se opakuje. Vzhledem k tomu, že přiřazená časová kvanta jsou velice malá, z vnějšího pohledu se jeví, jakoby vlákna pracovala současně. Ve skutečnosti procesor probíhá mezi vlákny a vykonává jejich kód.

Samozřejmě, manipulace s vlákny se liší na progresivnějších procesorech, do nichž byly zapracovány technologie pro podporu práce s více vlákny. Známá je kupříkladu technologie Hyper-Threading (HT) od společnosti Intel, jež se poprvé objevila v procesoru Intel Pentium 4. Díky ní operační systém nahlíží na jeden fyzický procesor jako na dva logické procesory.