RNDr.Zdeněk BOTEK

Katedra aplikované matematiky přírod.fakulty
UJEP Brno

KONVERZAČNÍ (STRUKTUROVANÉ) PROGRAMOVÁNÍ

1. Ú v o d

Limitujícím faktorem rozvoje výpočetní techniky v době před 15 - 20 lety byla především kapacita paměti stávajících počítačů. V [12] je uvedeno, že práce programátora na běžném programu byla v roce 1960 efektivní už tehdy, když ušetřil za jeden den práce dva příkazy vytvářeného programu. V roce 1970 stoupá tento počet na pět, 1975 na třicet a v současné době je denní práce programátora efektivní, zlepší-li program o 200 příkazů. Z toho je zřejmé, že se limitující faktor přesunul přes problematiku strojového času v 70. letech na otázku programátorské kapacity. Je proto přirozenou snahou rozšířit tuto kapacitu. Současný stav hardwaru uspokojivě řeší problém nabídkou malých počítačů vybavených terminály pro přímý přístup uživatele k počítači. Jaké komunikační prostředky dává k dispozici software těchto počítačů se pokusíme shrnout v tomto článku.

Rozšíření programátorské kapacity směřuje především na ty okruhy pracovníků, kteří ve své oblasti řeší problémy na tolik složité, aby bylo efektivní svěřit je počítači a také natolik opakované, aby bylo výhodné vytvořit pro ně program. Těmto tzv. aplikačním programátorům je ovšem nutné poskytnout prostředek, který je možné zvládnout bez rozsáhlé programátorské přípravy, ale který je na druhé straně natolik silný, aby byl schopen složité úlohy řešit. Uvedeným požadavkům plně odpovídají konverzační jazyky.

Konvenční programovací jazyky se svým rozšířením dostaly do podvědomí široké oblasti technických pracovníků. Nastupující konverzační způsob práce je ale nutí k přizpůsobení se novému trendu. Jazyky typu PASCAL, ALGOL či FORTRAN nyní používají výhodnější způsoby kompilace, jsou zpracovávány a editovány konverzačními prostředky. Efektivitou nasazení konvenčních programovacích jazyků v konverzačním prostředí se zabývá druhá část článku.

Konverzační jazyky zaměřené na numerické výpočty jsou známé již delší dobu. Nástup strukturovaného programování však zasáhl i tuto oblast. Konverzační jazyky jsou doplňovány řídicími strukturami, které umožňují vytvářet komplexnější posloupnosti výpočtů (tedy skutečný program), ale zároveň uchovávají sílu konverzačního jazyka. Touto problematikou se zabývá třetí část.

Moderní koncepce programovacích jazyků pro konverzační způsob práce si už ve svých cílech kladou strukturovanost, přehlednost a současně snadné vyjádření problémů. Snaží se poskytnout prostředek pro zapsání programu nikoliv v "prostředí" programovacích jazyků (formátování, deklarace, práce s prvkem nikoliv se strukturou...), ale přímo v "prostředí" problému (práce s celým vektorem či maticí, implicitní deklarace, vstupy i výstupy celé struktury jedním příkazem ...). Na jazyku CLIC budeme tuto koncepci demonstrovat ve čtvrté části článku.

2. Konvenční programovací jazyk v konverzačním prostředí

Řešme úlohu v jazyku FORTRAN nebo PASCAL např. na počítači PDP 11/34, který umožňuje zpracovávání úloh z terminálu. Po překladu zadaného programu kompilátor hlásí chybu, kterou je snadné odstranit úpravou jediného nebo několika řádků. Budou náklady na rekompilaci uměrné prováděné změně nebo se provede nová kompilace ? Odpověď je závislá na použitém způsobu překladu. Při generačním způsobu vyvolá každá změna překlad celého programu, běh výsledného kódu je ale efektivní. Při interpretaci je možné bez zvýšených nákladů měnit tvar libovolné části zdrojového programu, interpretovaný běh programu je ovšem pomalejší. Nastává boj mezi efektivitou rekompilace a rychlostí běhu výsledného kódu, řešení

v konkrétních systémech je řízeno prioritou požadavků. Jako spojující článek mezi dvěma diametrálně odlišnými způsoby kompilace se často užívá tzv. inkrementální kompilace. Jejím výsledkem je mimo výsledný kód také vnitřní representace logické struktury programu (tzv. skeleton). Tento způsob překladu umožňuje provádět změny na úrovni zdrojového jazyka bez rekompilace celého programu. Vyhodnocování takto přiloženého programu řídí monitor, který pracuje nad skeletonem a poskytuje programátorovi všechny výhody konverzačního zpracování programu (běh části programu, čtení a změna hodnot proměnných při přerušení...).

Například v[2] je popsán inkrementální kompilátor jazyka ALGOL 60. Za základní jednotku kompilace, representace i modifikace je po nepatrných úpravách jazyka brán příkaz. Každý příkaz zdrojového textu je v paměti representován informačním souborem, který mimo typ příkazu, zdrojový a výsledný kód obsahuje také množinu strukturálních ukazatelů. Tyto začleňují příkaz do seznamu příslušné urovně. Modifikace takto přeloženého programu je zadávána udáním IN (nebo OUT) příkazů a implementována vytvořením nových (nebo vymazáním neplatných) informačních souborů a jejich začleněním do struktury programu úpravou ukazatelů. Je-li při modifikaci respektována syntaktická definice příkazu je zaručeno, že nebude porušena globální syntaktická správnost programu, neboť může být měněn pouze celý příkaz. Z toho plyne také skutečnost, že se rekompilace téměř nedotýká příkazů, kterých se netýká modifikace. Tento způsob manipulace s programem vyznívá ze stručného popisu jako další omezování a komplikace pro programátora. Ve skutečnosti ale vyžaduje jen větší programátorskou kázeň (kterou již částečně zajišťuje strukturované programování) a přináší značné zisky při rekompilaci již průměrné délky. V [3] jsou uvedeny následující výsledky časového testu pro pět programů v jazyce JOVIAL zpracovávaných inkrementálním kompilátorem.

Číslo programu	1	2	3	4	5	
Počet příkazů	44	273	468	685	884	
Kompilace programu	21	81	146	216	296	sekund
Částečná rekompilace	9	21	26	31	36	sekund
Úspora času	57	75	82	85	88	%

U řádkově orientovaných jazyků nepřináší inkrementální kompilace zvláštní problémy. Každý příkaz je umístěn na zvláštním řádku. Rekompilace se týká jen řádků zadaných programátorem, neboť význam každého příkazu s vyjímkou deklarací a návěští je obvykle nezávislý na svém okolí. Návěští se většinou zpracovávají nepřímo přes tabulku a pouze modifikace deklarací vyvolává rekompilaci všech příkazů v působnosti modifikované deklarace.

Inkrementální systémy nacházejí s rozvojem konverzačního způsobu komunikace programátora s počítačem časté uplatnění - viz například [5] , [6] .

3. Konverzační jazyky

Nový způsob přístupu k počítači vyžaduje také nové programovací jazyky. Vzniká velké množství konverzačních jazyků zaměřených na použití ve speciálních oblastech. Mezi nejznámější patří BASIC, APL, JOSS, LCC, PPL, CPS. Jejich nevýhodou je úzce specializovaná struktura dat i příkazů, což neumožňuje řešení komplexních úloh, tedy nelze popsat složitější algoritmus jedním programem. Funkci chybějících řídicích struktur a datových typů musí nahrazovat sám programátor. Značná obliba a rozšířenost těchto jazyků však svědčí o jejich přednostech. Specializace umožňuje jednoduchou syntaxi a z toho plynoucí snadné zvládnutí a použití v praxi nejen pro programátory, ale také pro zaškolené pracovníky specializovaných oborů. Zaměření jazyků na konkrétní oblast dává podmínky pro jednoduché vyjádření problémů, jejichž zápis je v konvenčních programovacích jazycích složitější. Nebudeme se zabývat přesným popisem konkrétních konverzačních jazyků. Uvedeme jen charakteristické rysy představitelů různých skupin. Dělení bude vycházet z použité techniky konverzace s počítačem jak ve fázi sestavování, tak ve fázi testování a modifikace programu. Do první skupiny zařadíme řádkově orientované jazyky, u nichž je každý řádek samostatným příkazem a případné seskupování příkazů se provádí programovacími prostředky. Nejrozšířenější jazyk tohoto druhu je BASIC. Dále sem patří další fortranovské jazyky QUIKTRAN a XTRAN nebo z jazyka PL/1 vycházející konverzační jazyk CPS. Druhá skupina používá modulární techniku částí a kroků

Uvedeme z této třídy konverzační jazyk JOSS, dále sem patří jazyky MUMPS, LCC, TELCOMP a další. Ve třetí části si všimneme charakteristických rysů a vývojových trendů nejznámějšího z konverzačních jazyků -APL.

B A S I C

Je nejznámějším zástupcem skupiny konverzačních jazyků, u nichž zůstává důraz na programovacích prostředcích a prostředky konverzační umožňují pouze nutnou komunikaci s počítačem. Často je využíván jako přístupnější předchůdce jazyka FORTRAN.

V jazyku je možné používat jednoduché nebo indexované proměnné (jeden nebo dva indexy) v základní verzi jen číselného typu. Deklarace pole je obvykle povinná až od určité hranice. Aritmetické příkazy a standardní funkce zůstávají téměř beze změny, navíc je jen možnost operací nad maticemi či vektory uvedené klíčovým slovem (např. MAT X = Y + Z nebo MAT READ U). K programovacím prostředkům patří podmíněný a nepodmíněný skokový příkaz a příkaz cyklu s koncovou závorkou NEXT. Strukturace programu je obdobná jako v jazyku FORTRAN. Základní verze udává následující příkazy pro zpracování celého programu: NEW, RUN, příp. EXECUTE pro část textu, STOP, CONTINUE, LIST, SAVE. Podle typu počítače a oblasti nasazení jsou k dispozici klávesnice případně příkazy pro práci s řádky. Obvykle se používají následující: DELETE, INSERT, RECALL, FETCH. Při testování programu může programátor žádat krokové provádění programu příkazem STEP, při přerušení může zjistit hodnoty proměnných příkazem DISPLAY, případně výkonným příkazem nastavit novou hodnotu proměnné. Příkaz TRACE zajistí průběžné podávání zpráv o všech změnách hodnot proměnných či přerušení lineární posloupnosti v provádění příkazů. Navíc je možné používat příkazy pro práci s vnější pamětí a pro ovládání přídavných zařízení.

J O S S

Program v jazyku JOSS je rozdělen do částí a kroků (part a step). Každý krok je označen číslem tvaru p.q kde p,q jsou přirozená čísla. Hodnota p vyjadřuje pořadové číslo části programu, hodnota q pořadové číslo kroku v p-té části. Základní prostředky umožňují

provádět operace se skaláry (číselného nebo logického typu) nebo polí (pouze číselný typ). Rozšířením běžných fortranovských možností je alternativní přiřazovací příkaz a funkce SUM, která nahrazuje cyklus. Protože je možné jednoduše vytvářet složený příkaz začleněním příkazů do jedné části, programovací prostředky jen doplňují tento prostředek jednoduché a přehledné strukturace programu. Počet opakování příkazu cyklu je možné určit zadáním konkrétní hodnoty nebo vyjmenováním proměnných cyklu. Objektem typu formula je možné definovat jednopříkazové funkce. Jako v každém konverzačním jazyku i v JOSSu rozlišujeme příkazy přímé a nepřímé. Označené příkazy jsou chápány jako nepřímé a interpretují se až tehdy, jsou-li vyvolány např. DO STEP 3.4 DO PART 2. Zrušení už sestaveného kroku, části nebo programu je realizováno příkazem DELETE se specifikací dotyčného objektu.

K testování sestaveného programu slouží příkazy STOP, GO, DONE, QUIT a CANCEL. Příkaz STOP je možné použít pouze nepřímo a zajistí přerušení běhu programu na zvoleném místě. Naopak příkaz GO je možné zadat pouze přímo a jeho důsledkem je pokračování běhu přerušeného programu. Funkci známého příkazu EXIT zde plní příkazy DONE (pro část) a QUIT (pro program).

Z uvedeného popisu je patrné, že konverzační jazyk JOSS nachází své uplatnění pro specializované numerické výpočty. Přebírá mnoho výhod jazyka FORTRAN, mezi jeho přednosti lze počítat snadné zvládnutí a použití.

A P L

Práce z terminálu ovšem přináší některé příjemné stránky, které se v popsaných koncepcích jazyků příliš neprojevily. Vznikají proto také speciální konverzační jazyky, které neberou v úvahu von Neumannovskou koncepci výstavby programovacích jazyků, ale snaží se poskytnout prostředky pro řešení skutečných problémů v té které oblasti. Nejsou obecným nástrojem na řešení všech algoritmických problémů, ale rozšiřují přirozeným způsobem schopnosti člověka v žádaném směru.

Typickým představitelem je jazyk APL, který umožňuje zadávat jednoduchým způsobem složité operace jak nad čísly tak nad mati-

cemi. Jeho vznik však nebyl provázen pouze nadšením aplikačních vědců, ale také silnými protesty teoretiků. Elegance jazyka plyne především z rozšíření operátorů na všechny používané datové struktury, což dovoluje globální přístup k problému na rozdíl od předchozího zpracování prvek po prvku. Dalším styčným bodem diskusí o APL je neobvyklé pravidlo vyhodnocování zprava doleva bez priorit. Skutečnost, že pravý argument je hodnota celého následujícího výrazu, přináší programátorovi i určité výhody, neboť každý výraz je čitelný zleva doprava (první levá funkce je hlavní, následující je hlavní v jejím pravém argumentu atd.) a také zprava doleva (pořadí vyhodnocování). Skalární proměnné i proměnné typu pole mohou nabývat číselných nebo znakových hodnot. Zajímavá a dodejme velmi užitečná je možnost použít na pravé straně přiřazení libovolný typ proměnné.

Mimo běžné aritmetické, logické a relační operátory jsou navíc k dispozici například symboly pro výpočet faktoriálu, minima, maxima nebo zbytku po dělení. Pro práci s polem se výhodně používají operátory ⍳ (iota) a ⍴ (rhó). Monadické iota generuje vektor zadané délky z přirozených čísel, dyadické iota určí první výskyt zadaného čísla ve vektoru. Operátor rhó v monadickém použití určí rozmár proměnné, v dyadickém zápisu X⍴Y znamená generování proměnné o rozměrech X opakováním hodnoty Y. Pro práci s vektory je určen také operátor redukce /. V dyadickém významu vybírá jen některé prvky vektoru podle zadané mřížky, v monadickém použití má význam aplikace bezprostředně zleva předcházejícího operátoru na všechny složky vektoru.

Příklady:		znamená	
⍳3			1 2 3
2 3 5 1 4 ⍳3			2
0 1 0 1 0 / 5 6 7 8 9			6 8
+ / 1 2 3 4			10

Skalární součin vektorů je možné tedy zapsat +/AxB. Na ilustraci základních prostředků si vyřešme úlohu určení všech prvočísel menších než sto. Potřebujeme ještě dva další operátory:

A | B zápis pro zbytek po dělení B hodnotou proměnné A

A ↓ B proměnná B po škrtnutí prvních A složek.

Řešení:
```
X ←   ⍳ 100
Y ←  0≠ (2|X)×(3|X)×(5|X)×7|X
Z ←  2 3 5 7, 1↓Y/X
```

Struktura řádkování se omezuje na zápis přiřazení, podmíněného příkazu nebo cyklu na jeden řádek. Početnost a rozmanitost operátorů ale dovoluje zapsat v jazyku (i když ne vždy srozumitelně) téměř všechny konstrukce konvenčních programovacích jazyků. Např. funkce může být definována s dvěma, jedním nebo žádným argumentem. Toto omezení je však formální, neboť použitím pole za argument je dán počet argumentů počtem prvků pole. Podobně lze s trochou fantazie a zkušeností pomocí operátoru redukce a nepodmíněného příkazu simulovat fortranovský podmíněný příkaz:

```
→ (N=0, N<0, N≥-1)   / 5 7 9   - po vyhodnocení podmínek
                                 nastane například
→ ( 1    0    0 )   / 5 7 9   - operátor redukce zajistí
→ 5
```

Konverzační prostředky samozřejmě opět umožňují zahajovat a končit konverzaci, uchovávat nebo znovu povolávat dříve sestrojené funkce, modifikovat je, pořizovat výpisy hodnot proměnných atd. Blíže si všimneme dvou prostředků pro testování. Jsou to přerušovací (stop) a sledovací (trace) vektor. První zajistí přerušení interpretace po vykonání určených řádků, druhý zaznamenává hodnoty výrazů na těchto řádcích. Jejich definici lze zapsat následovně:

```
STOP VECTOR     S ∆ <function name> ← <vector>
TRACE VECTOR    T ∆ <function name> ← <vector>
```

Použijme sledovací vektor na funkci P s argumentem X.

```
     ▼ P X                      T∆P ← 1 2
[1]  Y ← X + 1                  P ⍳ 3
[2]  Z ← + / Y * 2          P[1] 2 3 4
[3]                         P[2] 29
```

Úspěchy strukturovaného programování zasáhly i konverzační jazyk APL. Mnoho autorů se snaží doplnit jazyk o konstrukce,

které by odstranily jeho nečitelnost. Většina návrhů však porušuje v určitém směru filosofii jazyka nebo je komplikovaně realizovatelná. Poměrně úspěšným řešením se ukazuje být jazyk APLGOL. Doplněním algolovských řídicích struktur do jazyka APL vzniká přehledný, strukturovaný a také výrazově silný konverzační jazyk. Nové řídicí struktury je možné samozřejmě zapsat také v APL, srovnejme například zápis podmíněného příkazu a cyklu:

APLGOL	APL

IF A≠0 THEN
B←C − A ;
ELSE
D←B + 4;
	→(~A≠0) / Q1
	B←C − A
	→Q2
	Q1 : D←B + 4
	Q2 :

WHILE A≤B DO ... END	Q1 :→(~A≤B) / Q2 ... →Q1 Q2 :

Použití řídicích struktur však podstatně usnadní větvení a čitelnost programů. Jedná se především o přehlednost dynamické struktury programu při libovolné hloubce vnoření příkazů (což je v APL velmi složité zapsat a už téměř nemožné přečíst).

Mimo uvedené dvě struktury jsou v původním návrhu jazyka APLGOL dále řídicí struktury cyklu REPEAT a FOR, příkaz CASE a klíčová slova BEGIN, END pro seskupení příkazů. Podmínka je nahrazována výrazem v APL s oborem hodnot 0, 1.

Konverzační jazyk APLGOL byl navržen v roce 1975. Implementace jazyka na počítači IBM 360/75 obsahuje 12 procedur o celkovém rozsahu 250 APL - příkazů a byla vypracována jedním člověkem za jeden měsíc. Za 1 minutu překládá kompilátor asi 100 APL - příkazů. Praxe ukazuje, že síla operátorů jazyka APL je vhodně doplněna řídicími strukturami. APLGOL nepředstavuje novou technologii programovacích jazyků. Je to snaha poskytnout prostředky strukturovaného programování bez narušení filosofie jazyka APL.

4. Strukturovaný konverzační jazyk

Konverzační jazyky uvedené v předešlé kapitole vycházejí - především na základě požadavku jednoduchosti - z řádkové struktury programu. Toto odráží orientaci starších programovacích jazyků (autokódy, FORTRAN), ale v pozdějších koncepcích se více prosazuje požadavek přehlednosti a strukturovanosti. Tato skutečnost se plně odráží v návrhu konverzačního jazyka CLIC (A conversational language for interactive computing). Autoři se snaží spojit prostředky algolovských jazyků (úplný podmíněný příkaz, přehledná struktura cyklů, explicitní složený příkaz) s funkcemi výhodnými v numerických výpočtech (aritmetické operace nad poli, vstupy a výstupy i složitějších struktur jedním příkazem a možnost práce s proměnnými bez deklarací a formátování).

Základní prostředky jazyka se příliš neliší od jiných konverzačních jazyků. Mimo pole reálného a booleovského typu lze pracovat s komplexním a znakovým polem. Jednopříkazové funkce se definují pomocí tzv. metavýrazu. Vedle klasických operací nad poli jsou k dispozici tzv. parametrické operace. Používají se tehdy, mají-li být prvky jedné nebo několika matic zpracovány systematickým způsobem nebo stejné skalární operace opakovány s různými parametry. Dále jsou k dispozici některé nové operátory (např. : = : pro výměnu hodnot proměnných, A⍴? určí počet prvků v A, A ⍴ ? vyčíslí počet dimenzí A).

Úkol čitelnosti a strukturovanosti programu je v jazyku CLIC řešen explicitním složeným příkazem jako základním programovacím prostředkem. To sebou přináší možnost seskupit několik příkazů do jediného složeného a současně omezení rozsahu platnosti použitých identifikátorů. Odsazování vnořených příkazů jak při sestavování (programátorem) tak při výpisu programu (systémem) je dalším důležitým znakem, kterým se jazyk CLIC liší od řádkově orientovaných konverzačních jazyků a naopak podobá svým strukturovaným vzorům. Do jazyka je zařazen algolovský podmíněný příkaz i několik typů cyklů. I když lze použít příkaz GOTO, k jeho minimalizaci slouží příkazy QUIT a CANCEL.

V jazyku lze definovat funkci s n parametry. Uvedeme příklad funkce pro výpočet největšího společného dělitele a její volání.

```
: ! GCD ( M, N )  '('
  1: DO '('
     1: M ///= N
     2: IF M = 0
     3: THEN GOTO EXIT
     4: M :=: N
     5: ')'
  2: EXIT: N
  3: ')'

: GCD ( 7, 35 )  ,  GCD ( 5733, 143 )
   7     13
:
```

Sestavení programu a jeho modifikaci si budeme ilustrovat na úloze přeskládání prvků vektoru. Zápis algoritmu v jazyku CLIC bude následující:

```
! FIND ( A, F )  '('
    ! M, N
    1: -- PŘESKLÁDÁNÍ PRVKŮ VEKTORU --
    2: M := 1 ! N := A&?  -- A&? JE VELIKOST A --
    3: WHILE M < N DO '('
       ! R, I, J
       1: R := A(F) ! I := M ! J := N
       2: WHILE I<=J DO '('
          1: WHILE A(I) < R DO I :=I + 1
          2: WHILE R < A(J) DO J -= 1
          3: IF I <= J
          4: THEN '('
                ! W
                1: W := A(I) ! A(I) := A(J) ! A(J) := W
                2: I := I + 1 ! J := J - 1
                ')'
          ')'
        3: IF F <= J
        4: THEN N := J
        5: ELSE '('
           1: IF I <= F
           2: THEN M := I
           3: ELSE GOTO L
           ')'
       ')'
    4: L:
    ')'
```

Upozorníme především na značku !, která specifikuje hlavičku funkce nebo lokální proměnné na příslušné úrovni.

Konverzační prostředky jsou determinovány použitou technikou strukturace programu. Je možné vypsat celý program příp. kteroukoliv vnořenou část. Při výpisu určité úrovně je možné nechat vypsat jen hlavičky všech příkazů a tím potlačit výpis případných vnořených příkazů. V následujícím příkladu je zadán výpis 3.příkazu z předchozího programu s potlačením a pak úplný text jeho 2. vnořeného příkazu.

```
:? FIND /3/ ...
    1: R := A(F) ! I := M ! J := N
    2: WHILE I<=J DO '('
    3: IF F<= J
    4: THEN N := J
    5: ELSE '('

:? FIND /3/ ALL 2
    2: WHILE I<=J DO  '('
       1: WHILE A(I) < R DO I := I + 1
       2: WHILE R <A(J) DO J -= 1
       3: IF I<= J
       4: THEN '('
            ! W
            1: W := A(I) ! A(I) := A(J) ! A(J) := W
            2: I := I + 1 ! J := J - 1
            ')'
       ')'
```

Obdobně se pracuje při modifikaci textu. Úpravy lze provádět pouze na právě otevřené úrovni, je ale také k dispozici operátor ? pro informaci o spojení úrovně s hlavním programem. Pokusme se sledovat dialog uživatele s počítačem o předešlém programu.

```
:! FIND /3/2/ '('
        5: ! ALTER 1
ALTER:  S |I := I + 1| = |I += 1|
        5: 4: THEN A(I) :=: A(J) ! I += 1 ! J -= 1
        5: ? ...
        1: WHILE A(I) <R  DO I += 1
        2: WHILE R <A(J)  DO J -= 1
        3: IF I <= J
        4: THEN A(I) :=: A(J) ! I += 1 ! J -= 1
        5: ?
```

```
! FIND (A, F) '('
    ! M, N, L
    3: WHILE M<N  DO  '('
        !  R, I, J
        2: WHILE I<= J  DO  '('
            5: ')'
```

Návrh a implementace jazyka je výsledkem práce tří lidí po dobu tří let. Nelze celkovou koncepci bez dobrozdání praxe hodnotit, ale mnohé konstrukce je možné považovat za velmi progresivní ve vývoji konverzačních jazyků.

LITERATURA :

1. KUPKA, I. WILSING,N: Dialogsprachen. Teubner Studienbücher Informatik, Stuttgart 1975

2. BOTEK,Z: Incremental compilation for languages with nested statement structure, SCRIPTA UJEP Brno, 1980

3. BOTEK,Z. ŠTETINA,I. TOMAN, P.: Interaktivní systémy Sborník SOFSEM 79, str. 249 - 282

4. BACKUS,J.: Can Programming Be Liberated from the von Neumann Style? CACM, Vol. 21, 1978, č.8, str. 613 - 637

5. ATKINSON,L.V.: CONA - A Conversational ALGOL System. Software - Pract. and Exp., Vol. 8, 1978, str. 699 - 708

6. REES, M.J.: SOBS - An Incremental BASIC System. Software - Pract. and Exp., Vol. 7, 1977, str. 631.- 643

7. BOTEK,Z.: Konverzační jazyky. Zprávy semináře o informatice ÚVT UJEP Brno, 1979, č. 3

8. BOTEK,Z.: Blokově orientované konverzační jazyky, Zprávy semináře o informatice ÚVT UJEP Brno, 1979, č. 14

9. BRENDEL,W.: Syntaxgesteuerte Programmierung und inkrementalle Compilation. Tagungsband der 7. Jahrestagung der GI, 1977.

10. KOLSKY, H.G.: APLGOL - a Structured Programming Language for APL. LN in CS 23 - Programming Methodology (1975).

11. GEORGES,J.: Design aspects of a language for interactive computing. Proc. Interactive systems, London 1975.

12. VALÁŠEK,M: Výhody a nevýhody interaktívneho využívania počítačov. MAA 1978,č.2