

CAPITOLO 2 – CONCETTI DI BASE SULLA PROGRAMMAZIONE

§2.1 A cosa serve questo capitolo?

Quando si vuole scrivere un'avventura testuale, bisogna prendere in considerazione la storia in sé ma anche il codice vero e proprio. Infatti, la sequenza di testi, domande e risposte che il vostro computer vi mostra a mano a mano che proseguite nel gioco, è il risultato di una sequenza di istruzioni Inform che nel loro insieme costituiscono il LISTATO o CODICE SORGENTE. Per scrivere un listato occorre però apprendere un insieme di regole relative alla programmazione (in questo caso specifiche su Inform, ma il discorso vale per qualsiasi linguaggio di programmazione). Ecco il motivo per cui ho scritto questo capitolo, che vi dà quindi le basi per scrivere poi un'AT vera e propria (argomento che verrà trattato nel terzo capitolo con la creazione della mini-avventura "Ruins").

Occorre tuttavia considerare il fatto che si può cominciare a scrivere un'avventura senza necessariamente sapere tutto quello che viene qui detto (ne è un esempio l'INFORM BEGINNER MANUAL di Roger Firth e Sonja Kesserich); in poche parole, potete saltare direttamente al capitolo 3, ritornando qui solo per apprendere le nozioni che vi servono. A voi la scelta quindi: o prima (dal mio punto di vista il metodo migliore), o dopo (una volta che vi sarete impraticchiti un po' di più con la programmazione delle avventure testuali in Inform).

§2.2 Benvenuto in Inform

Senza perdere tempo in ulteriori preamboli, direi che possiamo cominciare con il nostro primo programma:

```
! Esempio sull'utilizzo della funzione print - versione 1

[ Main temp;
  @erase_window $ffff;           ! pulisce lo schermo
  print "Benvenuto in Inform!";
  @read_char 1 0 0 temp;         ! legge un carattere dalla tastiera
];
```

Da IF-IDE, compilate il listato ed eseguitelo, come spiegato nel capitolo 1: se tutto è andato bene, sul video appare la seguente scritta:

```
Benvenuto in Inform!
```

il computer aspetta poi che l'utente abbia premuto un tasto qualsiasi per uscire e terminare il programma.

Nonostante l'estrema semplicità di quest'ultimo (e non poteva essere altrimenti, visto che è anche il primo) si possono fare su di esso diverse considerazioni:

- In primis, possiamo dire che, tutte le volte che abbiamo un punto esclamativo seguito da un testo, siamo di fronte a una riga di commento, molto utile per descrivere ad esempio cosa fa una determinata istruzione che possa in seguito essere utile a chi dovesse leggere il listato. I commenti, inoltre, vengono completamente ignorati dal compilatore e non hanno peso sul file eseguibile (potete scrivere tutti i commenti che volete, ma la dimensione del file eseguibile rimane sempre la stessa).

- L'istruzione print è, con molta probabilità, la più utilizzata nell'ambito della programmazione in Inform. In questo caso, la sua funzione è quella di stampare a video il testo racchiuso fra le virgolette, come d'altra parte è facilmente intuibile.
- Il punto e virgola (“;”) finale è una regola basilare nella programmazione in questo linguaggio; infatti, salvo casi particolari, TUTTE LE ISTRUZIONI IN INFORM DEVONO TERMINARE CON UN PUNTO E VIRGOLA pena la comparsa di un messaggio di errore durante la fase di compilazione.
- Le due istruzioni che cominciano con il carattere @ sono istruzioni in assembler (cioè istruzioni nel linguaggio stesso dell'interprete Z-Machine) e devono essere apprese così come sono: la prima pulisce lo schermo, la seconda legge invece il carattere corrispondente al tasto premuto dall'utente.
- Per quanto riguarda la formattazione del codice, è possibile scrivere il listato nel seguente modo:

```
! Esempio sull'utilizzo della funzione print - versione 1 [ Main temp;
@erase_window $ffff; ! pulisce lo schermo print "Benvenuto in Inform";
@read_char 1 0 0 temp; ! legge un carattere dalla tastiera ];
```

per il compilatore non cambia nulla, ma riuscite a capirci qualcosa da questa “macedonia”?

È meglio quindi mantenere un certo “ordine” all'interno di un nostro listato, in modo tale che risulti essere più comprensibile per se stessi e per gli altri (soprattutto a distanza di tempo).

§2.3 Le funzioni

In sostanza, possiamo definire una funzione come UNA SERIE DI ISTRUZIONI CONTENUTA ALL'INTERNO DI DUE PARENTESI QUADRE. Rifacendoci ancora una volta all'esempio precedente, in una funzione possiamo distinguere un nome, un argomento (opzionale), un'intestazione, un corpo e una fine:

```
[ Main temp; ←————— INTESTAZIONE DELLA FUNZIONE

  @erase_window $ffff;
  print "Benvenuto in Inform!";
  @read_char 1 0 0 temp; } CORPO DELLA FUNZIONE

]; ←————— FINE DELLA FUNZIONE
```

Il nome di questa funzione è Main, mentre l'argomento è la variabile temp.

All'interno di un listato possono essere definite più funzioni, come avviene in questo secondo esempio:

```
! Esempio sull'utilizzo della funzione print - versione 2

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  print "Benvenuto";
  Stampa2();
  Stampa3();
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Stampa2;
  print " in ";
```

```
];

[ Stampa3;
  print "Inform!";
];
```

il risultato è sempre lo stesso (la stampa a video della scritta Benvenuto in Inform!), ma cambia la modalità di esecuzione con cui questa stampa avviene. In questo esempio il programma, partendo dalla funzione Main, visualizza la parola Benvenuto; segue poi la chiamata alla funzione Stampa2 (che visualizza la parola in con tanto di spazi), il ritorno alla funzione Main, la chiamata alla funzione Stampa3 (che visualizza la parola Inform!), il ritorno alla funzione Main e la lettura del carattere dalla tastiera.

Possiamo quindi affermare che TUTTI I PROGRAMMI IN INFORM CHE NON SONO DELLE AVVENTURE TESTUALI COMINCIANO DALLA FUNZIONE MAIN (un'AT vera e propria comincia dalla funzione initialise, come vedremo nel capitolo 3). Inoltre, l'argomento di una funzione è opzionale (nel senso che può esserci o meno); nel nostro esempio, la funzione MAIN ha un argomento, mentre le funzioni Stampa2 e Stampa3 ne sono prive.

§2.4 Variabili numeriche e caratteri speciali

All'interno di un listato, vengono definiti dei valori che cambiano a seconda del verificarsi o meno di determinate condizioni: una variabile, può essere paragonata a una sorta di contenitore (per esempio una scatola) che contiene tutti questi valori. Ecco un esempio che illustra quanto detto:

```
! Esempio sull'utilizzo delle variabili numeriche

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Var_num();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Var_num numb;
  numb = 0;
  print "La variabile ~numb~ contiene il valore ", numb , ".^";
  numb = 1;
  print "Adesso la variabile ~numb~ contiene il valore ", numb , ".^";
];
```

Nella funzione Var_num viene definita una variabile numb alla quale viene dapprima assegnato, mediante l'operatore =, il valore 0 e successivamente il valore 1.

In Inform, le variabili vengono in genere usate per la gestione dei numeri e dei singoli caratteri. Di default, il range per un valore numerico valido da assegnare a una variabile varia da -32.768 a 32.767. Questo significa che È POSSIBILE ASSEGNARE UN VALORE COMPRESO TRA -32.768 E 32.767, MA NON MINORE DI -32.768 E NON MAGGIORE DI 32.767.

Ricordatevi inoltre, che PRIMA DI ESSERE UTILIZZATA, UNA VARIABILE DEVE SEMPRE AVERE ASSEGNATO UN VALORE INIZIALE (deve cioè essere INIZIALIZZATA) perché, nel momento in cui viene definita, il valore in essa contenuto può essere un numero qualsiasi che spesso e volentieri è al di fuori del range consentito. La conseguenza di tutto questo potrebbe essere davvero drastica, perché per il compilatore non ci sarebbe alcun tipo di errore, ma il valore sarebbe completamente sballato.

Provate come esercizio a sostituire la riga `number = 1;` con `number = 1234567;` per vedere cosa succede.

In Inform esistono anche dei caratteri speciali che vengono stampati in un modo un po' diverso da quello usuale. Nella riga `print "Adesso la variabile ~numb~ contiene il valore ", numb, ".^";` ce ne sono ben due:

- La TILDE (~), che stampa le virgolette a video; si ottiene tenendo premuto il tasto ALT sinistro + i tasti 1, 2 e 6 del tastierino numerico premuti in rapida successione. Per stampare invece la tilde vera e propria occorre scrivere `@@126` (ad esempio `print "@@126";`).
- Il segno di RITORNO A CAPO o di NEWLINE (^); se non si dispone di una tastiera italiana, questo carattere è ottenibile tenendo premuto il tasto ALT sinistro + i tasti 9 e 4 del tastierino numerico premuti in rapida successione. Per stampare invece il carattere ^ vero e proprio occorre scrivere `@@94`.

Ne esistono ovviamente anche degli altri. I più usati sono:

- `@^` che pone l'accento circonflesso sulle lettere a, e, i, o, u, A, E, I, O, U (es.: `print "@^e";`).
- `@c` che pone la cediglia sulle lettere c o C.
- `@:` che pone la diresesi sulle lettere a, e, i, o, u, A, E, I, O, U.
- `@'` (opzionale) che pone l'accento acuto sulle lettere a, e, i, o, u, y, A, E, I, O, U, Y.
- `@`` (opzionale) che pone l'accento grave sulle lettere a, e, i, o, u, A, E, I, O, U.
- `@@92` che stampa a video il carattere \ (in inglese backslash).
- `@@64` per stampare a video il carattere @.
- `@LL` per stampare il segno £.
- la parentesi graffa aperta { che si ottiene tenendo premuto il tasto ALT sinistro + i tasti 1, 2, 3 del tastierino numerico premuti in rapida successione.
- la parentesi graffa chiusa } che si ottiene tenendo premuto il tasto ALT sinistro + i tasti 1, 2, 5 del tastierino numerico premuti in rapida successione.

§2.5 Un po' di matematica non guasta mai

In Inform, quattro sono gli operatori aritmetici fondamentali:

- il segno di moltiplicazione (*) $\longrightarrow a = 10 * 5;$
- il segno di divisione (/) $\longrightarrow a = 10 / 5;$
- il segno di addizione (+) $\longrightarrow a = 10 + 5;$
- il segno di sottrazione (-) $\longrightarrow a = 10 - 5;$

Quando un'operazione aritmetica richiede l'utilizzo di più operatori aritmetici, ci troviamo di fronte a un'espressione $\longrightarrow a = ((10*5) + (10-3)) * 6 + (-9);$

In questo caso bisogna fare molta attenzione all'ordine in cui gli operatori lavorano. Infatti, se scriviamo `4 + 2 * 8` e `4 * 2 + 8`, avremo come risultato 20 nel primo caso e 16 nel secondo (vengono cioè eseguite, in ordine di priorità, prima le moltiplicazioni e poi le addizioni).

Possiamo quindi affermare che:

1. GLI OPERATORI ARITMETICI LAVORANO IN ORDINE DI PRIORITÀ: IN UN'ESPRESSIONE ARITMETICA VIENE ESEGUITA PER PRIMA LA MOLTIPLICAZIONE E, A SEGUIRE, LA DIVISIONE, L'ADDIZIONE E LA SOTTRAZIONE. NE CONSEGUO CHE:

2. IN UN'ESPRESSIONE ARITMETICA LE SINGOLE OPERAZIONI DEVONO ESSERE SEMPRE RACCHIUSE DA DELLE PARENTESI TONDE.
3. AD OGNI PARENTESI TONDA APERTA, NE DEVE SEMPRE CORRISPONDERE UNA TONDA CHIUSA.

Se vogliamo quindi sommare $4 + 2$ e moltiplicare poi il risultato ottenuto per 8, avremo: $(4 + 2) * 8$

In Inform è anche possibile usare l'operatore % per ottenere il resto di una divisione. Quindi:
 $15/5$ darà come risultato 3, mentre $15\%5$ darà come risultato 0;
 $17/5$ darà come risultato 3, mentre $17\%5$ darà come risultato 2.
 Non è invece possibile dividere per 0 come d'altra parte la stessa matematica ci insegna.

È anche possibile operare sui valori contenuti all'interno delle variabili. Ecco dei possibili esempi:

$a = b + 20;$ $b = (c * 60) + (b / 30);$ $d = (b + 7) * (b - 8);$

Spesso poi, il contenuto di una variabile può essere incrementato o decrementato di un certo valore, come avviene nei seguenti esempi:

$a = a + 2;$ $b = b + 5;$ $b = b - a$ $b = b - (a + 3);$

Quindi, supponendo che a abbia il valore 12 e che b abbia il valore 6, avremo:

$a = a + 2$	$a = 12 + 2$	$a = 14$
$b = b + 5$	$b = 6 + 5$	$b = 11$
$b = b - a$	$b = 6 - 12$	$b = -6$
$b = b - (a + 3)$	$b = 6 - (12 + 3)$	$b = -9$

Se poi vogliamo incrementare o decrementare il valore contenuto in una variabile di 1, esistono delle forme abbreviate: $a++$ al posto di $a = a + 1$ e $a--$ al posto di $a = a - 1$.

§2.6 Passaggio dei parametri e ritorno dei valori

Ora che sappiamo cosa sono le variabili, possiamo esaminare il seguente esempio:

! Esempio sul passaggio dei parametri - versione 1

```
[ Main temp x n;
  x = 5;
  @erase_window $ffff;      ! pulisce lo schermo
  n = Quadrato(x); print "Il quadrato di ", x, " è ", n, ".";
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Quadrato number;
  return number*number;
];
```

Il primo cambiamento possiamo notarlo subito all'inizio della funzione Main: gli argomenti questa volta sono tre (temp, x e n). Nulla di strano, perché UNA QUALSIASI FUNZIONE IN INFORM PUÒ AVERE UN MASSIMO DI 15 ARGOMENTI, SEPARATI TRA DI LORO DA UNO SPAZIO.

Quando poi viene chiamata la funzione Quadrato viene anche passato il valore tra parentesi (detto parametro) contenuto all'interno della variabile x. Ecco in sostanza quello che accade: alla riga `n = Quadrato(x)`; il programma salta alla funzione Quadrato e il valore contenuto nella variabile x (in questo caso 5) viene assegnato alla variabile number (definita nella funzione Quadrato). Viene poi eseguita l'elevazione al quadrato e il valore ottenuto (25) viene restituito, mediante l'istruzione `return`, e assegnato alla variabile n della funzione Main.

Se qualcuno di voi ha le idee confuse, si concentri su quello che sto per dire adesso: UNA VARIABILE PUÒ ESSERE LOCALE (VIENE CIOÈ VISTA SOLO ALL'INTERNO DELLA FUNZIONE IN CUI È DEFINITA) O GLOBALE (VIENE CIOÈ VISTA IN QUALSIASI PUNTO DEL PROGRAMMA). Provate a riscrivere la funzione Quadrato in questo modo:

```
[ Quadrato number;  
    return x*x;  
];
```

e compilate nuovamente il programma: il file eseguibile non viene creato, perché sono presenti ben due errori, nell'ordine:

1. "No such constant as x" che, tradotto in italiano, significa "Non c'è nessuna variabile denominata x". Questo accade perché la variabile locale x è stata definita nella funzione Main ed appartiene a essa soltanto. Ecco quindi il motivo per cui ho definito una nuova variabile number all'interno della funzione quadrato; se non lo avessi fatto, il valore 5 sarebbe andato irrimediabilmente perduto nel momento in cui il programma sarebbe passato dalla funzione Main alla funzione Quadrato.

Occorre anche notare che se definisco la funzione Quadrato nel seguente modo:

```
[ Quadrato x;  
    return x*x;  
];
```

il programma funziona perfettamente, ed è normale che sia così. Infatti, la variabile x definita all'interno della funzione Quadrato è diversa dalla variabile x definita all'interno della funzione Main.

2. "Local variable number declared but not used" che, tradotto in italiano, significa "La variabile locale number è stata dichiarata ma non usata". Ecco dunque un'altra regola sacrosanta di questo linguaggio: TUTTE LE VARIABILI DICHIARATE DEVONO ESSERE USATE. Nella funzione Quadrato viene definita number, ma al suo posto viene usata la x: ecco il motivo dell'errore.

Qualcuno di voi potrebbe però chiedersi: perché, al posto di usare più variabili locali non ne uso una sola globale? Perché di funzioni in un'AT ce ne sono moltissime e usare solo variabili globali diventa veramente un azzardo. Si genera troppa confusione e aumenta il rischio che Inform scambi un valore per un altro. Comunque, a scopo puramente didattico, ecco come usare una variabile globale con l'esempio di prima:

! Esempio sul passaggio dei parametri - versione 2

```
Global x;
```

```
[ Main temp n;  
  x = 5;  
  @erase_window $ffff;      ! pulisce lo schermo  
  n = Quadrato(); print "Il quadrato di ", x , " è ", n , ".^";  
  print "^Premi un tasto per uscire";
```

```

    @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Quadrato;
  return x*x;
];

```

Come vedete, la variabile `x`, essendo stata dichiarata al di fuori di una funzione, può essere vista e riconosciuta in qualsiasi punto del programma.

§2.7 L'input dei dati numerici: la funzione Getnumber

Riesaminando per un attimo l'esempio della funzione Quadrato:

```

[ Main temp x n;
  x = 5;
  @erase_window $ffff;
  n = Quadrato(x); print "Il quadrato di ", x , " è ", n , "."^";
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;
];

[ Quadrato number;
return number*number;
];

```

notiamo che, se vogliamo elevare al quadrato un numero diverso da 5, dobbiamo modificare il valore di `x` (ad esempio `x = 10`;) e poi ricompilare ed eseguire il tutto. Decisamente scomodo, non vi pare?

Inform stesso (che pur essendo un linguaggio di programmazione strepitoso è ben lungi dall'essere un prodotto perfetto), non ci viene minimamente incontro in un caso come questo, perché non esiste un'istruzione che chieda all'utente di inserire un numero dalla tastiera.

Fortunatamente però, è possibile includere in un programma delle nuove istruzioni che possono risolvere qualsiasi tipo di problema (rimanendo ovviamente relegati nell'ambito delle avventure testuali). Vediamo ora come possiamo scrivere un esempio che possa fare al caso nostro:

```

! Esempio sul passaggio dei parametri - versione 3

include "GetNumber.h";

[ Main temp x n;
  @erase_window $ffff;    ! pulisce lo schermo
  print "^Inserire il numero da elevare al quadrato: ";
  x = GetNumber(2);
  n = Quadrato(x); print "Il quadrato di ", x , " è ", n , "."^";
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Quadrato number;
  return number*number;
];

```

L'istruzione include informa il compilatore che il codice da compilare è diviso in due parti: il programma vero e proprio e il file GetNumber.h, contenente la funzione Getnumber (che acquisisce in input il numero digitato dall'utente). Il file deve trovarsi nella directory C:\inform\libraries e potete reperirlo nella directory inform/libraries inclusa a sua volta nel file Capitolo2.zip (quello che avete scaricato dal mio sito per poter leggere questo capitolo, tanto per intenderci).

GetNumber ha bisogno di sapere da quante cifre è composto il numero che l'utente deve digitare (1 o 2), e il valore che ritorna (il numero stesso) deve essere contenuto in una variabile (altrimenti viene perso, ricordate?). Non sono inoltre ammessi i numeri negativi e con la virgola.

Qualcuno in effetti può avere qualcosa da ridire su una funzione che è in grado di elaborare dei numeri con sole uno o due cifre (da un minimo di 0 fino a un massimo di 99), ma nell'ambito delle avventure testuali questo è più che sufficiente, e va quindi bene così.

§2.8 If (condition) then... else...

Nella stesura di un programma, le scelte sono quasi sempre d'obbligo, proprio come avviene in questo esempio:

```
! Esempio sull'utilizzo dell'istruzione IF THEN - versione 1

include "GetNumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x;
  print "^Ogni quanti anni compare la cometa di Halley? ";
  x = GetNumber(2);
  if (x == 75) print "La risposta è esatta!!!";
];
```

Tradotta in italiano, l'istruzione if (condition) then suona più o meno così: se (condizione) allora. In questo caso, se il numero introdotto dall'utente è proprio 75, allora il programma stampa a video La risposta è esatta!!!, altrimenti prosegue senza visualizzare nulla.

Fate molta attenzione a non confondere l'operatore = con l'operatore ==; il primo, come già sapete da quanto visto finora, assegna un certo valore a una variabile, mentre il secondo verifica che un certo valore (in questo esempio 75) sia UGUALE a un altro (in questo caso il valore contenuto nella variabile x, ovvero il numero digitato dall'utente mediante la funzione Getnumber).

Gli altri operatori sono:

(a ~= b) ➡ a è diverso da b
(a > b) ➡ a è maggiore di b
(a < b) ➡ a è minore di b
(a >= b) ➡ a è maggiore o uguale a b
(a <= b) ➡ a è minore o uguale a b

È anche possibile usarli insieme, ma in questo caso bisogna utilizzare anche gli OPERATORI LOGICI, nell'ordine:

- && detto anche "AND" → es: if ((a > 5) && (b < 10)) → restituisce true (la condizione è cioè soddisfatta) se il valore contenuto nella variabile a è maggiore di 5 E se il valore contenuto nella variabile b è minore di 10.
- || detto anche "OR" → es: if ((a > 5) || (b < 10)) → restituisce true se il valore contenuto nella variabile a è maggiore di 5 OPPURE se il valore contenuto nella variabile b è minore di 10. Questo operatore non deve essere confuso con l'operatore (non logico) or che viene usato per testare più valori in una volta sola (ad es.: if (a == 2 or 3 or 4) print...).
- ~~ detto anche "NOT" → es: if (~ (a > 5)) → restituisce true se il valore contenuto nella variabile a è minore di 5 (la condizione viene cioè NEGATA e di conseguenza invertita).

Così com'è, il nostro esempio stampa un messaggio nel caso in cui l'utente risponda correttamente alla domanda. E se invece sbaglia?

Non è un'idea così malvagia, quella di modificare il nostro programma in modo tale che stampi un messaggio anche nel caso di una risposta errata da parte dell'utente. Ecco una possibile soluzione:

```
! Esempio sull'utilizzo dell'istruzione IF THEN - versione 2

include "GetNumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x;
  print "^Ogni quanti anni compare la cometa di Halley? ";
  x = GetNumber(2);
  if (x == 75) print "La risposta è esatta!!!";
  else print "Spiacente, la risposta non è esatta...";
];
```

di nuovo c'è solo l'istruzione else, che in italiano significa altrimenti e viene usata solo insieme all'istruzione if-then. Ecco quello che accade: se il numero digitato dall'utente è uguale a 75 viene stampato il messaggio La risposta è esatta!!!, mentre se è diverso da 75 viene stampato il messaggio Spiacente, la risposta non è esatta...

§2.9 While e do...until

In Inform, così come in qualsiasi altro linguaggio di programmazione, oltre alle scelte esistono anche i cicli (o loop). Il primo di questi è while (condition) e il suo funzionamento è illustrato nel seguente esempio:

```
! Esempio sull'utilizzo del ciclo WHILE

include "GetNumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
```

```

    @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x;
  while (x ~= 75)
  {
    print "^Ogni quanti anni compare la cometa di Halley? ";
    x = GetNumber(2);
    if (x == 75) print "La risposta è esatta!!!";
    else print "Spiacente, la risposta non è esatta...";
  }
];

```

Nella funzione Domanda, viene definito il ciclo while (condizione), la cui funzione è quella di eseguire una serie di istruzioni, racchiusa da dalle parentesi graffe, fino a quando il valore contenuto nella variabile x è diverso da 75. In questo modo, il ciclo termina solo se x è uguale a 75 (se cioè la risposta esatta), altrimenti va avanti all'infinito (richiede tutte le volte la stessa domanda).

Il ciclo do...until (esegui... fino a quando vale la condizione) è simile al ciclo while ma, a differenza di quest'ultimo, esegue le istruzioni in esso contenute almeno una volta:

! Esempio sull'utilizzo del ciclo DO...UNTIL

```

include "GetNumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x;
  do
  {
    print "^Ogni quanti anni compare la cometa di Halley? ";
    x = GetNumber(2);
    if (x == 75) print "La risposta è esatta!!!";
    else print "Spiacente, la risposta non è esatta...";
  }
  until (x == 75);
];

```

Notate però, che nell'istruzione until la condizione è esattamente opposta a quella del ciclo while; infatti, mentre in quest'ultimo il ciclo va avanti fino a quando il valore contenuto nella variabile x è diverso da 75, in do-until le istruzioni racchiuse nelle parentesi graffe vengono eseguite fino a quando il contenuto della variabile x è uguale 75 (fino a quando, cioè, l'utente risponde correttamente alla domanda). Il risultato è lo stesso, ma la strada intrapresa è esattamente quella opposta.

§2.10 For, break e continue

Un altro ciclo da esaminare è il for. Ecco un esempio che ne illustra il funzionamento:

```
! Esempio sull'utilizzo del ciclo FOR - versione 1

include "GetNumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x count;
  for (count=0: count<5: count++)
  {
    print "^", count , "^";
    print "^Ogni quanti anni compare la cometa di Halley? ";
    x = GetNumber(2);
    if (x == 75)
    {
      print "La risposta è esatta!!!";
      count = 5;
    }
    else print "Spiacente, la risposta non è esatta...^";
  }
];
```

Questo ciclo è suddiviso in tre parti; la prima (count = 0) assegna il valore 0 alla variabile count; la seconda (count < 5) fa in modo che il ciclo va avanti fino a quando la variabile count contiene un valore minore di 5; la terza (count++) incrementa di 1 il valore di count fino a quando non arriva ad essere uguale a 5. A quel punto il programma termina.

Quando l'utente risponde correttamente alla domanda prima che i cinque tentativi a disposizione siano terminati, alla variabile count viene assegnato il valore 5, che fa così terminare il ciclo e di conseguenza anche il programma. Senza questo piccolo stratagemma, se l'utente indovina la risposta al secondo tentativo, il ciclo va avanti ancora tre volte (e viene quindi richiesta la domanda per altre tre volte).

Anche il comando if ha adesso delle istruzioni racchiuse da delle parentesi graffe; questo accade perché le istruzioni che gli appartengono sono più di una, al contrario di quello che avviene per l'istruzione else.

Il comando break si usa per fermare un ciclo, qualunque sia la sua condizione:

```
! Esempio sull'utilizzo del ciclo FOR - versione 2

include "GetNumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
```

```

];

[ Domanda x count;
  for (count=0: count<5: count++)
  {
    print "^", count , "^";
    print "^Ogni quanti anni compare la cometa di Halley? ";
    x = GetNumber(2);
    if (x == 75)
    {
      print "La risposta è esatta!!!";
      break;
    }
    else print "Spiacente, la risposta non è esatta...^";
  }
];

```

il risultato è esattamente identico a quello di prima, solo questa volta abbiamo usato l'istruzione `break` al posto di `count=5`.

Ecco un altro esempio che utilizza invece l'istruzione `continue`:

! Esempio che utilizza l'istruzione `continue`

```

[ Main temp k;
  @erase_window $ffff;      ! pulisce lo schermo
  for (k=1: k<=10: k++)
  {
    if (k == 5)
    {
      k = 8;
      continue;
    }
    print k, " ";
  }
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

```

quando `k` è uguale a 5 gli viene assegnato il valore 8; il ciclo continua, saltando però tutte le istruzioni che si trovano dopo la condizione `if` (in questo caso `print k, " "`) per `k == 5`. Il risultato in output (quello che appare sul video) è: 1 2 3 4 9 10.

§2.11 L'utilizzo delle costanti

Possiamo raffinare ulteriormente il nostro esempio sull'istruzione `for`, utilizzando una costante per identificare il numero di tentativi massimi possibili:

! Esempio sull'utilizzo del ciclo `FOR` - versione 3

```

Constant TENTATIVI 5;

include "GetNumber.h";

[ Main temp;

```

```

@erase_window $ffff;      ! pulisce lo schermo
Domanda();
print "^Premi un tasto per uscire";
@read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x count;
  for (count=0: count<TENTATIVI: count++)
  {
    print "^Ogni quanti anni compare la cometa di Halley? ";
    x = GetNumber(2);
    if (x == 75)
    {
      print "La risposta è esatta!!!";
      break;
    }
    else print "Spiacente, la risposta non è esatta...";
  }
];

```

in questo modo, se si vuole aumentare o diminuire il numero dei tentativi, basta andare a modificare il valore della costante, ad esempio:

```
Constant TENTATIVI 7;
```

Forse, alcuni di voi non riescono ancora a comprendere il vantaggio di utilizzare una costante in un listato così semplice. Immaginate allora cosa, accade in un listato più complesso se devo andare a cambiare i valori di tutte le condizioni di controllo dei cicli. Una cosa davvero noiosa, soprattutto quando ci sono più cicli che terminano dopo lo stesso numero di tentativi. DAL MOMENTO CHE UNA COSTANTE È DICHIARATA AL DI FUORI DI UNA QUALSIASI FUNZIONE O ALTRA STRUTTURA, È GLOBALE, E QUINDI VISIBILE IN TUTTO IL PROGRAMMA: con un solo cambiamento (quello del valore della costante stessa), metto a posto tutte le funzioni e tutti i cicli che si rifanno a quel valore. Insomma, è proprio il caso di dire che con un piccione prendo più di una fava...

Attenzione però a non confondere una costante con una variabile: UNA COSTANTE È UN VALORE FISSO, CHE NON MUTA DURANTE L'ESECUZIONE DEL PROGRAMMA, MENTRE UNA VARIABILE SÌ.

§2.12 Switch... case

Questa istruzione è una scelta multipla. Ecco un esempio che ne illustra il funzionamento:

```

! Esempio sull'utilizzo del ciclo SWITCH CASE

include "Getnumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x;
  do
  {

```

```

print "^Scrivi ogni quanti anni compare la cometa di Halley:";
print "^^50  60  70  75^^";
print "? "; x = Getnumber(2);
switch(x)
{
  50, 60: print "Assolutamente no!!!^";
  70: print "No, ma ci sei quasi...^";
  75: print "Bravo, la risposta è esatta...^";
  default: print "Sii serio. Il valore da te scritto non è tra
              quelli elencati.^";
}
}
until (x == 75);
];

```

Abbiamo la nostra solita domanda sulla cometa di Halley, ma questa volta c'è una novità: ora il computer propone quattro possibili risposte, e l'utente deve indovinare quella giusta. A differenza degli esempi precedenti però, ora viene stampato un messaggio per ogni risposta data, compreso un valore che non rientra nei quattro suggeriti.

Sottolineo ancora una volta l'importanza della formattazione del codice perché, rispettando un certo ordine di spazi e di tabulazioni, è possibile riuscire a capire quali siano le istruzioni che appartengano a un ciclo piuttosto che a un altro. Come potete vedere, dopo la riga switch(x) segue un elenco dei valori con, per ognuno, la stampa del relativo messaggio. La condizione default: è quella che permette al computer di capire se l'utente ha digitato un valore suggerito o no; essa dice al compilatore: PER TUTTO QUELLO CHE È DIVERSO DAI VALORI ELENCATI, SCRIVI "Sii serio. Il valore da te scritto non è tra quelli elencati.^";". Capito dove sta il trucco?

Notate che lo stesso risultato si può ottenere anche con l'istruzione if-then:

```

[ Domanda x;
do
{
  print "^Scrivi ogni quanti anni compare la cometa di Halley:";
  print "^^50  60  70  75^^";
  print "? "; x = Getnumber(2);
  if (x == 50 or 60) print "Assolutamente no!!!^";
  else
  {
    if ( x == 70) print "No, ma ci sei quasi...^";
    else
    {
      if (x == 75) print "Bravo, la risposta è esatta...^";
      else
      {
        print "Sii serio. Il valore da te scritto non è tra quelli
              elencati.^";
      }
    }
  }
}
until (x == 75);
];

```

tutto funziona alla perfezione (provare per credere), ma è proprio il caso di dire che con l'istruzione switch-case le cose si semplificano di molto. Nell'ambito della programmazione, più diventerete

bravi e più vi renderete conto che esistono diverse strade per arrivare alla risoluzione di un problema; alcune sono migliori, altre no. Sarete voi a deciderlo.

§2.13 Salti ed etichette

In Inform, esiste anche l'istruzione `jump` (in italiano salto) in grado di controllare l'esecuzione del programma, come accade nel seguente esempio:

```
! Esempio sull'utilizzo dell'istruzione JUMP

Constant TENTATIVI 5;

include "GetNumber.h";

[ Main temp;
  @erase_window $ffff;      ! pulisce lo schermo
  Domanda();
  print "^Premi un tasto per uscire";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];

[ Domanda x count;
  count = 0;

  .Inizio;
  print "^Ogni quanti anni compare la cometa di Halley? ";
  x = GetNumber(2);
  if (x == 75)
  {
    print "La risposta è esatta!!!^";
    count = TENTATIVI;
  }
  else print "Spiacente, la risposta non è esatta...^";
  count++; if (count < TENTATIVI) jump Inizio;
];
```

l'istruzione `.Inizio;` è un'etichetta (in inglese label) alla quale l'istruzione `jump` fa sempre riferimento. Alla fine della funzione `Domanda`, l'istruzione `if (count < TENTATIVI) jump Inizio;` dice al compilatore: SE IL VALORE CONTENUTO NELLA VARIABILE `COUNT` È MINORE DEL VALORE DELLA COSTANTE `TENTATIVI`, VAI ALL'ETICHETTA `INIZIO`, ALTRIMENTI PROSEGUI.

In questo modo abbiamo simulato un ciclo `do-until`, ovviamente a scopo puramente didattico e dimostrativo; non ha alcun senso, infatti, simulare qualcosa che esiste già. Questo stratagemma veniva utilizzato moltissimo con alcuni vecchi Basic degli anni 80, che di fatto non avevano i cicli `while` e `do...until` e dovevano in qualche modo “arrangiarsi”.

È bene comunque sapere che Inform mette a disposizione anche questa possibilità, perché, in alcune funzioni particolarmente complesse, potrebbe rivelarsi una valida scappatoia.

§2.14 Caratteri, stringhe e vettori

Se siete riusciti ad arrivare vivi fino a qui, devo farvi davvero i miei complimenti. Adesso però, prima di affrontare l'ultimo argomento di questo capitolo, vi consiglio vivamente di bere un po' d'acqua, rilassarvi completamente e magari uscire per un paio d'ore con la vostra moglie o fidanzata (e se siete dei single con qualche amica: non si sa mai, potrebbe nascere una nuova relazione amorosa).

Vi siete rilassati? Ok. Fino ad ora abbiamo visto come gestire i dati numerici (10, 13, 56 e così via), ma esistono anche i dati alfabetici (le sole lettere, come ad esempio a, b, c, casa, papà, ecc.) e alfanumerici (le lettere + i numeri, come ad esempio alpha202, omega3, ecc.).

In Inform una variabile, oltre ai valori numerici, può contenere anche dei caratteri e perfino delle stringhe (sequenze di numeri e lettere, come ad es. "Ogni mattina mi sveglio alle 6:00"):

! Esempio sull'utilizzo di variabili con dati non numerici - versione 1

```
[ Main temp carattere parola stringa;
  @erase_window $ffff;      ! pulisce lo schermo
  carattere = "a"; parola = "terrazzo"; stringa = "Questa è casa mia";
  print (string) carattere, "^" , (string) parola, "^" , (string)
    stringa, "^";
  @read_char 1 0 0 temp;    ! legge un carattere dalla tastiera
];
```

come potete vedere, il computer visualizza a, terrazzo, Questa è casa mia, esattamente quello che ci si aspetta. Questa volta però entra in gioco la regola di stampa (string) che dice all'istruzione print di stampare a video le stringhe contenute rispettivamente nelle variabili parola, carattere, e stringa. Altra cosa da notare, almeno per quelli un po' più esperti nella programmazione, è che IN INFORM LE VARIABILI NON SI DIFFERENZIANO PER TIPO.

Osserviamo ora quest'altro esempio:

! Esempio sull'utilizzo di variabili con dati non numerici - versione 2

```
include "Presskey.h";
```

```
[ Main carattere parola stringa car;
  @erase_window $ffff;      ! pulisce lo schermo
  carattere = "a"; parola = "terrazzo"; stringa = "Questa è casa mia";
  print (string) carattere, "^" , (string) parola, "^" , (string)
    stringa, "^";
  print "^Premi il tasto q per uscire^";
  do
  {
    car = Presskey();
  }
  until (car == 'q');
];
```

la funzione Presskey legge il carattere del tasto premuto dall'utente e ne restituisce il valore alla variabile car. Se il tasto premuto corrisponde alla lettera q, il programma termina, altrimenti richiede la pressione di un altro tasto.

Potete testare tutte le lettere che volete, ma se si volessero usare dei tasti "particolari" come ad esempio la barra spaziatrice o ESC?

In questo caso bisogna ricorrere all'utilizzo dei codici ZSCII (attenzione, non ASCII), perché ad ogni simbolo rappresentato sulla tastiera, corrisponde un determinato codice numerico indicante il carattere, e varia a seconda del sistema operativo e dei programmi che si stanno usando. Sul mio computer (un Pentium 2 a 350Mhz equipaggiato con Windows 98), ecco a quali codici corrispondono i seguenti tasti:

- 8 per il tasto DELETE
- 13 per il tasto INVIO
- 27 per il tasto ESC
- 32 per la BARRA SPAZIATRICE

volendo quindi terminare il programma premendo la barra spaziatrice, occorre modificare il ciclo do-until nel seguente modo:

```
print "^Premi spazio per uscire^";
do
{
    car = Presskey();
}
until (car == 32);
];
```

se usate Linux o il Mac-OS, per sapere a quali codici corrispondono i vostri tasti, potete inserire una `print car;` sotto la riga `car = Presskey();`. Provate a premere un tasto che sia diverso dalla barra spaziatrice, e vedete quale numero appare sul video: quello è il codice effettivo da usare nel test. Questo è tutto quello che ho da dire sui caratteri. E le stringhe? Osserviamo ora l'esempio che segue:

```
! Esempio sull'utilizzo di variabili con dati non numerici - versione 3

include "Presskey.h";

[ Main stringa1 stringa2 car;
  @erase_window $ffff;      ! pulisce lo schermo
  stringa1 = "Ciao Andrea!"; stringa2 = "Ciao Andrea!";
  print (string) stringa1 , " ", (string) stringa2, "^";
  if (stringa1 == stringa2) print "Le due stringhe sono uguali.^";
  print "^Premi il tasto q per uscire^";
  do
  {
      car = Presskey();
  }
  until (car == 'q');
];
```

le stringhe sono uguali, ma la condizione dell'istruzione `if` viene ignorata. A questo punto, le sole variabili non bastano più, e bisogna passare ai vettori (o in inglese array).

UN VETTORE PUÒ ESSERE CONSIDERATO COME UN INSIEME DI VARIABILI INDICIZZATE e in Inform viene definito in questo modo:

```
Array nome_array->lunghezza_max_array;
```

Scrivendo ad esempio Array numeri->20; abbiamo a disposizione 20 celle (o variabili) alle quali è possibile accedere tramite un indice:

```
! Esempio sull'utilizzo dei vettori - versione 1

include "Presskey.h";

Constant LUNGH_MAX 10;
Array numeri->LUNGH_MAX;

[ Main car;
  LoadArray(numeri);
  PrintNumericArray(numeri);

  print "^Premi il tasto q per uscire^";
  do
  {
    car = Presskey();
  }
  until (car == 'q');
];

[ LoadArray the_array i;
  for (i=1: i<=LUNGH_MAX: i++) the_array->(i - 1) = i;
];

[ PrintNumericArray the_array i;
  for (i=0: i<LUNGH_MAX: i++) print the_array->i, " ";
];
```

All'inizio, il vettore numeri si presenta così:

0	1	2	3	4	5	6	7	8	9
?	?	?	?	?	?	?	?	?	?

abbiamo 10 celle che devono ancora essere inizializzate (non sappiamo quindi cosa contengono, da qui il ?). I numeri che vanno da 0 a 9 sono gli indici delle celle (uno per ogni cella). Dopo la chiamata alla funzione LoadArray ecco come si ripresenta il nostro vettore:

i	0	1	2	3	4	5	6	7	8	9
the_array->i	1	2	3	4	5	6	7	8	9	10

Nel ciclo for di questa funzione, alla variabile i viene assegnato l'indice e l'istruzione:

```
the_array->(i - 1) = i;
```

fa riferimento alla cella corrispondente dell'indice (i - 1), e le assegna il valore della variabile i. Ricordatevi sempre che IN INFORM, L'INDICE DELLA PRIMA CELLA DI UN VETTORE NON È 1, MA 0. Ecco perché ho messo (i - 1) al posto di i: se non lo avessi fatto, il contenuto della decima cella sarebbe stato 9 anziché 10.

Ecco adesso, un esempio che usa un vettore con le stringhe:

```
! Esempio sull'utilizzo dei vettori - versione 2

include "Presskey.h";

Constant LUNGH_MAX 33;
Array nome->LUNGH_MAX;

[ Main car;
  ("Massimo è andato a casa").print_to_array(nome);
  PrintStringArray(nome);

  print "^Premi il tasto q per uscire^";
  do
  {
    car = Presskey();
  }
  until (car == 'q');
];

[ PrintStringArray the_array i;
  for (i=2: i<(the_array->1)+2: i++) print (char) the_array->i;
];
```

abbiamo ora un vettore nome con una lunghezza massima di 33 celle, che dopo l'istruzione:

```
("Massimo è andato a casa").print_to_array(nome);
```

si presenta così:

0	1	2	3	4	5	6	7	8	9	10	...
0	23	M	a	s	s	i	m	o		è	...

La cella numero 2 (nome->1) contiene il numero 23 che corrisponde alla lunghezza della stringa introdotta dall'utente. La stringa vera e propria comincia quindi dalla cella successiva (nome->2).

RICORDATEVI SEMPRE DI NON SUPERARE MAI LA LUNGHEZZA MASSIMA DEL VETTORE E DI AGGIUNGERE 3 NUMERI ALLA LUNGHEZZA MASSIMA DELLA VOSTRA STRINGA.

In poche parole, in questo esempio non potete inserire una stringa lunga più di 30 caratteri, e la lunghezza massima del vettore che la deve contenere deve essere 33 anziché 30. Piuttosto complicato, non è vero?

Vediamo ora come fare in modo che sia l'utente a inserire una stringa:

```
! Esempio sull'utilizzo dei vettori - versione 3

include "Array.h";
include "Presskey.h";

Constant LUNGH_MAX 18;
Array nome_player->LUNGH_MAX;
Array nome_computer->LUNGH_MAX;

[ Main car;
  print "^Come ti chiami? "; ReadArray(nome_player, LUNGH_MAX);
  ("massimo").print_to_array(nome_computer);
```

```

print "Ciao ", (PrintStringArray) nome_player, ", ";
if((CmpStr(nome_player, nome_computer))==1) print "abbiamo lo stesso
    nome.^^";
else print "io sono ", (PrintStringArray)nome_computer, ".^^";

print "Premi il tasto q per uscire";
do
{
    car = Presskey();
}
until (car == 'q');
];

```

La funzione ReadArray definita nel file Array.h (che contiene a sua volta anche le funzioni CmpStr e PrintStringArray), è quella che permette l'inserimento in input di una stringa da parte dell'utente. ReadArray(nome_player, LUNGH_MAX); chiama quindi questa funzione passandole, come parametri, il nome dell'array che deve contenere la stringa dell'utente e la sua lunghezza massima. La funzione CmpStr confronta due stringhe e restituisce il valore 1 se sono uguali, 0 se invece non lo sono. CmpStr(nome_player, nome_computer) chiama quindi questa funzione passandole, come parametri, i nomi degli array contenenti le due stringhe da confrontare. Occorre notare ancora una cosa: perché al vettore nome_computer viene passata la stringa “massimo” anziché “Massimo”? Dovete sapere che la funzione ReadArray da me scritta, (o meglio, l'istruzione Inform read usata dalla funzione ReadArray), trasforma automaticamente tutte le lettere maiuscole in minuscole. Se l'utente digita quindi Mario, il vettore nome_player conterrà mario anziché Mario. Di per sé il problema non sarebbe tanto grave, se non fosse per il fatto che due righe più sotto il nome dell'utente viene stampato insieme a quello del computer. Ecco allora come risolvere il problema:

! Esempio sull'utilizzo dei vettori - versione 4

```

include "Array.h";
include "Presskey.h";

Constant LUNGH_MAX 18;
Array nome_player->LUNGH_MAX;
Array nome_computer->LUNGH_MAX;

[ Main car;
    print "^Come ti chiami? "; ReadArray(nome_player, LUNGH_MAX);
    ("Massimo").print_to_array(nome_computer);

    nome_player->2 = (nome_player->2)-32;

    print "Ciao ", (PrintStringArray) nome_player, ", ";
    if((CmpStr(nome_player, nome_computer))==1) print "abbiamo lo stesso
        nome.^^";
    else print "io sono ", (PrintStringArray)nome_computer, ".^^";

    print "Premi il tasto q per uscire";
    do
    {
        car = Presskey();
    }
    until (car == 'q');
];

```

basta sottrarre il valore 32 a quello della prima lettera digitata dall'utente: il valore così ottenuto, nel codice ZSCII, è proprio quello della corrispondente lettera maiuscola.

Concludiamo il capitolo con quest'ultimo programma:

```
! Esempio sull'utilizzo dei vettori - versione 5

include "ReadChar.h";

Constant LUNGH_MAX 18;
Array nome->LUNGH_MAX;

[ Main car;
  do
  {
    @erase_window $ffff; ! pulisce lo schermo
    ("Massimo è andato a casa").print_to_array(nome);
    ChangeStringArray(nome);
    print "Ciao, ", (PrintStringArray) nome, ".^";
    print "^Vuoi continuare (s/n)? "; car = ReadChar();
  }
  until (car == 'n');
];

[ PrintStringArray the_array i;
  for (i=2: i<(the_array->l)+2: i++) print (char) the_array->i;
];

[ ChangeStringArray the_array i;
  for (i=2: i<=(the_array->l)+2: i++)
  {
    if (the_array->i == 'a' or 's') the_array->i = random('e', 'i',
      'u');
  }
];
```

la funzione ChangeStringArray cambia le lettere a e s della parola Massimo, scegliendo a caso fra le lettere e, i, u per mezzo dell'istruzione random. Quest'ultima, è anche in grado di generare dei numeri casuali come nel seguente esempio:

```
[ Main a count;
  for (count=1: count<=10: count++)
  {
    a = random(6);
    print a, " ";
  }
];
```

a può valere un numero a caso compreso tra 1 e 6; un'efficace simulazione del lancio di un dado. La funzione ReadChar legge un carattere inserito dall'utente proprio come nella funzione Presskey ma, a differenza di quest'ultima, deve essere seguito dalla pressione del tasto INVIO. Ricordate anche un'altra cosa: una stringa può essere costituita anche da soli numeri che però sono visti da Inform come dei caratteri. Se una stringa contiene il numero 5, il computer interpreta questo dato come il carattere 5 (e non il numero 5 come invece avviene se assegniamo questo valore a una

variabile). Se abbiamo quindi due stringhe che valgono rispettivamente 5 e 6, non possiamo sommarle tra di loro. Capite ora la differenza che intercorre tra un numero contenuto in una variabile e un numero contenuto in una stringa?

Siamo finalmente arrivati alla conclusione di questo terribile (ma necessario) capitolo, che tutto vi ha spiegato meno quello che vi ha spinto ad apprendere questo linguaggio. Nel prossimo capitolo invece, inizieremo a vedere come fare a scrivere un'avventura testuale vera e propria, un argomento decisamente più divertente rispetto a quanto visto finora.