Consiglio Nazionale delle Ricerche

Supplemento Maggio - Giugno 1982

Rapporto

Elementi di raffronto tra linguaggi

Servizio Elaborazione Dati

CNUCE

ELEMENTI DI RAFFRONTO TRA LINGUAGGI

(R.Medves)

1) Capita a volte di alternare, nella programmazione, due linguaggi diversi, spesso a notevole distanza di tempo uno dall'altro.
Ci si trova allora nella difficolta' (superata invero in breve tempo) di ricordare la sintassi del primo linguaggio, essendo portati ad utilizzare ancora per inerzia le strutture di quello usato in tempi piu' recenti.
Queste tabelle dovrebbero servire a dare un'indicazione delle caratteristiche di alcuni dei linguaggi piu' noti, in modo da facilitare una ripresa, anche a distanza di tempo, dell'uso di uno di essi.
Il contenuto delle tabelle non vuole essere formalmente ineccepibile, nel senso di avere una forma generale delle singole istruzioni presentate: si sono invece mescolate forme generali (es. GOTO n) ad altre particolari e sotto forma di esempi (es. DO 3 I =1,5,2) e ad altre ancora in forma veramente ridotta (es. I..N) o parziale (es. WRITE(6,2)...).
Questo perche' abbiamo pensato che queste tabelle, rivolgendosi a lettori che gia' conoscono tali linguaggi, sarebbero state piu' utili se avessero potuto colpire la memoria e suscitare un ricordo passato (una istruzione vista piu' volte in un programma), piuttosto che la forma generale di un'istruzione vista su un manuale una sola volta.

2) Le tabelle possono anche assolvere un secondo scopo meno pratico, ma pur sempre utile: quello di avere sott'occhio una visione panoramica della sintassi di un dato linguaggio, per poter paragonare i vari elementi con quelli presentati ad esempio da un linguaggio nuovo o da implementazioni diverse di uno stesso linguaggio su varie macchine.
A volte nello studio delle caratteristiche di tali prodotti, ci si lascia prendere la mano e si rimane invischiati nella descrizione dei nuovi comandi, perdendo un po' la visione d'insieme: uno sguardo a queste tabelle puo' far cadere l'attenzione su un gruppo di istruzioni o sulla forma di un'istruzione che li' per li' si era dimenticata e che, ci ricordiamo solo ora, non abbiamo ritrovato nel nuovo prodotto o abbiamo ritrovato in forma notevolmente diversa (non solo magari dal punto di vista formale, ma anche come risultato) rispetto a quella che eravamo abituati ad usare fino ad allora: e questo puo' portare a valutare positivamente o negativamente il nuovo prodotto, ricordando che spesso per i vecchi programmatori l'abitudine ad usare un certo modo di programmazione anche grossolano ha piu' peso che non nuove possibilita' raffinate quanto si vuole, ma che impongono cambiamenti nel modo di pensare.

3) Le tabelle non vogliono certo essere esaustive sia come varieta', sia come sintassi dei linguaggi.
Ho presentato cioe' solo i prodotti piu' comuni e piu' noti agli Utenti CNUCE (per questo non compaiono altri linguaggi, come ad esempio Algol, Simula, ecc. sui grandi calcolatori e Forth, Ratfor, C-language, ecc. sui micro) e per i quali potevo avere facilmente accesso alla documentazione (come si vede le colonne relative a PL/1 e Cobol sui micro sono state lasciate in bianco, anche se questi prodotti sono implementati anche su tali macchine): la descrizione risente comunque della mia scarsa (o nulla) conoscenza di Cobol e Pascal e della mia atavica diffidenza verso gli Algol-like.
Inoltre sono descritte in modo superficiale o tralasciate molte caratteristiche specifiche, anche se importanti quali ad esempio le istruzioni di I/O sui files (sequenziali, ad indici, OPEN, CLOSE, REWIND, ecc.), le istruzioni per il trattamento degli errori (riportate nel testo semplicisticamente con ON cond ...), le istruzioni per l'allocazione della memoria (statica, dinamica, strutture, ecc.), e molte altre a cui accenneremo in una chiaccherata finale.
Lo scopo non e' quello di confrontare i linguaggi tra loro per definire il migliore o il piu' adatto ad un determinato scopo: avrei avuto una notevole difficolta' a documentarmi in un tempo ragionevole su tutte le versioni dei prodotti, le loro caratteristiche, e a notarne le differenze macroscopiche o microscopiche; piuttosto ho voluto dare solo delle indicazioni generiche sulla forma di certe istruzioni che piu' frequentemente ricorrono nei programmi.

4) Due parole in piu' sui micro calcolatori.
Oggi queste macchine hanno una diffusione sempre maggiore, anche tra i lettori di questo Rapporto; e' un fenomeno che pertanto non va ignorato e che merita fin d'ora una particolare attenzione.
Per questo nel presente articolo ci siamo soffermati spesso sulle caratteristiche dei linguaggi per tali macchine, con particolare riferimento a Basic e Pascal, scendendo anche a dare qualche suggerimento pratico di programmazione.
La situazione di questi linguaggi e', al momento, molto variabile, nel senso che non esiste uno standard universalmente riconosciuto per ciascuno dei essi: ci si puo' pertanto trovare in difficolta' quando si cerca di programmare anche con uno stesso linguaggio su macchine diverse.
Anche per questi non abbiamo voluto assolutamente fare dei raffronti accurati di tutti i prodotti in commercio: ciascun micro si differenzia dall'altro in mille dettagli e un confronto sulle caratteristiche di tutti i linguaggi ci avrebbe portato fuori strada. Abbiamo pertanto anche qui cercato di dare una visione d'insieme delle caratteristiche di base del linguaggio, rimandando comunque alla parte descrittiva dopo le tabelle alcuni

commenti e considerazioni di dettaglio sulle differenze piu' rilevanti.
In tutti i casi un manuale specifico del linguaggio particolare da utilizzare sara' sempre insostituibile e assolvera' poi il compito di spiegarne nei dettagli le caratteristiche.

5) Non sono in possesso di una bibliografia completa sull'esame comparato dei vari linguaggi di programmazione. Per le caratteristiche dei singoli prodotti si consigliano naturalmente i manuali di utilizzo specifici, mentre per argomenti correlati piu' o meno strettamente con questo articolo, mi sono capitati tra le mani:

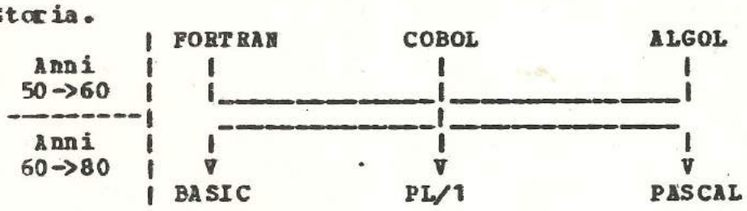
il libro:
'An Introduction to the Study of Programming Languages
(D.W.Barron - Cambridge Computer Science Text - 7 -
Cambridge University Press - 1977,1979,1980)

e due articoli comparsi su Byte:
'A High-Level Language Benchmark'
(Jim Gilbreath - Byte - Semptember 1981) e
'BASIC a confronto'
(Teri Li - Byte - January 1981)

Sul libro di Barron citato sono riportate, tra la bibliografia, le seguenti edizioni:

A Comparative Study of Programming Languages
(B.Hig,man - Macdonald/American Elsevier - 1967)
Programming Languages: History and Fundamentals
(J.E.Sammett - Prentice-Hall International - 1969)
High Level Languages-Infotech State of the Art Report No.7
(C.Boon editor - Infotech Information Ltd - 1972)
Concepts of Programming Languages
(M.Elson - Science Research Associates - 1973)

6) Storia.



TIPI DI DATI E ISTR. DICHIARATIVE		GRANDEZZA	FORTRAN-370	PL/1-370	COBOL-370	PASCAL-370	BASIC-370
RAPPR. ESTERNA	RAPPR. INTERNA	E PRECISION IBM/370	FORmula TRANslator	Programming Language /1	COmmon Businnes Oriented Language	da Blaise Pascal	Beginners Allpurpos Symbolic Instructio Code
			DATA	DECLARE (DCL)	DATA DIVISION.	VAR..: / TYPE..=	
INTERI XXXXX	B | 2B I | N | 4B A | R | 8B I | O |	-2**15 (2**15)-1 -2**31 (2**31)-1 -2**63 (2**63)-1	| *2 INTEGER | *4 | (I,..,N default)	(15) REAL FIXED BIN (31) (I,..,N default)	9(4) PICTURE 9(9) 9(18) COMPUTATIONAL	INTEGER; (RECORD)	1ch(+num): A,A1,... OPTIONS SPREC OPTIONS LPREC
	PA CK ED		-NO-	REAL FIXED DEC (15)	PICTURE 9(18) COMPUTATIONAL-3	-NO-	-NO-
REALI VIRGOLA FISSA xx.xxx REALI VIRGOLA MOBILE x.xExx x.xDxx	F | L | O | 4B A | T | 8B I | N |16B G |	10**(-78) 10**75 7 cifre 16 cifre 33 cifre	| *4 REAL | *8 | *16	(21) REAL | o (6) FLOAT BIN | (53) o DEC | o (16) | (109) | o (33)	PICTURE 9(6) COMPUTATIONAL-1 PICTURE 9(16) COMPUTATIONAL-2	REAL; (RECORD)	1ch(+num): A,A1,... OPTION SPREC OPTION LPREC
COMPLES.	" | "	"	COMPLEX |*8*16*32	COMPLEX " | "	-NO-	-NO-	-NO-
LOGICI	bit o Byte		LOGICAL | *1 *4 .FALSE. .TRUE.	BIT(n)	88 nome VALUE IS..	BOOLEAN; (RECORD) FALSE TRUE	-NO-
CARATT.	Byte		usate in variabili numeriche (4 char)	CHAR(n) (max 32767)	PICTURE A(n),X(n) (max 32767) DISPLAY	CHAR; STRING(n); PACKED ARRAY (.1..n.) OF CHAR; (max 255 char) (RECORD)	1ch+$: A$,B$,... (max: DIM A$255)
CARATT. NUMERICI			-NO-	PICTURE '...'	PICTURE 9(n),X(n)	-NO-	-NO-
Stringhe di lungh variabile ?			-NO-	CHAR(n) VAR;	-NO-	(Si-estensione)	No, ma LEN ignora a blank in coda
Dimensioni (per tutti i tipi di dati)			DIMENSION A(5,2), SK(80), SK(20)	DCL A(-1:3,5:6).., SK(80) CHAR(1), SK(20) CHAR(4), SK CHAR(80)	01 B OCCURS 5 TIMES 02 A OCCURS 2 TIMES max 3 indici ciascuno 1->n	A: ARRAY(.-1..3, 5..6.) OF ...; SK: PACKED ARRAY (.1..80.) OF CHAR;	DIM solo 1ch(+$), max 2 indici ciascuno 1->n
Conversioni e Espressioni miste	Int<-Real Real<-Int Num<-Char Char<-Num A+I/R		assegnaz. o IFIX assegnaz. o FLOAT No No (Si-estensione)	assegnaz. assegnaz. assegnaz. assegn.+3/6 blank Si	COMPUTE COMPUTE MOVE MOVE Si	No Si No No Si	NUM CHR Si

ISTRUZIONI		FORTRAN-370	PL/1-370	COBOL-370	PASCAL-370	BASIC-370
ISTRUZIONI DI CONTROLLO CONTROLLO	salto	GOTO n GOTO (n1,n2),J	GOTO label;	GO TO nome ...DEPENDING ON J	GO TO lbl;	GOTO n GOTO n1,n2 ON J

	ciclo	DO 3 I=1,5,2
		3 CONTINUE
			DO I=1 TO 5 BY 2;
			.. WHILE(cond);
			.. UNTIL(cond);
			END;
				PERFORM .. THRU..
				..TIMES/UNTIL...
				VARYING/AFTER...
				FROM..BY..UNTIL..
					FOR I:=1 TO 5 DO
					FOR I:=1DOWNTO-5DO
					WHILE cond DO
					REPEAT
					...; ...; ...
					UNTIL cond;
						FOR I=1 TO 5 STEP2
						NEXT I

	test (piu' istr.?)	IF (A.EQ.B) istr	IF A=B THEN
			DO;..;END;
			ELSE
			DO;..;END;
			
			SELECT (K);
			WHEN(1) istr;
			WHEN(2) istr;
			OTHERWISE istr;
			END;
				IF A cond B istr
				ELSE istr
					IF A=B THEN
					BEGIN..;.. END
					ELSE
					BEGIN..;.. END;
					
					CASE var OF
					val1,val2: istr1;
					val3: istr2
					OTHERWISE is3;is4
					END;
						IF A=B THEN istr
						ELSE istr
						(solo 1 istr)

	chiamata	CALL sub(...)	CALL sub(...);	PERFORM CALL sub USING...	nome(....);	GOSUB n GOSUB n1,n2 ON J
	ritorno	RETURN,END	RETURN;	GOBACK. / EXIT.	BEGIN .... END;	RETURN
	iniz.pr. fine pr.	- STOP, END	PROC OPTIONS(MAIN); END..;	IDENT...DIVISION. STOP RUN.	PROGRAM nome; BEGIN .... END.	(PAUSE), END
Sottoprogrammi interni-esterni		FUNCTION/SUBROUTINE solo esterni	nome: PROCEDURE interni e esterni	nome. interni-(esterni)	FUNCTION/PROCEDURE interni-(esterni)	DEF, FNEND solo interni
ISTRUZIONI DI I/O	open/clos			OPEN/CLOSE	RESET/REWRITE/ INTERACTIVE/CLOSE	
	senza formato	READ (7) ... WRITE(8) ...	READ ... INTO(..); WRITE .. FROM(..); GET LIST/DATA .. ; PUT LIST/DATA .. ;	READ..RECORD INTO.. AT END... WRITE...FROM... BEFORE/AFTER ADV.. ACCEPT...FROM.. DISPLAY..UPON..	GET/READ/READLN PUT/WRITE/WRITELN	READ../WRITE.. GET.../PUT... INPUT./PRINT.
	con formato	READ(5,1,END=998, ERR=999) ... 1 FORMAT (...) WRITE(6,2) ... 2 FORMAT (...) nX, Tn An In Fp.q (p>=q+3) Ep.q Dp.q (p>=q+7) Zn +,blank,0,1	GET EDIT(var)(form); PUT EDIT(var)(form); X(n), COL(n) A(n) F(p) F(p,q) (p>=q+2) E(p,q) (p>=q+6) P 'picture' SKIP(n),PAGE		READ/READLN WRITE/WRITELN var:±n var:±n var:s , var:±s:q PAGE	READ/WRITE USING PRINT USING n, .. n :.../FORM... Xn, POSn Cn PIC(...) SKIPn
	in memoria	-NO-	GET STRING(..).. PUT STRING(..)..	..ADVANCING.. CSP, n LINES, C01 MOVE		DATA/READ/RESTORE
Cicli impliciti?		(A(I),I=1,5)	(A(I) DO I=1 TO 5)	-NO-	-NO-	vettore completo

OPERATORI E FUNZIONI			FORTRAN-370	PL/1-370	COBOL-370	PASCAL-370	BASIC-370
OPERAT.	logici	and	.AND.	&	AND	& o AND	& o .AND.
		or	.OR.	|	OR	| o OR	| o .OR.
		not	.NOT.	¬	NOT	¬ o NOT	
	relaz.	<	.LT.	<	< o LESS THAN	<	<
		≤	.LE.	<=	NOT >	<=	<=
		=	.EQ.	=	= o EQUAL TO	=	=
		≠	.NE.	¬=	NOT =	<> o ¬=	<>
		≥	.GE.	>=	NOT <	>=	>=
		>	.GT.	>	> o GREATER THAN	>	>
	aritm.	+	+	+	+ o ADD	+	+
		-	-	-	- o SUBTRACT	-	-
		/	/	/	/ o DIVIDE	/	/
		*	*	*	* o MULTIPLY	*	*
		**	**	**	**	SQR(x)	** o ↑
OPER./ FUNZ.	matem.	mod	MOD,AMOD	MOD	p MOD q	a MOD b	
	string.	cat	-NO-	a||b	STRING	a||b	a||b o a.CAT.b
FUNZ.	matem.	abs,sgn	IABS,ABS,ISIGN,SIGN	ABS,SIGN	-NO-	ABS	ABS,SGN
		tru,rou	INT,AINT	TRUNC,ROUND		TRUNC,ROUND	INT
		flo,cei		FLOOR,CEIL			
		max,min	MAX0,MIN0,MAX1,MIN1	MAX,MIN		MAX,MIN	MAX,MIN
		sqr,rnd	SQRT,(RANDU)	SQRT,(RANDU)		SQRT,RANDOM	SQR,RND
		sin,cos	SIN,COS	SIN,COS		SIN,COS	SIN,COS
		tan,atn	TAN,ATAN	TAN,ATAN		,ARCTAN	TAN,ATN
		exp,log	EXP,ALOG	EXP,LOG		EXP,LN	EXP,LOG
	string.	chr,num		CHAR,	UNSTRING,MOVE		CHR,NUM
		len,sub	-NO-	LENGTH,SUBSTR	UNSTRING	LENGTH,SUBSTR	LEN,STR
		pos,ver		INDEX,VERIFY	EXAMINE,INSPECT	INDEX	IDX
		rep		SUBSTR	STRING		
Trattamento degli errori			READ (5,1,END=998, ERR=999) ...	ON cond ...	AT END.. ON ...	EOF / EOLN	ON cond ... EXIT cond ...
ISTRUZIONI DI ASSEGNAZIONE	multiple?		= No	= A,B,C= ... ;	COMPUTE / MOVE MOVE .. TO A B C.	:= No	LET var = ... LET A,B,C= ... MAT vet = (..)..
Formato scheda	campo		1-72 (7-72)	2-80	8-11/12-72	1-72	1-80
	piu'istr.		NO	SI; .. ;SI;	SI; ..;SI.	SI; ..;SI	NO
	label		numero	nome:	nome.	numero: nome:	numero
	commenti		C in colonna 6	/* .. */	* in colonna 7	LABEL n,nome,...;	REM
	contin.		col.6	libera	col.7 (e poi da 12)	{ }, (* *), /* */ libera	-NO-

TIPI DI DATI E ISTR. DICHIARATIVE	RAPPR. ESTERNA	RAPPR. INTERNA	GRANDEZZA E PRECISIONE MICRO	FORTRAN-MICRO FORmula TRANslator	PL/1-MICRO Programming Language /1	COBOL-MICRO COmmon Businnes Oriented Language	PASCAL-MICRO da Blaise Pascal	BASIC-MICRO Beginners Allpurpos Symbolic Instructio Code
				DATA			VAR..: / TYPE..=	
INTERI xxxxxx	B	da	da	INTEGER | *1 *2 (I,..,N default)			INTEGER; (RECORD)	1->2 ch% A1%,AB%,... DEFINT
	I	2B	-2**15					
	N	a	(2**15)-1					
	A	4B	a					
	R		-2**31					
	I		(2**31)-1					
	O							
	PACKED			-NO-			-NO-	-NO-
REALI VIRGOLA FISSA xx.xxx REALI VIRGOLA MOBILE x.xExx x.xDxx	F		da 10**(-39) 10**38 a 10**(-99) 10**99 da 7 cifre a 17cifre	REAL | *4 | *8 (DOUBLE PRECISION)			REAL; (RECORD)	1->2 ch (sing.pr.) A,A1,(AB)... DEFSNG 1->2 ch# (dop.pr.) A#,A1#,(AB#). DEFDBL
	L							
	O	da						
	A	4B						
	T	a						
	I	8B						
	N							
	G							
COMPLES.	*	*	*	-NO-			-NO-	-NO-
LOGICI	bit o Byte			LOGICAL *1 *4 .FALSE. .TRUE.			BOOLEAN; (RECORD) FALSE TRUE	-NO-
CARATT.	Byte			usate in variabili numeriche (4 char)			CHAR; STRING[n]; PACKED ARRAY [1..n] OF CHAR; (max 255 char) (RECORD)	1->2 ch$ A$,A1$,(AB$). (max 255 char)
CARATT. NUMERICI				-NO-			-NO-	-NO-
Stringhe di lungh variabile ?				-NO-				tutte
Dimensioni (per tutti i tipi di dati)				DIMENSION A(5,2) SK(80), SK(20) (max 3 indici)			ARRAY[-5..5] OF... (max 3 indici)	DIM (max 32767 indici) (ciascuno 0->n)
Conversioni e Espressioni miste			Int<-Real	assegnaz. o IFIX			No	
			Real<-Int	assegnaz. o FLOAT			Si	
			Num<-Char	No			No	VAL
			Char<-Num	No			No	STR$
			A+I/R	Si			Si	Si

ISTRUZIONI		FORTRAN-MICRO	PL/1-MICRO	COBOL-MICRO	PASCAL-MICRO	BASIC-MICRO
ISTRUZIONI DI CONTROLLO	salto	GOTO n GOTO (n1,n2),J			GOTO n;	GOTO n ON J GOTO n1,n2..

	ciclo	DO 3 I=1,5,2
		3 CONTINUE
					FOR I:=1 TO 5 DO
					FOR I:=1DOWNTO-5DO
					WHILE cond DO
					REPEAT
					...; ...; ...
					UNTIL cond;
						FOR I=1 TO 5 STEP2
						NEXT I

	test (piu' istr.?)	IF (A .EQ. B) istr			IF A=B THEN
					BEGIN..;.. END
					ELSE
					BEGIN..;.. END;
					
					CASE var OF
					val1,val2: istr1;
					val3: istr2
					END;
						IF A=B THEN ist:ist
						(ELSE ist:ist)
						si' piu' istr.

	chiamata	CALL sub(...)			nome(....);	GOSUB n ON J GOSUB n1,n2..
	ritorno	RETURN,END			BEGIN .... END;	RETURN
	iniz.pr. fine pr.	(PROGRAM nome) STOP, END			PROGRAM nome(...); BEGIN .... END.	(STOP), END
Sottoprogrammi interni-esterni		FUNCTION/SUBROUTINE solo esterni			FUNCTION/PROCEDURE interni e esterni	DEF solo interni
ISTRUZIONI DI I/O	open/clos				RESET/REWRITE/ INTERACTIVE/CLOSE	
	senza formato	READ (7) ... WRITE(8) ...			GET/READ/READLN PUT/WRITE/WRITELN	WRITE, GET, PUT INPUT(#)"..";v1,v2 PRINT(#) v1;,v2
	con formato	READ(5,1,END=998, ERR=999) ... 1 FORMAT (...) WRITE(6,2) ... 2 FORMAT (...) nX An In Fp.q (p>=q+3) Ep.q Dp.q (p>=q+7) +,blank,0,1			READ/READLN WRITE/WRITELN var:n var:n var:s , var:s:q	PRINT USING ...
	in memoria	DECODE,ENCODE PEEK,POKE				DATA/READ/RESTORE PEEK,POKE
Cicli impliciti?		(A(I),I=1,5)				

OPERATORI E FUNZIONI			FORTRAN-MICRO	PL/1-MICRO	COBOL-MICRO	PASCAL-MICRO	BASIC-MICRO
OPERAT.	logici	AND	.AND.			AND	AND
		OR	.OR.			OR	OR
		NOT	.NOT.			NOT	NOT
	relaz.	<	.LT.			<	<
		≤	.LE.			<=	<=
		=	.EQ.			=	=
		≠	.NE.			<>	<>
		≥	.GE.			>=	>=
		>	.GT.			>	>
	aritm.	+	+			+	+
		-	-			-	-
		/	/			/ e DIV	/ e \
		*	*			*	*
		**	**			SQR(x)	↑
OPER./ FUNZ.	matem.	mod	MOD,AMOD			a MOD b	a MOD b
	string.	cat				CONCAT(a,b)	a+b o a&b
FUNZ.	matem.	abs,sgn	IABS,ABS,ISIGN,SIGN			ABS	ABS,SGN
		tru,rou	INT,AINT			TRUNC,ROUND	FIX
		flo,cei					INT
		max,min	MAX0,MIN0,MAX1,MIN1				
		sqr,rnd	SQRT			SQRT	SQR,RND
		sin,cos	SIN,COS			SIN,COS	SIN,COS
		tan,atn	TAN,ATAN			,ARCTAN o ATAN	TAN,ATN
		exp,log	EXP,ALOG			EXP,LN	EXP,LOG
	string.	chr,num					STR$,VAL
		len,sub	-NO-			LENGTH	LEN
							LEFT$,MID$,RIGHT$
		pos,ver					INSTR
		rep				INSERT	REP
Trattamento degli errori			READ (5,1,END=998, ERR=999) ...				ON cond ...
ISTRUZIONI DI ASSEGNAZIONE	multiple?		= No			:= su matrici	LET var =
Formato scheda	campo		1-72 (7-72)			1-72	1-72 (1-255)
	piu'istr.		NO			SI; ..;SI	SI:...:SI
	label		numero			numero:	numero
						LABEL n1,n2,...;	
	commenti		C in colonna 6 o 1			(* ... *)	REM
	contin.		col.6			libera	-NO-

BREVI CONSIDERAZIONI SULLE TABELLE

1) Le tabelle sono formate da 2 gruppi di 3 elementi ciascuno: il primo gruppo si riferisce ai linguaggi disponibili al CNUCE sul 370 (CMS e OS), il secondo ai linguaggi disponibili sui micro. In quest'ultimo caso, come gia' si e' accennato nell'introduzione, le informazioni riportate sono molto meno precise e non si riferiscono ad un particolare prodotto, ma cercano di riportare le forme piu' comuni per le varie istruzioni.
Per il Fortran del 370 abbiamo cercato di mantenere le informazioni in forma generale, riferendoci pero' alle versioni G e H-esteso, senza tenere per il momento conto delle ulteriori estensioni annunciate col nuovo prodotto VS-Fortran e delle quali si da' un cenno nel seguito.
Dei tre elementi di ciascuna tabella il primo descrive i 'Tipi di dati' e le relative 'Istruzioni dichiarative', il secondo le 'Istruzioni eseguibili' (di controllo e di I/O), e il terzo gli 'Operatori e Funzioni' disponibili; si noti che per motivi di spazio la descrizione del 'Formato delle istruzioni' e delle 'Istruzioni di assegnazione' e' stata messa alla fine del terzo elemento, anche se la collocazione logica sarebbe stata nel secondo elemento insieme alle altre istruzioni eseguibili.

2) Nelle tabelle sono riportate, oltre le istruzioni dei linguaggi, anche la 'Grandezza' e 'Precisione' dei dati numerici (interi e reali). Questi sono chiaramente numeri che fanno riferimento all'hardware dell'elaboratore, dipendendo dalla struttura della 'voce' (N Bytes) utilizzata, piu' che dal linguaggio. Mentre per l'architettura 370 si possono dare valori quasi precisi (ricordiamo che internamente una tale macchina lavora in esadecimale mentre i valori da noi forniti sono espressi in numero approssimato di cifre decimali), per le tabelle riguardanti i micro abbiamo dato solo dei valori indicativi, essendo lo spettro delle macchine sul mercato molto vasto: cosi' ad esempio, mentre si dice che la 'Grandezza' piu' comune e' quella di 10**(-39) --> 10**38, si accenna anche al fatto che alcune macchine arrivano fino a 10**(-99) --> 10**99, ma si sa, anche se non se ne fa menzione, che vi sono anche macchine che accettano ben 3 cifre per l'esponente.
Tali dati vanno pertanto presi con le molle e vanno considerati solo una base di raffronto tra architetture diverse: quello cioe' che se ne ricava e' che 'Grandezza' e 'Precisione' dei dati non vanno messi in rapporto diretto con la 'potenza' dell'elaboratore: molti micro (e perfino calcolatori tascabili) hanno una grandezza ed anche un numero di cifre superiori a quello di macchine di dimensioni maggiori.

3) Istruzioni di controllo: per l'istruzione di test (IF) si e' specificato se e' accettata l'esecuzione condizionata di piu' istruzioni.

4) Istruzioni di assegnazione: si e' anche specificato se vengono accettate istruzioni di assegnazione multiple.

5) Nella pagina di descrizione degli OPERATORI e FUNZIONI, c'e' una riga comune OPER./FUNZ.: i simboli qui riportati sono infatti usati talvolta come operatori e talaltra come funzioni. Da notare che in Pascal anche l'elevazione al quadrato e' una funzione, non un operatore come in genere negli altri linguaggi.

6) Due parole sulle funzioni: si sono indicate in forma abbreviata le seguenti funzioni:

```
mod = modulo (resto della divisione tra interi)
abs = valore assoluto
sgn = segno
max = valore massimo
min = valore minimo
sqr = radice quadrata
rnd = random
sin = seno(rad)
cos = coseno(rad)
tan = tangente(rad)
atn = arco tangente(rad)
exp = esponenziale
log = logaritmo naturale (base e)
```

e poi Round, Trunc, Floor, Ceil, External (non riportata nelle tabelle precedenti), cosi' abbreviate e con i significati seguenti:

	rou	tru	flo	cei	ext
1.5	2	1	1	2	2
1.2	1	1	1	2	2
-1.2	-1	-1	-2	-1	-2
-1.5	-2	-1	-2	-1	-2

con, ad esempio, le relazioni:

```
rou(x)= flo(x  + sgn(x)*0.5)  |  a div b = tru(a/b)
tru(x)= sgn(x) * flo(abs(x))  |  a mod b = a-(b*(a div b))
cei(x)= flo(x) + 1            |            a≥0, b>0
```

Si noti che in vari casi la funzione INT corrisponde a Floor, mentre in altri corrisponde a Trunc: le differenze si notano naturalmente solo per valori negativi dell'argomento.

Per le stringhe si ha:

cat = concatenazione
chr = conversione da numero a carattere
num = conversione da carattere a numero
len = lunghezza
sub = sottostringa di una data stringa
pos = posizione di una sottostringa in una stringa
ver = verifica dell'esistenza di un insieme di caratteri in una data stringa
rep = sostituzione di parte di una stringa con un'altra fornita

Anche qui si noti che le funzioni chr e num sono implementate con i nomi CHR e NUM nel VSBasic, mentre nella maggior parte dei micro hanno nome STR$ e VAL: attenzione, perche' sui micro esistono anche CHR$ e NUM, ma con significato diverso (si riferiscono in genere alla rappresentazione ASCII di un dato carattere); anche la funzione CHR in Pascal e' di quest'ultimo tipo.

L'operazione rep compare a volte come vera e propria funzione (cioe' alla destra del segno = (es. INSERT in Pascal), altre volte come 'pseudo-variabile', cioe' a sinistra del segno = (es.SUBSTR in PL/1): in questo caso puo' comparire anche in istruzioni non di assegnazione, come ad esempio: IF SUBSTR(SK,1,3) ¬= 'ZOT' THEN ...

Altre brevi informazioni:
anche se sono riportati solo i riferimenti alle funzioni sin, cos, tan, atn, molti linguaggi hanno tutto l'insieme di funzioni dirette (sin,..) e inverse (asn,..), ed anche i logaritmi in base 10 o 2, ecc; le funzioni riportate nelle tabelle ci sono sembrate solo le piu' usuali.

Non ho inoltre parlato della priorita' degli operatori, per non addentrarmi in una giungla di convenzioni diverse: meglio usare esplicitamente le parentesi nei casi dubbi... e non.

Ed ancora: la funzione per generare numeri pseudo-casuali in Fortran (indicata in tabella con RANDU) e' messa tra parentesi in quanto non fa parte della normale libreria Fortran, ma di librerie matematiche e statistiche aggiuntive.

7) Per tutti i linguaggi 370, molti elementi riguardano estensioni del prodotto IBM rispetto alle norme standard; Indicativamente riportero' qui sotto alcune di tali estensioni:

Fortran:

VS Fortran	I/O in memoria dati di tipo carattere nomi simbolici per le costanti formato libero per le istruzioni IF maggiormente strutturato ecc.
Fortran H-esteso:	REAL*16, COMPLEX*32 uso di nomi generici per la funzioni I/O asincrono ecc.
Fortran-G :	INTEGER*2 , COMPLEX*16 , LOGICAL*1 parametri END e ERR nell'I/O literal tra apici espressioni miste piu' di 3 dimensioni formati T e Z chiamate per valore I/O diretto ecc.
Fortran-ANS :	COMPLEX . . . . DATA IF logico literal tra gli argomenti di un sottoprogramma ecc.
Basic-Fortran :	-

Cobol:
trattamento di stringhe
sottoprogrammi esterni
trattamento degli errori
ecc.

Pascal:
sottoprogrammi esterni
trattamento di stringhe (SUBSTR, acc.)
stringhe di lungh. variabile (STRING)
OTHERWISE nella istr. CASE
OPEN, CLOSE
funzioni MIN, MAX, RANDOM
label come nomi
numeri interi in 1 o 2 byte (PACKED)
ecc.

8) Pascal
Come il Cobol ha una struttura in cui la parte dichiarativa e' molto potente; presenta alcune limitazioni rispetto agli altri linguaggi (stringhe, I/O interattivo su terminale, ecc.) e varie caratteristiche diverse dai normali standard (dichiarazione di TIPI di variabili propri dell'utente, trattamento di insiemi, gestione di liste, ecc.) che pero' in questo contesto non vengono considerate.
Questo tentativo di confrontare i linguaggi sulla base di caratteristiche ritenute piu' o meno 'standard' porta sicuramente ad una critica da parte dei cultori di questo linguaggio; in verita', come diciamo anche noi in altra parte di questo stesso articolo, ciascun linguaggio ha una sua propria fisionomia e proprie caratteristiche che vanno sfruttate al massimo senza cercare invece analogie con altri linguaggi.
Per quanto riguarda i micro, come anche per il Basic, cosi' per il Pascal si hanno numerose versioni, sotto vari nomi: si va ad esempio da un Tiny-Pascal (con il trattamento solo di variabili intere, vettori di una sola dimensione e mancanza di I/O su disco) fino a versioni piu' complete che comprendono anche estensioni al linguaggio standard, quali ad esempio l'esistenza di un tipo STRING (equivalente a PACKED ARRAY...OF CHAR), la possibilita' di trattamento delle matrici anche nelle istruzioni di I/O, la possibilita' di definire l'occupazione di memoria dei numeri interi (1 Byte, 2 Bytes, ecc.), ed altre ancora.
Per quanto riguarda alcune note di confronto tra Pascal e Basic (attualmente tra i piu' diffusi sui micro) diciamo che, in genere, i Basic sono 'interpreti' e quindi piu' lenti in esecuzione, specie se il programma e' stato strutturato con salti tramite GOTO, mentre i Pascal generano un p-code ottimizzato, prima dell'interpretazione vera e propria (cio' non esclude che si trovino anche dgli ottimi compilatori per entrambe i linguaggi);
l'uso esclusivo della aritmetica in doppia precisione invece e' un fattore negativo per il Pascal, in tutti quei casi in cui la precisione semplice sarebbe sufficiente;
inoltre ha, in genere, una trattazione piu' limitata degli errori.

Post-Scriptum: si noti la differenza di significato del ; tra Pascal e PL/1:
in PL/1 il ; termina un'istruzione (e quindi e' sempre presente alla fine di un'istruzione)
in Pascal il ; e' un separatore tra due istruzioni (e quindi manca dopo l'ultima istruzione, sia del programma, sia di una istruzione composta).
Si noti anche che in Pascal l'uso di due punti consecutivi .. fa parte del linguaggio: pertanto non sempre nelle tabelle tali punti vanno considerati come segni di omissione.

9) Un altro discorso a parte merita il Basic, attualmente il linguaggio piu' diffuso sui micro calcolatori. Non ci soffermeremo su una descrizione dettagliata, ma ci limiteremo a valutare in maniera abbastanza superficiale il BASIC presente sotto VM/CMS con le versioni presenti sui micro-computer rimandando anche all'articolo ('Basic a confronto') citato ulteriori informazioni.

Il VSBasic, operando su calcolatori di dimensioni maggiori (ancora per quanto, visto che si parla gia' di micro con memorie dell'ordine dei MBytes?) ha il trattamento completo dei vettori e delle matrici, tramite l'istruzione MAT, che e' invece tralasciata su gran parte dei micro, ma presenta sorprendentemente delle limitazioni:

a) nei nomi delle variabili sia numeriche (accetta solo 1 lettera e 1 numero mentre su molti micro e' accettata una qualsiasi combinazione di caratteri), sia stringa (accetta solo 1 lettera), sia nelle dimensioni (dove accetta al massimo solo 2 indici e dove, come eccezione alla regola stabilita in precedenza per i nomi, accetta solo nomi di variabili -anche numeriche- con 1 sola lettera!)
b) nel formato delle istruzioni, in quanto non accetta istruzioni multiple sulla stessa linea.

Ed ancora, sui micro, a seconda della versione di Basic piu' o meno estesa (si parla di Tiny Basic, Basic 4K, 8K, Extended Basic, Disk Basic, ecc.), si possono avere notevoli diversita':
- presenza o meno di variabili stringa
- presenza o meno di variabili intere
- piu' istruzioni o no sulla stessa riga
- variabili solo in singola precisione
- opzione ELSE (nell'IF) assente
- trattamento piu' o meno esteso degli errori (ON cond)

10) Altri elementi di raffronto, non sufficientemente evidenziati nelle tabelle:
- Istruzione IF cond THEN: accetta variabili logiche per 'cond' (IF L THEN ...)?
  accetta 'cond' composte? (IF A=B AND C=D THEN ...)
  se si', le varie 'cond' vengono calcolate tutte o ci si arresta al momento in cui l'IF e' soddisfatto?
- Istruzione ciclo: accetta modifiche dell'indice o dei limiti? (alcuni si' altri no)
- Stringhe di bit: vengono trattate?
- Literal tra apici 'aaaa': vengono trattati?
- Uso di parole riservate (es. GOTO, DO, ecc.) vengono permessi come nomi di variabili dell'utente? (Cobol, Basic: no; Fortran, PL/1: si) (si pensi alle circa 500 parole 'riservate' del Cobol!)
- Gli spazi bianchi sono significativi o no nella codifica delle istruzioni, sia nei nomi di variabili e parole chiave (READ e' equivalente a RE A D ?) sia per gli operatori (A>3 o A > 3 ?) ?
- Esiste il concetto di variabile 'local' cioe' riconosciuta solo all'interno del blocco di istruzioni nel quale si trova, contrapposto a quello di variabile 'global' (nota cioe' a tutto il programma?)
- Sottoprogrammi: come vengono effettuate le chiamate ed il passaggio dei parametri? (per 'valore', per 'locazione', per 'nome', ecc.)
  e' possibile passare il nome di un sottoprogramma come parametro ad un altro sottoprogramma?
  nel caso di sottoprogrammi 'esterni', questi devono essere compilati separatamente o con un solo richiamo al compilatore? (Batch-compilation)
- Presenza o meno di funzioni 'generiche', cioe' che accettano operandi di tipi diversi (interi, reali, ecc.)
- Vettori e Strutture: memorizzazione per righe o per colonne
  dimensioni come costanti intere o reali o variabili o espressioni
  estremo inferiore nelle dimensioni 1 (come in Fortran) o a scelta (-3:5, ecc.)
  nomi qualificati delle strutture in forma 'top-down' (PL/1) o 'bottom-up' (Cobol)
- Possibilita' di attivare processi paralleli (multi-task)
- Istruzioni di I/O:
  accesso a files diretto o ad indici
  OPEN e CLOSE obbligatorie o automatiche
  espressioni nelle operazioni di lettura/scrittura (es. PRINT 1/3+5 o INPUT con dati 1/3+5)
- Struttura del prodotto:
  e' interprete, compilatore, p-code o altro?
  opera in memoria, su disco, in overlay o altro?
  possiede diagnostica completa e chiara?

CHIACCHERATA CONCLUSIVA

La Programmazione (propriamente detta) e' solo l'ultimo passo di una sequenza di attivita' di analisi e decisionali che accompagnano la risoluzione di un problema tramite un elaboratore:

- Determinazione del Problema:
  consiste nel chiarimento di cosa si vuole ottenere (es. vogliamo ordinare una lista in ordine alfabetico, oppure: vogliamo una funzione che fornisca dei numeri pseudo-casuali, oppure: vogliamo interpolare una sequenza di punti con una curva continua, ecc.).
  Questi accennati sono solo esempi banali, cosi' come banale puo' sembrare tutto questo punto, ma in realta' la determinazione esatta del problema e' il primo passo veramente fondamentale, ed a cui spesso per troppa fretta o troppa sufficienza o troppa presunzione non si attribuisce il dovuto peso. Partire col piede sbagliato o con una visione limitata e parziale dei problemi puo' portare in seguito a dover 'rimettere le mani', modificare e riadattare pesantemente un programma, con tutte le difficolta' che ben si conoscono e, in casi piu' gravi, puo' portare ad ad esempio ad un cattivo dimensionamento di una struttura di calcolo (archivi, spazio disco o addirittura parco macchine).

- Determinazione del Metodo:
  come si puo' risolvere il problema posto, cioe' con quali sistemi matematici, sistemistici o altro (ad esempio se, nel caso del Sort, per ordinare la lista: si utilizzera' un metodo per scambio - Bubble Sort, Shell-Metzner, ecc. - oppure per inserzione, oppure per ricerca binaria, ecc.; nel caso dei numeri pseudo-casuali, si utilizzera' il metodo dei quadrati centrali di Von Newmann, oppure quello dei prodotti centrali, oppure i metodi sequenziali di Fibonacci o additivo, di Lehmer o moltiplicativo, di Rotemberg o misto, ecc.).

  Altro esempio semplice, ma che anch'esso da' l'idea di quanti "metodi" si possono usare per uno stesso problema, ci e' dato dal gioco della tombola, cioe' dall'estrazione di 90 numeri pseudo-casuali (tra 1 e 90). Una prima stesura semplicistica del programma puo' portare alla chiamata del sottoprogramma rnd un numero indefinito di volte per coprire tutti i 90 numeri da estrarre: ma quante volte? mano a mano che ci si avvicina alla fine risulta piu' problematica l'estrazione dei numeri restanti cosi'

che in casi sfavorevoli il numero delle estrazioni puo' risultare altissimo e questo oltre al consumo di tempo di elaborazione, non e', tra l'altro, aderente alla realta' del gioco.
Occorre allora determinare un metodo di estrazione che in soli 90 passi ci fornisca i 90 numeri richiesti: bisogna cioe' trovare la maniera ogni volta, di "accorciare" il vettore dei 90 numeri eliminando l'elemento gia' estratto e compattando gli altri.
Ma anche questo procedimento puo' essere migliorabile, eliminando la fase di compattamento del vettore, se sfruttiamo la posizione occupata dall'ultimo elemento estratto (che ormai non serve piu' e puo' essere invece utilizzata per contenere uno dei valori "alti" ancora da estrarre): pensateci un po' e vedrete come anche nei problemi piu' banali come questi esempi si possa, con un minimo di ragionamento, trovare dei metodi "appropriati" al particolare problema.

- Determinazione dell'Algoritmo:
scelta cioe' della forma piu' adatta di una formula, onde evitare o minimizzare gli errori numerici (di arrotondamento, ecc.). Supponiamo ad esempio di dover avere a che fare con una formula del tipo C=(A/B - B/A) con A e B grandi e molto vicini tra loro (es. A= 10**5 e B= 10**5 -1).
La formula scritta sopra condurrebbe ad un notevole errore di arrotondamento causata dalla operazione di sottrazione di due numeri, risultanti dalle due operazioni di divisione, quasi uguali tra loro.
Con qualche manipolazione pero' potremo riscrivere la stessa formula come:
C=(DELTA/B) * ((DELTA/B)+2) / ((DELTA/B)+1)
nella quale compare la nuova variabile DELTA=(A-B)=1 e nella quale non compare piu' la differenza tra i due rapporti quasi uguali della formula precedente.

Un caso piu' semplice puo' essere il seguente:

```
A=200
B=0.5* 10**(-37)
C=10**(-5)
```

supponendo che la massima 'grandezza' ammessa sia 10**38, se si deve calcolare l'espressione (A/B)*C si avra' un errore di underflow dato che il quoziente A/B porta a superare il valore massimo ammesso per la macchina. Capovolgendo invece le operazioni (A*C)/B tutto va a posto.

Sono proprio queste due (ricerca del "metodo" e degli "algoritmi", secondo me, i passi piu' importanti di tutto il processo di programmazione: troppo spesso una scarsa conoscenza del substrato matematico per un dato problema, porta con troppa disinvoltura e faciloneria alla scrittura di programmi che, corretti solo in alcuni dei casi piu' semplici, presentano enormi pecche non appena si passa a casi piu' concreti e reali: ed ancora piu' grave e' allora il fatto che, se non si ha gia' un'idea degli ordini di grandezza delle misure che ci si debbono aspettare, si puo' anche arrivare ad accettare per validi i risultati cosi' erroneamente ottenuti. Attenzione percio': conoscere il linguaggio di programmazione in se' significa solo essere passati per le scuole elementari ed avere imparato a parlare, ma questo non e' sufficiente: la lingua va adoperata con proprieta' ed accortezza.

- Determinazione del Linguaggio:
per questo, personalmente mi sono fatto l'opinione che la conoscenza veramente approfondita di un linguaggio (qualunque esso sia) sia sufficiente per risolvere la maggior parte , per non dire tutti, dei problemi di programmazione, PIU' CHE la conoscenza superficiale di vari linguaggi.
Spesso la conoscenza di piu' linguaggi diversi porta ad usare le stesse forme di programmazione in ognuno di essi, senza cioe' sfruttare appieno le caratteristiche peculiari di ciascuno di essi, ma cercando piuttosto le analogie tra l'uno e l'altro.
Cosi' ad esempio si sara' portati ad usare in PL/1 sempre la stessa forma DO I = 1 TO 7, pur permettendo questo altre forme piu' esplicitamente mnemoniche e magari piu' adatte al particolare problema, quali ad esempio

```
DCL GIORNO CHAR(3); DO GIORNO = 'LUN', 'MAR', 'MER', ...
```

che permette di far assumere alla variabile GIORNO successivamente ad ogni ciclo, i valori LUN, MAR, ecc. oppure forme piu' strutturate quali DO..WHILE, ecc.

Come seconda conseguenza, si ha anche una scarsa conoscenza delle "trappole" nelle quali si puo' cadere involontariamente e che spesso non sono descritte sul manuale di uso.

Si ricordino ad esempio i problemi di conversione che sorgono in PL/1 - descritti anche in MANUT PLI - con particolari tipi di variabili:

```
DCL A FIXED DEC(15,10) INIT(1);
A = A + 0.1;
DISPLAY (A);     -----> 1.0625 invece di 1.1 !!!!

DCL A FIXED DEC (5,2)
A = 25 + 1/3
DISPLAY (A);     -----> 5.33333... o FIXED OVERFLOW !!!!
                        (si perde la prima cifra)

A = 25 +01/3
DISPLAY (A);     -----> 25.33333... CORRETTO con l'aggiunta
                        di uno Zero NON significativo!!!!
```

oppure

i problemi che possono sorgere, ad esempio in Fortran, nel passaggio di parametri ad un sottoprogramma, dove ad esempio la modifica di una variabile nel sottoprogramma puo' portare al cambiamento di una costante (Nota bene: costante!!!) nel programma chiamante:

```
CALL SUB(1)
N=1
WRITE (...) N ------> Stampa 3 !!!!
SUBROUTINE SUB(M)
M=M+2
RETURN
```

Altri motivi di errore possono essere, ad esempio, nell'IF, le differenti maniere con cui in linguaggi diversi le frasi THEN e ELSE si associano tra di loro (un'errata interpretazione porta all'esecuzione di istruzioni non volute o viceversa) oppure, sempre nell'IF, le differenti modalita' di trattamento di condizioni composte:

```
IF A=0 OR 1/A < 5 THEN ...
```

se A=0 si ha o meno errore a seconda che la seconda condizione 1/A venga tradotta ugualmente o saltata.

Programmando invece a fondo con un linguaggio si imparano tutte le sue caratteristiche, positive e negative, si impara ad evitare le trappole ed anzi, in certi casi, anche a sfruttarle coscientemente a proprio vantaggio.

Chiaramente alcuni linguaggi saranno piu' di altri orientati e adatti a determinati tipi di applicazioni, ma cio' non toglie che, nel campo dei problemi 'normali' (senza cioe' avere la necessita' di effettuare particolari elaborazioni troppo 'specialistiche', quali potrebbero ad esempio essere elaborazioni parallele, operazioni di I/O dirette o a chiave, gestione statica o dinamica della memoria, o altre ancora se non esplicitamente supportate dal compilatore) i linguaggi general-purpose oggi si equivalgano e permettano, con piu' o meno facilita', di effettuare le stesse elaborazioni.

Questo e' tanto piu' vero se si pensa non alla struttura di base del linguaggio (standard ANSI per il Fortran ecc.), ma alle varie estensioni che le case produttrici immettono sul mercato: cosi' ad esempio abbiamo visto il Fortran subire tutta una serie di estensioni con l'aggiunta del trattamento di condizioni particolari (EOF e errori) nelle istruzioni di I/O, la possibilita' di trattare processi paralleli, ecc; analogamente il Cobol ha subito, rispetto agli standard ANSI, modifiche ed estensioni analoghe (trattamento di stringhe, sottoprogrammi esterni, ecc.). In genere l'aggiunta di estensioni agli standard, porta sempre piu' i linguaggi ad equivalersi tra loro nelle funzioni: l'aspetto fonetico (insieme dei simboli di cui il linguaggio si compone), ortografico (regole per la combinazione dei simboli tra di loro), grammaticale (associazione delle parole tra di loro) e sintattico (combinazione di frasi tra di loro) del linguaggio divengono allora, nella scelta, una pura e semplice questione di moda e gusto personale.

- Determinazione della Struttura:
  e' la fase in cui si "butta giu'" uno scheletro di programma che non corrisponde a nessun linguaggio particolare ma che rispetta i canoni della programmazione strutturata (Si vedano gli articoli relativi, comparsi sui Rapporti 1980-4, 1980-5, 1981-1, 1981-2).

- Programmazione e ricerca dell'ottimizzazione:
  ed eccoci giunti all'ultima fase quella della programmazione vera e propria. Ormai il problema ha raggiunto una forma tale da essere facilmente tradotto in istruzioni specifiche;
  Di elementi variabili resta, a questo punto, solo la codifica ottimizzata del testo, cioe' il seguire tutti quegli accorgimenti che renderanno il nostro programma ottimizzato sia come memoria sia come tempo di esecuzione.

Nell'articolo di Jim Gilbreath citato all'inizio si fa ad esempio rilevare come la scelta di un Metodo (cfr. con quanto gia' detto in proposito) che non facesse uso di divisioni nella scrittura di un programma per la determinazione dei numeri primi comportasse un risparmio davvero notevole di tempo (si parla di ben 69 volte nei confronti di una macchina che effettua la divisione ad hardware; per altre che la simulano con metodi software si va naturalmente su cifre ancora piu' alte).

Per chi puo' servire, questi sono ad esempio alcuni degli accorgimenti che si possono adottare, sui micro, programmando in Basic:

Ottimizzazione di tempo:
- cercare Metodi ed Algoritmi appropriati (minimizzazione di divisioni, ecc.)
- sostituire GOTO con cicli FOR-NEXT in modo da ottimizzare la ricerca della label
- mantenere all'inizio del programma i cicli FOR-NEXT e le subroutine molto referenziate
- evitare chiamate a sottoprogrammi dall'interno di cicli
- codificare i cicli con istruzioni multiple sulla stessa linea
- strutturare i cicli nidificati in ordine crescente di indice in modo da minimizzare le assegnazioni dei valori alle variabili:
  I=1 TO 2
  J=1 TO 50
  K=1 TO 1000
- evitare di mescolare dati (variabili e/o costanti) di tipo diverso per risparmiare sul tempo di conversione
- ecc.

Ottimizzazione di memoria:
- minimizzare l'uso dei commenti
- minimizzare gli spazi tra le parole (abolendoli dove non esplicitamente necessari)
- utilizzare piu' istruzioni su una stessa riga
- utilizzare variabili (inizializzate una sola volta in cima al programma) al posto di costanti (numeriche o carattere), o peggio espressioni, ripetute in mezzo al programma
- utilizzare variabili intere (%) quando possibile
- utilizzare nomi corti per le variabili (alcuni Basic permettono nomi formati da piu' di due caratteri)
- al limite evitare di codificare intere istruzioni (es. END di chiusura) o parti di esse (es. indice variabile nell'istruzione NEXT) se il compilatore lo permette
- ecc.

Naturalmente queste norme si applicano a chi compie elaborazioni su micro calcolatori, dove il risparmio di memoria e' in stretta competizione con le regole dettate invece dalla programmazione strutturata e quindi con la chiarezza del testo: la virtu', ovviamente, consiste nel saper bilanciare le due cose secondo le proprie esigenze.

UN PROGRAMMA DI CONFRONTO

Per concludere, voglio riportare, scrivendolo nei linguaggi fin qui presi in considerazione, un semplice programma di esempio: si faccia conto di avere un file sequenziale con record lunghi 80 caratteri contraddistinti, in colonna 1, dal tipo 1 o 2.
Si vogliono elaborare solo i records di tipo 2, calcolando il quoziente tra i primi due numeri della scheda (che seguono cioe' il tipo) sostituendo il risultato al primo numero e stampando la nuova scheda cosi' ottenuta su carta.
Il contenuto del programma, come si vede, e' senza un particolare significato e non si propone neppure di coprire tutte le caratteristiche viste dei vari linguaggi; il programma vuol solo dare un'idea delle analogie e/o differenze dei vari tipi di programmazione.

Il listing dei programmi e' stato suddiviso in 5 parti: DICHIARAZIONI, LETTURA, ELABORAZIONI, SCRITTURA, FINE che rispecchiano un po' la struttura di un qualsiasi generico programma, anche complesso.
Le tecniche di programmazione nei vari casi sono leggermente diverse una dall'altra, come sara' brevemente spiegato sotto ciascun listing, in parte a causa delle diversita' dei linguaggi, in parte perche' i programmi sono stati scritti senza curarsi troppo di mantenere un'analogia troppo stretta tra di loro; naturalmente la codifica utilizzata e' ben lungi dall'essere l'unica o la piu' adatta e corretta per il problema, ed anzi spesso si e' usata una codifica ridondante al solo scopo di farsi un'idea delle strutture utilizzabili in ciascuno dei linguaggi.

FORTRAN

```
C
C  DICHIARAZIONI
C
      DIMENSION VAR(2),PAR(3)
   11 FORMAT (I1)
   12 FORMAT (A1,2F8.0,3F6.0)
   13 FORMAT (1X,A1,2(1X,F8.0),3(1X,F6.0))
C
C  LETTURA
C
    3 READ (8,11,END=100) ITYP
      IF (ITYP .NE. 2) GOTO 3
      BACKSPACE 8
      READ (8,12) ITYP,(VAR(I),I=1,2),(PAR(I),I=1,3)
C
C  ELABORAZIONI
C
      IF (VAR(2) .NE. 0.) VAR(1)=VAR(1)/VAR(2)
C
C  SCRITTURA
C
    7 WRITE (6,13) ITYP,(VAR(I),I=1,2),(PAR(I),I=1,3)
      GOTO 3
C
C  FINE
C
  100 STOP
      END
```


Dati di input

```
 1ASASASASAAASASASASASASAAAASASASASASASASASA
 20000999.0000333.00999.00888.00777.
 1CVCVCCVCCCVCVCVCVCCVCVCVCVCVCVCVCVCVCVCVCV
 20000066.0000033.00099.00088.00077.
 1DSSSDSDSSDSDSSDSDSDSDSDSDSDSDSDSDSDSDSDSDS
 20000003.0000003.00009.00008.00007.
```

Dati di output

```
 2       3.     333.    999.    888.    777.
 2       2.      33.     99.     88.     77.
 2       1.       3.      9.      8.      7.
```

Considerazioni Fortran:

Il problema da cui siamo partiti, pur nella sua semplicita', comporta subito qualche difficolta' in Fortran. Si deve infatti ricordare che in genere in Fortran non abbiamo a disposizione istruzioni di lettura senza formato di un record in una data area di memoria, in cui esaminarlo e leggere solo successivamente le variabili interessate col formato corretto: la lettura del file deve gia' tener conto delle variabili e del loro formato. La tecnica qui usata consiste allora nel leggere uno stesso record due volte (BACKSPACE): la prima volta per controllare il tipo e la seconda, se il tipo era quello voluto, per inizializzare le variabili. L'elaborazione procede poi in modo normale.
Da notare anche, nel formato di stampa, la codifica 1X iniziale: il Fortran infatti al contrario di PL/1 ed altri interpreta il primo carattere della linea di stampa come carattere di controllo per la paginazione (salto riga, salto pagina, ecc.): pertanto i dati significativi (ITYP in questo caso) vanno iniziati da colonna 2. Inoltre ogni istruzione READ o WRITE opera su un intero record ed inizia sempre una nuova linea di stampa: non e' pertanto necessaria un'indicazione esplicita di salto riga, per ogni istruzione di uscita (cosi' come con SKIP in PL/1 o BASIC).

```
PROG: PROC    OPTIONS(MAIN);

/*  DICHIARAZIONI  */

 DCL INP FILE RECORD INPUT;
 ON ENDFILE(INP) GOTO FINE;

 DCL 1 BUFFIN,
     2 TYPE    CHAR(1),
     2 VAR(2)  CHAR(8),
     2 PAR(3)  CHAR(6),
     2 FILLER CHAR(45);
 DCL 1 SCHEDA,
     2 TYPE    CHAR(1),
     2 VAR(2)  BIN FIXED(31),
     2 PAR(3)  BIN FIXED(31),
     2 FILLER CHAR(45);

/*  LETTURA  */

LEGGI: READ FILE(INP) INTO(BUFFIN);
       IF BUFFIN.TYPE ¬= '2' THEN GOTO LEGGI;

/*  ELABORAZIONI  */

       SCHEDA = BUFFIN, BY NAME;
       IF SCHEDA.VAR(2) ¬= 0
          THEN SCHEDA.VAR(1) = SCHEDA.VAR(1)/SCHEDA.VAR(2);

/*  SCRITTURA  */

       PUT SKIP EDIT (SCHEDA.TYPE,
                 (SCHEDA.VAR(I) DO I=1 TO 2),
                 (SCHEDA.PAR(I) DO I=1 TO 3))
                 (A,2 (X(1),F(8)),3 (X(1),F(6)));
       GOTO LEGGI;

/*  FINE  */

 FINE: END PROG;
```


Dati di input

```
 1ASASASASAAASASASASASASAAAASASASASASASASASA
 20000999.0000333.00999.00888.00777.
 1CVCVCCVCCCVCVCVCVCCVCVCVCVCVCVCVCVCVCVCVCV
 20000066.0000033.00099.00088.00077.
 1DSSSDSDSSDSDSSDSDSDSDSDSDSDSDSDSDSDSDSDSDS
 20000003.0000003.00009.00008.00007.
```

Dati di output

```
 2        3      333    999    888    777
 2        2       33     99     88     77
 2        1        3      9      8      7
```

Considerazioni PL/1:

la tecnica usata in PL/1 e' quella di definire delle strutture che servano a contenere i dati in ingresso (BUFFIN) e in elaborazione (SCHEDA) la struttura di ingresso e' definita come caratteri, quella di elaborazione contiene dati numerici.
Letta una scheda e controllato che sia di tipo 2, vengono fatte le opportune elaborazioni e stampato il risultato.

COBOL

```
 IDENTIFICATION DIVISION.
     PROGRAM-ID. PROG.
*
 ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT INP ASSIGN TO UR-2540-S-INP1.
     SELECT OUT ASSIGN TO UR-1403-S-OUT1.
*
 DATA DIVISION.
  FILE SECTION.
   FD INP
      LABEL RECORDS ARE OMITTED,
      RECORDING MODE IS F,
      RECORD CONTAINS 80 CHARACTERS,
      DATA RECORD IS REC1.
   01 REC1.
      02 TYP       PICTURE IS 9.
      02 VAR       OCCURS 2 TIMES.
         03 VAR1      PICTURE IS 9(7)V.
         03 FILLER    PICTURE IS X.
      02 PAR       OCCURS 3 TIMES.
         03 PAR1      PICTURE IS 9(5)V.
         03 FILLER    PICTURE IS X.
      02 FILLER    PICTURE IS X(45).
   FD OUT
      LABEL RECORDS ARE OMITTED,
      RECORDING MODE IS F,
      RECORD CONTAINS 133 CHARACTERS,
      DATA RECORD IS RIGA.
   01 RIGA.
      02 CC        PICTURE IS X(1).
      02 FILLER    PICTURE IS X(132).
 WORKING-STORAGE SECTION.
   77 TEMP PICTURE IS 9(8) USAGE IS COMP.
   01 SCHEDONE.
      02 FILLER    PICTURE IS X VALUE IS SPACES.
      02 TYP       PICTURE IS 9.
      02 BL1       PICTURE X(16) VALUE IS SPACES.
      02 VAR       REDEFINES BL1 OCCURS 2 TIMES.
         03 FILLER    PICTURE IS X.
         03 VAR1      PICTURE IS 9(7)V.
      02 BL2       PICTURE X(18) VALUE IS SPACES.
      02 PAR       REDEFINES BL2 OCCURS 3 TIMES.
         03 FILLER    PICTURE IS X.
         03 PAR1      PICTURE IS 9(5)V.
      02 FILLER    PICTURE IS X(97).
```



```
*
 PROCEDURE DIVISION.
     OPEN INPUT INP, OUTPUT OUT.
 LEGGI. READ INP AT END GO TO FINE.
     IF TYP OF REC1 NOT = 2 GO TO LEGGI.
     MOVE TYP  OF REC1 TO TYP  OF SCHEDONE.
     MOVE VAR1 OF REC1 (1) TO VAR1 OF SCHEDONE (1).
     MOVE VAR1 OF REC1 (2) TO VAR1 OF SCHEDONE (2).
     MOVE PAR1 OF REC1 (1) TO PAR1 OF SCHEDONE (1).
     MOVE PAR1 OF REC1 (2) TO PAR1 OF SCHEDONE (2).
     MOVE PAR1 OF REC1 (3) TO PAR1 OF SCHEDONE (3).
     IF VAR1 OF SCHEDONE (2) NOT = 0
        COMPUTE VAR1 OF SCHEDONE (1) =
        VAR1 OF SCHEDONE (1) / VAR1 OF SCHEDONE (2).
     WRITE RIGA FROM SCHEDONE AFTER ADVANCING 1 LINES.
     GO TO LEGGI.
*
 FINE. CLOSE INP OUT. STOP RUN.
```

Dati di input

```
 1ASASASASAAASASASASASASAAAASASASASASASASASA
 20000999.0000333.00999.00888.00777.
 1CVCVCCVCCCVCVCVCVCCVCVCVCVCVCVCVCVCVCVCVCV
 20000066.0000033.00099.00088.00077.
 1DSSSDSDSSDSDSSDSDSDSDSDSDSDSDSDSDSDSDSDSDS
 20000003.0000003.00009.00008.00007.
```

Dati di output

```
 2 0000003 0000333 00999 00888 00777
 2 0000002 0000033 00099 00088 00077
 2 0000001 0000003 00009 00008 00007
```

Considerazioni Cobol:

Particolare attenzione nell'evitare l'uso di parole riservate (il Cobol ne ha circa 500).
Alcune caratteristiche piu' limitative nei confronti degli altri linguaggi: GO TO va scritto con un blank in mezzo, OPEN e CLOSE sono obbligatorie, tutti gli operatori vanno separati con spazi bianchi dal resto del testo; inoltre, siccome non accetta il carattere . nella lettura di numeri reali fixed-point, si sono dovute leggere solo le cifre a sinistra e ignorare, nella lettura, il punto per mezzo di un campo FILLER.
Nel complesso si notera' la maggiore dimensione del programma dovuta allo stile colloquiale del linguaggio e la preponderanza della parte dichiarativa su quella esecutiva.

PASCAL

```
PROGRAM PROG;
(*  DICHIARAZIONI  *)
LABEL
  10;
CONST
  PI = 3.14159;
TYPE
  INT1 = PACKED  0..255;
VAR
  INP,OUT     : TEXT;
  I           : INT1;
  R           : RECORD
                  TYP : INT1;
                  VA  : ARRAY (.1..2.) OF REAL;
                  PA  : ARRAY (.1..3.) OF REAL;
                END;
(*  LETTURA  *)
BEGIN
  RESET(INP); REWRITE(OUT);
  WHILE NOT EOF(INP) DO
  BEGIN
    WITH R DO
    BEGIN
      10: READ(INP,TYP:1);
      IF TYP <> 2 THEN
         BEGIN
           READLN(INP); GOTO 10
         END;
      FOR I := 1 TO 2 DO
        READ(INP,VA(.I.):8);
      FOR I := 1 TO 3 DO
        READ(INP,PA(.I.):6);
      READLN(INP);
(*  ELABORAZIONI  *)
      IF VA(.2.) <> 0 THEN VA(.1.):=VA(.1.)/VA(.2.);
(*  SCRITTURA  *)
      WRITE(OUT,TYP:1,' ');
      FOR I := 1 TO 2 DO
        WRITE(OUT,VA(.I.):8:0,' ');
      FOR I := 1 TO 3 DO
        WRITE(OUT,PA(.I.):6:0,' ');
      WRITELN(OUT);
    END;
  END;
(*  FINE  *)
END.
```


Dati di input

```
 1ASASASASAAASASASASASASAAAASASASASASASASASA
 20000999.0000333.00999.00888.00777.
 1CVCWCCVCCCVCVCVCVCCVCVCVCVCVCVCVCVCVCVCV
 20000066.0000033.00099.00088.00077.
 1DSSSDSDSSDSDSSDSDSDSDSDSDSDSDSDSDSDSDSDS
 20000003.0000003.00009.00008.00007.
```

Dati di output

```
 2        3      333    999    888    777
 2        2       33     99     88     77
 2        1        3      9      8      7
```

Considerazioni Pascal:

Come per il Cobol, abbiamo dovuto evitare l'uso di alcune parole riservate (TYPE, VAR), inoltre abbiamo utilizzato la codifica (. e .) al posto delle parentesi quadre.
Nel programma sono state inserite istruzioni superflue (LABEL, CONST) al solo scopo di dare un'idea leggermente piu' completa della varie possibilita' dichiarative; in particolare l'uso della label e del GOTO fara' inorridire i cultori di questo linguaggio, che avrebbero sicuramente sostituito le istruzioni in oggetto utilizzando una delle tante strutture (WHILE, REPEAT, ecc.) a disposizione.
Una nota merita il diverso trattamento della condizione di EOF durante la lettura di un file.
In Fortran, PL/1, Cobol e Basic la condizione di EOF viene rilevata durante lo stesso ciclo di lettura delle variabili: al fine di evitare l'esecuzione delle istruzioni successive e' pertanto necessario intrappolare subito tale condizione eseguendo un test nella stessa istruzione di lettura (Fortran, Cobol, Basic) o prevederla comunque in anticipo (ON condition in PL/1).
Diversamente si comporta il Pascal, che operando con una tecnica di puntatori, opera in maniera asincrona la lettura del file fisico e il trasferimento dei dati nelle variabili richieste: questo permette di conoscere 'in anticipo' la presenza dell'EOF (durante l'ultima lettura 'buona' il puntatore si sara' spostato in avanti preavvertendo la presenza dell'EOF) e quindi di effettuare il test dopo che i dati della precedente lettura sono stati elaborati.

BASIC

```
 10 REM
 20 REM  DICHIARAZIONI
 30 REM
 40 DIM V(2),P(3),76
 50 FORM SKIP,PIC(#),X1,2*PIC(Z#######.),X1,3*PIC(Z#####.)
 60 REM
 70 REM  LETTURA
 80 REM
 90 GET 'NEWDAT1', T, EOP 260
100 IF T = 2 THEN 130
110 GET 'NEWDAT1', F
120 GOTO 90
130 GET 'NEWDAT1', MAT V, MAT P
140 REM
150 REM  ELABORAZIONI
160 REM
170 IF V(2) <> 0 THEN V(1)=V(1)/V(2)
180 REM
190 REM  SCRITTURA
200 REM
210 PRINT USING 50, T, MAT V, MAT P
220 GOTO 90
230 REM
240 REM  FINE
250 REM
260 END
```


Dati di input

```
 1 'ASASASASAAASASASASASASAAAASASASASASASASASA'
 2 0000999. 0000333. 00999. 00888. 00777.
 1 'CVCVCCVCCCVCVCVCVCCVCVCVCVCVCVCVCVCVCVCVCV'
 2 0000066. 0000033. 00099. 00088. 00077.
 1 'DSSSDSDSSDSDSSDSDSDSDSDSDSDSDSDSDSDSDSDSDS'
 2 0000003. 0000003. 00009. 00008. 00007.
```

Dati di output

```
 2  0000003. 0000333.  00999. 00888. 00777.
 2  0000002. 0000033.  00099. 00088. 00077.
 2  0000001. 0000003.  00009. 00008. 00007.
```

Considerazioni Basic:

Un discorso un po' particolare necessita il Basic in quanto il prodotto VSBasic utilizzato per questo esempio ha, tra le altre limitazioni, quella di non permettere l'accesso con formato a files che non siano VSAM; i dati contenuti nel file utilizzato ad esempio per Fortran e PL/1 non sarebbero elaborabili con quel Basic. Abbiamo pertanto dovuto modificare il file dei dati, aggiungendo i necessari spazi di separazione tra le variabili (caratteri e numeriche) e gli apici all'inizio e alla fine delle sequenze di caratteri (per evitare errori di lettura). Questo falsa un po' il problema, ma permette di scrivere un programma analogo a quello negli altri linguaggi.
Un'altra limitazione e' che non e' ammesso l'uso del FILEDEF per associare il nome simbolico interno di file (INP in PL/1 e 08 in Fortran) con il vero file fisico su disco: in VSBasic il nome fornito nel programma deve essere il nome del file su disco, mentre il tipo deve essere obbligatoriamente VSBDATA.