Parte xxxi

Basic

«

Basic: introduzione

Il Basic è un linguaggio di programmazione nato solo per scopi didattici, anche se ormai non si può più considerare tanto adatto neanche per questo. La semplicità di questo linguaggio fa sì che si trovino quasi sempre solo interpreti e non compilatori.

Struttura fondamentale

«

Di linguaggi Basic ne esistono tanti tipi, anche con estensioni che vanno molto lontano rispetto all'impostazione originale, facendone in realtà un linguaggio completamente diverso. In questa descrizione, si vuole fare riferimento al Basic tradizionale, con tutte le sue limitazioni antiche. In questo senso, l'interprete Basic per GNU/Linux che più si avvicina al livello essenziale è Bywater BASIC. [1]

Numerazione delle righe

«

La caratteristica tipica di un programma Basic è quella di avere le righe numerate. Infatti, non gestendo procedure e funzioni, l'unico modo per accedere a una subroutine è quello di fare riferimento alla riga in cui questa inizia. In pratica, le istruzioni iniziano con un numero di riga, progressivo, seguito da almeno uno spazio; quindi continuano con l'istruzione vera e propria:

```
110 PRINT "ciao a tutti"
120 PRINT "come va?"
```

Si può intendere che questa dipendenza dalla numerazione delle righe costituisca poi un problema per il programmatore, perché il cam-

biamento della numerazione implica la perdita dei riferimenti alle subroutine.

Istruzioni

Le istruzioni Basic, oltre al fatto di iniziare con il numero di riga, non hanno altre caratteristiche particolari. Generalmente utilizzano una riga e non richiedono la conclusione finale con un qualche simbolo di interpunzione.

È interessante notare invece che i commenti vanno espressi con l'istruzione '**REM**', seguita da qualcosa che poi viene ignorato, e che le righe vuote non sono ammissibili in generale, anche se iniziano regolarmente con il numero di riga.

La natura del linguaggio Basic è tale per cui le istruzioni e i nomi delle variabili dovrebbero essere espressi sempre utilizzando le sole lettere maiuscole.

Esecuzione di un programma

L'esecuzione di un programma Basic dipende dal modo stabilito dall'interprete prescelto. L'interprete tradizionale obbliga a caricare il programma con il comando '**LOAD**' e ad avviarlo attraverso il comando '**RUN**'.

Interprete tradizionale

L'interprete Basic tradizionale è una sorta di shell che riconosce una serie di comandi interni, oltre alle istruzioni Basic vere e proprie. In pratica, attraverso l'invito di questa shell si possono eseguire singole istruzioni Basic, oppure comandi utili a gestire il file di un program-

ma completo. Per esempio, avviando il Bywater BASIC, si ottiene quanto segue:

```
$ bwbasic [Invio]

  bwBASIC:
```

In pratica, '**bwBASIC:**' rappresenta l'invito. L'esempio seguente mostra l'inserimento di alcune istruzioni Basic, allo scopo di eseguire la moltiplicazione 6*7.

```
bwBASIC: A=6 [Invio]

bwBASIC: B=7 [Invio]

bwBASIC: C=A*B [Invio]

bwBASIC: PRINT C [Invio]

  42
```

Comandi tipici dell'interprete

«

L'interprete Basic tipico mette a disposizione alcuni comandi, che risultano essenziali per la gestione di un programma Basic.

Opzione	Descrizione
`LIST` [*riga_iniziale* ↩ ↪ [*-riga_finale*]] [, ...]	Elenca le righe del programma selezionate dagli intervalli indicati come argomento. Se non viene indicato alcun argomento, la visualizzazione viene fatta a partire dalla prima riga; se viene indicata solo la riga iniziale, la visualizzazione riguarda esclusivamente quella riga. L'esempio seguente serve a visualizzare la riga 100 e poi l'intervallo da 150 a 200: '**`LIST 100, 150-200`**'
`RUN` [*riga_iniziale*]	Il comando '**`RUN`**' viene usato normalmente senza argomenti, per avviare il programma caricato nell'interprete. Se si aggiunge il numero di una riga, quel punto viene utilizzato per iniziare l'interpretazione ed esecuzione del programma.
`NEW`	Cancella il programma che eventualmente fosse caricato nell'interprete.
`LOAD` *file*	Carica il programma indicato dal nome del file posto come argomento. Se esiste già un programma in memoria, quello viene eliminato. Solitamente, il nome del file deve essere indicato delimitandolo tra apici doppi. È probabile che l'interprete aggiunga un'estensione predefinita od obbligatoria.

Opzione	Descrizione
`SAVE` ***file***	Salva il programma con il nome specificato come argomento. Solitamente, il nome del file deve essere indicato delimitandolo tra apici doppi. È probabile che l'interprete aggiunga un'estensione predefinita od obbligatoria.
`DEL` ***riga_iniziale*** [−***riga_finale***] [,…]	Elimina le righe indicate dall'argomento. Può trattarsi di una sola riga, o di un intervallo, o di una serie di intervalli.
`RENUM` [***riga_iniziale*** [, ↪ ↪***incremento***]]	Rinumera le righe del programma, aggiornando i riferimenti alle subroutine. È possibile indicare il numero iniziale e anche l'incremento. Di solito, se non viene specificato alcun argomento, la riga iniziale ha il numero 10 e l'incremento è sempre di 10.
`BYE` `QUIT`	Termina il funzionamento dell'interprete Basic.

L'inserimento delle righe di programma attraverso l'interprete Basic, avviene iniziando le istruzioni con il numero di riga in cui queste devono essere collocate. Ciò permette così di inserire righe aggiuntive anche all'interno del programma. Se si utilizzano numeri di righe già esistenti, queste vengono sostituite.

Quando un'istruzione Basic viene inserita senza il numero iniziale, questa viene eseguita immediatamente.

Tipi di dati ed espressioni

I tipi di dati gestibili in Basic sono generalmente solo i numeri reali (numeri a virgola mobile con approssimazione che varia a seconda dell'interprete) e le stringhe.

I numeri vengono indicati senza l'uso di delimitatori; se necessario, è possibile rappresentare valori decimali con l'uso del punto di separazione; inoltre è generalmente ammissibile la notazione esponenziale. L'esempio seguente mostra due modi di rappresentare lo stesso numero.

```
123.456
1.23456E+2
```

Le stringhe si rappresentano delimitandole attraverso apici doppi (possono essere ammessi anche gli apici singoli, ma questo dipende dall'interprete) e sono soggette a un limite di dimensione che dipende dall'interprete (spesso si tratta di soli 255 caratteri).

Le variabili sono distinte in base al fatto che servano a contenere numeri o stringhe. Per la precisione, le variabili che contengono stringhe, hanno un nome che termina con il simbolo dollaro ('**$**'). I nomi delle variabili, a parte l'eventuale aggiunta del dollaro per le stringhe, sono soggetti a regole differenti a seconda dell'interprete; in particolare occorre fare attenzione al fatto che l'interprete potrebbe distinguere tra maiuscole e minuscole. In origine, si poteva utilizzare una sola lettera alfabetica!

L'assegnamento di una variabile avviene attraverso l'operatore '**=**', secondo la sintassi seguente:

```
[LET] variabile=valore
```

L’uso esplicito dell’istruzione ‘**LET**’ è facoltativo.

Espressioni numeriche

«

Gli operatori tipici che intervengono su valori numerici, restituendo valori numerici, sono elencati nella tabella u133.6.

Tabella u133.6. Elenco degli operatori utilizzabili in presenza di valori numerici, all’interno di espressioni numeriche. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
var = ***valore***	Assegna alla variabile il valore alla destra.
− ***op1***	Inverte il segno dell’operando.
op1 + ***op2***	Somma i due operandi.
op1 − ***op2***	Sottrae dal primo il secondo operando.
op1 * ***op2***	Moltiplica i due operandi.
op1 / ***op2***	Divide il primo operando per il secondo.
op1 MOD ***op2***	Modulo: il resto della divisione tra il primo e il secondo operando.
op1 ^ ***op2***	Eleva il primo operando alla potenza del secondo.
SQRT ***op1***	Calcola la radice quadrata dell’operando.
SIN ***op1***	Calcola il seno dell’operando.

Operatore e operandi	Descrizione
`COS` ***op1***	Calcola il coseno dell'operando.
`TAN` ***op1***	Calcola la tangente dell'operando.
`ARCTAN` ***op1***	Calcola l'arcotangente dell'operando.
`LOG` ***op1***	Calcola il logaritmo naturale dell'operando.
`ABS` ***op1***	Calcola il valore assoluto dell'operando.

Le parentesi tonde possono essere utilizzate per indicare esplicitamente l'ordine dell'elaborazione delle espressioni.

Espressioni stringa

L'unico tipo di espressione che restituisce una stringa a partire da stringhe, è il concatenamento che si ottiene con l'operatore '**+**':

stringa_1*+*stringa_2

Espressioni logiche

Le espressioni logiche si possono realizzare a partire da dati numerici, da dati stringa e dal risultato di altre espressioni logiche. La tabella u133.7 mostra gli operatori fondamentali.

Tabella u133.7. Elenco degli operatori utilizzabili nelle espressioni logiche. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
op1 `=` ***op2***	I due numeri, o le due stringhe sono uguali.
op1 `<` ***op2***	Il primo operando è minore del secondo.
op1 `>` ***op2***	Il primo operando è maggiore del secondo.
op1 `<=` ***op2***	Il primo operando è minore o uguale al secondo.
op1 `>=` ***op2***	Il primo operando è maggiore o uguale al secondo.
op1 `<>` ***op2***	I due operandi sono diversi.
cond1 `AND` ***cond2***	Le due condizioni sono entrambe vere.
cond1 `OR` ***cond2***	Almeno una delle due condizioni è vera.
`NOT` ***cond1***	Inverte il risultato logico della condizione.

Espressioni miste

«

Alcuni operatori utilizzano valori di tipo diverso dal tipo di dati che restituiscono. La tabella u133.8 mostra alcuni di questi.

Tabella u133.8. Elenco di altri operatori.

Operatore e operandi	Descrizione
`VAL` ***stringa***	Valuta la stringa trattandola come un'espressione numerica.
`LEN` ***stringa***	Restituisce la lunghezza della stringa in caratteri.
`STR$` ***numero***	Restituisce una stringa contenente il numero indicato come argomento.

Array

Gli array in Basic possono essere a una o più dimensioni, a seconda dell'interprete. In ogni caso, dovrebbero essere distinti in base al contenuto: solo numeri o solo stringhe. L'indice del primo elemento dovrebbe essere zero. La dichiarazione avviene nel modo seguente:

```
DIM nome(dimensione_1 [, dimensione_2] ...)
```

```
DIM nome$(dimensione_1 [, dimensione_2] ...)
```

Nel primo caso si tratta di un array con elementi numerici, nel secondo si tratta di un array con elementi stringa.

Primi esempi banali

L'esempio seguente è il più banale, emette semplicemente la stringa '**`"Ciao Mondo!"`**' attraverso lo standard output:

```
10 print "Ciao Mondo!"
```

Per eseguire il programma basta utilizzare il comando '**RUN**':

RUN [*Invio*]

```
Ciao Mondo!
```

L'esempio seguente genera lo stesso risultato di quello precedente, ma con l'uso di variabili:

```
10 A$ = "Ciao"
20 B$ = "Mondo"
30 PRINT A$; " "; B$
```

L'esempio seguente genera lo stesso risultato di quello precedente, ma con l'uso del concatenamento di stringa:

```
10 A$ = "Ciao"
20 B$ = "Mondo"
30 PRINT A$+" "+B$
```

L'esempio seguente mostra l'uso di una costante e di una variabile numerica:

```
10 A$ = "Ciao"
20 B$ = "Mondo"
30 N = 1000
40 PRINT N; "volte "; A$; " "; B$
```

Il risultato che si ottiene dovrebbe essere il seguente:

```
1000 volte Ciao Mondo!
```

Strutture di controllo del flusso

«

Il Basic è un linguaggio di programmazione molto povero dal punto di vista delle strutture di controllo. In modo particolare sono assenti funzioni e procedure. Per fare riferimenti a porzioni di codice occorre sempre indicare un numero di riga, attraverso le istruzioni '**GOTO**' o '**GOSUB**'.

Istruzione «GOTO»

```
GOTO riga
```

Si tratta dell'istruzione di salto incondizionato e senza ritorno. In pratica, l'esecuzione del programma prosegue dalla riga indicata come argomento, perdendo ogni riferimento al punto di origine.

Istruzione «GOSUB»

```
GOSUB riga
```

Si tratta dell'istruzione di salto incondizionato con ritorno. L'esecuzione del programma prosegue dalla riga indicata come argomento e, quando poi viene incontrata l'istruzione '**RETURN**', il programma riprende dalla riga successiva a quella in cui è avvenuta la chiamata. Questo è l'unico modo offerto dal Basic tradizionale per la realizzazione di subroutine

L'esempio seguente mostra un programma completo che visualizza il messaggio '**"Ciao"**' e poi il messaggio '**"Mondo"**':

```
10 GOTO 50
20 A$ = "Ciao"
30 PRINT A$
40 RETURN
50 GOSUB 20
60 B$ = "Mondo"
70 PRINT B$
```

Istruzione «IF»

«

```
IF condizione THEN istruzione [ELSE istruzione]
```

Se la condizione si verifica, viene eseguita l'istruzione posta dopo la parola chiave '**THEN**', altrimenti, se esiste, quella posta dopo la parola chiave '**ELSE**'. La situazione è tale per cui le istruzioni condizionate sono prevalentemente '**GOTO**' e '**GOSUB**'.

L'esempio seguente emette la stringa '**"Ottimo"**' se la variabile '**N**' contiene un valore superiore a 100; altrimenti esegue la subroutine che inizia a partire dalla riga 50.

```
150 IF N > 100 THEN PRINT "Ottimo" ELSE GOSUB 50
```

Istruzione «FOR»

«

```
FOR variabile_num = inizio TO fine [STEP incremento]
istruzioni
...
NEXT
```

Esegue le istruzioni e ogni volta incrementa la variabile numerica indicata, assegnandole inizialmente il valore posto dopo il simbolo '**=**'. Il blocco di istruzioni viene eseguito fino a quando la variabile raggiunge il valore finale stabilito; l'incremento è unitario, a meno che sia stato indicato diversamente attraverso l'argomento della parola chiave '**STEP**'.

Istruzioni «END» e «STOP»

La conclusione, o l'interruzione del programma può essere indicata esplicitamente utilizzando l'istruzione '**END**' oppure l'istruzione '**STOP**'. La prima corrisponde all'interruzione dovuta a una conclusione normale, la seconda serve a generare un messaggio di errore e si presta per l'interruzione del programma in presenza di situazioni anomale.

Input e output

L'input e l'output del Basic tradizionale è molto povero, riguardando prevalentemente l'acquisizione di dati da tastiera e l'emissione di testo sullo schermo.

Istruzione «PRINT»

```
PRINT operando [{, | ;}…]
```

L'istruzione '**PRINT**' permette di emettere sullo schermo una stringa corrispondente agli operandi utilizzati come argomenti. Eventuali valori numerici vengono convertiti in stringhe automaticamente. Gli operandi possono essere elencati utilizzando la virgola o il punto e virgola. Gli esempi seguenti sono equivalenti:

```
10 PRINT 1234, "saluti"
```

```
10 PRINT 1234; "saluti"
```

```
10 A = 1234
20 PRINT A; "saluti"
```

```
10 A = 1234
20 M$ = "saluti"
30 PRINT A; M$
```

Se come operando si vuole utilizzare il risultato di un'espressione, di qualunque tipo, può essere necessario l'uso di parentesi tonde, come nell'esempio seguente, in cui si vuole emettere il risultato del coseno di zero:

```
10 PRINT ( COS 0 )
```

Istruzione «INPUT» o «?»

«

```
INPUT [invito ;] variabile [, variabile] ...
```

```
? [invito ;] variabile [, variabile] ...
```

Attraverso questa istruzione è possibile inserire un valore in una variabile, o una serie di valori in una serie di variabili. Se viene indicata la stringa dell'invito, questa viene visualizzata prima di attendere l'inserimento da parte dell'utente; altrimenti viene visualizzato semplicemente un punto interrogativo.

Se si indica un elenco di variabili, queste devono essere dello stesso tipo (tutte numeriche o tutte stringa) e il loro inserimento viene atteso in modo sequenziale da parte dell'utente.

L'esempio seguente rappresenta l'inserimento di una stringa senza invito e di una coppia di numeri con invito:

```
10 INPUT A$
20 INPUT "Inserisci la coppia di numeri "; X, Y
```

[1] **Bywater BASIC** GNU GPL

Basic: esempi di programmazione

In questo capitolo si raccolgono solo alcuni esempi molto semplici di programmazione in Basic. Infatti, questo linguaggio di programmazione non si presta per la rappresentazione di algoritmi complessi.

Somma tra due numeri positivi 2542
Moltiplicazione di due numeri positivi attraverso la somma . 2542
Divisione intera tra due numeri positivi 2543
Elevamento a potenza 2543
Radice quadrata 2544
Fattoriale 2544
Ricerca sequenziale 2545

Somma tra due numeri positivi

«

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione 62.3.1.

```
1000 REM ============================================================
1010 REM somma.bas
1020 REM Somma esclusivamente valori positivi.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = X
1080 FOR I = 1 TO Y
1090 LET Z = Z + 1
1100 NEXT
1110 PRINT X; "+"; Y; "="; Z
1120 END
1130 REM ============================================================
```

Moltiplicazione di due numeri positivi attraverso la somma

«

Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione 62.3.2.

```
1000 REM ============================================================
1010 REM moltiplica.bas
1020 REM Moltiplica esclusivamente valori positivi.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = 0
1080 FOR I = 1 TO Y
1090 LET Z = Z + X
1100 NEXT
1110 PRINT X; "*"; Y; "="; Z
1120 END
1130 REM ============================================================
```

Divisione intera tra due numeri positivi

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione 62.3.3.

```
1000 REM ============================================================
1010 REM dividi.bas
1020 REM Divide esclusivamente valori positivi.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = 0
1080 LET I = X
1090 IF I < Y THEN GOTO 1130
1100 LET I = I - Y
1110 LET Z = Z + 1
1120 GOTO 1090
1130 PRINT X; "/"; Y; "="; Z
1140 END
1150 REM ============================================================
```

Elevamento a potenza

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione 62.3.4.

```
1000 REM ============================================================
1010 REM exp.bas
1020 REM Eleva a potenza.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = 1
1080 FOR I = 1 TO Y
1090 LET Z = Z * X
1100 NEXT
1110 PRINT X; "^"; Y; "="; Z
1120 END
1130 REM ============================================================
```

Radice quadrata

«

Il problema della radice quadrata è descritto nella sezione 62.3.5.

```
1000 REM ================================================================
1010 REM radice.bas
1020 REM Radice quadrata intera.
1030 REM ================================================================
1040 REM
1050 INPUT "Inserisci il valore "; X
1060 LET Z = 0
1070 LET T = 0
1080 REM Inizio del ciclo di calcolo
1090 LET T = Z * Z
1100 IF T > X THEN GOTO 1130
1110 LET Z = Z + 1
1120 GOTO 1080
1130 REM Riprende il flusso normale
1140 LET Z = Z - 1
1150 PRINT "radq("; X; ") ="; Z
1160 END
1170 REM ================================================================
```

Fattoriale

«

Il problema del fattoriale è descritto nella sezione 62.3.6.

```
1000 REM ================================================================
1010 REM fatt.bas
1020 REM Fattoriale.
1030 REM ================================================================
1040 REM
1050 INPUT "Inserisci il valore "; X
1060 LET Z = X
1070 FOR I = (X - 1) TO 1 STEP -1
1080 LET Z = Z * I
1090 NEXT
1100 PRINT "fatt("; X; ") ="; Z
1110 END
1120 REM ================================================================
```

Ricerca sequenziale

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione 62.4.1.

```
1000 REM ============================================================
1010 REM ricercaseq.bas
1020 REM Ricerca sequenziale.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il numero di elementi "; N
1060 DIM A(N)
1070 FOR I = 0 TO N-1
1080 PRINT "A("; I; ") ="
1090 INPUT A(I)
1100 NEXT
1110 INPUT "Inserisci il valore da cercare "; X
1120 FOR I = 0 TO N-1
1130 IF X = A(I) THEN GOTO 1170
1140 NEXT
1160 GOTO 1190
1170 PRINT "L'elemento A("; I; ") contiene il valore "; X
1180 END
1190 PRINT "Il valore "; X; " non è stato trovato"
1200 END
1210 REM ============================================================
```