edice PRŮVODCE

příklady ke stažení na WWW.GRADA.CZ

Pascal

programování pro začátečníky

Miroslav Virius

- Psaní, překlad a ladění programů
- Základní algoritmické konstrukce
- Rozklad programu na moduly
- Lehký úvod do objektového programování
- Úvod do programování grafického uživatelského rozhraní

GRADA

edice PRŮVODCE

Pascal

programování pro začátečníky

Miroslav Virius

Upozornění pro čtenáře a uživatele této knihy
Všechna práva vyhrazena. Žádná část této tištěné či elektronické knihy nesmí být reprodukována a šířena v papírové, elektronické či jiné podobě bez předchozího písemného souhlasu nakladatele. Neoprávněné užití této knihy bude **trestně stíháno**.

Pascal
programování pro začátečníky

Miroslav Virius

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
jako svou 4704. publikaci

Odpovědný redaktor Pavel Němeček
Sazba Tomáš Brejcha
Počet stran 256
První vydání, Praha 2012

© Grada Publishing, a.s., 2012

V knize použité názvy programových produktů, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Vytiskly Tiskárny Havlíčkův Brod, a. s.
Husova ulice 1881, Havlíčkův Brod

ISBN 978-80-247-4116-1 (tištěná verze)
ISBN 978-80-247-7730-6 (elektronická verze ve formátu PDF)
ISBN 978-80-247-7731-3 (elektronická verze ve formátu EPUB)

Obsah

Předmluva ... 9

1.

Než začneme programovat

1.1 Počítač ... 11
- 1.1.1 Operační paměť ... 11
- 1.1.2 Datové typy a proměnné ... 12

1.2 Programy a programovací jazyky ... 13
- 1.2.1 Vyšší programovací jazyky ... 13
- 1.2.2 Operační systém ... 13

1.3 Program a algoritmus ... 14
- 1.3.1 Algoritmus ... 14

1.4 Objekty a třídy ... 19
- 1.4.1 Zapouzdření ... 19
- 1.4.2 Dědění ... 22
- 1.4.3 Polymorfizmus ... 23
- 1.4.4 Dědění versus skládání ... 24

1.5 Náměty pro cvičení ... 24

2.

První příklad

2.1 První program ... 25
- 2.1.1 Píšeme, překládáme a spouštíme program ... 25
- 2.1.2 Když se něco nepodaří ... 26
- 2.1.3 Co jsme naprogramovali ... 29

2.2 První program podruhé ... 32
- 2.2.1 Lazarus ... 32

2.3 Náměty pro cvičení ... 38

3.

Jednoduché příklady

3.1 Druhý program ... 39
- 3.1.1 Dvojnásobek celého čísla ... 39

3.2 Třetí program ... 42
- 3.2.1 Výpočet faktoriálu ... 42
- 3.2.2 Když uživatel zadá nesprávnou hodnotu ... 43
- 3.2.3 Problémy, které zatím nebudeme tak úplně řešit ... 44
- 3.2.4 Složitější podmínky a konstanty ... 46

3.3 Program a podprogramy ... 47
- 3.3.1 Podprogram ... 47
- 3.3.2 Procedura ... 50
- 3.3.3 Když chceme měnit skutečný parametr ... 52

3.4 Knihovna, modul ... 53
- 3.4.1 Používáme knihovny ... 53
- 3.4.2 Dělíme program na moduly ... 54

3.5 Náměty pro cvičení ... 56

4.

Složitější příklady

4.1 Pokusy s textovými řetězci ... 57
4.1.1 Znak a řetězec ... 57
4.1.2 Kolikrát se v řetězci vyskytuje daný znak ... 58
4.1.3 První výskyt znaku v řetězci ... 59
4.1.4 Poslední výskyt znaku v řetězci ... 61
4.2 První seznámení s objekty ... 62
4.2.1 Třída představující nápis ... 62
4.2.2 Několik poznámek k prvnímu objektovému programu ... 65
4.3 Náměty pro cvičení ... 68

5.

Ladění programu

5.1 Testování ... 69
5.2 Obvyklé nástroje pro ladění ... 70
5.2.1 Krokování ... 70
5.2.2 Hodnoty proměnných ... 73
5.2.3 Posloupnost volání ... 76
5.2.4 Ladicí tisky ... 76
5.3 Příklad ... 77
5.3.1 Zadání: převod řetězce na velká písmena ... 77
5.3.2 Test a ladění ... 78

6.

Začínáme naostro

6.1 Jak budeme Pascal popisovat ... 81
6.1.1 Syntaktické diagramy ... 81
6.1.2 Slovní popis ... 82
6.2 Základní součásti programu ... 83
6.2.1 Program se skládá ze znaků a slov ... 83
6.2.2 Program a blok ... 87
6.2.3 Jednotka ... 88
6.2.4 Deklarace ... 91

7.

Základní datové typy

7.1 Čísla a počítání ... 95
7.1.1 Celá čísla ... 95
7.1.2 Reálná čísla ... 100
7.2 Znaky a řetězce ... 104
7.2.1 Datový typ char ... 104
7.2.2 Datový typ string ... 107
7.3 Logické hodnoty ... 111
7.4 Náměty pro cvičení ... 114

8. Příkazy a výrazy

8.1 Příkazy 115
8.1.1 Základní příkazy 115
8.1.2 Cykly 117
8.1.3 Větvení programu (rozhodování) 122
8.1.4 Přenos řízení (skoky) 124
8.1.5 Příkazy, o nichž jsme nehovořili 128
8.2 Výrazy 128
8.2.1 Priorita neboli pořadí operátorů 129
8.2.2 Typ výrazu 130
8.3 Náměty na cvičení 131

9. Některé další datové typy

9.1 Výčtové typy 133
9.2 Intervalové typy 134
9.3 Množiny 136
9.4 Pole 139
9.4.1 Jednorozměrné pole 140
9.4.2 Vícerozměrná pole 147
9.4.3 Dynamická pole 156
9.5 Náměty na cvičení 158

10. Záznamy a objekty

10.1 Záznam 159
10.1.1 Deklarace záznamu 159
10.1.2 Záznam s pevnou částí 159
10.1.3 Záznam s variantní částí 162
10.2 Objektové typy 166
10.2.1 Deklarace objektového typu 166
10.2.2 Metody 170
10.2.3 Složky třídy 171
10.2.4 Alternativní model objektových typů 174
10.3 Náměty na cvičení 175

11. Procedury a funkce

11.1 Deklarace podprogramu 177
11.1.1 Procedura 178
11.1.2 Funkce 178
11.1.3 Předběžná deklarace 179
11.2 Parametry podprogramů 181
11.2.1 Specifikace formálních parametrů 181
11.2.2 Předávání parametrů hodnotou 182
11.2.3 Předávání parametrů odkazem 183

11.2.4 Předávání parametrů konstantou ... 183
11.2.5 Pole proměnné délky jako parametr ... 184
11.3 Bloková struktura programu ... 185
11.3.1 Bloky a jejich hierarchie ... 185
11.4 Rekurze ... 188
11.5 Funkcionální typy ... 190

12.

Komunikace s programem

12.1 Vstupy a výstupy orientované na soubory ... 195
12.1.1 Pojetí souboru v Pascalu ... 195
12.1.2 Operace se souborem ... 196
12.1.3 Nástroje pro práci se soubory ... 197
12.1.4 Základní operace s binárním souborem ... 198
12.1.5 Základní operace s textovým souborem ... 202
12.1.6 Standardní vstup a výstup ... 210
12.1.7 Chyby ... 212
12.2 Parametry programu a proměnné prostředí ... 214
12.2.1 Parametry programu ... 214
12.2.2 Proměnné prostředí ... 222
12.3 Práce s konzolou ... 222
12.3.1 Vlastnosti konzolového okna ... 222
12.3.2 Přímá práce s konzolou a klávesnicí ... 222
12.4 Náměty na cvičení ... 225

13.

Začínáme s okny

13.1 První program s oknem ... 227
13.1.1 Prázdné okno ... 227
13.1.2 Pohled pod pokličku ... 232
13.1.3 Pokusy s komponentami, vlastnostmi a událostmi ... 233
13.1.4 Některé důležité vlastnosti a události ... 237
13.2 Příklad: kalkulačka ... 237
13.2.1 Předběžné úvahy ... 237
13.2.2 Analýza zadání a návrh aplikace ... 238
13.2.3 Grafické uživatelské rozhraní našeho programu ... 239
13.2.4 Obsluha událostí levého panelu ... 241
13.2.5 Obsluha událostí pravého panelu ... 243
13.2.6 Výpočetní stroj ... 245
13.2.7 Okno O programu ... 246
13.2.8 Princip jediné zodpovědnosti ... 247
13.3 Náměty na cvičení ... 248

Rejstřík ... 249

Předmluva

Kniha, kterou držíte v ruce, vám nabízí možnost naučit se programovat v jazyce Pascal. Předpokládá, že jste úplnými začátečníky, a klade si za cíl dovést vás na úroveň mírně pokročilého programátora.

Než začnete

Od čtenáře očekávám, že umí zacházet s počítačem na běžné uživatelské úrovni: umí ho zapnout a vypnout, umí zacházet s textovým editorem, umí instalovat program. Vedle toho budete potřebovat chuť se něco naučit a čas vyzkoušet si vše, s čím se v této knize seznámíte. Existuje totiž jen jediný rozumný způsob, jak se naučit programovat: psát programy. Ještě lepší je vzít programování jako hru a zkoušet si „co to dělá". Každá kapitola končí několika náměty na vlastní pokusy s programováním.

Co v této knize najdete

Tato kniha má třináct kapitol. V první z nich najdeme obecné povídání o tom, co to je počítač, jak vypadá jeho paměť, jak se v ní ukládají data. Postupně v ní dojdeme k pojmu *datový typ*, který je jedním z pilířů programování v jakémkoli jazyce. Vysvětlíme si, co je to algoritmus a jak ho zapisujeme, a poznáme základní pojmy objektově orientovaného programování, jako je třída, instance nebo dědění. Výklad v této kapitole nezávisí na programovacím jazyce.
Ve druhé kapitole napíšeme první program v Pascalu. Ukážeme si, jak vytvořit zdrojový text, jak ho přeložit a jak hotový program spustit. V první části použijeme jen nástroje pracující s příkazovou řádkou, ve druhé části využijeme k témuž účelu integrované vývojové prostředí Lazarus.
Ve třetí a čtvrté kapitole napíšeme několik programů, na nichž si ukážeme, jak naprogramovat čtení a zápis číselných hodnot, jak naprogramovat opakovaný výpočet a rozhodování, jak naprogramovat dílčí algoritmy, abychom je mohli používat na více místech programu (tedy jak naprogramovat proceduru nebo funkci) a jak vytvořit programovou knihovnu. Popis jazyka v těchto dvou kapitolách je velmi zjednodušený, stačí však k tomu, abyste od počátku mohli psát smysluplné programy a na nich si zkoušet konstrukce, o nichž si budeme vykládat.
V páté kapitole se seznámíme s nástroji pro ladění – hledání chyb v programech a jejich odstraňování –, které nabízí integrované vývojové prostředí Lazarus. Prakticky stejné nástroje a postupy lze ale využít i v jiných vývojových prostředích.
Od šesté kapitoly začíná výklad jazyka Pascal naostro. V této kapitole se seznámíme se syntaktickými diagramy a dalšími nástroji pro popis Pascalu. Dále se seznámíme s pravidly pro tvorbu identifikátorů a pro zápis programu a poznáme strukturu programu a jednotky. V následujících kapitolách postupně projdeme základní datové typy, příkazy a výrazy, uživatelem definované typy (včetně základních informací o objektových typech), procedury a funkce. Ve dvanácté kapitole se seznámíme s nástroji pro vstupní a výstupní operace a pro komunikaci uživatele s programem. Poslední kapitola je věnována základům tvorby grafického uživatelského rozhraní („oken").

Příklady

Výklad pochopitelně doprovází řada příkladů. Začínáme u jednoduchých prográmků nebo samostatných funkcí, které ukazují základní možnosti jazyka Pascal, a končíme u aplikací, které lze i prakticky využít – například u programu, který slouží k převodu textových souborů z jednoho kódování do jiného.
Úplné zdrojové texty programů, a v některých případech i celé projekty pro vývojové prostředí Lazarus, si můžete stáhnout z webových stránek nakladatelství Grada Publishing, http://www.grada.cz.

Co v této knize nenajdete

Tato kniha je určena začátečníkům, a proto nemůže pokrýt programování v Pascalu v plném rozsahu. Asi nejdůležitější témata, která jsem v této knize ponechal stranou, jsou ukazatele a dynamické

datové struktury, dědění a polymorfizmus a zpracování výjimek. Také programování grafického uživatelského rozhraní je zde jen uvedeno na scénu; ponechal jsem stranou takové věci, jako je použití předdefinovaných dialogů, vytváření vlastních komponent, kreslení, práce s databázemi a mnohé další. Tato témata se do naší knihy prostě nevešla.

Jazyk Pascal

Programovací jazyk Pascal navrhl v 70. letech minulého století prof. N. Wirth pro výuku programování a pojmenoval ho po francouzském matematikovi, fyzikovi a filozofovi z první poloviny 17. století Blaisu Pascalovi. Od té doby prošel tento jazyk bouřlivým vývojem a stal se z něj nástroj, který je vedle výuky vhodný i pro vývoj profesionálních aplikací. Protože byl navržen jako jazyk pro výuku, je jeho zvládnutí pro začátečníka snazší než zvládnutí některých jiných jazyků; zápis programu v něm je zpravidla také přehlednější.

Nástroje

V této knize se seznámíte implementací Pascalu označovanou Free Pascal a s integrovaným vývojovým prostředím Lazarus. Toto prostředí, spolu s překladačem, je k dispozici zdarma pod licencí GPL a lze si ho stáhnout na adrese http://sourceforge.net/projects/lazarus/files/. Tam lze získat i dokumentaci, v níž lze – po rozbalení např. programem 7-zip – vyhledávat informace o funkcích a třídách z knihoven Free Pascalu.

Lazarus je multiplatformní: lze ho používat pod 32bitovými i pod 64bitovými Windows, pod Linuxem i pod operačním systémem Mac OS X.

I když v celé knize předpokládám, že máte k dispozici lokalizovanou verzi Lazarusu 0.9.30, jejíž součástí je i překladač Free Pascalu verze 2.4.2 pod Windows, téměř vše, o čem v této knize hovořím, platí i v ostatních prostředích. Jedinou výjimkou, o které vím, jsou problémy s kódováním češtiny: ty jsou specifikou operačního systému Windows.

Poděkování

Na závěr bych chtěl poděkovat všem, kteří se svými radami a připomínkami podíleli na vzniku této knihy. Přes veškerou péči, kterou jsem její přípravě věnoval, se mohou vyskytnout chyby. Najdete-li nějakou, budu vděčný za upozornění. Na webové stránce http://tjn.fjfi.cvut.cz/~virius/errata.htm uveřejním následně případnou opravu.

Praha 28. 9. 2011 Miroslav Virius

1. Než začneme programovat

Než začneme programovat, měli bychom se seznámit s pojmy, které budeme později potřebovat. Povíme si krátce něco nejen o počítačích a programovacích jazycích, ale také o algoritmech, a řekneme si i několik slov o objektově orientovaném programování. Mnohé z toho jistě znáte; přesto vám doporučuji tuto kapitolu alespoň zběžně přečíst, abychom si sjednotili terminologii.

1.1 Počítač

Počítače se často označují jako stroje na zpracování informací. To je samozřejmě pravda, ale programátor, a to i začátečník, musí o počítačích vědět přece jen o něco víc. S trochou zjednodušení můžeme říci, že běžný počítač se skládá ze čtyř základních částí:

- *Procesor* opravdu „počítá", tedy zpracovává informace, a také řídí činnost všech ostatních částí počítače.
- K přechodnému ukládání dat, tedy informací, které počítač zpracovává, a programů, tedy příkazů, které určují, co má počítač dělat, slouží *operační paměť*. Používá se pro ni také zkratka RAM pocházející z anglických slov *Random Access Memory*, doslova „paměť s náhodným přístupem". Vše, co je v operační paměti, se při vypnutí dnešního počítače ztratí, počítač to „zapomene".
- Počítač si musí nějak vyměňovat informace s okolím. K tomu slouží *vstupní a výstupní zařízení*. Obvyklá vstupní zařízení jsou klávesnice, myš, skener atd.; nejobvyklejší výstupní zařízení osobních počítačů jsou monitor a tiskárna. Pro vstupní a výstupní zařízení se používá zkratka V/V nebo I/O pocházející z anglického input/output.
- Data a programy, které si má počítač pamatovat i po vypnutí, obvykle ukládáme na pevný disk nebo jiné podobné zařízení, na kterém se data po vypnutí neztratí. I když je dnes pevný disk typicky uvnitř počítače, z historických důvodů se často označuje za *vnější* (*trvalou*) *paměť*. Vnější paměť má zpravidla mnohonásobně větší kapacitu než operační paměť, ale práce s ní je také mnohonásobně pomalejší než práce s operační pamětí (v případě pevného disku je to přibližně tisíckrát). Jako vnější paměť mohou vedle magnetických disků sloužit optické disky, CD, DVD apod.

1.1.1 Operační paměť

Bity

Operační paměť počítače je tvořena elektronickými obvody, které mají mít dva dobře rozlišitelné stavy – např. vypnuto nebo zapnuto. Jeden z těchto stavů obvykle odpovídá číslici 0, druhý číslici 1. Jakékoli údaje do paměti proto můžeme zapisovat jen pomocí nul a jedniček, v tzv. dvojkové soustavě. Každé místo, na které můžeme zapsat jednu číslici 0 nebo 1, označujeme jako *bit*; pro bity používáme značku **b**. Operační paměť je tedy dlouhá řada bitů.

Bajty

Pracovat s jednotlivými bity je nepohodlné. Proto se bity v operační paměti sdružují do větších celků. V dnešních počítačích se zpravidla používají skupiny o velikosti 8 bitů, které se nazývají bajty (anglicky byte, tj. slabika). Pro bajty používáme značku **B**; zopakujme si, že platí 1 B = 8 b (jeden bajt je osm bitů).

Adresa

Aby mohl počítač s pamětí snadno pracovat, jsou jednotlivé bajty, které paměť tvoří, očíslovány. Počáteční bajt má číslo 0, následující má číslo 1 atd. Toto pořadové číslo se nazývá *adresa* bajtu. (Na některých počítačích, včetně PC, je záležitost s adresami trochu složitější, ale to pro nás nebude důležité.)

1.1.2 Datové typy a proměnné

Není těžké zjistit, 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 žalostně málo; v běžných programech se obvykle objevují daleko větší čísla. Proto se pro ukládání dat používají různě velké skupiny za sebou následujících bajtů.

Ani to ovšem nestačí. Vezmeme-li např. dva za sebou následující bajty, můžeme do nich uložit celá čísla v rozmezí od 0 do 65 535 (od 0 do 1111 1111 1111 1111 ve dvojkové soustavě). Když nám ani to nebude stačit, můžeme vzít skupinu 4 nebo třeba 8 bajtů. Jenže ani to neřeší problém, co dělat, když budeme potřebovat záporná čísla, reálná čísla, znaky nebo třeba logické hodnoty, jež vyjadřují, že nějaké tvrzení platí nebo neplatí.

Musíme tedy najít nějaký způsob, který nám umožní reprezentovat data různých „druhů“ v paměti počítače. Jinými slovy, musíme najít způsob, jakým určité hodnotě jednoznačně přiřadíme skupinu bitů, která bude tuto hodnotu představovat – jak tuto hodnotu v počítači *zakódovat*.

Datový typ

Můžeme se např. dohodnout, že bajt s hodnotou `01000010` bude představovat znak `'B'`. Táž skupina bitů může za jiných okolností také představovat celé číslo, které má v desítkové soustavě hodnotu 66. Bajt se stejnou hodnotou ale může být i součástí většího celku s naprosto jiným významem.

Odtud je vidět, že počítači nestačí znát adresu bajtu nebo bajtů, s nimiž pracuje. Vedle adresy musí vědět, jak velký úsek – kolik bajtů – má vzít a jak má jeho obsah interpretovat. Jinými slovy, musí znát *datový typ* hodnoty, která je tam uložena – musí vědět, zda jde o celé číslo, znak, logickou hodnotu atd.

Datový typ také určuje operace, které lze s danou hodnotou provádět. Celá čísla lze například sčítat a odečítat, znaky lze spojovat do řetězců, tedy do souvislého textu.

Proměnná

Až dosud jsme se hodnotami, uloženými v paměti, zabývali z hlediska počítače; nyní se na ně podívejme z hlediska programátora. Hodnotu, s níž budeme chtít ve svém programu pracovat, potřebujeme uložit někam do paměti. Musíme si vyhradit místo a říci, jakého typu budou údaje, které bude obsahovat. Takovéto místo pro ukládání hodnoty budeme nazývat *proměnná*.

Počítač bude s proměnnou zacházet pomocí její adresy (pořadového čísla jejího počátečního bajtu). Pro programátora by ale práce s adresou byla nepohodlná, a proto proměnnou pojmenujeme, dáme jí *identifikátor*. Tomu se v programování říká *deklarace proměnné*.

1.2 Programy a programovací jazyky

Budeme-li od počítače chtít, aby zpracovával data, která mu předložíme, musíme mu také říci, co s nimi má vlastně dělat – musíme mu dát *program*. Program, podobně jako data, uložíme do operační paměti.

Procesor umí s daty provádět různé jednoduché operace. Umí např. sečíst, odečíst, vynásobit nebo vydělit dvě čísla, zadáme-li mu jejich adresy, umí vzít znak a zobrazit ho na displeji atd. Protože však do jeho paměti nelze uložit nic jiného než čísla, musí být tyto příkazy vyjádřeny – zakódovány – také čísly. Číselné vyjádření instrukcí (příkazů) pro procesor se nazývá *strojový kód* a je to jediná věc, které procesor rozumí.

Aby nebyl život příliš jednoduchý, používají různé procesory různé strojové kódy, takže programy ve strojovém kódu nejsou obvykle přenositelné mezi počítači s různými procesory.

1.2.1 Vyšší programovací jazyky

Je asi jasné, že programování ve strojovém kódu je velice namáhavé a nepřehledné. Také to téměř nikdo nedělá; místo toho se používají tzv. vyšší programovací jazyky, jako je třeba Pascal.

Program ve vyšším programovacím jazyku obsahuje popis řešené úlohy vyjádřený pomocí vybraných anglických slov a matematických výrazů zapsaných podobně jako v matematice. Napsat program ve vyšším programovacím jazyku je pochopitelně daleko snazší než napsat odpovídající program ve strojovém kódu.

Ovšem nic není zadarmo. Program ve vyšším programovacím jazyku nelze na počítači přímo spustit, protože počítač mu nerozumí. Takovýto program 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.

Překlad

Textový soubor, který obsahuje zápis programu ve vyšším programovacím jazyku, se zpravidla nazývá *zdrojový kód* nebo *zdrojový program*, v programátorské hantýrce „zdroják". K překladu do strojového kódu (kompilaci) slouží program zvaný překladač neboli kompilátor. Často s ním spolupracuje ještě sestavovací program neboli linker, který dokáže 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. Dnešní překladače však často zastanou i práci sestavovacího programu.

Překladem (a popřípadě následným sestavením) programu vznikne *spustitelný soubor*, tedy soubor obsahující strojový kód, který již lze na cílovém počítači spustit. Mezi typické překládané programovací jazyky patří např. právě Pascal.

Interpretace

Místo překladu však můžeme v některých případech použít speciální program, který bude číst zdrojový text a interpretovat ho, tj. provádět příkazy, které v něm najde. Typickým interpretovaným jazykem je klasický Basic.

Interpretované programy obvykle běží podstatně pomaleji než překládané programy. Vedle toho musíme na cílový počítač spolu s naším programem instalovat také interpretační program.

1.2.2 Operační systém

Počítač bez programového vybavení by nám nebyl mnoho platný. Proto se s ním zpravidla dodává alespoň jeden základní program, který se rozeběhne automaticky hned po spuštění počítače a běží po celou dobu jeho provozu. Tento program se nazývá *operační systém* a „oživuje" počítač, tj. přijímá pokyny uživatele, stará se o jejich provedení a informuje uživatele o výsledcích. Pokyny

mohou být vyjádřeny příkazem zapsaným v příkazové řádce, kliknutím myší na ikonu nebo jiným způsobem.
Operační systém má ovšem ještě řadu dalších úloh, z nichž nejdůležitější pro nás je, že poskytuje služby dalším programům. Stará se o jejich spouštění, o přidělování paměti, poskytuje nástroje pro práci se soubory atd. Když např. chceme v programu otevřít soubor, program předá odpovídající požadavek operačnímu systému a ten se postará o vše potřebné.
Mezi nejznámější operační systémy na osobních počítačích patří různé verze Windows a Linuxu.
Nepříjemné je, že překladač musí program přeložit tak, aby mohl běžet pod určitým operačním systémem. Například program určený pro Windows nepoběží pod Linuxem nebo pod systémem MacOS.

1.3 Program a algoritmus

Jazyk Pascal, podobně jako ostatní programovací jazyky, slouží k zápisu programu. Už víme, že program je nějaký soubor instrukcí (příkazů), které počítači říkají, co má dělat.
Každý program představuje návod k řešení určitého problému. Při programování si ale musíme uvědomit, že počítač neumí najít způsob, jak problém vyřešit. To musíme udělat my. Když už známe cestu, jak najít řešení, můžeme to naučit také počítač. Počítač umí jen to, co ho my – nebo někdo jiný – naučíme, na co má program.

1.3.1 Algoritmus

Návod, který chceme předložit počítači, musí mít určité vlastnosti; některé z nich mohou vypadat jako samozřejmé, ale přesto je zde uvedeme.

- Návod *musí vést k požadovanému výsledku*. I když to možná vypadá směšně, přesvědčit se o tom není vždy jednoduchá záležitost. Musíme např. uvážit, za jakých předpokladů určitý postup vede ke správnému výsledku a zda budou tyto předpoklady splněny pro data, s nimiž bude náš návod pracovat.
- Musí se skládat z kroků, kterým počítač rozumí – tzv. *elementárních kroků*. To pro nás znamená, že ho musíme umět zapsat v některém programovacím jazyce, nejlépe v Pascalu. Jednotlivým krokům algoritmu budou v ideálním případě odpovídat jednotlivé příkazy (instrukce) programu.
- Po každém kroku musí být jasné, zda algoritmus skončil nebo kterým krokem bude pokračovat (říkáme také „kam přejde řízení“ nebo „kam bude přeneseno řízení“).
- Těchto kroků nesmí být nekonečně mnoho. Nejde o to, že bychom mohli napsat program, který je nekonečně velký; to se nám asi nepodaří. Není ale nic těžkého napsat krátký program, který nikdy neskončí, protože se v něm bude do nekonečna opakovat určitá skupina instrukcí.

Návod, který tyto podmínky splňuje, obvykle označujeme jako *algoritmus*. Na *počítačový program* se můžeme dívat jako na zápis algoritmu v programovacím jazyce.

Metoda shora dolů

Než začneme programovat, musíme tedy znát způsob, jak daný problém vyřešit. Pak musíme postup řešení zapsat jako algoritmus, tedy jako posloupnost elementárních kroků. Přitom se používá obvykle tzv. *metoda shora dolů*: návod se rozkládá na menší a menší úseky, až dospějeme k elementárním krokům. Během toho se samozřejmě může stát, že budeme muset předchozí rozdělení upravit, některé kroky spojit, zjistíme, že jsme vynechali jiné kroky, které nám připadaly tak samozřejmé, že jsme o nich vůbec nepřemýšleli, atd.

Zápis algoritmu

Algoritmy obvykle zapisujeme jako očíslovanou posloupnost kroků. Přitom rozumíme, že kroky se provádějí v pořadí, ve kterém jsou zapsány, pokud některý z kroků nepřikazuje něco jiného. Vždy musí být jasné, kterým krokem algoritmus pokračuje.

Modelovací jazyk UML

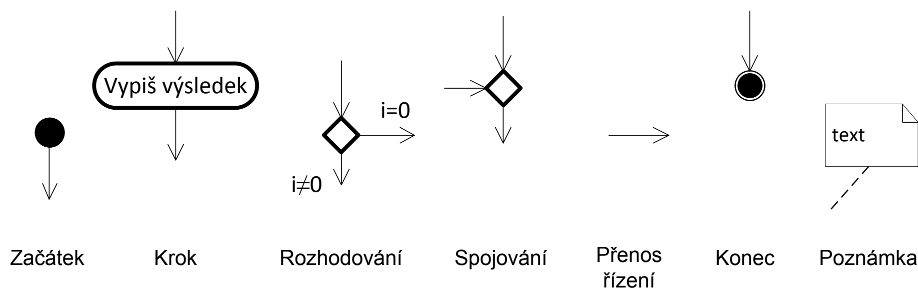
Při návrhu a popisu řešeného problému – nejen použitých algoritmů – se dnes často využívá tzv. Unifikovaný modelovací jazyk (Unified Modeling Language, UML). To je soubor pravidel pro vytváření diagramů, které popisují různé aspekty řešené úlohy. K popisu algoritmů slouží tzv. *diagram činností* (*diagram aktivit*, anglicky *activity diagram*).

Tento diagram začíná černě vyplněným kruhem, který označuje, že zde algoritmus začíná. Jednotlivé kroky jsou v něm vyznačeny ovály, do nichž stručně zapíšeme odpovídající činnost. Tyto ovály jsou spojeny šipkami, které směřují vždy od předchozího k následujícímu kroku (tedy vyjadřují *přenos řízení*).

Větvení algoritmu, tedy kroky, v nichž podle nějaké podmínky rozhodujeme, kterým krokem budeme pokračovat, vyznačujeme čtverečkem postaveným na koso. Každé z možností pokračování algoritmu odpovídá jedna spojnice, která z něj vychází. U každé spojnice je připsáno, jaké možnosti odpovídá – jaká podmínka musí být splněna, aby se použila tato spojnice.

Také místo, kde se několik možných cest výpočtu spojuje, označujeme čtverečkem postaveným na koso; do něj vchází několik spojnic, ale vychází z něj jediná.

Diagram činností končí terčíkem – kruhem, v němž je uzavřen soustředný menší vyplněný kruh. Diagram činností může obsahovat ještě další symboly; ty, o nichž jsme zde hovořili, ukazuje obrázek 1.1. Příklad diagramu činností uvidíme dále.



Obrázek 1.1: Nejběžnější symboly používané v diagramech činností (šipky vyznačující přenos řízení nejsou součástí jednotlivých symbolů, ale naznačují, jak se u těchto symbolů typicky používají)

Diagram činností představuje skvělý nástroj pro znázornění algoritmu pro člověka, který sám nebude psát kód. Pro programátora je ale – alespoň podle mé zkušenosti – výhodnější popis algoritmu v bodech, a proto mu budeme v této knize dávat přednost.

Dříve se často používaly tzv. *vývojové diagramy* (anglicky nazývané flowchart), které sloužily jako univerzální nástroj pro grafický popis programu. V 70. letech minulého století ale proti nim byla vznesena řada námitek, především že vedly ke špatně strukturovaným programům, a proto byly s velkou slávou pohřbeny jako přežitek. Nicméně diagramy aktivit nejsou nic jiného než přejmenované vývojové diagramy, které používají jiné symboly a navíc mohou vyjádřit i věci, s nimiž původní vývojové diagramy nepočítaly. Jsou ale jen jedním z mnoha druhů diagramů, které se v modelovacím jazyce UML používají.

Příklad 1.1: Řazení neboli třídění

Máme skupinu N číselných proměnných, které označíme A[1], A[2], … A[N]. Takovouto skupinu proměnných stejného typu, se kterými můžeme zacházet jako s jedním celkem, nazýváme *pole*. Zde tedy máme pole A s *prvky* A[1], A[2], … A[N]. Čísla, označující jednotlivé prvky pole, nazýváme *indexy*.

Naším úkolem je uspořádat pole A tak, aby platila podmínka

A[1] ≤ A[2] ≤ … ≤ A[N]. (1)

Úloha „zpřeházet" hodnoty uložené v prvcích pole, aby splňovaly podmínku (1), se nazývá *třídění* pole. Naším úkolem tedy je setřídit pole A.

Náš první pokus o napsání algoritmu třídění by mohl vypadat takto:

1. Najdi v poli A prvek, který obsahuje nejmenší hodnotu, a vyměň jeho obsah s prvkem, který je na prvním místě (na místě s indexem 1).
2. Najdi v poli A prvek, který obsahuje druhou nejmenší hodnotu, a vyměň ho s prvkem, který je na druhém místě.
3. … atd., dokud není pole setříděné.

Je zřejmé, že tento postup povede k cíli. Uvedený popis se ale pro naše účely příliš nehodí, neboť program nemůže obsahovat nějaké „… atd.". Musíme proto najít lepší formulaci.

Klíčem k úspěchu bude následující úvaha: poté, co v prvním kroku vyhledáme nejmenší prvek z celého pole a dáme ho na první místo, je nejmenší hodnota již na svém místě a nemusíme se o ni starat. Stačí tedy setřídit zbytek pole.

Ve druhém kroku nám postačí najít nejmenší prvek ze zbylého úseku pole A[2], A[3], … A[N] a ten dát na druhé místo, ve třetím kroku najít nejmenší hodnotu mezi poli A[3], A[4], … A[N] atd. V posledním kroku budeme hledat nejmenší z prvků A[N-1], A[N]. Pak bude pole setříděné, tj. bude vyhovovat podmínce (1).

Všimněte si, že vlastně stále opakujeme podobnou akci: vyhledáme nejmenší prvek v nějakém úseku pole a dáme ho na první místo tohoto úseku. Přitom v každém následujícím kroku pracujeme se stále menším úsekem pole. Zkoumaný úsek pole začíná postupně prvkem s indexem 1, 2, … N-1. Zařadíme-li na své místo předposlední prvek, skončíme, neboť největší prvek už musí být na správném místě. Jinými slovy, jednoprvkový úsek začínající prvkem s indexem N už řadit nemusíme.

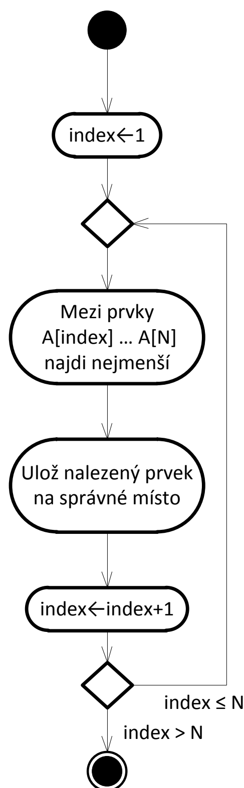
Zapišme výsledek předešlých úvah jako posloupnost kroků a podívejme se, zda jsou elementární. Při výpočtu budeme používat pomocnou proměnnou index, která nám poslouží jako „počitadlo".

1. *Příprava*. Do proměnné index vlož hodnotu 1.
2. *Nalezení nejmenšího prvku v daném úseku pole*. Mezi prvky A[index], … A[N] najdi ten, který obsahuje nejmenší hodnotu.
3. *Uložení minima na správné místo*. Vyměň jeho hodnotu s hodnotou prvku, který je na místě určeném indexem.
4. *Příprava na zpracování zbytku pole*. Zvětši hodnotu proměnné index o 1.
5. *Test konce*. Je-li index < N, pokračuj krokem 2, jinak skonči.

Tento postup můžeme znázornit diagramem činností na obrázku 1.2.

Vezměte prosím jako fakt, že kroky 1, 4 a 5 můžeme považovat za elementární, zatímco kroky 2 a 3 je třeba ještě dále rozložit. Až totiž poznáte základy Pascalu, bude vám jasné, že kroky 1, 4 a 5 odpovídají jednotlivým příkazům v tomto jazyce, zatímco zbývající dva jsou stále ještě příliš komplikované.

Podívejme se nejprve na krok 2. V něm budeme potřebovat dvě pomocné proměnné. V jedné, kterou pojmenujeme min, si budeme pamatovat index nejmenšího zatím nalezeného prvku, a druhá, kterou pojmenujeme např. k, nám poslouží jako „počitadlo" při procházení zkoumaného úseku pole. Rozepsaný krok 2 bude vypadat takto:



Obrázek 1.2: Diagram aktivit popisující první verzi algoritmu pro třídění pole

2. *Nalezení nejmenšího prvku v daném úseku pole* `A[index],… A[N]`.
   2a. *Příprava – nastavení počátečních hodnot*. Do obou pomocných proměnných, `k` a `min`, ulož hodnotu proměnné `index`. (To je index počátečního prvku zkoumaného úseku. Zatím jsme prozkoumali jen jeden prvek a tak ho pokládáme za nejmenší.)
   2b. *Přechod na další prvek v poli*. Zvětši `k` o jedničku.
   2c. *Vyčerpali jsme všechny prvky v úseku?* Je-li `k > N`, přejdi na krok 3.
   2d. *Zapamatování indexu nejmenšího prvku*. Je-li `A[k] < A[min]`, ulož do `min` hodnotu `k`.
   2e. *Opakování*. Přejdi na bod 2b.

Po skončení kroku 2 bude proměnná `min` obsahovat index nejmenšího prvku v prohledávaném úseku.
Podívejme se ještě, jak bude vypadat rozepsaný bod 3. Zde budeme potřebovat pomocnou proměnnou `pom`.

3. *Uložení nejmenšího prvku na správné místo*. Vyměň prvek `A[index]` s prvkem `A[min]`.
   3a. Do pomocné proměnné `pom` ulož hodnotu `A[index]`.
   3b. Do `A[index]` přesuň hodnotu z `A[min]`.
   3c. Do `A[min]` přesuň hodnotu `pom`.

Až se naučíme Pascal, budeme vědět, že tyto kroky již můžeme pokládat za elementární.
Podrobně rozepsaný algoritmus třídění pole ukazuje obrázek 1.3.

Obrázek 1.3: Podrobný diagram aktivit popisující třídění pole

1. Popsaná metoda se nazývá třídění výběrem (selection sort) a hodí se pro pole s menším počtem prvků. Pro rozsáhlejší pole se používá především tzv. rychlé třídění (quicksort). To je velmi efektivní postup, a vrátíme se k němu v 11. kapitole v příkladu 11.10. Podrobnější informace o těchto i dalších metodách třídění najdete např. ve třetím dílu knihy [1] nebo v [2].

2. **Označení *třídění* pro uvedený postup je vlastně terminologický nesmysl. Slovo *třídění* znamená rozřazování do tříd podle nějakého znaku; můžeme například třídit brambory na dobré a zkažené nebo třešně na zralé a nezralé. Zde nám ale jde o *řazení*, tedy o úlohu uspořádat – seřadit – prvky podle velikosti.**
   V programátorském žargonu se však pro tuto úlohu z nějakých záhadných důvodů ujalo třídění, a to nejen v češtině, ale třeba i v angličtině, kde se používá *sorting*, nikoli *ordering*.
3. **Algoritmům a metodám jejich návrhu je věnována řada knih; většina z nich ale předpokládá znalost určitého konkrétního programovacího jazyka, často Pascalu (např. [2]). Asi nejznámější jsou knihy D. Knutha [1], které znalost žádného konkrétního programovacího jazyka nepředpokládají.**

Metoda zdola nahoru

Vedle metody shora dolů se občas se také hovoří o tzv. metodě *zdola nahoru*. To znamená, že naučíme svůj počítač nové elementární kroky, které poskládáme z kroků, které počítač už umí. Jestliže například často třídíme pole, naprogramujeme si výše uvedený algoritmus jednou provždy jako *podprogram* (proceduru nebo funkci), uložíme si ho do knihovny a později ho budeme už jen používat jako elementární krok.

Knihovna

Součástí každého programovacího jazyka jsou připravené knihovny, které obsahují řadu běžných algoritmů již naprogramovaných jako podprogramy, které můžeme ve svých programech používat jako elementární kroky. Už jsme se zmínili, že programátor si může také vytvářet své vlastní knihovny.

1.4 Objekty a třídy

Objektově orientované programování (OOP) je od devadesátých let minulého století naprosto převládajícím přístupem, a proto si o něm musíme povědět alespoň nejzákladnější informace i v knize určené naprostým začátečníkům. V tomto oddílu si vysvětlíme jeho základní principy. Až se trochu seznámíme s Pascalem, budeme si moci povědět více.

1.4.1 Zapouzdření

Zatím jsme uvažovali o programování a programech převážně z hlediska počítače, hardwaru. Od zobrazování dat v paměti jsme došli k datovým typům, od strojového kódu k programovacím jazykům. Pokusme se nyní na programování podívat z hlediska programátora, který stojí před úkolem něco vyřešit – ať už jde o program pro vedení účetnictví malé či velké firmy, počítačovou hru nebo systém řídící vstřikování paliva v automobilovém motoru. Program musí v každém případě odrážet vybrané stránky řešené úlohy, musí být jejím *počítačovým modelem*.

Proč objekty?

Je asi jasné, že programátorovi se bude lépe pracovat, bude-li moci uvažovat v termínech řešeného problému, než když bude muset uvažovat v termínech datových typů, které jsou jednou pro vždy dány jako součást programovacího jazyka. Jinými slovy, bylo by vhodné, aby si programátor mohl definovat své vlastní datové typy, které by mohl použít k programovému popisu řešené úlohy. V OOP se jim říká *objektové typy* neboli *třídy*, neboť popisují třídy objektů, které se vyskytují v řešeném problému.

-

Jako příklad si představme, že programujeme grafický editor. Jeho uživatel si v něm bude chtít kreslit obrázky složené ze základních geometrických tvarů – bodů, úseček, kružnic ap.
Programátor bude muset ve svém programu tyto geometrické tvary nějak reprezentovat. Bude si muset vytvořit programový popis – model – bodu, úsečky atd. a bude muset naprogramovat také operace, které lze s těmito grafickými objekty dělat.

Bod jako objekt

Co to znamená? Abychom si to ujasnili, zamysleme se např. nad bodem. Z geometrie si nepochybně pamatujete, že bod v rovině (v našem případě na obrazovce monitoru) je určen souřadnicemi – dvojicí reálných čísel. Vedle toho ovšem potřebujeme určit také jeho barvu; tu můžeme v počítači vyjádřit jedním celým číslem. Z toho plyne, že k reprezentaci bodu stačí skupina tří čísel. Je jasné, že tvoří jeden celek – bod –, a proto budeme chtít s touto skupinou v programu také zacházet jako s celkem.
Abychom mohli s bodem snadno pracovat, musíme popsat (naprogramovat) také operace, které s ním chceme dělat. Musíme ho umět vytvořit, zrušit, zobrazit, přemístit, zjistit jeho barvu, změnit jeho barvu, zjistit jeho polohu, změnit jeho polohu atd.

Třída

V programu budeme určitě potřebovat větší množství bodů. Abychom nemuseli popisovat každý zvlášť, definujeme `Bod` jako nový datový typ – jako třídu. Jednotlivé body, které nakreslí uživatel našeho grafického editoru, budou v programu představovány proměnnými (budeme také říkat *objekty* nebo *instancemi*) třídy `Bod`. Každá instance třídy `Bod` bude obsahovat datové složky, které popisují jeho polohu a barvu.

Metoda

K zacházení s jednotlivými body budeme používat operace, které jsme k tomu účelu naprogramovali. Také tyto operace budou součástí definice třídy; v OOP se nazývají *metody*.
Při definici nového typu představujícího bod jsme udělali (zatím jen naznačili) dvě věci. Určili jsme:

- datovou reprezentaci nového typu;
- operace s ním (metody).

Zapouzdření a ukrývání implementace

Jedno z pravidel objektového programování říká, že s datovými složkami dané třídy bychom měli manipulovat pouze prostřednictvím jejích metod. Později si vysvětlíme, proč je to tak důležité.
V objektovém programování se tomu říká zapouzdření (anglicky encapsulation). Poznamenejme, že pro některé složky můžeme specifikovat tzv. přístupová práva – můžeme např. předepsat, že je smějí používat pouze metody dané třídy (tzv. soukromé složky) nebo že je smí používat kdokoli (složky veřejně přístupné). O veřejně přístupných složkách hovoříme také jako o *rozhraní* třídy.

Objektový program

V teorii OOP se říká, že objektový program se skládá pouze z objektů, tedy z instancí objektových typů, a tyto objekty si navzájem posílají zprávy. To zní asi velice nesrozumitelně, ale v podání jazyka Pascal to prostě znamená, že volají (spouštějí) své metody.
Vezměme opět grafický editor a podívejme se na něj jako na celek. Celá aplikace bude nejspíš představována instancí třídy `Editor`, jednotlivé součásti obrázku budou představovány instancemi tříd `Bod`, `Úsečka` atd. Když uživatel stiskne tlačítko, kterým určí, že chce někde vytvořit nový bod, tedy instanci třídy `Bod`, dozví se to editor. Ten požádá o vytvoření instance třídy `Bod` (pošle třídě `Bod` zprávu, která bude představovat žádost o vytvoření instance určitých vlastností). Bude-li třeba vybraný bod přemístit, pošle editor této instanci zprávu „přemísti se“, tj. zavolá odpovídající metodu.