COMAL: la strana storia di un linguaggio dimenticato

Sommario

Tra il 1973 e il 1974 il matematico danese Børge R. Christensen e il suo collega informatico Benedict Løfstedt definirono le specifiche della prima versione di *COMAL* (*COMmon Algorithmic Language*): un nuovo linguaggio che si impose rapidamente in tutte le scuole della Danimarca e successivamente in gran parte del Nord europeo grazie alle sue peculiari caratteristiche. Semplice come il *BASIC* ma strutturato come il Pascal, prettamente pensato per la didattica come Logo e Forth, finirà per essere utilizzato anche in campo applicativo per tutti gli anni Ottanta e oltre, grazie ad interpreti e compilatori disponibili per una vasta quantità di home e personal computer.

«If you want more effective programmers, you will discover that they should not waste their time debugging: they should not introduce the bugs to start with.»
(Edsger W. Dijkstra, 1930-2002)

1 Introduzione.

Ripercorriamo sinteticamente la storia e le caratteristiche di un linguaggio che ha avuto il suo momento di gloria negli anni Ottanta, in ambito didattico ma anche applicativo. *COMAL* (acronimo di *COMmon Algorithmic Language*) nasce in Danimarca per una precisa esigenza didattica: l'impiego in corsi di alfabetizzazione informatica agli inizi degli anni Settanta, per i quali il *BASIC* era considerato inadeguato a causa della sua mancanza di struttura, ma al contempo il Pascal risultava eccessivamente avanzato e complesso. Durante gli anni Ottanta *COMAL* è stato implementato su due dozzine di home computer: particolare fortuna hanno avuto la versione interpretata 0.14 *Public Domain* su disco per Commodore 64, seguita dalla versione 2.0 su cartridge (molto più perfezionata, con una notevole quantità di memoria a disposizione dei programmi utente), e il compilatore per Atari con il quale sono stati prodotti anche numerosi applicativi commerciali, sia per usi domestici che burotici in area *SOHO*, distribuiti in tutta Europa.

2 La genesi di COMAL.

In una lunga e semiseria intervista rilasciata a «COMAL Today» e apparsa sul numero 25 (1989), il matematico danese Børge R. Christensen illustra la genesi del linguaggio e la sua a tratti rocambolesca implementazione su un Data General NOVA 1200 dotato unicamente di una unità a nastro carta perforato per la memorizzazione. Vogliamo qui riassumerne almeno i passaggi più salienti, pur rifuggendo dall'idea di proporre una pedissequa traduzione integrale che in quest'ambito divulgativo e colloquiale finirebbe per divenire uno sterile esercizio di acribia filologica, annoiando i lettori.

Figura 1: Data General NOVA 1200

Nel 1972 il college danese di Tønder (vicino al confine con la Germania) presso il quale il professor Christensen insegna matematica decide di dotarsi di uno dei primi calcolatori commerciali per applicazioni di ricerca e per la didattica. I primissimi corsi di informatica non sembrano tuttavia avere un particolare successo, e questo viene attribuito in particolare alla versione notevolmente fallata di *Extended BASIC* preinstallata sul calcolatore. Christensen lascia in realtà trasparire che anche la sua limitata esperienza in materia ha il suo peso, rendendogli difficile la lettura dei programmi *BASIC*: per superare

questi dubbi, si reca presso la vicina università di Årus, nota per avere un dipartimento di informatica piuttosto efficiente. Sfruttando la contingenza di dover preparare i problemi d'esame per il suo primo (piuttosto disastroso) corso di informatica, chiede una consulenza a Benedict Løfstedt che lavora appunto in tale università. La risposta di Løfstedt è perfettamente in linea col pensiero informatico dell'epoca: la colpa non è del docente né degli studenti, bensì del linguaggio. Si ricordi che in quell'epoca, tra gli anni Sessanta e Settanta, siamo ancora nel pieno di un dibattito molto acceso nella comunità informatica sulla contrapposizione tra programmazione strutturata e non, tra linguaggi come *ALGOL* da un lato e *BASIC* dall'altro, con numerosi punti di contrasto e posizioni fortemente contrapposte, ad esempio, in merito all'uso della keyword `GOTO`: dibattito dal quale, oltre a fondamentali articoli come quelli di Edsger W. Dijkstra[1] [Dij87, Dij82] e accorati interventi di altri padri nobili dell'informatica come C. A. R. (Tony) Hoare (1934-), Niklaus Wirth (1934-) e Donald E. Knuth (1938-), scaturisce tra l'altro nel 1966 anche un fondamentale teorema di estensione della tesi di Church-Turing sulla computabilità, dovuto agli italiani Corrado Böhm (1923-2017) e Giuseppe Jacopini (1936-2001) [BJ66], il quale sancisce che qualsiasi algoritmo computabile secondo Church-Turing può essere implementato in un linguaggio minimale dotato unicamente di costrutti sequenziali, selezioni e iterazioni (cicli), il che quindi esclude la stretta necessarietà di uno statement come `GOTO`.

In breve, Christensen viene prontamente indirizzato ad un libro seminale di Niklaus Wirth [Wir73] che è una pietra miliare nella storia dell'informatica applicativa e ne riceve una vera e propria illuminazione. Tuttavia, pur affascinato dal Pascal e dalla programmazione imperativa strutturata, si rende presto conto che la difficoltà di utilizzo per i suoi allievi, nella fascia d'età del college, sarebbe comunque eccessiva. Decide quindi di mantenere un ambiente interattivo e un linguaggio interpretato, ma durante tutto il 1973 in una lunga serie di scambi epistolari col collega Løfstedt costruisce la definizione di un linguaggio sostanzialmente nuovo, ibridando la struttura del Pascal con l'intuitività del *BASIC* e gettando le basi per quello che diventerà lo standard *COMAL 75*.

Se la genesi della definizione risulta relativamente semplice, la prima implementazione del linguaggio si rivela ben presto assai più ardua. Dopo avere invano tentato di rivolgersi ad alcune aziende locali, Christensen incarica due dei suoi migliori studenti di aiutarlo a codificare in Assembly sul NOVA 1200 l'intera implementazione dell'ambiente interattivo e dell'interprete, ed è qui che il suo racconto assume un tono vagamente epico, sebbene sempre intriso di autoironia. La pressoché totale mancanza di esperienza dell'improvvisato team, nonostante la consulenza intermittente di Løfstedt, e la fissazione su alcune caratteristiche didatticamente utili ma onerose dal punto di vista computazionale, come i nomi di variabile estesi a ben (sic!) 8 caratteri rispetto ai due consentiti dal *BASIC*, provocano un accumulo di ritardi. Un primo prototipo risulta effettivamente operativo il 5 agosto 1974, ma una versione realmente utilizzabile viene completata solo nel febbraio 1975. Christensen indulge anche nel colorire il racconto sottolineando come uno dei due studenti prescelti, incaricato in particolare del debugging, avesse all'epoca il poco edificante vizio dell'alcool, risultando spesso fuori combattimento per lunghi periodi a causa di sbornie colossali e ritardando così ulteriormente lo sviluppo. Altro aspetto pionieristico dell'intera vicenda è sicuramente l'imbarazzante limitatezza dei mezzi a disposizione, decisamente primitivi anche per l'epoca: soprattutto il terrificante nastro carta perforato, in mancanza di qualsivoglia mezzo di memorizzazione magnetico nell'installazione Data General in uso, ma anche il costo complessivo dell'intera operazione, che si attesta su un budget di soli 300 dollari.

Figura 2: Esemplari di nastro carta perforato, larghezza 5 (giallo) e 8 (rosa) fori.

Tuttavia, proprio l'universale diffusione nelle scuole danesi di ogni ordine e grado di macchine NOVA identiche o simili a quella utilizzata per lo sviluppo risulta essere il fattore strategico che decreta una immediata diffusione di *COMAL* nel mondo della didattica: in meno di un anno la maggioranza degli istituti sul territorio nazionale ha abbandonato il *BASIC* dell'installazione di default in favore del nuovo linguaggio, anche se in molti casi il caricamento del nastro carta originale con una telescrivente con lettore a 10 caratteri/secondo richiede poco più di un'ora di tempo. Christensen si compiace nell'attribuire tale successo principalmente alla presenza di nomi di variabile di lunghezza relativamente significativa, come risulta dalle interviste agli studenti, e ovviamente alla possibilità di usare costrutti strutturati. Come terza motivazione viene citata la possibilità di attribuire un nome alle subroutines, in realtà implementata fin dai primissimi linguaggi di alto livello (*COBOL* in primis) ma all'epoca

[1] (1930-2002) Fisico teorico per formazione, è stato uno dei più influenti informatici teorici dell'ultimo mezzo secolo. Pioniere e anticipatore dell'uso dei metodi formali di specifica e verifica, è ben noto in ambito informatico generalista anche come autore di alcuni dei più taglienti aforismi in merito a taluni linguaggi di programmazione in voga all'epoca del dibattito di cui discutiamo, ancor oggi citati sovente - sebbene quasi sempre a sproposito, fuori dal loro precipuo contesto storico.

inesistente nei *BASIC* tradizionali. Tutto ciò sottolinea come *COMAL 75* nasce dal traumatico incontro con le limitazioni di un *BASIC* decisamente primitivo e quindi ne costituisce un tentativo di superamento, anche in termini prestazionali, pur rimanendo nell'ambito di un linguaggio interpretato. Accorgimenti come l'uso estensivo di jump tables dinamiche, puntatori e link diretti alle procedure nascono esplicitamente per superare meccanismi realmente arretrati e farraginosi come la ricerca sequenziale effettuata dall'interprete nella vera e propria lista linkata semplice costituita dall'insieme delle linee *BASIC* in caso di `GOSUB`, che nel decennio successivo saranno progressivamente abbandonati anche nella maggioranza degli home *BASIC* interpretati.

Si noti che, a questo punto della sua storia, il linguaggio è ancora caratterizzato da un impronunciabile nome danese e probabilmente, sotto tale egida e legato a doppio filo com'era con una serie di macchine non particolarmente diffuse su scala globale e con un mezzo di distribuzione poco pratico come il nastro perforato, non avrebbe mai superato i confini del regno di Danimarca. In realtà è lo stesso Christensen, secondo la sua narrazione, a ideare il nome *COMmon Algorithmic Language* per banale analogia con l'*ALGOL* (*ALGOrithmtic Language*), all'epoca uno dei protagonisti di maggior rilievo nel campo della programmazione strutturata: progettato e standardizzato accuratamente, tanto da essere usato (nella versione *ALGOL 60* in particolare) come linguaggio ufficiale per l'espressione di algoritmi negli articoli scientifici per oltre tre decenni. Anche la metasintassi conosciuta come Forma di Backus–Naur (BNF), ben nota a qualunque informatico che abbia compiuto un minimo di studi istituzionali, è stata in origine concepita specificamente per descrivere la sintassi dei programmi in *ALGOL*: al pioniere John Backus (1924-2007) si deve l'ideazione di tale formalismo per *ALGOL 58*, perfezionato poi da Peter Naur (1928-2016) con *ALGOL 60*, come magistralmente illustrato dall'ineffabile Donald Knuth [Knu64]. Nella pratica l'unica aggiunta alla BNF (standardizzata dalla ISO/IEC 14977:1996, rev. 2018) che non sia riconducibile direttamente ad *ALGOL* è semplicemente l'uso delle parentesti quadre [ ] per delimitare i parametri opzionali: introdotto in realtà pochi anni dopo *ALGOL 60* con la definizione del linguaggio *PL/I* di IBM e poi diffusosi universalmente anche nelle sinossi informali di comandi e funzioni d'ogni genere, dai comuni linguaggi di programmazione ai batch, alle shell Unix, fino all'help online di singoli programmi applicativi, eccetera.

Piccolo inciso: un aspetto decisamente saliente della BNF e delle sue estensioni (come EBNF o le specializzazioni come TBNF, ABNF o RBNF) è l'espressività. Essa risulta sufficiente a descrivere anche la meta-metasintassi stessa di BNF in modo molto conciso, autoreferenziale e ricorsivo, come riportiamo di seguito da un classico esempio presente nella maggioranza dei testi, abbreviato per mere ragioni tipografiche. Risulta poi intuitivo come ad esempio in EBNF, grazie all'uso delle espressioni regolari, l'autodefinizione risulti ancora più concisa a livello di <letter>, <digit>, <symbol>.

```
<syntax>          ::= <rule> | <rule>
<syntax> <rule>   ::= <opt-whitespace> "<" <rule-name> ">"
<opt-whitespace> "::=" <opt-whitespace> <expression> <line-end>
<opt-whitespace>  ::= " " <opt-whitespace> | ""
<expression>      ::= <list> | <list> <opt-whitespace> "|" <opt-whitespace> <expression>
<line-end>        ::= <opt-whitespace> <EOL> | <line-end> <line-end>
<list>            ::= <term> | <term> <opt-whitespace> <list>
<term>            ::= <literal> | "<" <rule-name> ">"
<literal>         ::= '"' <text1> '"' | "'" <text2> "'"
<text1>           ::= "" | <character1> <text1>
<text2>           ::= '' | <character2> <text2>
<character>       ::= <letter> | <digit> | <symbol>
<letter>          ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | ...
<digit>           ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<symbol>          ::=  "|" | " " | "!" | "#" | "$" | "%" | "&" | "(" | ")" | ...
<character1>      ::= <character> | "'"
<character2>      ::= <character> | '"'
<rule-name>       ::= <letter> | <rule-name> <rule-char>
<rule-char>       ::= <letter> | <digit> | "-"
```

Una volta ideato un nome convincente per il nuovo linguaggio, dopo la presentazione e la rapida diffusione di *COMAL 75* negli istituti scolastici danesi, il professor Christensen si dedica alla stesura della documentazione del progetto, creando così le basi per quello che sarà il primo libro di testo del linguaggio [Chr82], originariamente pubblicato in danese qualche anno dopo la prima versione del linguaggio, nel 1978. Nel frattempo però i minicomputer

NOVA si avviano rapidamente verso l'obsolescenza, a causa dell'invasione del mercato da parte dei primi PET: il mero timore di poter essere dantescamente rispedito ad insegnare qualche *BASIC*, così ci racconta coloritamente il simpatico matematico danese, è sufficiente a spingere il suo gruppo di lavoro a perfezionare una ulteriore versione del linguaggio, più completa ed efficiente, destinata alle piattaforme emergenti. Così il gruppo di lavoro si amplia, unendo i suoi sforzi ad un team in un altro college e nel giro di un paio d'anni si arriva alla versione *COMAL 80*, così denominata sia in base al fattore cronologico (1980 è l'anno di presentazione), sia perché sviluppata prevalentemente su una macchina basata sullo Zilog Z80. Nel maggio 1982 viene ratificato lo standard internazionale denominato *COMAL KERNAL*, poi sottoposto a riapprovazione 1983. In quegli stessi anni, a maggior gloria del nazionalismo danese, una società locale ha iniziato a progettare e realizzare computer specificamente dedicati alla didattica, grazie soprattutto all'uso di un moderno bus che rende facile l'espansione della macchina e anche la sperimentazione elettronica negli istituti tecnici: ovviamente tale macchina è dotata di *COMAL*. Al linguaggio però in questo momento mancano ancora molte delle caratteristiche che lo hanno reso famoso nei due decenni successivi: grafica e sprites, suono, un editor avanzato, un ambiente completo che consenta di gestire direttamente nastri, dischi e altre memorie di massa. Tuttavia, si è già ricavato una solida posizione di predominio nell'ambito didattico, tanto che a partire dai primi anni Ottanta nessun istituto danese accetta forniture di calcolatori che non abbiano *COMAL* a bordo.

Nel 1982 il danese Mogens Kjær, dietro consiglio di Christensen e sulla base dello standard *COMAL KERNAL*, inizia a sviluppare quella che sarà la famosa e diffusa serie di versioni 0.1x *Public Domain* per Commodore PET e successivamente per C64, avviando assieme ad un gruppo di pionieri ed entusiasti (Jens Erik Jensen, Helge Lassen e Lars Laursen) una società denominata *UniComal ApS*, che di fatto succede al team originale nello sviluppo del linguaggio e nel porting verso altre piattaforme. L'ambiente di lavoro viene espanso e perfezionato per sostituire completamente quello del *BASIC V2* e il relativo *DOS Wedge*, dando la possibilità di gestire con semplici e intuitivi comandi i file su nastro e disco, le stampanti e altre periferiche. Vengono aggiunti i moduli binari, denominati «packages», che sono librerie di routine in codice macchina richiamabili direttamente da *COMAL*.

Pochi anni dopo nasce anche una versione sviluppata appositamente per gli Amstrad, successivamente anche un compilatore per Atari e una lunga serie di porting per altri home computer, inclusa una versione avanzata di *COMAL 80* su cartridge (la 2.0) per C64 commercializzata nel 1985 che resterà in auge a lungo, praticamente per l'intera storia di tale home computer. Viene presto sviluppata anche una versione con compilatore per i potentissimi VAX-11, evento più unico che raro nella storia dei linguaggi didattici nati su home computer. Negli anni Ottanta e Novanta il linguaggio, nelle sue varie versioni, viene utilizzato per la didattica in numerosi paesi anglofoni del Nord Europa, ma anche per svariate applicazioni commerciali, in particolar modo su C64 con apposita cartridge e soprattutto su Atari e VAX nella versione compilata.

Chiudiamo questa succinta cronistoria con una doverosa citazione di Børge R. Christensen: «*COMAL* is first of all for people who are not professional... to make it possible for people to program computers, even if they were not programming people.». Come talora avviene, si tratta però di un linguaggio talmente ben concepito da attrarre immediatamente e per lungo tempo a venire l'attenzione di molti professionisti, che data l'epoca pionieristica della sua diffusione non ha rischiato di diventare un Santo Graal di pasticcioni e improvvisatori, come è invece è tristemente accaduto alla svolta del millennio con linguaggi concepiti nominalmente con le medesime intenzioni. Inoltre è indiscutibile merito di questo linguaggio l'aver abbreviato il percorso verso il conseguimento di solide competenze di *problem solving* e logica della programmazione per intere generazioni di studenti in gran parte del sistema scolastico europeo anglofono.

3 Un primo sguardo a *COMAL 80* per C64.

La fortunata serie di release per Commodore avviata dal *COMAL 0.11* rilasciato da *UniComal ApS* nel 1982 è culminata con una versione di larghissima diffusione, la 0.14, alla quale si riferiscono anche numerosi manuali. Tale versione *Public Domain*, caricata da floppy o da nastro, non lasciava tuttavia molto spazio disponibile per i programmi utente (circa 9.900 bytes sul C64): in ultima analisi, ciò non costituiva in realtà un problema per il principale utilizzo del linguaggio, quello didattico e formativo. Dopo una versione intermedia 1.2 destinata principalmente al Commodore PET 8096, compare la versione 2.0, meglio nota come *COMAL 80 per C64* prodotta nel 1984 e distribuita su cartuccia CBM nei primi mesi del 1985. Tale versione, disponibile anche su schede con ROM associate ad espansioni di memoria per i PET, garantiva sul C64 30.714 bytes di spazio per i programmi utente, più del triplo delle versioni precedenti. Essa risultava notevolmente più potente e completa delle prime release, ed è stata utilizzata con successo nell'arco di due decenni non solo per la didattica, ma anche per numerosi programmi commerciali.

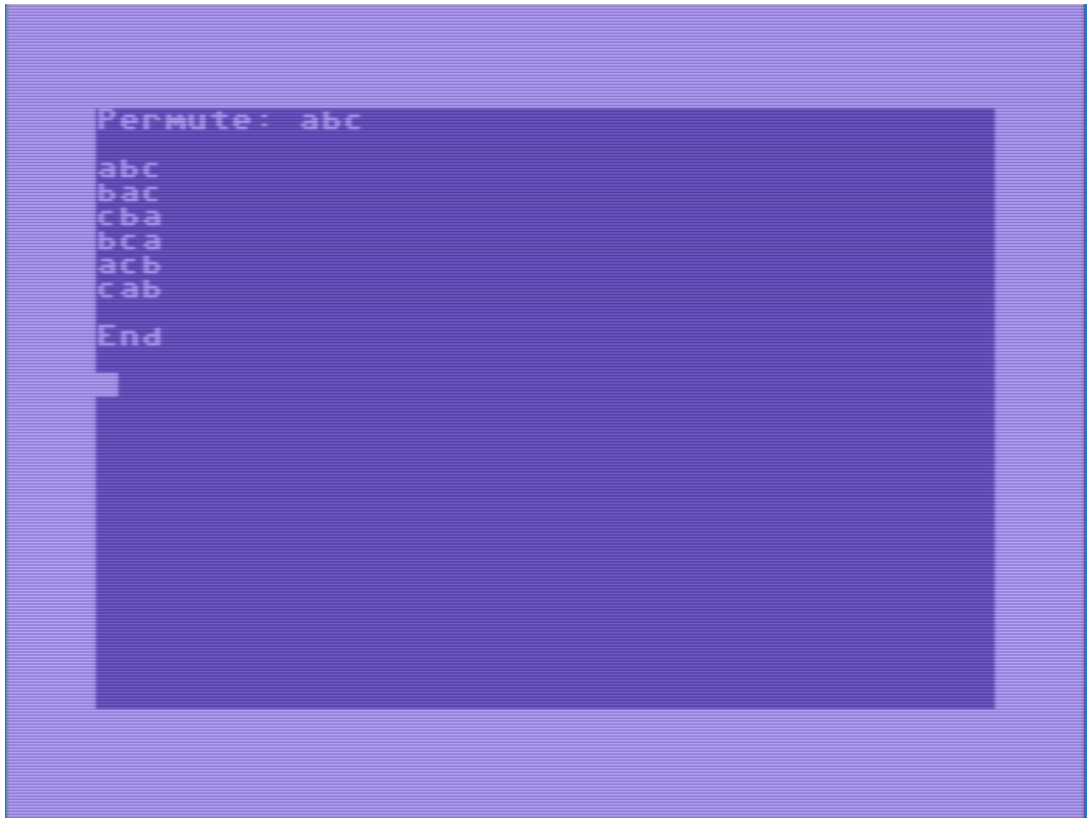
L'ambiente interattivo sostituiva completamente l'originale del C64, rendendo disponibili potenti comandi per la gestione di dischi e nastri, la stampa, le periferiche su porta utente e seriali, librerie grafiche, perfino un sottosistema

LOGO Turtle, editor di programma fullscreen e molto altro sotto forma di moduli binari abilitabili a piacimento. Risulta inoltre molto semplice espandere il linguaggio con nuove *keyword* create dall'utente e con moduli binari scritti in Assembly.

In questa sede preferiamo però evitare di iniziare con una sterile, pedissequa elencazione delle caratteristiche del linguaggio e delle sue peculiarità sintattiche (tutti aspetti peraltro ampiamente sviscerati nella manualistica indicata in bibliografia), lasciando invece «parlare» degli esempi concreti di codifica come primo primo assaggio della potenza e della reale modernità di *COMAL 80*. Senza voler appesantire la trattazione con puntuali citazioni dai testi di riferimento fondamentali in materia come [Seb15, FW08], si preferisce che il lettore valuti intuitivamente, *hands-on*, la leggibilità e la manutenibilità del linguaggio come caratteristiche oggettive di software engineering e language design. Per cominciare proponiamo quindi un brevissimo programma che in sole 23 LOC genera esaustivamente tutte le permutazioni dei simboli *PETSCII* immessi in input in una stringa arbitraria (lunghezza massima 20 caratteri, comunque sconsigliata anche ai più temerari, sia pure su emulatori con warp mode). Il programma è tratto direttamente dal disco di esempi che accompagnava la cartridge *COMAL 80*.

```
0010 // SAVE "permute"
0020 //
0030 // (c) 1984 by UniComal ApS.
0040 //
0050 PAGE
0060 DIM a$ OF 20
0070
0080 PROC permute(a$,k)
0090    IF k<=1 THEN
0100       PRINT a$
0110    ELSE
0120       permute(a$,k-1)
0130       FOR i:=1 TO k-1 DO
0140          permute(a$(:i-1)+a$(k)+a$(i+1:k-1)+a$(i)+a$(k+1:),k-1)
0150       ENDFOR i
0160    ENDIF
0170 ENDPROC permute
0180
0190 INPUT "Permute: ": a$
0200 PRINT
0210 permute(a$,LEN(a$))
0220
0230 END "End"
```

Balzano immediatamente agli occhi la sintesi, l'intuitività e l'assoluta chiarezza della struttura logica del programma, tipiche della programmazione strutturata quanto antitetiche ai primitivi *BASIC* dei PET. I numeri di linea vengono generati automaticamente dall'editor e sono allineati per default a 4 cifre, aumentando notevolmente la leggibilità. Il programma non fa uso di nomi lunghi di variabile, in quest'occasione. Si nota immediatamente l'uso di un classico algoritmo ricorsivo, sul quale non ci soffermeremo se non per sottolineare come il parametro stringa passato di volta in volta alla funzione sia in realtà l'unione di sottostringhe ricavate dinamicamente con «modernissime» espressioni di *slicing*, che molti studenti e practitioners credono nate con Python negli anni Novanta o con linguaggi anche più recenti, ma in realtà supportate da oltre mezzo secolo dai benemeriti *FORTRAN*, *APL*, *ALGOL*, Ada e perfino da qualche home *BASIC* dei primi anni Ottanta, come ad esempio il *Sinclair BASIC*. Si notino inoltre i commenti che qualcuno potrebbe essere tentato di definire «in stile C++», in realtà altra «modernità» degli anni Settanta: alla quale ovviamente Bjarne Stroustrup nei primi anni Ottanta può essersi liberamente ispirato, a margine del notevole sforzo di concepire il suo *C with classes* come evoluzione e superamento delle possibilità di *SIMULA*.



Come secondo assaggio, nulla di meglio che un classico quicksort come proposto dal disco *COMAL 80*. Algoritmo oggettivamente non facile da implementare in *BASIC V2* in meno di 60 LOC, mantenendo l'assoluta chiarezza del seguente listato. Chiunque abbia programmato intensivamente in *BASIC V2* saprà sicuramente apprezzare, tra l'altro, la robustezza e l'intuitività dello statement `WHILE NOT EOD DO` che cancella con un colpo di spugna l'incubo dell'*off-by-one* e più in generale il temibile errore `OUT OF DATA`.

Si noti anche l'eleganza degli statement di assegnazione composita, che eliminano le inguardabili istruzioni di autoassegnazione come `A = A + 1`, probabilmente uno degli aspetti soggettivamente più odiati dai programmatori HLL alle prese con il *BASIC V2*.

```
0010 // save "quicksort"
0020 //
0030 // by UniComal ApS. may 1984
0040 //
0050
0060 PROC quicksort(REF a$(),left,right,reclen) CLOSED
0070   DIM pivot$ OF reclen, buffer$ OF reclen
0080    partition(left,right,left,right) // sort a$(left:right)
0090
0100   PROC partition(left,right,i,j)
0110     pivot$:=a$((left+right) DIV 2) // get middle element as pivot
0120     REPEAT // perform swappings
0130       WHILE pivot$>a$(i) DO i:+1
0140       WHILE pivot$<a$(j) DO j:-1
0150       IF i<=j THEN swap(a$(i),a$(j)); i:+1; j:-1
0160     UNTIL i>j
0170     IF left<j THEN partition(left,j,left,j) // sort a$(left:j)
0180     IF i<right THEN partition(i,right,i,right) // sort a$(i:right)
0190   ENDPROC partition
0200
```

```
0210    PROC swap(REF a$,REF b$)
0220       buffer$:=a$; a$:=b$; b$:=buffer$
0230    ENDPROC swap
0240 ENDPROC quicksort
0250
0260 PAGE
0270 DIM message$(1:50) OF 35
0280 messageno:=0
0290
0300 WHILE NOT EOD DO
0310    messageno:+1
0320    READ message$(messageno)
0330 ENDWHILE
0340
0350 // sort error messages:
0360
0370 PRINT "sorting started:"
0380 quicksort(message$(),1,messageno,35)
0390
0400 // print the sorted messages:
0410
0420 FOR i:=1 TO messageno DO PRINT i,":";message$(i)
0430
0440 END "End of sort"
0450
0460 // start-of-data
0470
0480 DATA "format error"
0490 DATA "syntax error"
0500 DATA "type conflict"
0510 DATA "function argument error"
0520 DATA "statement too long or too complicated"
0530 DATA "system error"
0540 DATA "name too long"
0550 DATA "bracket error"
0560 DATA "overflow"
0570
0580 // end-of-data
```

Un terzo esempio, sempre tratto dal disco originale, genera esaustivamente tutte le soluzioni al classico problema scacchistico detto «delle otto regine»: data una scacchiera standard bicolore da 64 caselle, si devono posizionare 8 regine in modo tale che nessuna di esse minacci le altre, ossia ponendo al più un solo pezzo per ogni riga, colonna e diagonale. Sebbene non si disponga di una formula chiusa per calcolare il numero delle soluzioni a partire dalla dimensione n della scacchiera data, è noto che con 8 regine esistono 92 soluzioni distinte in totale, derivate per rotazioni e riflessioni da 12 soluzioni di base, dette «uniche». Il breve programma qui proposto genera sequenzialmente, con accattivanti animazioni grafiche, tutte le 92 possibili configurazioni che risolvono il problema. Si noti che i dati degli sprites vengono elegantemente letti da un file dati sequenziale e non sono incorporati nel codice.

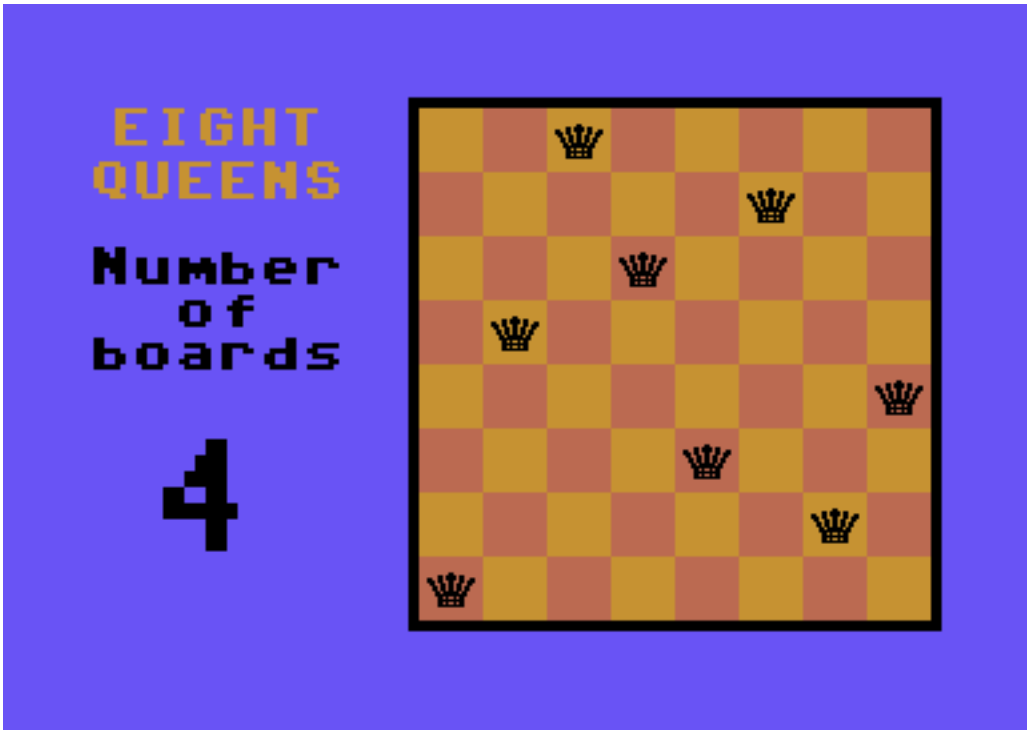
Si invita il lettore ad apprezzare la chiarezza del programma, grazie alla forte strutturazione, all'uso estensivo di nomi lunghi per le varabili e per le procedure, oltre alla presenza dei commenti: ricordando inoltre che tutto ciò avveniva nel lontano 1985 (ed era comunque possibile anche prima su Commodore PET e C64, con *COMAL 0.1x*), quando gli home *BASIC* su hardware di pari classe offrivano livelli di leggibilità, sintesi e mantenibilità nettamente inferiori e i compilatori per HLL come Pascal erano una rarità. Il programma, nello specifico, fa uso di backtracking: una soluzione decisamente classica sebbene dalle prestazioni non entusiasmanti.

```
0010 // SAVE "queens"
0020 //
0030 // (c) 1984 by UniComal ApS.
0040 //
0050 // Find all possible placements
0060 // of 8 queens on a chess board
0070 // in such a fashion that none
0080 // is checking any other piece,
0090 // i.e. each row, column and
0100 // diagonal must contain at most
0110 // one piece.
0120
```

```
0130 PAGE
0140 m:=24 // size of squares //
0150 l:=8*m // size of board
0160 x0:=(320-l)-4
0170 y0:=(200-l) DIV 2
0180 homex:=x0-27
0190
0200 red:=2; blue:=6
0210 black:=0; orange:=8
0220
0230 USE graphics
0240 graphicscreen(1)
0250 // blue background //
0260 background(blue)
0270 border(-1)
0280 clearscreen
0290
0300 USE sprites
0310 queen:=0
0320 DIM xpos(0:7)
0330 // define queen shape //
0340 OPEN FILE 1,"queen.spr",READ
0350 define(queen,GET$(1,64))
0360 CLOSE
0370
0380 DIM row(1:8)
0390 // row(i) = "no queen on i-th row" //
0400 DIM d1(1:2*8)
0410 // d1(i) = "no queen on i-th upleft to lowright diagonal //
0420 DIM d2(1:2*8-1)
0430 // d2(i) = "no queen on i-th lowleft to upright diagonal" //
0440
0450 //------main-------//
0460
0470 clear.and.draw.board
0480 trycol(1)
0490 FOR i:=0 TO 7 DO
0500   slideto(i,homex)
0510 ENDFOR i
0520 END "End Queens"
0530
0540 //-------procedures--------//
0550
0560 PROC paint(c,x,y,w,h)
0570   // paint rectangle //
0580   viewport(x,x+w-1,y,y+h-1)
0590   pencolor(c)
0600   fill(x,y)
0610 ENDPROC paint
0620
0630 PROC clear.and.draw.board
0640   FOR i:=1 TO 8 DO row(i):=TRUE
0650   FOR i:=1 TO 2*8 DO d1(i):=TRUE
0660   FOR i:=1 TO 2*8-1 DO d2(i):=TRUE
0670   // draw text //
0680   pencolor(orange)
```

```
0690    type(0,180,1,2,"EIGHT")
0700    type(0,160,1,2,"QUEENS")
0710    pencolor(black)
0720    type(0,128,1,2,"Number")
0730    type(0,112,1,2,"   of")
0740    type(0,96,1,2,"boards")
0750    boards:=-1
0760    newboard
0770    // draw border //
0780    paint(black,x0-4,y0-4,8+l,4)
0790    paint(black,x0-4,y0,4,l)
0800    paint(black,x0-4,y0+l,8+l,4)
0810    paint(black,x0+l,y0,4,l)
0820    // define colors of the queens //
0830    y:=y0+22
0840    FOR sprite:=0 TO 7 DO
0850       spritecolor(sprite,black)
0860       xpos(sprite):=homex
0870       spritepos(sprite,homex,y)
0880       identify(sprite,queen)
0890       showsprite(sprite)
0900       spritesize(sprite,0,0)
0910       y:+m
0920    ENDFOR sprite
0930    // draw board //
0940    y:=y0
0950    color:=red
0960    FOR row:=1 TO 8 DO
0970       x:=x0
0980       FOR col:=1 TO 8 DO
0990          paint(color,x,y,m,m)
1000          x:+m
1010          color:=red+orange-color
1020       ENDFOR col
1030       y:+m
1040       color:=red+orange-color
1050    ENDFOR row
1060    last:=-1
1070    viewport(0,319,0,199)
1080 ENDPROC clear.and.draw.board
1090
1100 PROC pause
1110    FOR i:=1 TO 7000 DO NULL
1120 ENDPROC pause
1130
1140 PROC slideto(sprite,x1)
1150    y1:=y0+sprite*m+22
1160    stp:=SGN(x1-xpos(sprite))
1170    IF stp<>0 THEN
1180       FOR x:=xpos(sprite) TO x1 STEP stp DO
1190          spritepos(sprite,x,y1)
1200       ENDFOR x
1210       xpos(sprite):=x1
1220    ENDIF
1230 ENDPROC slideto
1240
```

```
1250 PROC place.queen(i,j)
1260   row(i):=FALSE; d1(i+j):=FALSE; d2(8+i-j):=FALSE
1270   IF last>0 AND last<>j THEN
1280     slideto(last-1,homex)
1290   ENDIF
1300   slideto(j-1,x0+(i-1)*m)
1310   last:=-1
1320 ENDPROC place.queen
1330
1340 PROC remove.queen(i,j)
1350   row(i):=TRUE; d1(i+j):=TRUE; d2(8+i-j):=TRUE
1360   IF last>0 THEN
1370     slideto(last-1,homex)
1380   ENDIF
1390   last:=j
1400 ENDPROC remove.queen
1410
1420 PROC trycol(j)
1430   FOR i:=1 TO 8 DO
1440     IF row(i) AND d1(i+j) AND d2(8+i-j) THEN
1450       place.queen(i,j)
1460       IF j<8 THEN
1470         trycol(j+1)
1480       ELSE // all queens placed //
1490         newboard
1500         pause
1510       ENDIF
1520       remove.queen(i,j)
1530     ENDIF
1540   ENDFOR i
1550 ENDPROC trycol
1560
1570 PROC newboard
1580   // draw new number //
1590   pencolor(black)
1600   boards:+1
1610   type(0,24,3,6,STR$(boards))
1620 ENDPROC newboard
1630
1640 PROC type(x0,y0,xsize,ysize,text$)
1650   IF LEN(text$) MOD 2=1 THEN
1660     x0:+xsize*8
1670   ENDIF
1680   textstyle(xsize,ysize,0,0)
1690   plottext(x0,y0,text$)
1700 ENDPROC type
```



4 Bibliografia essenziale.

I testi originali di Børge R. Christensen, creatore del linguaggio con Benedict Løfstedt, sono [Chr82, Chr84]. Altri testi generici sul linguaggio includono [Ath82, BP88, Gra85, Kel84, LO90].

COMAL 80 per C64 è interamente documentato nel manuale originale Commodore «COMAL 80» (a cura di Frank Bason e Leo Højsholt-Poulsen, 1985) pubblicato in duplice edizione inglese e danese, relativo alla versione 2.0 su cartuccia con disco allegato. Specifici per C64 anche i testi di Lindsay [Lin83] e J. William Leary [Lea86].

5 Conclusioni.

Si è presentata una brevissima cronistoria della definizione e creazione del linguaggio *COMAL* (in particolare le versioni *milestone* 75 e 80 e lo standard 1982/83 noto come *COMAL KERNAL*) da parte del matematico danese Børge R. Christensen e del suo collega informatico Benedict Løfstedt, seguita dalle principali tappe dello sviluppo commerciale europeo del linguaggio in numerose versioni, concentrandoci in particolare sulle principali release per PET e Commodore 64. Si sono proposti alcuni semplicissimi sorgenti di esempio al fine di incuriosire il lettore, mostrando solo alcuni aspetti della potenza e della leggibilità di un linguaggio di notevole innovatività nell'ambito degli home computer: con la speranza di avere suscitato anche qualche gradito ricordo in chi all'epoca avesse provato ad utilizzare il linguaggio su C64 come alternativa al *BASIC V2* o su altre piattaforme. Si è infine esposta una bibliografia di riferimento sul linguaggio, comprendente pressoché tutti i testi in lingua inglese ritenuti più autorevoli e di maggiore diffusione.

Qualora dopo la pubblicazione del presente articolo si riscontrasse sufficiente interesse attorno al linguaggio, ciò che l'autore auspica, si potrà dare continuità alla trattazione con la presentazione di ulteriori esempi e quesiti risolti con l'uso di *COMAL 80* per C64, anche in parallelo con le relative soluzioni in Assembly 6510 e *BASIC V2*.

Riferimenti bibliografici

[Ath82] Roy Atherton, *Structured programming with comal (ellis horwood series in computers and their applications)*, Ellis Horwood Ltd , Publisher, 1982.

[BJ66] Corrado Boehm and Giuseppe Jacopini, *Flow diagrams, turing machines and languages with only two formation rules*, Commun. ACM **9** (1966), no. 5, 366–371.

[BP88] Marcus D. Bowman and Stephen Pople, *Computing studies in context: Comal programming guide*, Heinemann Educational Publishers, 1988.

[Chr82] Borge Christensen, *Beginning comal*, Harwood & Charles Pub Co, 1982.

[Chr84] Borge R Christensen, *Comal reference guide*, Toronto PET users group, 1984.

[Dij82] Edsger W. Dijkstra, *How do we tell truths that might hurt?*, SIGPLAN Not. **17** (1982), no. 5, 13–15.

[Dij87] Edsger W. Dijkstra, *Go to statement considered harmful*, Commun. ACM **30** (1987), no. 3, 195–196.

[FW08] Daniel P. Friedman and Mitchell Wand, *Essentials of programming languages, 3rd ed.*, MIT Press, 2008.

[Gra85] Ingvar Gratte, *Starting with comal*, Simon & Schuster (Paper), 1985.

[Kel84] John Kelly, *Foundations in computer studies with comal*, The Educational Company of Ireland, Ltd., 1984.

[Knu64] Donald E. Knuth, *Backus normal form vs. backus naur form*, Commun. ACM **7** (1964), no. 12, 735–736.

[Lea86] J. William Leary, *Introduction to computer programming with comal 80 and the commodore 64/128*, COMAL Users Group, Ltd, 1986.

[Lin83] Len Lindsay, *Comal handbook*, Brady (Robert J.) Co ,U.S., 1983.

[LO90] Thomas Lundy and Rory O'Sullivan, *Beginning structured programming in basic and comal*, Gill & Macmillan, 1990.

[Seb15] Robert W. Sebesta, *Concepts of programming languages (11th edition)*, Pearson, 2015.

[Wir73] Niklaus Wirth, *Systematic programming: An introduction*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1973.