Qualche articolo fa avevo proposto un esercizio che riguardava l'input di una password da parte del giocatore. E' il tipico esempio che si fa quando si isegna a confrontare delle stringhe con un linguaggio di programmazione.
Una piccola nota a riguardo, un po' specialistica: chi avrà letto l'articolo, avrà probabilmente visto nell'esempio non spiegato che i caratteri vanno indicati, nel caso di array stringhe, tra singoli apici (anche nel caso fossero numeri). Questo perché Inform legge ciò che viene digitato in codice ASCII, e quindi, di fatto, ad ogni stringa è associato il numero dei singoli caratteri, quindi i caratteri, onde evitare che vengano stampati come numeri, vanno indicati tra apici. Una piccola spiegazione che poteva non essere ovvia. Rimane comunque il fatto che bisogna tenere presente questa "ambiguità".
Un altro esempio classico che si fa con le password, è quello che permette di ripetere la sua immissione un certo numero di volte. Questo si fa tramite alcuni *cicli*, la cui spiegazione è ciò che si proporrà questo articolo.
Cominciamo con questo codice sorgente:

```
Include "Parser";
Include "Verblib";
Include "Replace";

[ Ciclo digitato;
digitato=0;
print "Quanto fa 6x7?^^> ";
digitato = GetNumber(2);
while (digitato ~= 42)
{
  print "Risposta errata!^Prego digitare quanto fa 6x7^^> ";
  digitato = GetNumber(2);
  }
print "Risposta corretta!^";
];

[ Initialise;
Ciclo();
quit;
];

Include "Io";
Include "ItalianG";
```

Vediamo che abbiamo creato una routine di nome Ciclo, e che in essa abbiamo dichiarato una variabile di nome digitato. La seconda riga di questa routine assegna il valore zero alla variabile. E' indispensabile altrimenti essa contiene un valore "incognito", che dipende dalla particolare configurazione di bit presente nella cella di memoria allocata per la variabile stessa [se non siete programmatori e/o non avete capito cosa significhi, non preoccupatevene: ricordate solo che bisogna dare un valore alle variabili di questo tipo onde evitare spiacevoli inconvenienti].
Dopodiché viene stampata una domanda. Alla riga successiva è presente la funzione **GetNumber()**, la quale permette di incamerare ciò che viene digitato con la tastiera e di assegnarne il valore alla variabile digitato. Tra parentesi è indicato il numero massimo di caratteri (ebbene sì: il giocatore potrebbe scegliere di digitare anche delle lettere, che verranno interpretati come numeri per il discorso fatto prima) che possono essere digitati (se fate una prova e cercate di digitarne 3, il terzo non lo vedrete comparire).

Successivamente abbiamo il ciclo **while**, che in italiano significa "finché". Il suo significato è proprio questo: finché è vera la condizione indicata tra parentesi tonde dopo la parola *while*, esegui tutto ciò che è contenuto tra le parentesi graffe.
La condizione tra le parentesi tonde significa *digitato è diverso da 42* (42 è infatti la risposta che ci si aspetta), e quindi il ciclo si traduce con: finché il giocatore non digita 42 stampa "Risposta errata..." e aspetta di nuovo che digiti un numero. Poi il ciclo ricomincia, ovvero verrà valutato ancora se il giocatore ha digitato proprio 42 ecc...
La condizione di uscita dal ciclo è proprio che il giocatore digiti 42: se lo fa, ciò che è contenuto tra parentesi graffe chiaramente non viene eseguito (poiché è falsa la condizione tra parentesi tonde) e passa all'istruzione successiva, ovvero stampare che la risposta è corretta (e chiaramente lo è, altrimenti il programma non sarebbe mai arrivato a quell'istruzione).

Un altro modo per ottenere questo risultato è utilizzare, al posto del ciclo *while*, il ciclo **do until**:

```
Include "Parser";
Include "Verblib";
Include "Replace";

[ Ciclo digitato;
digitato =0;
print "Quanto fa 6x7?^^> ";
do
{
  digitato = GetNumber(2);
  if (digitato == 42)  "Risposta esatta!^";
  else print "Risposta errata!^Prego digitare quanto fa 6x7^^> ";
  }
  until (digitato == 42);
]

[ Initialise;
Ciclo();
quit;
];

Include "Io";
Include "ItalianG";
```

Il ciclo *do until significa:* esegui ciò che è presente tra le parentesi graffe fintantoché non si verifica la condizione espressa tra parentesi tonde dopo *until.* Il risultato è (quasi) esattamente lo stesso del cido *while*, c'è solo qualche differenza: la prima, importante è che non è stato necessario scrivere la prima istruzione *Getnumber()* fuori dal ciclo (il che, tra l'altro, significa che in generale il ciclo *do until* viene eseguito almeno una volta, il ciclo *while* potrebbe anche venire eseguito zero volte! - certo quando non si tratta di leggere testo dalla tastiera), il secondo è che la condizione di uscita del ciclo *do until* è esattamente la opposta del ciclo *while*.

Tra gli altri cicli, poi, è presente il ciclo **for**, di cui abbiamo già trattato. Di fatto si tratta di un ciclo *do until* "specializzato". Vediamone la corrispondenza:

```
do
{
istruzioni generiche
```

```
i++;
} until (i<=numero);

for(i=0: i<=numero: i++)
{
istruzioni generiche
}
```

Notiamo che nel caso dell'immissione della password non è corretto impostare solamente come condizione di uscita i<=numero (numero in questo caso corrisponderà al numero di tentativi concessi per l'immissione): se infatti preimpostiamo numero col valore di 3 (ossia concediamo al giocatore 3 tentativi) e il giocatore indovinasse al secondo, il ciclo do si ripeterebbe, ancora una volta, fino a raggiungere il terzo tentativo. In questo caso bisognerà scrivere tra le parentesi dell'*until* qualcosa come

```
until (i<=numero || Cmpstr(password_predefinita, password_inserita)==1)
```

L'operatore **||**, già introdotto negli esempi passati (e mai spiegato: mea culpa!) è un operatore logico che restituisce il valore vero (true)se almeno una delle condizioni è vera e falso (false) se tutte e due sono false.
(è possibile utilizzare anche i valorii **true** e **false** associati alle cosiddette "variabili booleane" - ne parleremo in seguito)
Un esempio:

```
x<=3 || x>5
```

restituirà il valore vero se la x avrà un valore minore o uguale a tre, o maggiore di cinque. Se la x dovesse avere un valore pari a 4, risulterà un valore falso.
Un altro operatore logico è **&&** che restituisce il valore vero se entrambe le condizioni sono vere e falso in tutti gli altri casi. Ad esempio:

```
x==4 && y>5
```

restituirà il valore vero se contemporaneamente x ha il valore 4 e y ha un valore maggiore di 5.

Si può anche negare il valore logico di un'espressione tramite l'operatore **~** (come abbiamo visto per gli attributi). Ad esempio:

```
~(x<=3 || x>5)
```

restituirà valori di verità opposti rispetto a quelli descritti in precedenza.
Come appare da quest'ultimo esempio, gli operatori logici possono essere combinati tra di loro (come fossero operazioni aritmetiche, ad esempio:

```
(condizione_1==true || variabile_1>5) &&(condizione_2==false || ~(variabile_2<5 && variabile_3<0))
```

L'importante è prestare attenzione, soprattutto quando compaiono come condizione di uscita da un ciclo, che non siano sempre vere o sempre false. Ad esempio:

```
x<3 || x>0
```

restituirà sempre il valore vero, indipendentemente da quanto vale x!

Esauriamo poi le strutture di controllo con l'istruzione **switch** che non è un ciclo ma può tornare comunque molto utile:

```
Include "Parser";
Include "Verblib";
Include "Replace";

[ Ciclo vocale carattere;
print "Digita una vocale^^";
vocale = PressKey();
switch(vocale)
{
'a': print "Hai digitato la prima vocale^";
'e', 'i', 'o': print "Non hai digitato n@'e la prima n@'e l'ultima vocale^";
'u': print "Hai digitato l'ultima vocale^";
default: print "Non hai digitato una vocale.^";
}
print "^Premi spazio per uscire^";
do
{
carattere = Presskey();
}
until (carattere == 32);
];

[ Initialise;
Ciclo();
quit;
];

Include "Io";
Include "ItalianG";
```

In questo caso vediamo un'impostazione, nelle prime due righe della routine Ciclo, simile a quella dell'esempio di while. La differenza è che in questo caso non abbiamo l'istruzione *GetNumber()* ma l'istruzione **PressKey()** che significa: attendi che l'utente prema un tasto generico. Nel nostro caso abbiamo associato questa istruzione a una variabile, e quindi il carattere digitato verrà immagazzinato nella variabile (vocale). Dopodiché è presente l'istruzione *switch()* che dirà al programma di valutare cosa è presente nella variabile compresa tra le parentesi dello *switch* (nel nostro caso è sempre la variabile vocale), di vedere a cosa corrisponde tra le scelte seguenti e di stampare un testo di conseguenza (indicate, nell'esempio, dalle varie vocali dell'alfabeto). E' presente anche una risposta di *default*, che nello switch va sempre indicata come ultima alternativa, nel caso il giocatore abbia deciso di digitare un carattere che non sia una vocale.
Come detto all'inizio, Inform interpreta i caratteri digitati in codice ASCII, cioè secondo numeri, e questo ci torna utile nel ciclo *do* che segue lo *switch*: il programma continuerà ad aspettare che venga premuto il tasto spazio (che in codice ASCII corrisponde proprio al valore 32) per proseguire, anche nel caso in cui vengano premuti altri tasti.

Per il momento non mi vengono in mente esercizi creativi, casomai fate una capatina sul mio sito in futuro per vedere se mi sarà venuto in mente qualcosa di soddisfacente da proporvi.