P R O G R A M O V A C Í J A Z Y K Y

Ing. Andrej Boldiš, Ing. Jiří Diviš

1. Úvod

V článku je učiněn pokus o charakteristiku prvního čtvrtstoletí existence vyšších programovacích jazyků. Článek čerpá především z obsáhlého přehledu J. E. Sammetové /1/. Navíc několik příkladů má představit a ilustrovat v mnoha směrech pozoruhodný jazyk ADA.

Doposud neexistuje přesná definice programovacího jazyka vyšší úrovně (nebo-li prostě programovacího jazyka). Není jasné například zda a kdy lze parametrizovatelné programy a generátory (např. RPG), nebo soustavy maker, prohlásit za programovací jazyk. Za hlavní znaky programovacího jazyka lze považovat násl. skutečnosti :

a) Uživatel jazyka nemusí znát strojové operace, nebo ani paměťovou reprezentaci objektů.
b) Jazyk je nezávislý na počítači.
c) Zápis v programovacím jazyce je v jistém smyslu "přirozený".
d) Jazyk poskytuje možnost sestavování libovolných procedur.

Mohlo by se mluvit též o logických úrovních jednotlivých jazyků, zatím se ale zdá, že by to bylo předčasné. Čtvrtstoletí existence programovacích jazyků je doba příliš krátká na krystalizaci teorie programování.

2. Stručná retrospektiva

V roce 1956 byl dán do užívání FORTRAN (J.Backus, IBM) a byly zahájeny práce na APT, jazyku pro numerické řízení strojů (D.T.Ross, MIT). Roku 1958 byla publikována zpráva o ALGOLu 58, předchůdci ALGOLu 60, který je považován za nejvýznamnější programovací jazyk vůbec. V této době byly zahájeny práce na LISPu, určeného pro aplikace v umělé inteligenci. V květnu 1959 se sformovala skupina CODASYL a urychleně vypracovala jazyk COBOL. Do tohoto období spadá i mnoho dalších méně známých jazyků. Kromě uvedených jazyků rok 1980 přežil ještě jazyk JOVIAL.

Období 1961-1970 lze charakterizovat jako období dospívání

a vítězství programovacích jazyků vyšší úrovně nad programováním v strojovém kódu. Z nových jazyků stojí za zmínku PL/I; interaktivní a on-line jazyk BASIC; SNOBOL pro zpracování řetězců a simulační jazyky SIMSCRIPT a SIMULA. Byly vypracovány standardy pro FORTRAN a COBOL.

V období 1971-1980 bylo uvedeno do užívání pouze několik významných jazyků, mezi nimi ALGOL 68, Pascal a ADA. Za zmínku stojí též dílčí teoretické výsledky, jakými jsou koncepce datových abstrakcí a funkcionálního programování a řada experimentálních jazyků jako CLU, EUCLID a SETL.

Toto poslední období se vyznačuje též krystalizací aplikačních oblastí a specializací jazyků. Např. pro rok 1977 se v /1/ uvádí násl. výčet oblastí a počtu jazyků : vědecké a numerické výpočty (20), hromadné zpracování dat (4), zpracování řetězců a seznamů (11), manipulace s formulemi (10), specializované aplikace (90), ostatní (34). V hromadném zpracování dat COBOL si udržuje výlučné postavení. Řada programovacích jazyků byla dopracována jako ANSI nebo ISO standardy (BASIC, COBOL, FORTRAN, Basic FORTRAN, PL/I, APT, MUMPS, PL/I subset) nebo jejich standardizace je v řízení (Pascal, APL, ADA).

3. Několik důležitých teoretických výsledků

3.1 Datové abstrakce

Termín datové abstrakce je v posledních létech velmi modní a byl nejdříve spojován s koncepcí "zásobníku" a operacemi PUSH a POP. Protože tyto poslední zase neměly valného použití, termín "datové abstrakce" zněl poněkud tajemně a vystředně. Ve skutečnosti schopnost abstraktního myšlení a datové abstrakce užíváme aniž bychom o tom explicitně mluvili. Horší situace nastane, když chceme zavést novou abstrakci. Jazyk umožňující zavádět datové abstrakce musí splňovat dvě podmínky :

a) Pro datové objekty se zvolí jistá struktura, případně i reprezentace, definují se povolené operace a algoritmy těchto operací.
b) Povolené operace charakterizují úplně datové objekty, t.j. uživatel nemusí se zabývat reprezentací a strukturou objektů a algoritmy operací a dokonce jazyk by měl omezit přístup k objektům a do algoritmů operací.

3.2 Ověřování správnosti programu

Jde v podstatě o to, jak ukázat nebo dokázat, že program dělá to co má dělat. Otázkou je, zda funkci programu lze specifikovat i v jiném případně vyšším jazyce, než je samotný použitý programovací jazyk. Dosavadní výsledky nejsou povzbudivé a jsou vážné důvody domnívat se, že problém je neřešitelný ze své podstaty a že tato problematika zůstane doménou pouhých akademických diskusí.

3.3 Spolehlivost software

Úměrně tomu jak svět se stal závislejším na počítačích, kvalita programů se posuzuje především podle spolehlivosti na rozdíl od dřívější efektivity a portability jazyka. Zatímco hardware dociluje až neuvěřitelnou spolehlivost, spolehlivost software je pořád stejná jako před mnoha léty. Zdá se, že ani zavedení datových typů, ani přísné metodické předpisy (např. zásady strukturovaného programování) nezajistí žádoucí spolehlivost. Řada lidí považuje programování za technickou, inženýrskou disciplinu. Jsou ale vážné důvody se domnívat, že programování je vědou velmi podobnou matematice, která doposud se ještě dostatečně nevykrystalizovala.

3.4 Funkcionální programování

Otázky, které si položil John Backus, zní: je von Neumannova koncepce počítače posledním slovem, lze se obejít bez přiřazovacího příkazu? Jsou to skutečně zásadní otázky. Backus je optimistický, většina z nás nikoliv.

3.5 Dotazovací jazyky

Dotazovacím jazykem se obyčejně rozumí prostředek pro styk s databází, vyznačující se jistou "přirozeností" a umožňující odvozování odpovědí v reálném čase. Tyto dvě skutečnosti jsou v podstatě určující pro komplexnost dotazů a technologii přípravy procesů odvozování. Ostatní logické problémy dotazovacích systémů jsou totožné s klasickými systémy.

4. Programovací jazyk ADA

Jazyk ADA je prohlašován za představitele současného stavu v programování. Jestliže je to pravda, potom se stačí seznámit s Adou a získáme obraz o současném stavu v programování. Jazyk ADA není experimentální jazyk, byl navržen na objednávku Ministerstva obrany USA v konkurenním řízení. Jazyk vyhovuje asi 100 předem stanoveným požadavkům. V připomínkovém řízení se zúčastnila stovka odborníků a institucí z 15 zemí a bylo zpracováno asi 900 připomínek a návrhů. Podle manuálu se jazyk vyznačuje značnou vyjadřovací schopností a pokrývá širokou aplikační oblast, jako numerické výpočty, systémy v reálném čase a systémové programování. Oblast hromadného zpracování dat není uváděna explicitně.

Pomocí několika příkladů pokusíme se představit základní prvky a některé význačné možnosti jazyka.

Příklad 1 (dohromady nedává smysl) pouze znázorňuje základní podobu programu. Program je procedure bez parametrů.

```
Procedure PRIKLAD_1 is
I,J:INTEGER;     -- I,J,K,L, celočíselné objekty; I,J,K proměnné
K  :INTEGER:=5; -- K, proměnná s počáteční hodnotou
L  :constant INTEGER:=22; -- konstanta nesmí být na levé straně
                          -- výrazu
X,Y:FLOAT        -- X,Y,Z,PI, reálná čísla (objekty)
Z  :FLOAT:=5L.23E-12;
PI :constant FLOAT:=3.14159_26536;
A,B:BOOLEAN;     -- boolovské proměnné A,B,C
C  :BOOLEAN:=TRUE;
P:STRING(1..10); Q:STRING:="DOBRY"; R:STRING:="DEN!"; --řetězce
begin I:=2; J:=(K+L)*I + ABS(I-K)/2; -- výrazy s celými čísly
      Y:=4.2; X:=Y**3 +(PI*Z)/24.5;  -- výrazy s reálnými čísly
      P:=Q & R; -- &=operace řetězení. P="Dobrý den!"
      if (C or I >=0) and X <=PI -- >= čti "větší nebo roven"
         then I:=0;Z:=3.2;A:=FALSE; -- <= "menší nebo roven"
         else I:=7; A:=C and_ PI> 0;
      end if; end PRIKLAD_1;
```

Procedura se skládá z deklarací a posloupnosti příkazů. Rezervovaná slova jsou psána malými písmeny a podtržena, jazyk jich obsahuje úhrnem pouze 62. Komentáře začínají dvěma pomlčkami a končí řádkem. Datové objekty mohou být proměnné nebo konstanty a

všechny musí být deklarovány s uvedením typu a případně počáteční hodnoty. Základním příkazem je přiřazovací příkaz, jehož levou stranou může být pouze proměnná, pravou stranou přiřazovacího příkazu může být výraz obsahující objekty, operace a funkce, které musí být stejného typu. Řídící struktura if-then-else je obecně známá.

Druhý příklad ilustruje koncepci package (modul, knihovna), sloužící pro definování souboru logicky příbuzných objektů, typů, podprogramů (procedur a funkcí) a operací. V našem příkladě package obsahuje všechno, co souvisí se souborem "ZAMĚSTNANCI", t.j. především struktura věty, výčtové typy a instance generických (obecných) procedur a funkcí pro vstup a výstup dat.

```
With INPUT_OUTPUT, TEXT_IO; --moduly tvořící okolí příkladu 2
Package PRIKLAD_2 is
   type POHLAVI is (MUŽ,ŽENA); --výčtový typ (enumeration type)
   type MESICE is (LEDEN,UNOR,...,PROSINEC); --zjednodušeně!
   type TITUL  is (DRSC,CSC,MUDR,RNDR,ING);
   type KCS    is delta 0.01 renge -1.00E12..1.00E12;
                  -- pevná čárka (fixed point), další reálný typ
   subtype PLAT is KCS renge 400.00..8000.00;
   type VETA is record
      JMENO:STRING(1..15); PRIJMENI:STRING(1..20);
      DEN: NATURAL range 1..31;     -- datum narození
      MESIC: MESICE; RCK: INTEGER range 1901..2099;
      X: POHLAVI; T: TITUL; Z: PLAT;
      end record;
   MAX: constant NATURAL:=2500;  -- max. počet zaměstnanců
   package I_O is new INPUT_OUTPUT (ELEMENT_TYPE => VETA);
   package ENUM_IO is new ENUMERATION_IO (TITUL);
   package INT_IO is new INTEGER_IO (NATURAL);
end;
```

Deklarace package je jedním z knihovních prvků. Ke každému knihovnímu prvku se musí uvést, které další knihovní prvky jsou využity. Nemusí se uvádět pouze package STANDARD, obsahující základní typy jazyka (INTEGER,FLOAT,...) a odpovídající funkce (+, -, /, *, ...), (viz. příklad 1). Package STANDARD, INPUT-OUTPUT, TEXT-IO, CALENDAR, a další jsou tzv. predefinované jazykem. Uživatel si může definovat vlastní package jako v příkl.2.

```
With PRIKLAD2,TEXT_IO,CALENDAR;  -- okolí příkladu 3
Procedure PRIKLAD_3 is           -- program se jmenuje Příklad3
use PRIKLAD_2,TEXT_IO,CALENDAR;  -- zpřístupnění identifikátorů
type RADEK is record             -- struktura řádku tabulky
   DEN: NATURAL range 1..31; MES: MESICE;
   JM : STRING (1..15); PR: STRING (1..20);
   TI : TITUL; VEK: NATURAL range 1..99;
   end record;
type TABULKA is array (1..MAX) of RADEK;
A: VETA; B: TABULKA; I: INTEGER:=0; C: INTEGER; -- objekty
procedure TRID (T: in out TABULKA; N: in INTEGER) is -- třídění
   I,P: INTEGER;   POM: RADEK;                       -- tabulky
   begin P:= N;
      loop P:= P/2; exit when  P=0;
         for J in 1..N-P loop
            I := J ;
            loop exit when
                 T(I).PR & T(I).JM <= T(I+P).PR & T(I+P).JM;
                 POM:=T(I); T(I):=T(I+P); T(I+P):=POM; --výměna
                 I := I-P; exit when  I<1;
            end loop; end loop; end loop; end TRID;

begin     -- začátek procedurální části programu
C:=CLOCK().YEAR  -- hodnota fce CLOCK, package CALENDAR
OPEN (VSTUP,"ZAMESTNANCI"); -- OPEN,READ,CLOSE - package I_O
while not END_OF_FILE (VSTUP) loop  -- package I_O, příklad 2
   I:=I+1; exit when  I> MAX;
   READ (VSTUP,A);
   B(I).DEN:=A.DEN; B(I).MES:=A.MESIC; B(I).JM:=A.JMENO;
   B(I).PR:=A.PRIJMENI; B(I).TI:=A.T; B(I).VEK:=C+1+A.ROK;
end loop; CLOSE(VSTUP);
TRID (B,I); -- volání procedury; B,I aktuální parametry
PUT("KALENDAR NAROZENIN NA ROK"); PUT(C,5);
for J in 1..I loop NEW_LINE; PUT(B(J).DEN,2); PUT(B(J).MES,10);
   PUT(B(J).JM); PUT(B(J).PR); PUT(B(J).TI); PUT(B(J).VEK,2);
end loop;end PRIKLAD_3;
```

V příkladě 2 je uveden výčtový typ (poprvé zaveden v jazyce Pascal), jako nejobecnější diskrétní typ. Speciální diskrétní typy jsou BOOLEAN, INTEGER a CHARACTER. Výčtový typ je vlastně konečná množina hodnot danná výčtem. Interní zobrazení hodnot je věcí jazyka, pokud uživatel nepředepíše určitou reprezentaci. Moduly I-O, ENUM-IO, INT-IO jsou instance generických modulů, které budeme potřebovat v příkladě 3. Při zřizování instancí se generické formální parametry (ELEMENT_TYPE) nahrazují aktuálním parametrem (VETA,TITUL,INTEGER).

Příklad 3 ilustruje další možnosti a hlavně představuje smysluplný proces. Funkci programu lze formulovat takto: vytisknout kalendář narozenin zaměstnanců, využít k tomu soubor "ZAMĚSTNANCI" a třídění v paměti. Klausule use umožňuje se odvolávat jednoduchým způsobem na identifikátory v uvedených modulech, jinak by muselo uvádět např. CALENDAR.CLOCK().YEAR. Druhým parametrem PUT je šířka sloupce.

Příklad 4 ilustruje jednu z nejprogresivnějších nových možností v programování, generické (obecné) podprogramy a moduly.

```
--Příklad 4: deklarace a tělo generické procedury SORT
generic   -- generické formální parametry
   type PRVEK is private;
   type TABULKA is array (INTEGER range <>) of PRVEK;
   with function SPRAVNE_PORADI(X,Y:PRVEK) return BOOLEAN;
procedure SORT (T: in out TABULKA; N:INTEGER); -- specifikace p.
-- následuje tělo procedury (implementace p.)
procedure SORT (T: in out TABULKA; N:INTEGER) is
   I,P: INTEGER; POM: PRVEK;
   begin P:=N;
      loop  P:=P/2;
         exit when P=0;
         for J in 1..N-P loop
            I:=J;
            loop exit when SPRAVNE_PORADI(T(I),T(I+P));
                 POM:=T(I); T(I):=T(I+P); T(I+P):=POM;
                 I:=I-P; exit when  I< 1;
            end loop; end loop; end loop; end SORT;
```

Generický podprogram je analogií assemblerovských maker. V našem příkladě je implementová obecná procedura třídění v paměti.

Generický typ private bude dodatečně zpřesněn uživatelem procedury. Range <> je libovolný celočíselný typ. Generický formální podprogram (with function ...) v našem případě musí si sestavit uživatel.

Příklad 5 ilustruje použití generického podprogramu.

```
With PRIKLAD_2, TEXT_IO, CALENDAR; -- Příklad použití generické
Procedure PRIKLAD_5 is                -- procedury SORT z př. 4
-- opíšeme PRIKLAD_3 až do "procedure TRID"
procedure TRID is new SORT (PRVEK=> RADEK,TABULKA=> TABULKA,
   SPRAVNE_PORADI=> SP) -- instance gener.p. SORT
function SP(X,Y:RADEK) return BOOLEAN is  -- aktuální fce pro
                                          -- SORT
   begin if X.PR & X.JM <= Y.PR & Y.JM
            then return TRUE;
            else return FALSE;
         end if; end SP;
begin    --- z a č á t e k  procedurální části programu
         --  opíšeme zbytek příkladu 3
```

Knihovními prvky jsou: deklarace podprogramu, tělo podprogramu, deklarace modulu a tělo modulu. Možnost oddělení deklarací od vlastní implementace představuje důležitý moment v metodologii programování.

Moduly (packages), generické moduly a podprogramy, tasks (paralelní procesy, o kterých zde nebyla zmínka) a možnost implementace datových abstrakcí (privátní typy) jsou zcela novými prvky v programovacích jazycích.

5. Budoucí trendy

5.1 Přirozený jazyk

Dorozumívání s počítačem v přirozeném jazyce je zatím snem a touhou, nicméně pracuje se na tom a je předmětem matematické lingvistiky. Problematika těsně souvisí s obecnou reprezentací znalostí a dalšími teoretickými disciplinami. V dohledné budoucnosti nelze očekávat řešení netriviálních dotazovacích systémů.

5.2 Vyšší formalizované (umělé) jazyky

Toto téma je od roku 1960 předmětem zatím bezúspěšných výz-

kumů. Jazyk ADA a specielně jeho generické podprogramy a moduly představují nový prostředek pro budování právě takovýchto vyšších formalizovaných jazyků v základním rámci jazyka (Pozn.: PARAM/II, o kterém bylo referováno před rokem, lze realizovat jako generické procedury a funkce). Lze očekávat, že těžiště experimentování se posune právě do oblasti generických podprogramů.

5.3 Méně procedurální jazyky

Rozhodujícím přínosem prvních programovacích jazyků (FORTRAN, ALGOL, COBOL) bylo snížení procedurálnosti ve srovnání se strojovým kódem (ASSEMBLER) asi o 1 řád, především zásluhou zavedení výrazů (složených funkcí). Toto snížení procedurálnosti se týká pouze základní úrovně objektů a operací, t.j. skalárních. V současné době dokážeme formulovat i vyšší datové struktury, nedokážeme ale zatím přistupovat k nim jako k abstraktním objektům. Lze se domnívat, že právě zde je veliká rezerva a snižování procedurálnosti je zcela realistická cesta, kterou se další vývoj programování bude ubírat. Naproti tomu úplné potlačení procedurálnosti (tzv. deskriptivní a funkcionální jazyky) je nereálné.

5.4 Zdokonalování informačních systémů a dotazovacích jazyků

V blízké budoucnosti se podstatně změní náš přístup k problematice databází, systémů pro řízení databází a dotazovacích jazyků. Především se vykrystalizují technické a logické aspekty problematiky. Lze se domnívat už teď, že logické aspekty databází jsou totožné s klasickými informačními systémy a že lze formulovat společný základní a referenční jazyk pro značně širokou třídu softwarových systémů. Datové modely a modelování není specifická problematika pouze databázová, resp. počítačové simulace. Lze předvídat, že entitno-relační model se nakonec ukáže jako logicky nejkonzistnější.

5.5 Využívání vyšších programovacích jazyků pro systémové a real-time úlohy

Kritické požadavky na čas a paměť se částečně zvládnou ma-

ximálně efektivní kompilací a reprezentací objektů, jako např. v Adě. Problematiku systémového programování úloh v reálném čase včetně distribuovaných systémů lze pokrýt jednotným jazykem, rozdílnosti budou v operačních systémech a v hardware.

5.6 Ověřování správnosti programů

Lze předvídat, že ověřování správnosti se nestane nikdy prakticky použitelné.

5.7 Nadvýroba jazyků

Jazyk (přirozený, formalizovaný i programovací) to je náš myšlenkový svět. Nový jazyk pro jeho tvůrce je předmětem jeho seberealizace. Nový jazyk ale si vyžaduje od uživatele v lepším případě restrukturaci jeho myšlenkového světa, v horším případě vyžaduje od něho proniknutí do úplně nového světa myšlenek a především abstrakcí. Proto programátoři nejraději používají jazyk, který se naučili jako první. Takto lze vysvětlit nadvýrobu jazyků a současně odpor k novým jazykům. Lze předvídat, že COBOL, FORTRAN a BASIC budou i nadále nejpoužívanějšími jazyky.

6. Závěr

Jsme svědky a účastníky bouřlivého vývoje programování. Základní teoretické problémy se zatím pouze mlhavě rýsují a zdá se, že jazyk patří k ústředním problémům teorie i praxe programování a počítačů vůbec.

7. Literatura

/1/ Sammet, J.E.: An overview of high-level languages. Advances in Computers, Vol. 20, Academic Press, New York, 1981.

/2/ Reference manual for the ADA programming language. Proposed standard document, United States Departement of Defense, July, 1980.