Co je vám, pane Wirth?

aneb Pohled na svět strukturovaného programování

Miroslav Beneš
Ústav informatiky a výpočetní techniky, FEI VUT Brno

Abstrakt

Cílem tohoto příspěvku je podat kritický přehled vývoje jazyků, jejichž autorem je prof. Wirth, a poukázat na základní principy těchto jazyků, jejich výhody a nevýhody.

1. Na počátku všeho byl Algol 60

Technologie strukturovaného programování je od počátku spjata se jménem profesora Niklause Wirtha z curyšské ETH, který ve svém jazyce Pascal (1970) navázal na úspěchy zejména jazyka Algol 60 (1960). Tato řada pak pokračuje jazykem Modula-2 (1979) a prozatím končí jazykem Oberon (1988).

Algol 60 byl vytvořen skupinou třinácti expertů z mnoha zemí. Jeho cílem bylo vytvoření společné notace pro zápis programů nejen pro účely práce s počítačem, ale rovněž i pro komunikaci mezi programátory. To s sebou neslo i nutný požadavek na nezávislost definice jazyka na konkrétním technickém vybavení. Jazyk Algol 60 s sebou přinesl rovněž do té doby neobvyklý přístup k autorům překladačů - byl to vlastně první jazyk, u kterého se dala jednoduše kontrolovat syntaxe (na rozdíl od např. jazyka Fortran, kde mezery neoddělovaly symboly od sebe a kde klíčová slova jazyka nebyla rezervovaná, což pocítili američtí daňoví poplatníci, přispívající na kosmický program).

Jazyk Algol 60 zavedl do programování pojem strukturovanosti. Tato technologie pohlíží na program ne jako na posloupnost jednotlivých příkazů zapsaných v jazyce symbolických instrukcí nebo ve Fortranu, ale jako na hierarchickou strukturu tvořenou kombinací jednoduchých a strukturovaných příkazů. V ideálním případě tyto příkazy lze reprezentovat jako bloky s jediným vstupem a jediným výstupem, pokud neuvažujeme existenci příkazů skoku. Tyto vlastnosti však ještě v 60. letech nebyly nijak ceněné a jazyk se příliš neujal; Algol 60 neumožňoval práci s heterogenními strukturami a dynamickými proměnnými a obsahoval kromě mnoha elegantních konstrukcí rovněž některé prvky, které byly složité jak implementačně, tak i pro uživatele (např. předávání parametrů jménem vzniklo vlastně nepochopením rozdílu mezi podprogramem a makrodefinicí). Zřejmě hlavní chybou při návrhu bylo to, že jazyk Algol-60 vytvořili matematici pro popis svých vlastních (obvykle numerických) problémů, avšak bez znalostí širšího kontextu.

2. Počátky strukturovaného programování

Programovací jazyk Pascal se ihned po svém vzniku na konci 60. let začal rychle rozšiřovat, zejména v akademickém prostředí. Jeho úspěch nebyl dán jen značnou jednoduchostí syntaxe i sémantiky, ale také například použitím speciálně navrženého přenositelného mezikódu, tzv. P-kódu, jenž umožňoval velmi rychlou dostupnost překladače na velkém množství platforem. Tento jazyk navázal na Algol 60 zejména svým přístupem ke strukturovanosti programu a stal se současně i se svým autorem horlivým šiřitelem principů *strukturovaného programování*. Tyto principy byly uplatněny nejen na strukturalizaci příkazů jako v Algolu, ale rovněž i na datové

typy. Kromě v té době běžných polí byly zavedeny záznamy, množiny a soubory, umožňující vytvářet hierarchické struktury i na úrovni dat. Tím se jazyk Pascal dostal do oblasti skutečně obecně použitelných jazyků pro numerické i nenumerické výpočty, a to i přes jeho původní výlučně výukové účely.

Ani Pascal se však nevyhnul nedostatkům, které se začaly projevovat především až při jeho nasazení do vývoje běžných komerčních a systémových aplikací. Původní verze jazyka totiž předpokládala, že celý program bude uložen v jediném souboru, což zdaleka nepostačovalo pro vývoj rozsáhlejších programových systémů. Postupně se objevují nové metody strukturalizace programů na vyšší úrovni, které se dají souhrnně pojmenovat jako metody *modulárního programování*. Tento pojem v sobě zahrnuje dva základní aspekty: systém je rozdělen na relativně izolované moduly, pro které existují definovaná rozhraní. Ta zakrývají vnitřní implementaci modulů a umožňují jejich oddělený překlad a vytváření knihoven modulů.

Masové rozšíření jazyka Pascal mezi programátory nastalo po uvedení na trh produktu Turbo Pascal firmou Borland, tehdy ještě pod systémem CP/M; tato implementace se však již od počátku odpoutala od původní definice jazyka, což vrhá učitele na školách do nekonečných diskusí o tom, co vlastně jako Pascal učit. Na jedné straně existuje ISO standard jazyka, který je ale dodržován zejména v implementacích pod systémem Unix. V našich podmínkách, kde ale ještě stále převládají osobní počítače pracující v systému MS-DOS, však právě produkty firmy Borland převládají a vytvářejí tak jakýsi "de facto" standard jazyka Pascal; a právě jeho znalost je pak požadována v praxi. Autoři jazyka Borland Pascal velice rychle reagují na aktuální trendy ve vývoji programovacích jazyků, a proto již od prvních verzí určených pro PC umožňují například pracovat s odděleně překládanými moduly. Pokud jde o vztah k původnímu návrhu jazyka Pascal, je třeba prof. Wirthovi vzdát hold, neboť na rozdíl od vývoje jazyka C s sebou Pascal nenese ani v nejnovějších variantách balast konstrukcí zachovávaných jen z piety kvůli tzv. zpětné kompatibilitě (viz možnost definice typů parametrů funkce v C až za hlavičkou funkce).

3. Období modulárního programování

Modulární programování si získalo své nadšené příznivce především mezi programátory rozsáhlejších programových celků. Vzhledem k tomu, že začalo vznikat mnoho různých, navzájem nekompatibilních implementací tzv. modulárního Pascalu, dozrál na konci 70. let čas ke zcela novému jazyku. Tímto jazykem se stala Modula-2, opět z dílny prof. Wirtha. Jazyk Modula-2 již umožňuje oddělený překlad modulů, samostatnou specifikaci rozhraní modulu a jeho implementace. Na rozdíl od odděleného překladu známého ze strojově orientovaných jazyků jsou překladače modulárních jazyků schopny provádět veškeré nutné kontroly konzistence i přes hranice jednotlivých modulů na základě explicitně definovaných rozhraní. Právě zachování plné typové kontroly i při odděleném překladu bylo velkou výhodou tohoto přístupu.

Ve srovnání s jazykem Pascal je v Module-2 značně posílena typová kontrola. Jsou zavedeny dva celočíselné typy - INTEGER (se znaménkem) a CARDINAL (bez znaménka), které spolu nejsou kompatibilní v aritmetických operacích a programátor je pak nucen explicitně konvertovat ve smíšených výrazech typy operandů. To na jedné straně zvyšuje bezpečnost jazyka, ale zase znepříjemňuje práci programátorů tam, kde jde jen o nadbytečnou formalitu. Značné odlehčení typové kontroly je zase umožněno v parametrech podprogramů, kde lze předávat dynamická pole (s libovolnou délkou). Vzhledem k tomu, že jazyk Modula-2 vznikl - jak tomu bylo i u Pascalu - původně pro poměrně speciální účely, obsahuje i prostředky podporující kvaziparalelní procesy; modul může mít např. přiřazenu prioritu, jejíž přímé zařazení do syntaxe jazyka není právě nejšťastnější.

Tomuto jazyku lze jistě vytknout určité nedostatky, zejména přílišný "strukturovaný dogmatismus", díky kterému byl z definice zcela odstraněn příkaz skoku, aniž by byly dány k dispozici jeho dostatečné náhrady v obvyklých situacích (příkaz typu break pro nucené opuštění těla cyklu apod.) - ostatně u nás nejznámější implementace jazyka TopSpeed Modula-2 příkaz skoku opět zavádí. Proti Pascalu je však další výhodou zejména odstranění některých syntaktických pohrobků Algolu jako nejednoznačnosti příkazu IF (nyní je vždy ukončen klíčovým slovem END), umožnění posloupnosti příkazů všude tam, kde v Pascalu byl možný pouze jeden příkaz a oddělení funkcí pro vstup a výstup od vlastní definice jazyka.

Práce s modulárními programovacími jazyky po čase ukázala, že jednou z nejdůležitějších vlastností jazyka je jeho *rozšiřovatelnost*, a to nejen na úrovni procedurální, kdy můžeme vytvářet hierarchie modulů a podprogramů na různé úrovně abstrakce, ale i na úrovni datové. Zde však narážíme na to, že nelze současně rozšířit datový typ a přitom zachovat jeho kompatibilitu.

4. Objektově orientované programování

Vhodným řešením problému rozšiřovatelnosti bylo zavedení *dědičnosti* jako základního principu pro rozšiřování datových typů. Tento trend, patrný již v 80. letech, ale zesilující zejména v poslední době, vedl ke vzniku disciplíny *objektově orientovaného programování*. Do centra pozornosti se dostávají jazyky, které vycházejí z pohledu na program jako na model reality tvořené objekty. S tím současně dochází ke značnému zmatení obvyklých pojmů, neboť typům se začíná říkat třídy, proměnným instance, procedurám metody a voláním procedur zasílání zpráv.

Rozvoj objektově orientovaných technologií, nejen v programování, ale i na vyšší úrovni analýzy a návrhu systémů, vedl k silnému tlaku na rozšíření programovacích jazyků o možnost definice objektů s dědičností, metodami a selektivním zapouzdřováním dat. Tyto prvky, opět jako tomu bylo v případě jazyka Pascal, jsou postupně začleňovány do stávajících jazyků formou různých objektových rozšíření. Výsledkem toho byl např. jazyk C++, v němž snaha po zpětné kompatibilitě s jazykem C vedla k návrhu velmi obtížně implementovatelného jazyka se složitou sémantikou. Podobná rozšíření najdeme i v posledních verzích jazyka Borland Pascal nebo TopSpeed Moduly-2; obě tyto implementace se tím však ještě více vzdálily od standardní definice jazyků, jejichž jména i přesto stále nesou.

Nelze se proto divit, že ani tým prof. Wirtha nezahálel a ve zcela opačném duchu byl v roce 1986 let uveden do světa jazyk Oberon, podstatně jednodušší než jazyk Modula-2, a v roce 1991 po revizi a drobných úpravách jazyk Oberon-2. Opět i tento jazyk byl vytvořen ze zcela jiných pohnutek, než je posun laťky univerzálních jazyků o kus nahoru - jeho cílem bylo podpořit implementaci nového operačního systému založeného na objektově orientovaných principech. Oberon je totiž ve skutečnosti nejen programovací jazyk, ale zároveň i operační systém s vlastním uživatelským prostředím, které je možné postupně rozšiřovat a modifikovat.

Jazyk Oberon je založen na pojmu rozšiřovatelnosti podpořeném vhodně definovanými pojmy typu a procedury. Na rozdíl od jiných objektově orientovaných jazyků je zde rozšiřitelnost podpořena pouze úpravou definice záznamu a nevyžaduje tedy zavádění soustavy zcela nových pojmů. Odvozování datových typů probíhá tak, že nový datový typ (záznam) obsahuje proti původnímu některé složky navíc, přičemž je zachována jednostranná kompatibilita mezi odvozeným a bázovým typem, případně mezi ukazateli na takové typy. Skutečný typ lze v programu testovat a provádět bezpečné přetypování typu na jeho podtypy. Aplikací pouze těchto konstrukcí můžeme například bez nutnosti změny definice typu rozšiřovat množinu jeho operací (metod).

V porovnání s jazykem Modula-2 zde dochází k mnoha zjednodušením (definice jazyka Oberon má pouhých 16 stran oproti Module-2 na 45 stranách), motivovaným snahou o odstranění vlastností, které "mohou být ignorovány bez ztráty obecnosti a vyjadřovací síly" jazyka, jak uvádí sám prof. Wirth. Jde zejména o následující zjednodušení (a připomínky k nim):

* odstranění variantních záznamů (jejich funkci mohou dostatečně zastávat rozšiřitelné datové typy);
* odstranění výčtových typů a intervalů (včetně všech příslušných kontrol rozsahů)
* množiny mohou být pouze nad přirozenými čísly, tj. nelze například vytvořit množinu znaků(!) - množiny nad jinými typy byly označeny za redundantní pojem vzhledem k neexistenci výčtů a intervalů (co bylo dříve, výčet nebo množina?);
* zrušení bezznaménkového celočíselného typu CARDINAL (nově zavedeného právě v Module-2);
* omezení ukazatelů pouze na pole a záznamy (v praxi postačující);
* polí pouze přirozenými čísly počínaje nulou, odůvodněné neúměrnou výpočetní složitostí při generování mapovací funkce (ať si ji programátoři píší sami, aspoň mají kde udělat chybu!);
* sloučení definice rozhraní a implementace modulu do jediného textu (při změně implementace pak po překladu může vzniknout i nová verze rozhraní, která si vynutí rekompilaci zbytku projektu);
* povolení pouze kvalifikovaných jmen pro dovezené objekty a v podobném duchu i odstranění původního příkazu WITH (vedoucí k nutnosti rozsáhlejších optimalizací kódu, zcela v protikladu s proklamovanými principy návrhu jazyka);

Při bližším seznámení se s jazykem Oberon je přece jen vidět vliv jazyka C, a to často v místech, kde bychom to od prof. Wirtha čekali nejméně. Pokud jsme se snažili sledovat vývoj od Algolu k Module-2, je tento obrat zcela nepochopitelný, neboť jazyk Oberon ustupuje na plné čáře od těch zásad, které se zdály v předchozích jazycích být v centru pozornosti. Někdy z hlediska implementace velmi jednoduše implementovatelné a srozumitelné pojmy jsou odstraněny za cenu podstatně snížené sémantické vyjadřovací schopnosti programu. Do syntaxe jsou zaváděny tajuplné a nic neříkající hvězdičky a pomlčky pro odlišení veřejných prvků a prvků přístupných pouze pro čtení, čímž je čitelnost programu opět snížena.

Sledujeme-li vývoj koncepce programovacích jazyků vytvořených profesorem Wirthem, není možné přejít uvedené připomínky bez povšimnutí. Naznačený ústup od základních myšlenek srozumitelnosti textu programu a minimalizace potenciálních chyb programátora za cenu nepříliš významného zjednodušení překladače jistě přináší vtíravou otázku: Jak bude vypadat Oberonův následník? Bude to jazyk spíše podobný C nebo Pascalu?

I přes tyto připomínky rozhořčeného programátora, je však jazyk Oberon velmi zajímavý jako nástroj pro výuku základů programování a programovacích technik, neboť podporuje moderní metody tvorby programů a přitom je dostatečně jednoduchý a pochopitelný i pro úplné začátečníky. Rovněž z hlediska implementace překladače je velmi zajímavý vzhledem ke své jednoduchosti a přesné specifikaci. V současné době se dokonce i u nás některé firmy zabývají profesionálním vývojem překladačů jazyka Oberon-2, zejména pro podporu specializovaných procesorů pro průmyslové řídicí aplikace. Jaká bude jejich úspěšnost, to je závislé zejména na tom, jak rychle bude tento jazyk dostatečně rozšířený mezi programátory. Prozatím se zdá, že proti modulárním jazykům ještě stále stojí "všeho schopné" jazyky C a C++, ovšem snaha o větší bezpečnost programů, snadnější a efektivnější vytváření rozsáhlejších systémů a v neposlední řadě i o větší srozumitelnost a tím i udržovatelnost textů programu nakonec zvítězí - i když, alespoň podle autorova názoru, to možná nebude právě Oberon. Například jednou z dalších možností, která se v současné době v naznačené vývojové linii vyskytuje, je jazyk Modula-3

vyvinutý výzkumnými centry DEC a Olivetti a pojmenovaný s posvěcením prof. Wirtha. Že by právě toto byl ten správný chybějící článek?

Literatura:

Beneš M., Honzík J. M.: *Modula-2 versus Pascal na počítačích PC*. Sborník semináře Programování'89. DT Ostrava, 1989.

Reiser M., Wirth N.: *Programming in OBERON*. Step beyond Pascal and Modula. Addison-Wesley, 1992.

Snášel V., Sklenář V.: *Oberon*. Sborník semináře Programování'94. DT Ostrava, 1994, str. 210--215.

Wirth N.: *From Modula to Oberon*. Součást dokumentace systému Oberon-2 dostupného na adrese ftp.inf.ethz.ch:/pub/Oberon

Kontaktní adresa:

Ing. Miroslav Beneš
Ústav informatiky a výpočetní techniky FEI VUT v Brně
Božetěchova 2, 612 66 Brno, tel. (05)7275238
benes@dcse.fee.vutbr.cz