ALGORITMI DI BASE (2° Parte):
CARATTERISTICA, TUTTI CON CARATTERISTICA, INSERIMENTO-ELIMINAZIONE DA UN VETTORE

Come già visto, ricordiamo che, nell'esposizione degli Algoritmi di Base, indicheremo genericamente con il nome **"Vet"** il vettore che contiene la *Lista di Dati* da gestire, con il nome **"K"** (o "K1", "K2") l'*indice dei cicli di scansione* e con il nome **"N"** la variabile indicante il *numero di dati reali* presenti nel vettore.

Elaborare solo gli Elementi di un Vettore che hanno una determinata Caratteristica

Spesso capita di dover scorrere un Vettore, considerando ed elaborando *solo gli elementi che hanno una data caratteristica*.

☞ Esempi tipici sono: contare i voti insufficienti, sommare solo i numeri positivi, contare gli "1" in un numero binario, ecc.

Per effettuare una **Elaborazione dei Soli Elementi di un Vettore aventi una Data Caratteristica**, si effettua una *Scansione Completa del Vettore* e, ad ogni passo, si esegue un *if* la cui condizione verifica se l'elemento in esame *Vet[K]*, gode della caratteristica richiesta:

```
for ( int K = 0;  K <= N-1;  K++)          … scansione completa del vettore
   if  ( … Vet [ K ] gode della caratteristica … )   … ad ogni passo, verifica se Vet[K] gode della caratteristica
      {  … operazione su  Vet [ K ] …  }     … se gode della caratteristica, allora fai l'operazione richiesta
```

Se *Vet[K]* gode della caratteristica, allora è uno degli elementi da elaborare, quindi, nell'*if*, si procede effettuando su di esso le operazioni richieste.

☞ Ad esempio, volendo conteggiare i soli voti sufficienti presenti in un vettore *VetVoti*, si utilizza un contatore *ContVotiSuff* inizialmente nullo, e lo si incrementa ogni volta che si scandisce un voto sufficiente (caratteristica):

```
int ContVotiSuff = 0;                  … dichiara e azzera il contatore ContVotiSuff
for ( int K = 0;  K <= N-1;  K++)      … scansione completa del vettore
      if  ( VetVoti [K]  >=  6 )       … ad ogni passo, verifica se il voto in esame VetVoti[K] è sufficiente
            ContVotiSuff++;            … se lo è, allora incrementa il contatore ContVotiSuff
```

Verificare se Tutti gli Elementi di un Vettore hanno una determinata Caratteristica

Spesso capita di dover verificare se, in un Vettore, *tutti gli elementi hanno una data caratteristica*.

☞ Esempi tipici sono: tutti i voti sono sufficienti, non c'è nessun numero negativo, non ci sono "0" in un numero binario, ecc.

Il risultato di questo tipo di elaborazione è un valore di tipo *bool*: **true** (*TUTTI godono della caratteristica*) oppure **false** (*ALMENO UNO non gode della caratteristica*).

Per **Verificare se Tutti gli Elementi di un Vettore hanno una determinata Caratteristica**, si utilizza una Variabile di tipo *bool* che chiameremo **TuttiHannoCaratteristica**, inizialmente impostata a **true**.

Si effettua una *Scansione del Vettore* e, ad ogni passo, si esegue un *if* la cui condizione verifica se l'elemento *Vet[K]* **non gode** della caratteristica: in tal caso, si pone la variabile *TuttiHannoCaratteristica* a **false** e si abbandona il ciclo.

```
bool TuttiHannoCaratteristica = true;     … dichiara e inizializza a true la variabile di tipo bool
for ( int K = 0;  K <= N-1;  K++)         … scansione del vettore
   if  ( … Vet [K] non gode della caratteristica … )   … verifica se Vet[K]  NON GODE  della caratteristica
   {
     TuttiHannoCaratteristica = false;    … non è più vero che tutti godono della caratteristica!
     break;                               … interrompi il ciclo
   }
```

Nel caso in cui *tutti gli elementi hanno la caratteristica richiesta*, la condizione dell' *if* non si verifica mai e nella variabile *TuttiHannoCaratteristica* rimane il valore iniziale, ossia *true*.

Al termine del ciclo, nella variabile *TuttiHannoCaratteristica* è presente il risultato desiderato (*true o false*).

☞ Ad esempio, volendo verifica se un alunno è promosso, ossia se TUTTI i suoi voti sono sufficienti, si utilizza una variabile bool *TuttiSufficienti* inizialmente posta a *true*. Nel ciclo, la si pone a *false*, appena il voto in esame *VetVoti[K]* risulta insufficiente:

```
bool TuttiSufficienti = true;        … dichiara e pone a true la variabile bool TuttiSufficienti
for ( int K = 0;  K <= N-1;  K++)    … scansione del vettore
      if  ( VetVoti[K] < 6 )         … per ogni passo, verifica se il voto in esame VetVoti[K] è insufficiente
      {
        TuttiSufficienti = false;    … se lo è, allora pone TuttiSufficienti a false
        break;                       … interrompi il ciclo
      }
```

Inserire un nuovo Elemento nella Lista di Dati contenuta in un Vettore

La struttura di memoria chiamata **Vettore**, come è noto, viene *dimensionata* in modo da poter ospitare un numero massimo di dati (**costante Max**). Nell'uso reale, però, di tutte le posizioni allocate, ne vengono realmente utilizzate solo un certo numero (**variabile N**) e, precisamente: dall'elemento in *posizione 0* a quello in *posizione N-1*.

Per **Inserimento in un Vettore**, si intende l'aggiunta di un nuovo Dato, in una data Posizione, senza sovrascrivere, né perdere in alcun modo i dati preesistenti o il loro ordinamento.

☞ Ad esempio, se un vettore di **Max = 10** elementi, inizialmente, contiene **N = 7** dati …

B	T	R	F	Q	P	A	----	----	----
0	1	2	3	4	5	6	7	8	9

inserire un **nuovo dato "S"** in **posizione 3**, significa ottenere:

B	T	R	S	F	Q	P	A	----	----
0	1	2	3	4	5	6	7	8	9

Si noti come, per evitare perdite di dati o modifiche dell'ordine preesistenti, gli elementi in grigio sono stati tutti "traslati" di una posizione in avanti.

Le variabili **NuovoDato** e **PosIns** contengono rispettivamente il *dato da inserire* e la *posizione di inserimento*.

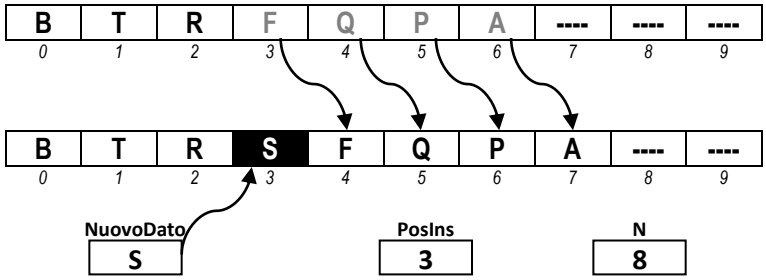
Per realizzare correttamente l'*Inserimento in un Vettore* è necessario "***shiftare***" (ossia, *spostare*), di una posizione in avanti, tutti gli elementi *dalla posizione* ***PosIns*** *alla posizione* ***N-1***.

Per farlo, si utilizza un normale *for* che scandisce gli elementi da spostare e che, ad ogni passo, copia l'elemento in esame **Vet[K]** nella posizione successiva (ossia in **Vet[K+1]**). Per evitare di sovrascrivere gli elementi, il ciclo deve scandire le posizioni **all'indietro** (*da N-1 a PosIns)*:

```
for ( int K = N-1;  K >= PosIns;  K--)   … scansione parziale del vettore "all'indietro" (da N-1 a PosIns)
   Vet[K+1] = Vet[K];                    … copia l'elemento Vet[K] nella posizione successiva Vet[K+1]
Vet[PosIns] = NuovoDato;                 … creato lo "spazio", si pone il nuovo dato nella posiz. richiesta
N++;                                     … avendo aggiunto un elemento, si incrementa la variabile N
```

Completato il *for* e, quindi, lo "shifting" dei dati, è possibile porre il nuovo dato (variabile *NuovoDato*) nella posizione richiesta (*PosIns*), in quanto si è creato lo "spazio" necessario a memorizzarlo, senza sovrascrivere i dati esistenti.

Inoltre è fondamentale *incrementare la variabile N*, in quanto il numero di dati reali nel vettore è aumentato di 1.



Eliminare un Elemento Esistente dalla Lista di Dati contenuta in un Vettore

Per **Eliminazione in un Vettore**, si intende la rimozione del dato che si trova in una data posizione, garantendo che gli elementi successivi a quello rimosso siano traslati per occupare la posizione e mantenere il corretto ordinamento.

☞ Ad esempio, se un vettore di **Max = 10** elementi, inizialmente, contiene **N = 7** dati …

B	T	R	F	Q	P	A	----	----	----
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

eliminare il dato in **posizione 3**, significa ottenere:

B	T	R	Q	P	A	----	----	----	----
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

Si noti come, per far si che *la "Q" diventi il 4° elemento del vettore* (ossia, occupi la posizione 3), che la "P" diventi il 5° e che la "A" diventi il 6°, gli elementi successivi a quello rimosso sono stati tutti "traslati" di una posizione all'indietro.

Indichiamo nuovamente il vettore con il nome **Vet**, mentre la variabile **PosElim** contiene la *posizione del dato da rimuovere*.

Per realizzare correttamente l'*Eliminazione in un Vettore* è necessario "***shiftare***" (ossia, *spostare*), di una posizione all'indietro, tutti gli elementi *dalla posizione PosElim+1 alla posizione N-1*. Per farlo, si utilizza un normale *for* che scandisce gli elementi da spostare e che, ad ogni passo, copia l'elemento in esame *Vet[K]* nella posizione precedente (ossia in *Vet[K-1]*). Al contrario dell'inserimento, il ciclo può scandire le posizioni in avanti (*da PosElim+1 a N-1)*:

```
for ( int K = PosElim+1;  K <= N-1;  K++)     … scansione parziale del vettore da PosElim+1 a N-1
   Vet[K-1] = Vet[K];                         … copia l'elemento Vet[K] nella posizione precedente Vet[K-1]
N--;                                          … avendo rimosso un elemento, si decrementa la variabile N
```

Completato il *for* e, quindi, lo "shifting" dei dati, è fondamentale *decrementare la variabile N* in quanto il numero di dati reali nel vettore è diminuito di 1.

☞ Si noti come il dato che si trova in quella che prima era l'ultima posizione valida nel vettore (la posizione 6, nella figura), dopo lo shifting, *non viene effettivamente cancellato* … semplicemente la sua posizione viene considerata "libera" e quindi sarà sovrascritta alla prossima aggiunta o al prossimo inserimento di nuovi dati. Per questo motivo è fondamentale *decrementare il valore della variabile N* (prima era 7, dopo l'eliminazione deve diventare 6, in modo tale che 6 sia considerata *la prima locazione libera del vettore*):

B	T	R	Q	P	A	A	----	----	----
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

ZONA NON UTILIZZATA DEL VETTORE