In questo articolo cominceremo a parlare di array: è un argomento abbastanza complesso e, per certi versi, ostico (Scarpa stesso lo ammette nel manuale), quindi verrà sviluppato in più articoli in modo da includere anche, una volta per tutte, alcune funzioni particolarmente interessanti che possono sempre tornare utili. Rimane il fatto che comunque sono basilari, dal momento che è proprio con gli array che è possibile assegnare un punteggio a determinate azioni.
[Cominciamo subito con una nota di game design: altrove si è posta la domanda: vedremo mai un'avventura testuale - per quanto già alcune ne esistano - in cui per progredire nella storia non c'è bisogno di un punteggio? A voi la risposta]
Per questo motivo, anziché iniziare con un codice sorgente, faremo degli esempi volta per volta per chiarire le spiegazioni.

Prima di ogni cosa diciamo che per lavorare efficacemente sugli array è stata creata una libreria apposita che ci semplifica la vita: si trova nel pacchetto di librerie consigliate qualche articolo fa, il cui nome è **Io.h** e che quindi va scompattata nella cartella *libraries*. Questo file, peraltro, contiene numerose altre funzioni molto utili che vedremo in seguito.

Iniziamo col definire un array: possiamo identificarlo come una serie di "celle" adiacenti in cui sono presenti determinati valori (numeri/caratteri), ciascuna di queste celle è identificata da un indice. Ad esempio, possiamo creare un array di 10 celle il cui indice, però, varia progressivamente da 0 (è sempre il primo indice) a 9 (notare che sono comunque 10 valori): ossia, se con *i* indichiamo il numero di celle del nostro array, gli indici arriveranno fino a *i-1*.
Vediamo come si dichiara un array, che chiameremo vettore, di 10 caselle:

```
! gli array si dichiarano dopo Include "Replace";
Array vettore -> 10;
```

che ricalca esattamente la dichiarazione di un oggetto. In questo momento le celle sono "vuote" o meglio *non inizializzate* (hanno un valore, ma non sappiamo quale).
Vediamo per il momento **ARRAY NUMERICI**.
Per inizializzare un array (numerico) possiamo utilizzare un codice simile:

```
Include "Parser";

Include "Verblib";

Include "Replace";


Array vettore -> 10;


[ Operazioni vettore_passato i;

for (i=1: i<=10: i++) vettore_passato->(i-1)=i;

vettore_passato->(5) = 145;

PrintNumericArray (vettore, 10);

];


[ Initialise;
```

Operazioni(vettore);

quit;

];

Include "Io";

Include "ItalianG";

E già qui sono dolenti note. Prendiamo una bella boccata d'aria e facciamoci coraggio.
Vediamo che abbiamo dichiarato un array. Passiamo alla funzione *Initialise*, che è quella da cui parte il programma. La prima istruzione che leggiamo è *Operazione(vettore);* che viene detta "chiamata di funzione" e che significa, nell'esempio, "esegui la routine Operazioni prendendo come parametro ciò che hai dichiarato col nome di vettore". Quindi, comportandoci come il programma, passiamo alla routine Operazioni. Vediamo che assomiglia molto alla routine *Initialise*: abbiamo aperto una quadra, scritto il nome della routine (aggiunto qualche altra roba) e dichiarato delle istruzioni, poi abbiamo chiuso la quadra e infine, al solito, terminato con un punto e virgola. Quello che abbiamo dichiarato **dopo** il nome della routine sono i parametri di cui avremo bisogno **all'interno della routine stessa**, cioè un array e una variabile i.
Abbiamo chiamato questo array vettore_passato perché abbiamo creato una corrispondenza tra l'array dell'istruzione *Operazioni(vettore);* e l'array all'interno della routine Operazioni (abbiamo aggiunto _passato al nome vettore perché questo procedimento si chiama "passaggio di valori", ma potevamo anche chiamarlo pinco), ossia: siccome abbiamo due routine differenti che agiscono sullo stesso array denotiamo anche questi array (per non fare confusione) con lo stesso nome, ossia: vettore_passato è come viene chiamato l'array vettore (solo e soltanto) nella routine Operazioni. Attenzione: il nome vettore_passato può essere utilizzato soltanto all'interno della routine Operazioni, perché è lì che è stato dichiarato.
La prima istruzione della routine Operazioni è un **ciclo for**. La sintassi di questo ciclo è

for(variabile=valore_iniziale: condizione1: modifica valore variabile) azione;

il che significa: prendi una variabile e assegnale un valore di inizio (nell'esempio *i=1*), finché vale la condizione1 (cioè, nell'esempio, finché i è minore o uguale a 10) modifica il valore della variabile come indicato (*i++* significa "aumenta il valore di i di 1 e corrisponde alla forma abbreviata di *i=i+1*; - da notare che la forma abbreviata è utilizzabile solo se l'incremento è unitario; analogo è il decremento unitario) ed esegui l'azione. Cioè, per il nostro esempio: esegui l'azione per tutti i valori di i da 1 a 10, con incremento di i di 1, o, in altro modo, esegui 10 volte la azione. I valori di i assumono i valori 1, 2, 3... 10.
[Nota per i programmatori che conoscono C++: Inform 6 prevede l'utilizzo di : per separare le tre espressioni!]
Ora vediamo la nostra "azione": alla cella *(i-1)* del vettore passato assegna il valore di i.
Quindi al primo ciclo i vale uno (il nostro valore iniziale), verifichiamo che 1 è minore o uguale a 10, siccome lo è eseguiamo l'azione. Ricordiamo che l'indice iniziale delle celle di un vettore è zero, quindi non è corretto scrivere *vettore_passato->i = i;* perché altrimenti assegneremmo il valore 1 alla **seconda** cella (cioè quella con indice 1).
Dopodiché si incrementa il valore di i e si ripete il ciclo.
Quindi il risultato è un array:
di nome vettore_passato
di 10 caselle
di indici: 0 1 2 3 4 5 6 7 8 9

con valori: 1 2 3 4 5 6 7 8 9 10

A posteriori capiamo anche perché all'inizio, di fianco al nome della routine Operazioni, oltre all'array di nome vettore_passato abbiamo dichiarato anche la variabile i. Da notare, però, che come è stato necessario indicare il nome del vettore dopo il nome della routine, così è stato necessario indicare la variabile i dopo il nome dell'array, altrimenti avremmo passato la parola vettore (della chiamata *Operazione(vettore)* nella *Initialise*) a i e non a vettore_passato (ossia avremmo ottenuto lo stesso risultato, nell'esempio, di scrivere *Operazioni(i);* il che è chiaramente sbagliato, a meno di invertire successivamente nella routine Operazioni i due nomi).
Naturalmente l'"azione" da eseguire poteva essere diversa, ad esempio:

*for (i=1: i<=10: i++) vettore_passato->(i-1) = i*2;*

in questo caso avremmo avuto

indici:	0	1	2	3	4	5	6	7	8	9
valori:	2	4	6	8	10	12	14	16	18	20

L'inizializzazione tramite un *ciclo for* può chiaramente essere utilizzata laddove si individui una relazione tra i valori delle celle dell'array e gli indici della loro posizione, ovvero laddove si possa costruire quella che in matematica viene chiamata una *successione* (ebbene sì: per la gestione di array numerici è consigliabile avere alcune nozioni matematiche, per quanto "relativamente semplici" - vedere esercizi). Nel caso non fosse possibile (o anche se lo fosse ma non volessimo utilizzare un *ciclo for*) è possibile creare un array già inizializzato. Ad esempio, il vettore precedente poteva essere dichiarato come

Array vettore -> 1 2 3 4 5 6 7 8 9 10;

che di solito è il modo in cui si dichiara l'array dei punteggi, non essendo generalmente identificata una legge che regoli i valori delle celle.
Un'altra cosa tipica è quella di non dichiarare la dimensione di un array con un valore numerico, ma di associare quel valore numerico ad una costante che poi viene utilizzata al posto di esso. Spieghiamo: anziché dichiarare l'array vettore come abbiamo fatto potevamo anche scrivere

! dopo Include "Replace"
Constant LUNGH_MAX = 10;
Array vettore -> LUNGH_MAX;

cioè da adesso in poi, al posto di scrivere il valore 10 possiamo scrivere il nome che gli abbiamo associato; ad esempio nella terza istruzione della routine Operazioni avremmo potuto scrivere *PrintNumericArray(vettore, LUNGH_MAX);* ottenendo lo stesso risultato. Il vantaggio è che è più facile ricordarsi un nome che un numero, inoltre, se nello sviluppo dell'avventura dovessimo accorgerci che una LUNGH_MAX da noi definita non è sufficiente, basterà modificarne il valore nella dichiarazione della *Constant* perché sia modificato automaticamente ovunque compaia LUNGH_MAX.

La seconda istruzione della routine Opzioni ci dimostra come modificare un singolo valore di un array. Nella fattispecie accediamo (*->*) al sesto elemento (*5*, perché l'indice va scalato di uno!) del vettore (array) e gli assegniamo (*=*) il nuovo valore (145). Da notare che la sola espressione *vettore_passato->(5)* può anche identificare il valore che nell'array ha indice 5!

La istruzione *PrintNumericArray* (inclusa nella libreria *Io*) permette di stampare su schermo i valori di un array numerico: come è facile vedere basta specificare all'interno delle parentesi tonde il nome dell'array e la sua dimensione massima.

Una volta che la funzione chiamata è terminata, cioè una volta che sono state eseguite tutte le istruzioni dichiarate al suo interno, il programma ritorna alla routine chiamante, in questo caso ancora la *Initialise*, procedendo dalla chiamata di funzione appena eseguita. Quindi, nell'esempio, dopo aver eseguito la *Operazioni(vettore);* si prosegue con **quit**, che ha l'effetto di terminare il programma.

Prima di lasciare qualche esercizio, diciamo che l'array che definisce i vari punteggi si chiama **task_scores** e i suoi valori vengono definiti nella dichiarazione. Si associano sempre ad esso anche una costante **MAX_SCORE** che definisce il massimo punteggio ottenibile nell'avventura (dato dalla somma dei valori di inizializzazione dell'array *task_score*), una costante **NUMBER_TASKS** che definisce di quanti indici abbiamo bisogno per identificare i valori dell'array (cioè la nostra precedente LUNGH_MAX) e una costante **TASKS_PROVIDED**.
Finiamo la parte teorica col dire che Inform 6 non considera numeri superiori a 255, così come non è possibile definire un array con più di 255 celle (a meno di usare artifici), e che l'array *task_scores* non può contenere valori negativi (ma esistono comunque metodi per decurtare il punteggio nel corso dell'avventura).

Per gli array numerici direi che è tutto, il consiglio è di impratichirsi sul loro uso prima di vedere le **stringhe** (array di caratteri) che sono leggermente più complesse e a cui sono associate più funzioni di libreria che, se opportunamente usate, concorrono alla buona riuscita di un'avventura testuale.

Per esercizio:
Creare un programma che, tramite opportune routines, stampi i numeri:

- da 1 a 100 in ordine crescente
- da 1 a 100 in ordine decrescente
- il doppio dei numeri da 1 a 100 (in ordine crescente)
- i numeri da 1 a 50 (in ordine crescente)
- i numeri da 51 a 100 (in ordine crescente)
- i primi 13 numeri della serie di Fibonacci (cercando di capire perché non se ne possono chiedere 14 o più) [cercare su internet cosa sono i numeri di Fibonacci se non si conoscono. Quest'ultima richiesta potrebbe risultare difficile, ma è altamente consigliato cercare di risolverla!]

e che al termine produca un saluto all'utente informandolo che il programma è stato eseguito.

(La soluzione, come sempre, sul mio sito personale)