MINIMÁLNÍ SLOVNÍK PROGRAMOVÁNÍ

Ing. Andrej Boldiš, FMF Praha

1. Úvod

Podle B. Russella /1/ minimální slovník určité vědní disciplíny je množina jistých počátečních slov, pomocí kterých lze nominálně definovat všechny ostatní termíny disciplíny a žádné počáteční slovo nelze nominálně definovat v termínech tohoto minimálního slovníku. Úměrně tomu jak se vědní disciplína stává více systematickou, zmenšuje se i její minimální slovník.

S vývojem programování a programátorského myšlení prodělává změny i jazyk a slovník programování, o čemž svědčí například i jazyk manuálu programovacího jazyka Ada /3/.

Vymezíme si téma příspěvku takto. Programováním budeme rozumět běžné proceduráiní programování, speciálně nebudou programováním myšleny: simulace dynamických systémů, logické programování, programování robotů, numericky řízených strojů a technologických procesů a pod. Vyššími jazyky jsou zde míněny jazyky jako Fortran, Cobol, PL/1, Pascal, Ada, a pod. Programování má jak známo dva aspekty: (a) aspekt logicko-sémantický, t. j. programování jako systematické vytváření a formulování algoritmů a (b) aspekt technologicko-syntaktický, t. j. programování jako efektivní výroba software. Oba tyto aspekty se odráží v programovacích jazycích a v jazyku programování, nám půjde ale výhradně o logicko-sémantickou stránku programování. Tématem pojednání jsou tedy logicko-sémantické kategorie proceduálního programování.

Podle posledního vývoje lze usuzovat, že minimální slovník programování sestává z následujících základních pojmů: objekty, hodnoty, typy, procedury a funkce. Při formulování minimálního slovníku musíme řešit dvě zásadní otázky. Za prvé je to otázka obecnosti, t. j. ptáme se, zda pojmy minimálního slovníku jsou dostatečně obecné nebo naopak jsou přehnaně obecné a tudíž málo výstižné. Za druhé je to samotný obsah minimálního slovníku. V našem případě se ptáme, zda by nebylo možné uvedenou soustavu základních pojmů nahradit jinou soustavou pojmů, založenou například na jiných abstrakcích. Takový alternativní minimální slovník programování by mohla např. představovat soustava pojmů: datové struktury a řídící struktury.

2. Objekty nebo-li datové objekty

Podle /3/ "objekty jsou entity obsahující (mající) hodnotu určitého typu". Zcela konkrétní představa datového objektu je paměťové místo. Pojem "objekt" má řadu předchůdců, např. proměnná, operand, položka, a pod. Každý z těchto pojmů dobře vystihoval jistou stránku povahy objektů, přičemž stranou zůstávala podstata věci, a sice konstruktivní a manipulativní povaha datových objektů. Základní detailní typologie objektů bude uvedena v odstavci o typech, čímž se přesněji vymezí i obsah pojmu.

Další užitečné členění objektů je na proměnné a konstanty a na objekty globální a lokální. Toto poslední členění je zvlášť důležité pro pochopení podstaty funkcí a vůbec podstaty vyšších jazyků. Pojem objektu je spjat s procedurálními jazyky všech úrovní, o objektech lze mluvit též v mikroprogramování.

V běžné terminologii datové objekty jsou označovány rolí, kterou oni v programech, např. "indikátor", "klíč", a pod. Mluvíme-li v obecné a neurčité rovině označujeme objekty přímo typem. Říkáme "věta" u "souboru" a míníme tím objekty typu věta, resp. soubor, protože v tradičním pojetí se neoddělovaly deklarace typů od deklarace objektů.

3. Hodnoty

Hodnota je kategorie charakteristická pro vyšší programovací jazyky, vyznačující se jistou abstraktní úrovní. Konkrétnější představa hodnoty je datový obsah objektu. K pojmu "hodnota" dospíváme odhlédnutím od materiálního nositele datového obsahu, od objektu. Souvislost mezi objektem a hodnotou si nejčastěji připomínáme a uvědomujeme, když mluvíme o počáteční hodnotě objektu a o přiřazení hodnoty objektu. Abstraktnější povaha hodnoty ale vyniká v souvislosti s funkcí. Říká se, že funkce je výpočet (computation) hodnoty, nebo že funkce vrací (returns) hodnotu.

K označování explicitních hodnot se používají literály. Zatímco hodnota je kategorie sémantická, literál je odpovídající kategorie syntaktická.

S hodnotami těsně souvisí kódování obsahu objektů, nebo-li reprezentace různých typů hodnot. V jazycích vysoké abstraktní úrovně, jako např. Ada, se abstrahuje od reprezentace hodnot i takového typu jakým je typ výčtový definovaný uživatelem.

4. Typy

Trojice kategorií "hodnoty, typy a funkce" je typická a výstižná pro vyšší programovací jazyky, vyznačující se vysokou abstraktní úrovní. U těchto jazyků je typ nutné dávat do souvislosti nejprve s hodnotou a mluvit o typech hodnot a teprve pak mluvit např. o typech objektů nebo typech funkcí.

V novějším pojetí typ charakterizuje jistou množinu hodnot a jistou množinu funkcí (operací) použitelných na tyto hodnoty. V tomto smyslu kategorie typu je konkretizací jistých teoretických principů známých pod názvem "datové abstrakce". Podle těchto principů podstatu určitého typu hodnot vystihují právě funkce (operace) použitelné na daný typ hodnot. Tak např. celočíselný typ (integer) je množina hodnot, na kterou lze použít operace sčítání, odčítání, atd.

Uvedeme si klasifikace typů podle /3/. Skalární typy jsou typy, které nemají komponenty. Skalární typy se dělí na diskrétní a reálné. Diskrétní typ tvoří typ celočíselný (integer) a typ výčtový (enumeration). Výčtové typy jsou: typ boolovský, znakový (character) a typ definovaný uživatelem. Reálné typy mohou být typ číselný s pohyblivou desetinnou tečkou (floating point) a s pevnou desetinnou tečkou (fixed point). Typy skalární, diskrétní, reálný, číselný, výčtový jsou generické typy ostatní jsou základní (base) typy. Množinu hodnot určitého základního skalárního typu lze omezit aniž by se to dotklo použitelných funkcí (operací). Za tímto účelem jsou zavedeny podtypy (subtype) a odvozené (derived) typy. Všechny skalární typy charakterizují úplně uspořádané množiny hodnot, t. j. na všechny skalární typy lze použít operace srovnání.

Složený (composite) typ je typ složený ze skalárních hodnot. Složené typy jsou věta (record) a pole (array). Věta a pole jsou generickými typy, jejich instance se obdrží uvedením komponentů. Tedy typy komponentů určují jednotlivé typy vět a polí.

Soubor je další a hierarchicky vyšší typ. Intuitivně vzato je soubor posloupností vět. Samotný soubor je typ generický, jeho instance se obdrží uvedením typů vět, z nichž soubor je vytvořen. V souvislosti se složenými typy a souborem je řada ještě otevřených teoretických otázek.

V jazycích Pascal a Ada se využívá typ přístupový (access). Tento typ je nutné chápat jako paměťové adresy objektů. Oprávněnost tohoto typu

ve vyšších jazycích se zakládá asi na stejných argumentech jako oprávněnost příkazu go to. Oba mají umožňovat efektivní implementaci nestandardních objektů a nestandardních operací.

Privátní typ zavedený v Adě má se chápat ve dvou smyslech : (a) Typ je vyspecifikován v privátní části modulu a pro uživatele modulu je skryt, čímž se umožňuje implementovat nové datové abstrakce. (b) Libovolný typ, neznámý autoru generického podprogramu představuje nejobecnější generický typ.

Typy je vhodné ještě rozlišovat na standardní a nestandardní typy. Každý jazyk má své standardní typy, v jazyce Ada jsou to zmíněné základní typy kromě typu definovaného uživatelem a některé generické typy.

5. Procedury

Jsme u nejzákladnějšího a současně nejproblematičtějšího pojmu. Wirth /4/ navrhuje považovat za základní pojem akci a příkaz považovat za opis akce. Nakonec ale přiznává, že "procedura je jeden z mála základních pojmů programování, jehož zvládnutí má rozhodující vliv na styl a kvalitu práce programátora". Protože se jedná o základní pojem, nelze proceduru formálně přesně definovat pomocí žádného jiného pojmu.

Proceduru chápeme jako posloupnost akcí. Ve výrazech "procedurální jazyk", "neprocedurální jazyk", "snižování procedurálnosti" přívlastek "procedurální" zdůrazňuje přítomnost sledů příkazů, operací, akcí, a pod. Konečným efektem procedury bývá změna stavu objektů, na rozdíl od funkce, jejíž efektem a výsledkem je vypočtená hodnota.

Každá procedura obsahuje v sobě složku konstruktivní nebo i složku řízení. Výsledkem procedury je, že objekty nabývají nových hodnot nebo i určení procedury, která se má provádět jako následující.

Procedurou rozumíme jednak celou posloupnost akcí a též i určité části této posloupnosti a speciálně i jednotlivé elementární prvky této posloupnosti. Elementární prvky netriviální posloupnosti tvořící proceduru se tradičně nazývají příkazy (statements) a u nižších jazyků též operace. K tomu, aby se příkaz a operace chápaly jako procedury, je i závažný neformální důvod. Každý příkaz nebo operace jsou implementovány v jazyce nižším jako netriviální procedury. Strojové operace jsou např. implementovány jako mikroprogramy, tedy jako procedury.

Na pojem procedura bezprostředně navazují pojmy in-line procedura a out-line procedura mající základní důležitost v programování. V předcházející definici se říká, že i určité netriviální části hlavní posloupnosti příkazů jsou též procedury. Jsou to ty částí hlavní posloupnosti příkazů, které lze vyjmout z posloupnosti a vytvořit z nich novou samostatnou posloupnost příkazů, t. j. out-line proceduru. In-line procedura je každá část posloupností příkazů, kterou lze vyjmout z hlavní posloupnosti jako out-line proceduru, speciálně elementární prvky posloupností t. j. příkazy jsou též in-line procedury. Out-line procedura je samostatná posloupnost příkazů vyjmutá z hlavní posloupnosti a též celá hlavní posloupnost příkazu. K tomu, aby i celá hlavní posloupnost příkazů byla považována za out-line proceduru, je i vážný neformální důvod, a sice každá procedura je částí hierarchicky vyšší procedury, v limitním případě např. celý program je out-line procedurou hierarchicky vyšší procedury, označované jako práce (job), proces, a pod.

Výše uvedená koncepce procedur a out-line procedur se odráží v programovacích jazycích pouze částečně. Programy a podprogramy představují především kompilační jednotky, z hlediska logického mohou obsahovat celou hierarchii procedur a out-line procedur, ale též i nesouvisející procedury.

Neobsahuje-li procedura konstruktivní složky, t. j. její efekt se projevuje pouze v řízení, můžeme jí označit jako řídící proceduru. Nejelementárnější řídící procedury jsou příkazy skoku (go to, branch), volání out-line procedury (call, branch and link) a návratu (return) z out-line procedury. Obdobně můžeme mluvit o konstruktivních procedurách, je-li efektem každého kroku procedury změna hodnoty objektů. Nejelementárnější konstruktivní procedurou je příkaz přiřazení (assignement) hodnoty objektu.

Každý příkaz vyššího jazyka, jehož ekvivalentem v nižším jazyce není prostý příkaz, ale netriviální procedura, lze považovat a označit z hlediska nižšího jazyka jako standardní proceduru. Formálně přesná definice příkazu vyššího jazyka a částečně i jeho sémantika je dána algoritmem příkazu v nižším jazyce. Standard bychom měli chápat v tom nejintuitivnějším smyslu, jako dohodu a ustálenou normu a též připouštět i jeho omezenější či širší platnost. Takto chápaný standard se odráží např. v pojmech "obecný programovací jazyk" a "problémově orientovaný jazyk".

Z dosavadního vývoje vyšších procedurálních jazyků je patrno několik cest tvorby příkazů vyššího jazyka, tedy obecných postupů standardiza-

ce. Popis těchto postupů by si ale vyžadoval delší exkurzi do problematiky algolských procedur, speciálně do sémantiky parametrů algolských procedur a generických procedur. Zůstaneme proto při intuitivním chápání příkazů vyššího jazyka jako standardní procedury, což odpovídá zkušenosti každého programátora, který programoval jak v nižším, tak i ve vyšším jazyce.

6. Funkce

Praktický význam funkcí spočívá v možnosti skládat funkce. Uplatnění konceptu funkcí v programování má v podstatné míře zásluhu na pozvednutí programovacích jazyků na abstraktnější úroveň a na snížení proceduránosti jazyků.

Termín "operace" je nutné ve vyšším jazyce chápat jako variantu označení pro elementární funkce. Obdobně složené funkci odpovídá v nejbližší terminologii výraz (expression). Argumentu funkce odpovídají termíny operand a parametr. Jsou známy tři druhy zápisů funkcí: prefixový, infixový (tradiční) a sufixový (tzv. polský). Forma zápisu je ze sémantického hlediska nepodstatná.

Jak již bylo řečeno, funkci chápeme intuitivně jako výpočet hodnoty, alespoň v případě skalárních funkcí. Tuto neurčitou formulaci lze dosti dobře zpřesnit. Funkci odpovídá na nižší jazykové úrovni procedura, jejíž konstruktivní efekt je uchováván v lokálním objektu a je zajišťován přístup k tomuto lokálnímu objektu pro případ využití funkce v přiřazovacím příkazu nebo jako argument další funkce. K pojmu funkce dospíváme tedy odhlédnutím od existence lokálních objektů funkce a od fungování přístupu k lokálním objektům obsahujícím výslednou hodnotu. Dospíváme takto též k zcela konstruktivnímu pojetí programování a sice k primárnosti konstruktivních objektů.

Vysvětlíme si ještě, proč např. operaci sčítání považujeme ve vyšším jazyku za funkci, zatímco v nižším jazyku za proceduru. V nižším jazyku (např. jazyk strojů 3. generace) je výsledek operace automaticky přiřazován globálnímu objektu, zatímco ve vyšším jazyku výsledek operace nemusí být přiřazován objektu, ale operace může být součástí složené operace, resp. složené funkce.

Možnost skládání funkcí je podstatná a určující vlastnost funkcí. Implementace funkcí in-line nebo out-line a mechanizmy předávání mezivýsledků jsou z hlediska vyššího jazyka technologickou záležitostí, přenechává-

nou realizátorům překladačů nebo dokonce hardware.

O standardních a nestandardních funkcích se mluví už od začátku existence vyšších jazyků. Sémantika jednotlivých funkcí obdobně jako u procedur je dána algoritmem procedury v nižším jazyce.

7. Alternativní minimální slovník

Zdá se, že pojmy datové struktury a řídící struktury prosadili do programování čistí teoretici. Tyto pojmy byly nejpravděpodobněji vytvořeny podle vzoru matematických struktur a měly by představovat v programování obdobně silné existenciální abstrakce. Proti nim jsou námitky dvojího druhu: (a) v povědomí široké programátorské obce vyvolávají neurčité a nejednotné představy a (b) představují málo účelné abstrakce.

Datová struktura se obyčejně používá jako synonymum pro složený (composite) objekt i když do programování byla zavedena pro označení množiny (hodnot) s určitými pravidly o specifické manipulaci s prvky této množiny. Datové struktuře odpovídá tedy přibližně dvojice pojmů hodnoty a typy při odhlédnutí od objektů.

Řídící struktura je abstrakce, ve které se nepřihlíží ke konstruktivní složce procedur. Tato abstrakce se dobře uplatnila např. v teorii strukturovaného programování. Podle této teorie dobře sestavená procedura představuje tzv. (částečně) uspořádanou množinu příkazů. Se zvyšující se logickou úrovní jazyků řídící složka procedur je silně potlačována, proto na pojmu "řídící struktura" bude asi obtížné vybudovat výstižný slovník programování.

8. Slovník programátorů - praktiků

Praktici reprezentovaní např. Codasylem setrvávají v intuitivní rovině a odmítají obecnější a abstraktnější slovník. Chrakteristickým produktem této tendence je např. termín a pojem "výskyt typu věty" v databázi. Uvážíme-li, že pojem "soubor" pochází vlastně též od praktiků, potom zavedení výskytu typu věty byl zpátečnickým pokynem.

9. Závěr

Jazyk programování a speciálně jazyk manuálů programovacích jazyků je konkretizace programátorského myšlení. Proto slovník programování

si zasluhuje značnou pozornost.

V pojednání byl učiněn pokus o formulování soustavy logicko-sémantických kategorií programování, na které lze vybudovat celý slovník programování. V pojednání byl kladen důraz na zdůvodnění oprávněnosti tohoto minimálního slovníku. Jako součást tohoto zdůvodňování bylo ukázáno, jak pomocí pětice základních pojmů lze formálně částečně přesně definovat další pojmy základní důležitosti pro programování. Všechny tyto pojmy jsou neutrální k jednotlivým programovacím jazykům, protože se týkají jejich společného logicko-sémantického aspektu.

Prosazovaný minimální slovník a slovník z něho odvozený dobře vystihují podstatu programování a umožňují přesně vymezit programování jako disciplínu.

Výuka a příprava profesionálních programátorů by měla být vedena tak, aby programátor dospěl k uvedeným logicko-sémantickým kategoriím. Sémantická analýza existujících nebo nově vytvářených jazyků a softwarových produktů by měla být vedena v pojmech tohoto slovníku programování a měla by předcházet analýzu technologicko-syntaktickou.

10. Literatura

/1/ B. Russell: Logika, jazyk a věda. Nakladatelství Svoboda, Praha, 1967

/2/ N. I. Kondakov: Logičeskij slovar - spravočnik, Nauka, Moskva, 1975

/3/ Reference Manual for the Ada Programming Language, US DoD, 1980

/4/ N. Wirth: Systematické programovanie. Alfa, Bratislava, 1981

/5/ Z. Horský: Množiny a matematické struktury, SNTL, Praha, 1979

/6/ S. Lacko: Struktura dat v ASRP, Příloha MAA, 8/1983.