Strutture di controllo, Diagrammi di flusso, codifica in C degli algoritmi e dicotomia

Vitoantonio Bevilacqua

bevilacqua@poliba.it

Sommario. Il presente paragrafo si riferisce alle lezioni del corso di Fondamenti di Informatica e Laboratorio di Informatica dei giorni 12, 16 e 17 marzo 2010.

Parole chiave: Algoritmo, Diagramma di flusso, Ricerca dicotomica.

1 Introduzione

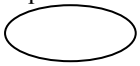
Un algoritmo è una sequenza ordinata di passi elementari che consente di passare da dati in ingresso a dati in uscita; alcune delle proprietà fondamentali di un algoritmo sono:
- eseguibilità: ogni parte, ogni diramazione dell'algoritmo deve essere percorribile;
- non ambiguità: la scelta di un percorso rispetto ad un altro è legata ad una condizione (cioè da ogni punto del programma non devo poter andare dove voglio);
- finitezza: l'algoritmo deve finire altrimenti si va in "loop".

Un diagramma di flusso è una rappresentazione simbolica di un algoritmo e ha lo scopo di esprimere il meccanismo di interazione con l'utenza e le operazioni da fare a partire dai dati inseriti.

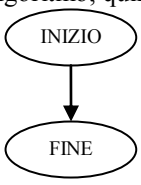
Le operazioni esplicitate in un algoritmo necessitano di essere eseguite sotto il controllo di un esecutore che le interpreta ed esegue nell'ordine logico di esecuzione.
Poiché gli algoritmi che incontreremo dovranno essere eseguiti in maniera automatica, l'esecutore è l'elaboratore e quindi l'algoritmo dovrà essere tradotto in una sequenza di istruzioni scritte in un linguaggio che l'elaboratore è in grado di interpretare ed eseguire. Un linguaggio che ci consente di fare ciò è appunto il linguaggio C.

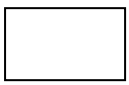
2 Struttura di un diagramma di flusso

Un qualsiasi diagramma di flusso è gestito da quattro schemi fondamentali, che opportunamente combinati attraverso delle frecce rappresentano la sequenza logica e temporale dell'evoluzione dell'algoritmo.
I quattro schemi hanno un ordine e un valore logico completamente diverso, e sono:

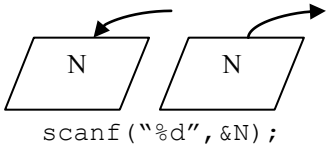


1. una "ellisse" che contiene al suo interno un label (etichetta) che può indicare l' "inizio" o la "fine" dell'algoritmo; quindi ciascun algoritmo avrà questa forma:





2. il "blocco delle assegnazioni", che contiene appunto un'assegnazione[1]; in particolare si assegna la parte destra del segno di = alla variabile a sinistra del segno; un'assegnazione del tipo `i=0` è un'assegnazione "di inizializzazione".



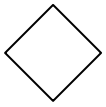

```
scanf("%d",&N);
```

3. il "blocco di input/output": se di input, serve ad assegnare alla variabile contenuta nel blocco (ad es. N) un valore immesso dall'esterno e in C corrisponde all'istruzione:

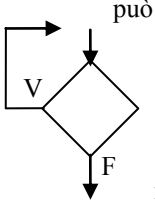
[1] vedi Prima Dispensa, paragrafo 4 "Verso il C"

se di output, serve a stampare su video il valore della variabile contenuta nel blocco, precedentemente modificato dall'algoritmo e in C corrisponde all'istruzione:

```
printf("%d",N);
```



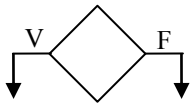
4. il "blocco condizionale", che controlla se la condizione contenuta sia vera o no e restituisce l'esito vero o falso come output; quindi in un diagramma di flusso si trova sempre dopo altre operazioni di assegnazione o di input; si può presentare sotto due forme principali:



- rappresenta un "ciclo con controllo in coda": controlla il risultato di un'azione e se è vera viene rieseguita; in C corrisponde ad una struttura di tipo "do while":

```
do
{
  istruzione
}
while(condizione);
```

in cui l'istruzione viene eseguita fintanto che la condizione risulta vera;

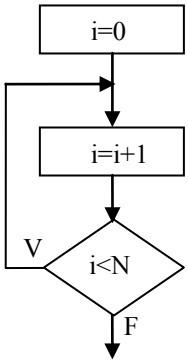


- a seconda della veridicità o meno ell'espressione inserita decide di eseguire un ramo o l'altro dell'algoritmo; in C corrisponde alla struttura dell' "if":

```
if(condizione)
{
  istruzione conseguenza della verità;
}
else
{
  istruzione conseguenza della falsità;
}
```

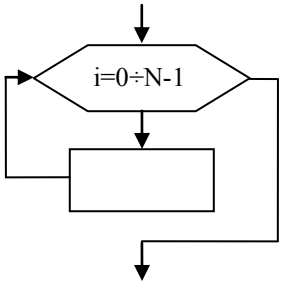
in cui la 1^ istruzione viene eseguita solo se la condizione è vera, altrimenti viene eseguita la 2^ istruzione presente nel corpo dell' "else"; tuttavia quest'ultima può anche essere assente e in tal caso se non risulterà vera la condizione iniziale, l'elaboratore passerà direttamente alla linea di codice successiva saltando l'intero corpo dell'if.

2.1 Controllo con contatore o di iterazione



Il costrutto di controllo con contatore è simile al ciclo con controllo in coda, però il ciclo controlla il valore di una variabile detta "contatore", abitualmente indicata con la lettera "i", confrontandola con il valore di un'altra variabile al termine di ogni ciclo. Se la condizione da verificare risulta vera, il ciclo si ripete e, contestualmente all'operazione eseguita all'interno del ciclo, si incrementa ogni volta di 1 la variabile contatore.
Nell'inserimento di una variabile contatore risulta necessario inizializzarla con valore 0, altrimenti non conoscendone il valore iniziale non la si può confrontare con un'altra variabile.

2.2 Ciclo for



Il "ciclo for" è un ciclo ad iterazione che in alcuni casi può sostituire il ciclo del "do while".
Consente di eseguire una determinata operazione o un insieme di operazioni in successione (il corpo del ciclo) fintanto che il contatore i

varia tra due valori numerici (es: 0 e N-1) indicati nel blocco del "for"; al termine delle operazioni contenute nel corpo del ciclo il contatore viene incrementato di 1, quindi si ripassa dal blocco for, dove viene controllato il valore del contatore e viene ripercorso l'intero corpo del ciclo; ciò finché il contatore i non avrà superato il valore massimo ammesso. In C corrisponde ad una struttura del seguente tipo:

```
for(i=0;i<N;i++)
{
    istruzione
}
```

dove

1. viene eseguita l'operazione `i=0` inserita tra la parentesi tonda aperta e il primo punto e virgola, detta "inizializzazione", che da al contatore un valore iniziale da cui partire e non verrà più eseguita successivamente;
2. viene verificata la condizione `i<N` inserita tra il primo e il secondo punto e virgola e se risulterà vera, viene eseguita `istruzione`, altrimenti il ciclo non viene percorso;
3. dopo aver percorso il corpo del ciclo, viene eseguita la terza operazione `i++`[2] inserita tra il secondo punto e virgola e la parentesi tonda chiusa e di nuovo valutata la condizione `i<N` che se risulterà vera, darà luogo all'esecuzione di. `istruzione`. Il tutto si ripeterà finché `i<N` non risulterà falsa.

Nel costrutto del for non è necessario che tutte e tre le operazioni contenute all'interno della parentesi tonda siano presenti, tuttavia è necessario che siano presenti i due punti e virgola:

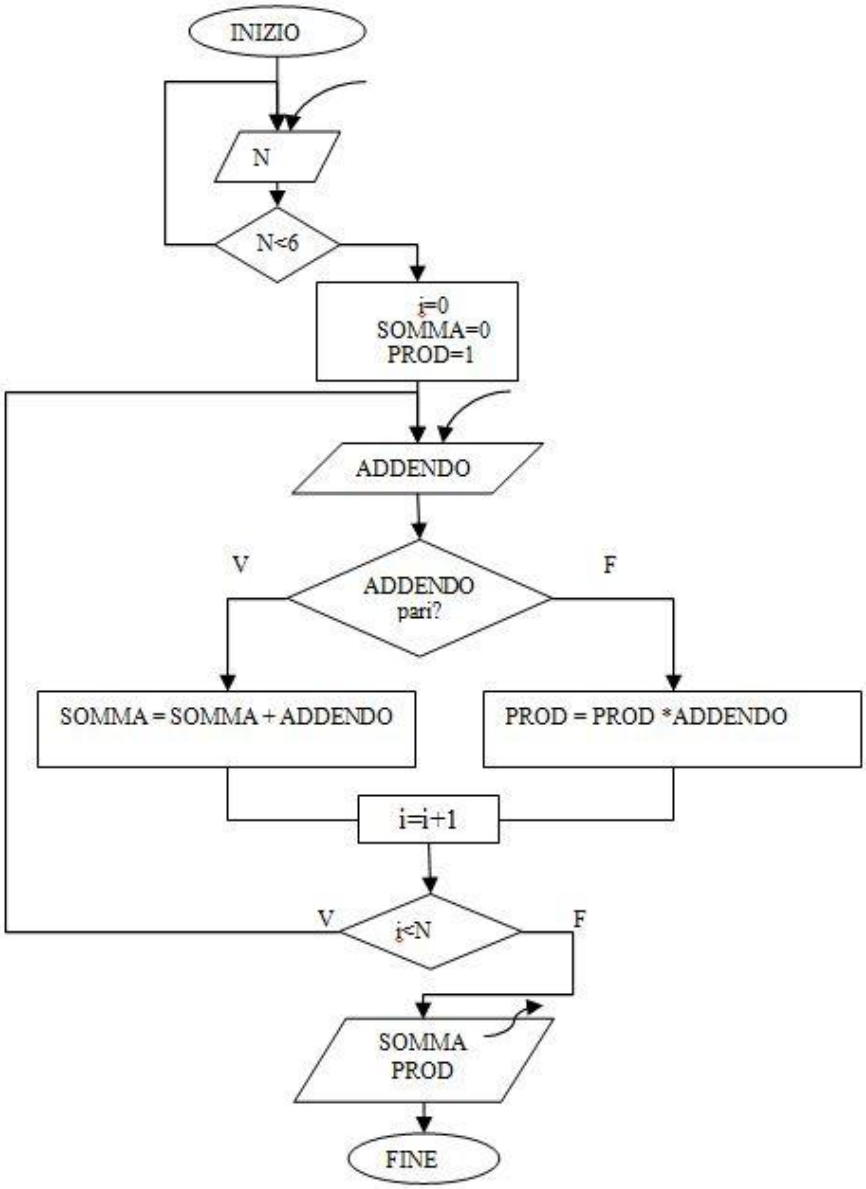
```
for(;i<N;i++)
{
    istruzioni
}
```

nell'esempio il contatore i non viene inizializzato.

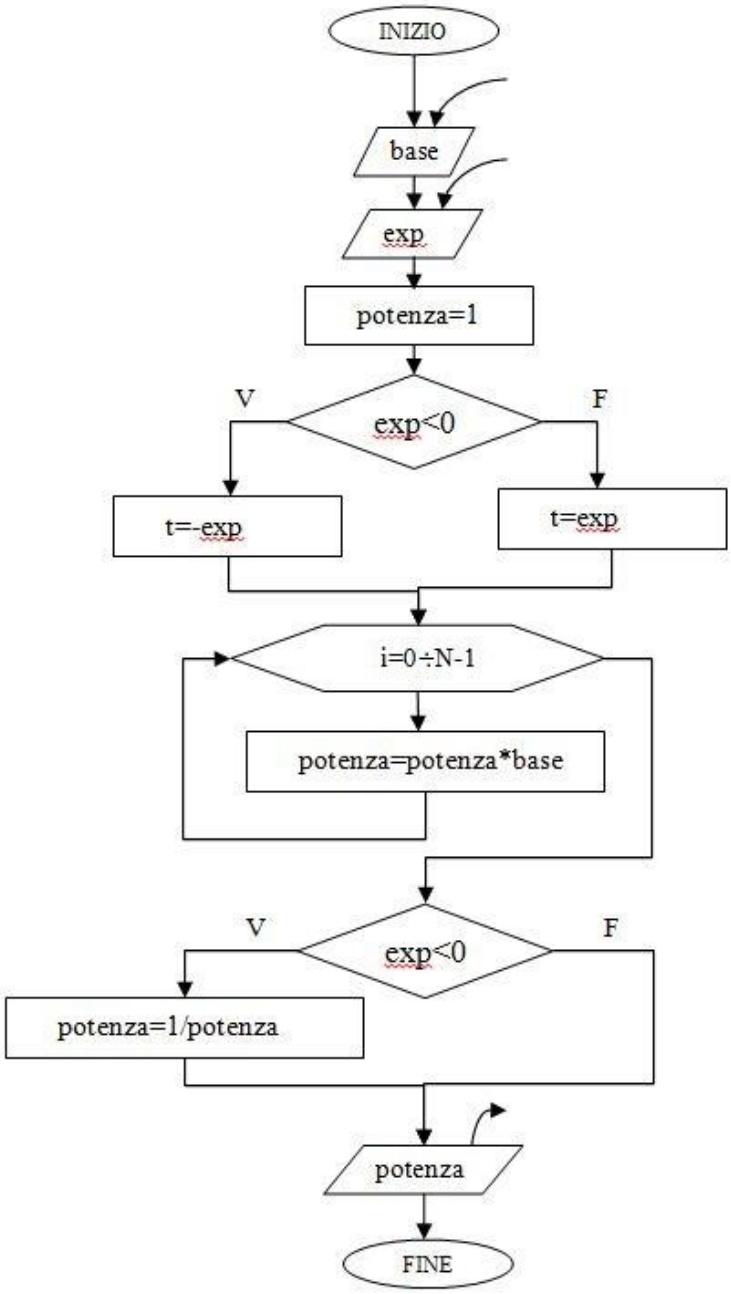
3 Esempi

Algoritmo delle somme e prodotto parziali: il seguente algoritmo chiede di quanti numeri si vuol fare la somma (minimo 6), quindi richiede tutti i numeri e successivamente stampa a schermo la somma dei pari e il prodotto dei dispari.

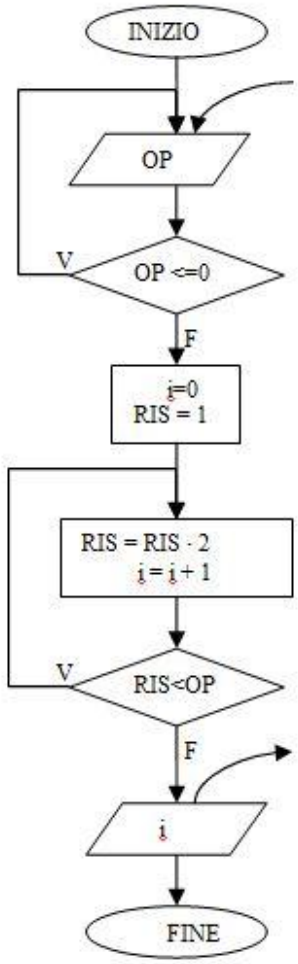
[2] equivalente all'istruzione `i=i+1`



Algoritmo del calcolo della potenza: il seguente algoritmo chiede un valore della base e un valore positivo dell'esponente e stampa a schermo la potenza.



In appendice troverete il codice corrispondente all'algoritmo del calcolo della potenza, ma nel caso in cui è possibile immettere valori di base ed esponente sia positivi che negativi.
Algoritmo del calcolo del logaritmo in base 2 per eccesso: il seguente algoritmo chiede un valore intero che per le proprietà del logaritmo dovrà essere necessariamente positivo, quindi ne calcola e stampa a schermo il logaritmo in base 2.



4 Operazione di “casting” e di “modulo”

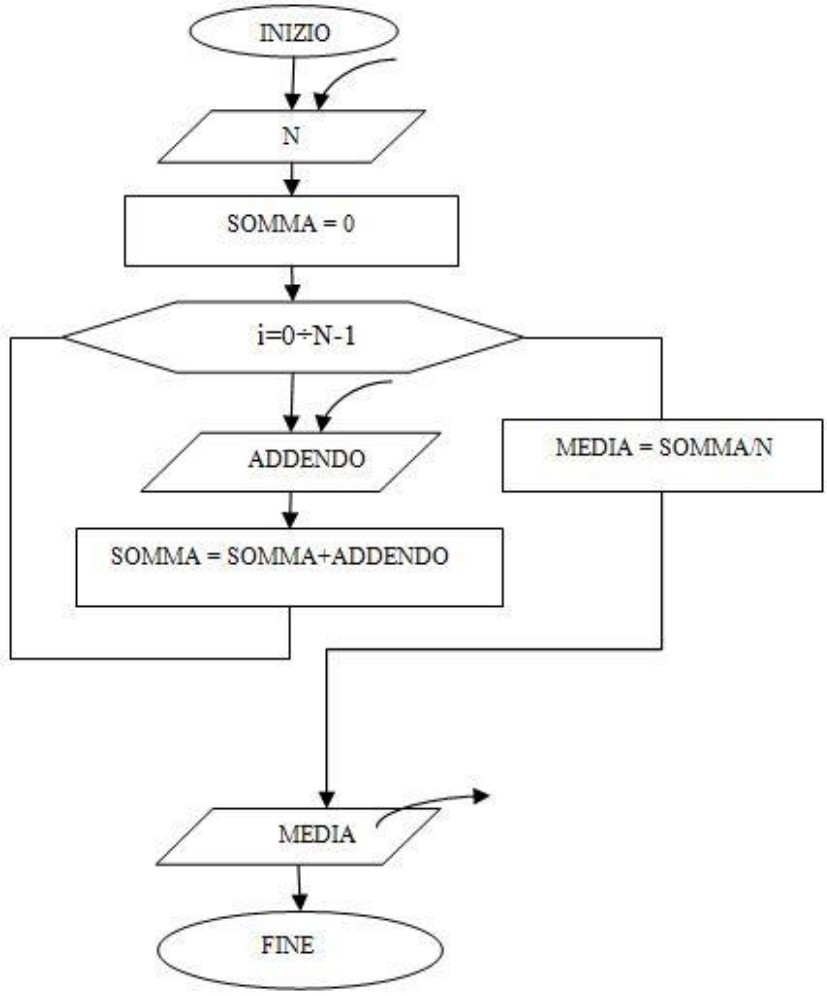
Quando si lavora con tipi di dati diversi tra loro (int, char, float, etc.), può essere necessario convertire valori da un tipo ad un altro. Questa operazione si chiama “casting” e forza la conversione esplicita di un tipo ad un altro tipo, conversione magari non prevista automaticamente.

In C, per convertire esplicitamente un tipo ad un altro tipo, si usa l'operatore ( ), detto operatore di cast; all'interno delle parentesi bisogna mettere il nuovo tipo al quale vogliamo passare, e fuori il valore che si vuole modificare. Inoltre l’argomento di tipo stringa “%d” è sostituito da “%f” per indicare che si tratta di un numero decimale in virgola mobile.

```
int somma,N;
float media;
media = (float)somma/N;
printf(“La media vale %f”,media);
```

4.1 Applicazione: algoritmo della media

Il seguente algoritmo chiede il numero di addendi da inserire, il valore degli addendi e ne restituisce la media.



5 Concetto di "Vettore"

Per trattare insiemi omogenei di dati si possono utilizzare i vettori.

Dal punto di vista algebrico il vettore è un estensione multidimensionale del concetto di scalare. Dal punto di vista informatico un vettore è una variabile strutturata dove è possibile memorizzare un numero n di valori dello stesso tipo; dunque un vettore è costituito da una lista si variabili, ciascuna delle quali è una componente.

Come per qualsiasi altra variabile, deve essere definito il nome e il tipo; inoltre, per il vettore deve essere esplicitata la lunghezza , cioè il numero di componenti, tra parentesi quadre, e questo numero dovrà essere necessariamente un intero positivo.

```
int vettore[10];
```

Questa dichiarazione permette di riservare in memoria RAM uno spazio contiguo di 10 word di tipo int, ciascuna delle quali è una componente del vettore.

Per accedere ad una singola componente del vettore, si deve specificare il nome di quest'ultimo seguito dall'indice in parentesi quadre:

```
vettore[0]=10;
```

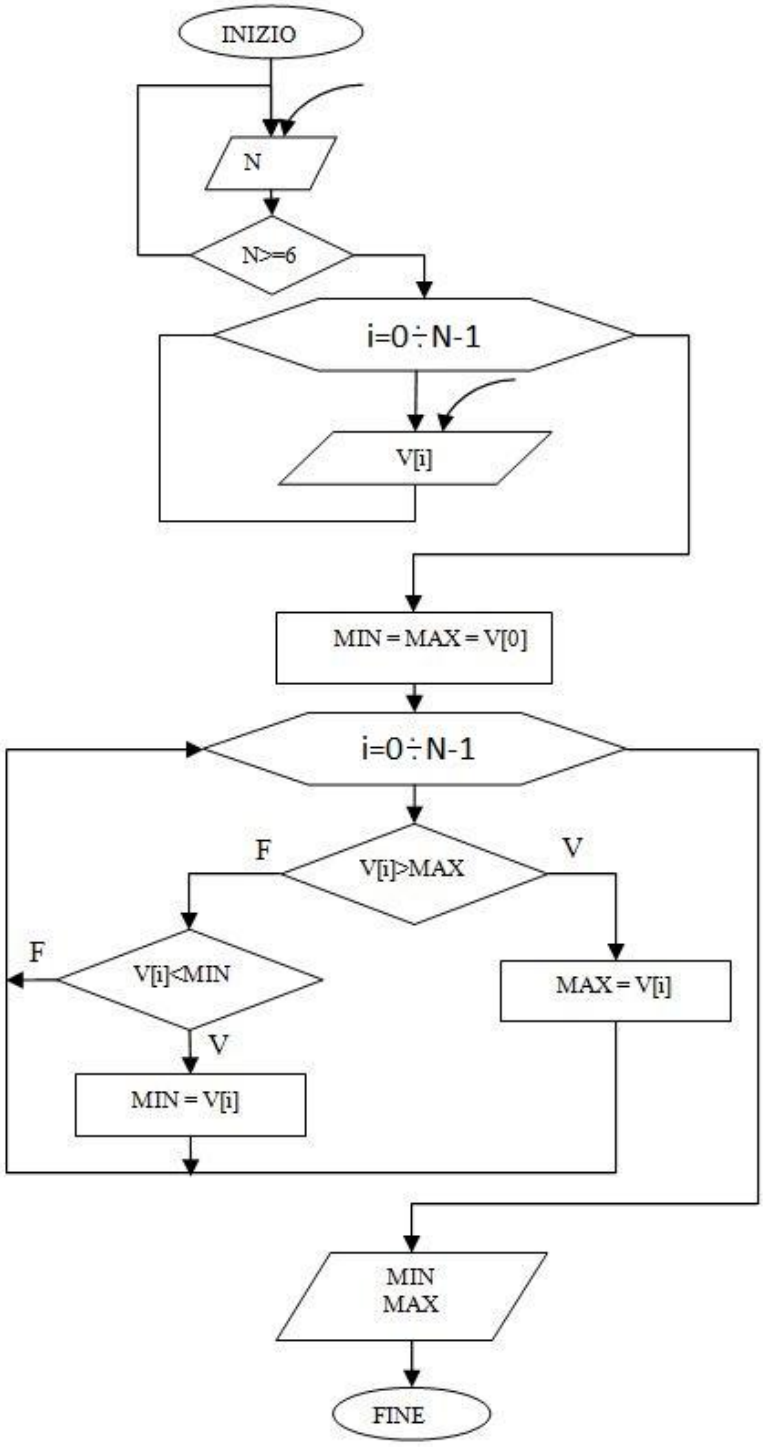
Questa istruzione assegna alla prima componente del vettore valore numerico 10.

Se si conosce il numero massimo di componenti del vettore che un programma in runtime (durante l'esecuzione) dovrà utilizzare, ma non si conosce il numero preciso, è possibile dichiarare un vettore "sovradimensionato" con lunghezza uguale al numero massimo di componenti da utilizzare. Si ha però lo

svantaggio di occupare più memoria di quanta in realtà ne sarebbe necessaria; per ovviare a quest'inconveniente, successivamente introdurremo il concetto di "allocazione dinamica della memoria".

5.1 Applicazione: diagramma di flusso con vettori

Il seguente algoritmo acquisisce una sequenza di valori numerici interi, li memorizza in un vettore e stampa a schermo il valore minimo e massimo.



6 Algoritmi di ordinamento

Gli algoritmi di ordinamento sono algoritmi che operano all'interno dei vettori.

Esistono quattro tipi di algoritmi di ordinamento: quick sort, bubble sort, merge sort, selection sort. In questo paragrafo tratteremo degli ultimi due.

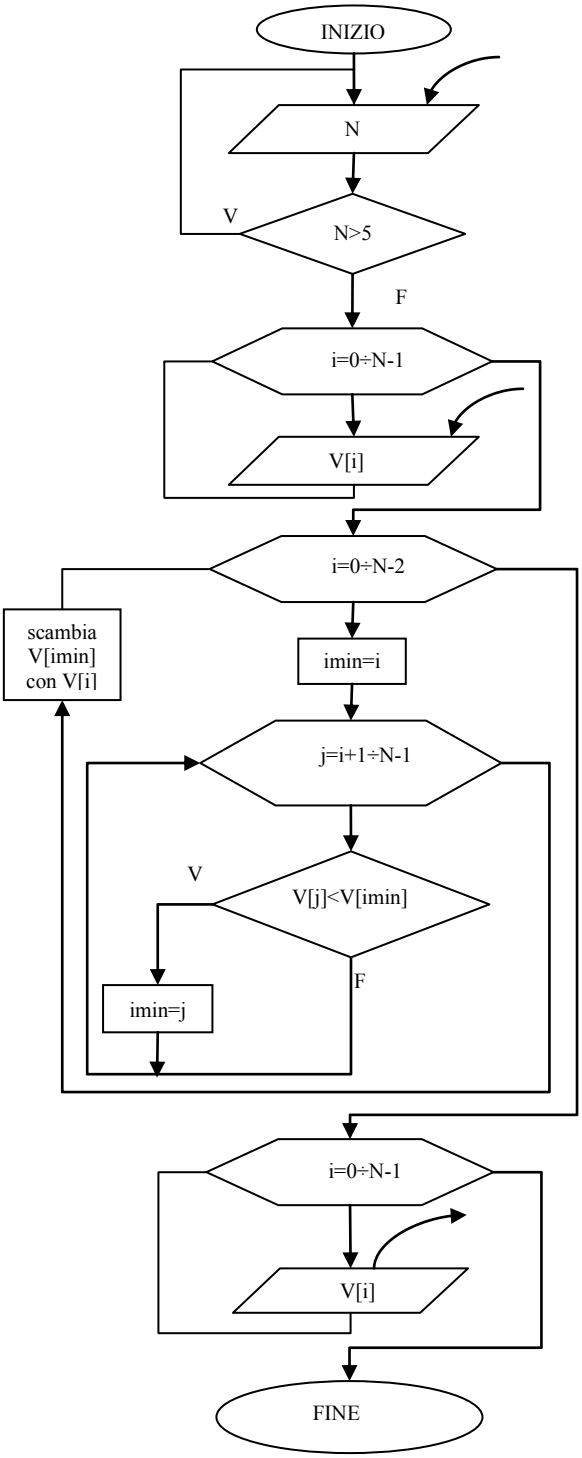
Il cuore di qualsiasi algoritmo di ordinamento è lo scambio, che consiste nello scambiare di posto due componenti di un vettore mediante una variabile `temp` ausiliaria

```
int A=5, B=3;
temp = A;
A = B;
B = temp;
```

dopo queste assegnazioni, avremo A=3 e B=5.

6.1 Selection Sort

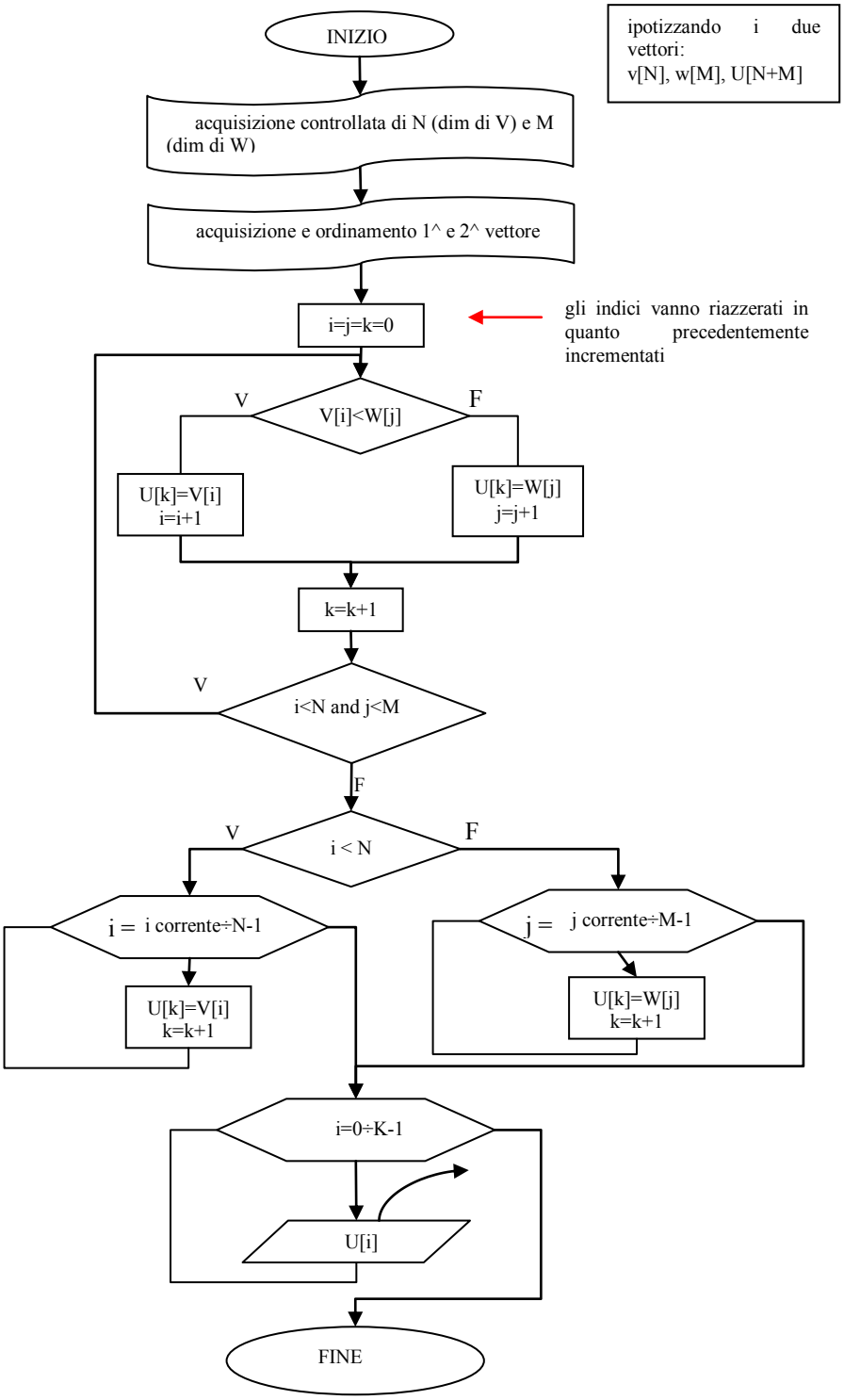
Il seguente algoritmo ordina gli elementi di un vettore in ordine crescente; la logica alla base di quest'algoritmo è quella di far lavorare due indici sul vettore: il primo, inserito nel primo ciclo for, fa riferimento ad una singola componente e il secondo, inserito nel secondo ciclo for, scorre, a partire da quella cui fa riferimento il primo indice, tutte le restanti componenti esclusa l'ultima, cercandone una minore di quella cui fa riferimento il primo indice.



6.2 Merge Sort

Partendo dal presupposto di avere due vettori già ordinati con lo stesso criterio e nelle ipotesi che la loro lunghezza non coincida, che ogni vettore non contenga elementi uguali e che non ci sono elementi comuni tra i due vettori, il seguente algoritmo crea un terzo vettore ordinato dato dalla fusione dei primi due.

La logica alla base dell'algoritmo è quella di far lavorare in un ciclo due indici ciascuno su un vettore, scorrendo i quali, si aggiunge al vettore fusione volta per volta l'elemento minore a cui fanno riferimento i due indici, quindi l'indice dell'elemento aggiunto scorre, finché uno dei due vettori non finisce e si aggiungono al vettore fusione i restanti elementi dell'altro vettore.



7 Concetto di “Matrice”

Una matrice è una estensione del concetto di vettore, in quanto il vettore è una matrice monodimensionale.

In una matrice bidimensionale i dati sono organizzati in righe e colonne, come se fossero inseriti in una tabella.

Come per le variabili e i vettori, si definisce il nome e il tipo; inoltre si specificano, tra parentesi quadre, il numero di componenti per ciascuna delle due dimensioni che la costituiscono:

```
int mat[4][4];
```

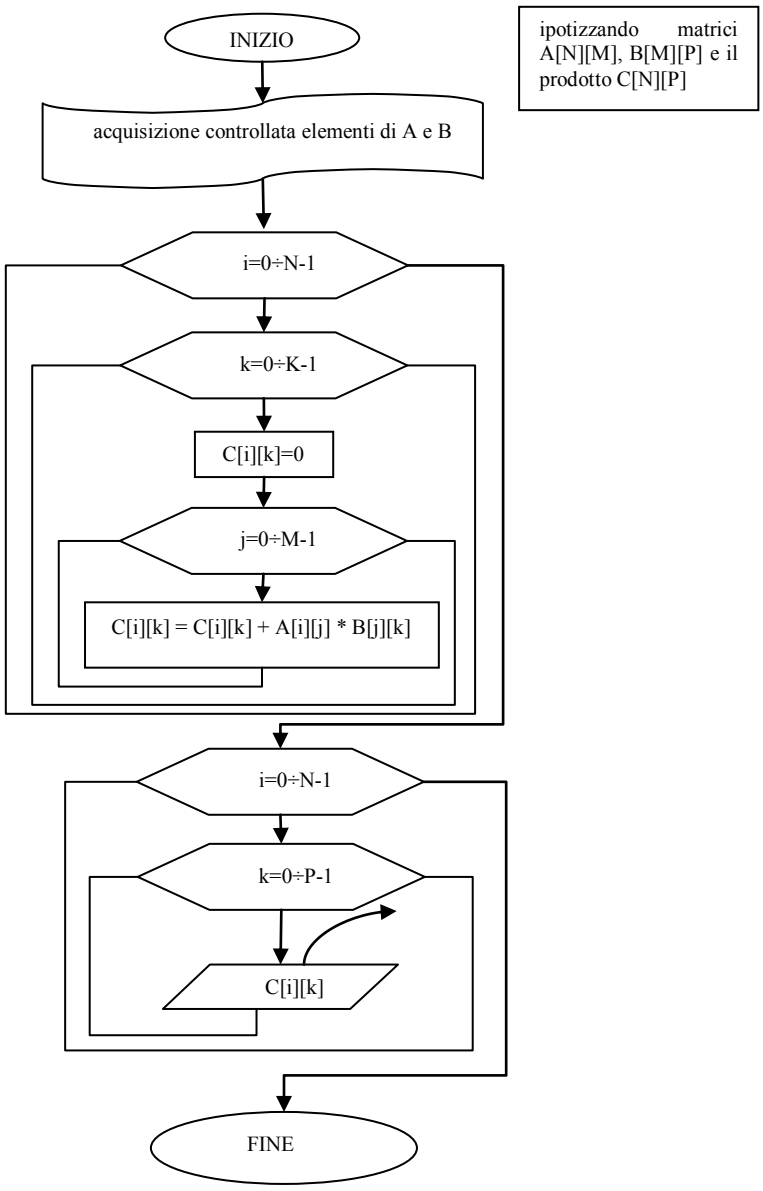
La matrice mat che abbiamo dichiarato contiene 4 righe e 4 colonne per un totale di 16 componenti; quest’ultima dichiarazione permette di riservare in RAM uno spazio contiguo di 16 word e per accedere a ciascuna di essi si utilizzano due indici: il primo specifica la riga, il secondo, la colonna:

```
mat [1][2]
```

rappresenta l’elemento presente nella seconda riga della terza colonna.

Si noti che il C gestisce le matrici per riga, dunque le componenti di una stessa riga sono sempre adiacenti in memoria, mentre quelle di una stessa colonna non lo sono.

7.1 Prodotto riga-colonna tra matrici



8 Ricerca sequenziale di un valore all'interno di un vettore

Uno tra i problemi comuni riguardanti i vettori è quello di determinare se un valore è presente all'interno di quest'ultimo.

Per far ciò, un primo algoritmo applicabile anche ad un vettore non ordinato è quello in cui, con un ciclo for, gli elementi del vettore vengono scanditi in modo sequenziale e vengono confrontati con il valore ricercato; nel momento in cui vi è esito positivo, viene restituito l'indice dell'elemento all'interno del vettore.

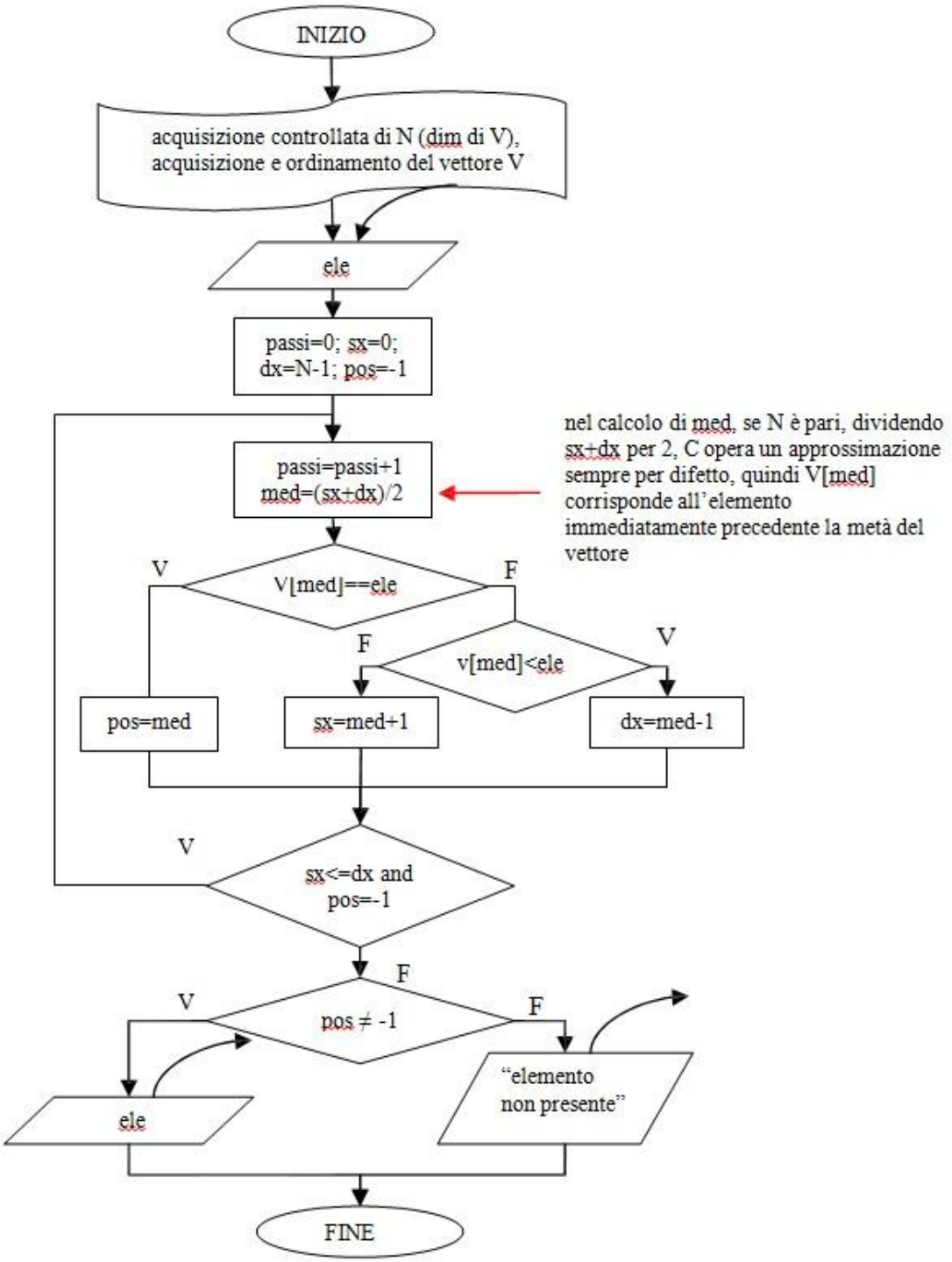
8.1 Ricerca dicotomica

Quando il vettore è ordinato, dopo il selection sort, la ricerca di un valore al suo interno può avvenire mediante criteri particolari volti alla diminuzione del numero di passi necessari, uno dei quali è detto "ricerca binaria/dicotomica": si confronta il valore da ricercare, memorizzato nella variabile `ele`, con l'elemento intermedio del vettore, il cui indice è dato dalla semisomma dell'indice del primo elemento del vettore (0), memorizzato nella variabile `sx`, e dell'indice dell'ultimo elemento (n-1) memorizzato

nella variabile `dx`. Poiché il vettore è ordinato si possono presentare tre casi:

1. l'elemento intermedio è quello ricercato, la ricerca finisce positivamente, si memorizza l'indice dell'elemento intermedio nella variabile `pos`
2. l'elemento intermedio è minore di quello ricercato, la ricerca continua nella parte del vettore compresa tra l'elemento immediatamente successivo all'elemento intermedio, il cui indice viene memorizzato in `sx`, e l'ultimo elemento, il cui indice rimane `dx`, quindi si confronta l'elemento intermedio della porzione di vettore con `ele`
3. l'elemento intermedio è maggiore di quello ricercato, la ricerca continua nella parte del vettore compresa tra l'elemento immediatamente precedente all'elemento intermedio, il cui indice viene memorizzato in `dx`, e il primo elemento, il cui indice rimane `sx`, quindi si confronta l'elemento intermedio della porzione di vettore con `ele`

Il seguente algoritmo, escluso il fortunato caso 1, prevede il continuo mutare dei valori assegnati a `sx` e `dx` fino al reperimento del valore desiderato o finché `sx` non sia minore di `dx`; è per questo che le due condizioni affinché la ricerca continui alla fine di ogni controllo del valore intermedio sono che `sx` deve rimanere minore di `dx` e che la variabile `pos` deve continuare ad assumere valore -1 assegnatole prima dell'inizio del ciclo di ricerca (il fatto che il suo valore rimane -1 alla fine di ogni controllo segnala che non è stato ancora trovato il valore cercato; per tale comportamento, essa è detta "variabile
flag/bandierina").


INIZIO
acquisizione controllata di N (dim di V), acquisizione e ordinamento del vettore V
ele
passi=0; sx=0; dx=N-1; pos=-1
passi=passi+1 med=(sx+dx)/2
nel calcolo di med, se N è pari, dividendo sx+dx per 2, C opera un approssimazione sempre per difetto, quindi V[med] corrisponde all'elemento immediatamente precedente la metà del vettore
V[med]==ele
V
F
v[med]<ele
F
V
pos=med
sx=med+1
dx=med-1
sx<=dx and pos=-1
V
F
pos ≠ -1
V
F
ele
"elemento non presente"
FINE

Appendice: Codice in linguaggio C

```
  //ALGORITMO SOMMA E PRODOTTO
#include <stdio.h>
void main()
{
      int nop, i=0, somma=0, pr=1, op;
      do
      {
            printf("nop=");
            scanf("%d",&nop);
      }
      while(nop<5);
      do
      {
            printf("op=");
            scanf("%d",&op);
            if(op%2==0)
                  somma=somma+op;
            else
                  pr=pr*op;
            i++;
      }
      while(i<nop);
      printf("%d\n%d",somma,pr);
}
//ALGORITMO DEL CALCOLO DELLA POTENZA
#include <stdio.h>
void main()
{
      int base, exp, vexp, i;
      float ris;
      printf("Inserisci la base:\t");
      scanf("%d",&base);
      printf("Inserisci l'esponente:\t");
      scanf("%d",&exp);
      ris=1;
      vexp=exp;
      if(vexp<0)
            vexp=-vexp;
      for(i=0;i<vexp;i++)
            ris=ris*base;
      if(exp<0)
            ris=1/ris;
      printf("Risultato\t%f\n", ris);
}

//ALGORITMO DELLA MEDIA
#include <stdio.h>
```

```
void main()
{
      int nad, addendo, somma=0, i;
      float media=0;
      printf("Inserisci il numero di addendi:\t");
      scanf("%d",&nad);
      for(i=0;i<nad;i++)
      {
            printf("Inserisci il %d^ addendo:\t",i+1);
            scanf("%d",&addendo);
            somma=somma+addendo;
      }
      media=(float)somma/nad;
      printf("La media risulta uguale a:\t%f\n",media);
}

//LOGARITMO IN BASE 2 PER ECCESSO
#include <stdio.h>
void main()
{
      int op, i=0, ris=1;
      do
      {
            printf("Immettere l'argomento del log\n");
            scanf("%d",&op);
      }
      while(op<=0);
      do
      {
            ris=ris*2;
            i=i+1;
      }
      while(ris<op);
      printf("log %d = %d\n",op,i);
}

//ALGORITMO DI MINIMO E MASSIMO
  #include <stdio.h>
  void main()
  {
      int N,i,min,max;
      int v[10];
      do
      {
            printf("Immetti larghezza vettore(max 10)\n");
            scanf("%d",&N);
      }
      while(N>10);
```

```
      printf("Inserisci SOLO il PRIMO Numero\n");
      scanf("%d",&v[0]);
      min=max=v[0];
      for(i=1;i<N;i++)
      {
            scanf("%d",&v[i]);
            if(v[i]<min)
                  min=v[i];
            else
                  if(v[i]>max)
                        max=v[i];
      }
      printf("Minimo=\t%d\nMassimo=\t%d\n",min,max);
  }

//SELECTION SORT
#include <stdio.h>
void main()
{
      int N,i,imin,j,temp;
      int v[10];
      do
      {
            printf("Immetti larghezza vettore (max 10)");
            scanf("%d",&N);
      }
      while(N>10);
      for(i=0;i<N;i++)
      {
            printf("Inserisci il %d^ numero ",i+1);
            scanf("%d",&v[i]);
      }
      for(i=0;i<N-1;i++)
      {
            imin=i;
            for(j=i+1;j<N;j++)
            {
                  if(v[j]<v[imin])
                  {
                        imin=j;
                  }
                  temp=v[i];
                  v[i]=v[imin];
                  v[imin]=temp;
            }
      }
      printf("Vettore ordinato:\n");
      for(i=0;i<N;i++)
```

```
	{
		printf("%d\t",v[i]);
		}
	printf("\n");
}
//MERGE SORT
#include <stdio.h>
void main()
{
	int v[10],u[15],z[25],N,M,i,j,k,imin,temp;
	do
	{
		printf("Immetti larghezza 1^vett(max 10)\n");
		scanf("%d",&N);
	}
	while(N>10);
	for(i=0;i<N;i++)
	{
		printf("Inserisci il %d^ numero ",i+1);
		scanf("%d",&v[i]);
	}
	do
	{
		printf("Immetti larghezza 2^vett(max 15)\n");
		scanf("%d",&M);
	}
	while(M>15);
	for(i=0;i<M;i++)
	{
		printf("Inserisci il %d^ numero ",i+1);
		scanf("%d",&u[i]);
	}
	for(i=0;i<N-1;i++)
	{
		imin=i;
		for(j=i;j<N;j++)
			if(v[j]<v[imin])
				imin=j;
		temp=v[i];
		v[i]=v[imin];
		v[imin]=temp;
	}
	for(i=0;i<M-1;i++)
	{
		imin=i;
		for(j=i;j<M;j++)
			if(u[j]<u[imin])
				imin=j;
```

```
            temp=u[i];
            u[i]=u[imin];
            u[imin]=temp;
      }
      i=j=k=0;
      do
      {
            if(v[i]<u[j])
            {
                  z[k]=v[i];
                  i=i+1;
            }
            else
            {
                  z[k]=u[j];
                  j=j+1;
            }
            k=k+1;
      }
      while(i<N && j<M);
      if(i<N)
            for(;i<N;i++)
            {
                  z[k]=v[i];
                  k=k+1;
            }
      else
            for(;j<M;j++)
            {
                  z[k]=u[j];
                  k=k+1;
            }
      printf("Vettore fusione:\n");
      for(k=0;k<M+N;k++)
            printf("%d\t",z[k]);
      printf("\n");
}

//PRODOTTO TRA MATRICI
#include <stdio.h>
void main()
{
      int L,M,N,A[10][10],B[10][10],C[10][10],i,j,k;
      do
      {
            printf("Inserire il nm righe della 1^ matrice
(max 10)\n");
            scanf("%d",&L);
```

```
	}
	while(L>10);
	do
	{
		printf("Inserire il nm colonne della 1^ matrice
= nm righe della 2^ matrice (max 10)\n");
		scanf("%d",&M);
	}
	while(M>10);
	do
	{
		printf("Inserire il nm colonne della 2^ matrice
(max 10)\n");
		scanf("%d",&N);
	}
	while(N>10);
	for(i=0;i<L;i++)
	{
		for(j=0;j<M;j++)
		{
			printf("Inserisci A(%d,%d)\t",i+1,j+1);
			scanf("%d",&A[i][j]);
		}
	}
	for(i=0;i<M;i++)
	{
		for(j=0;j<N;j++)
		{
			printf("Inserisci B(%d,%d)\t",i+1,j+1);
			scanf("%d",&B[i][j]);
		}
	}
/*Prodotto tra le due matrici:*/
	for(i=0;i<L;i++)
	{
		for(k=0;k<N;k++)
		{
			C[i][k]=0;
			for(j=0;j<M;j++)
				C[i][k]=C[i][k]+A[i][j]*B[j][k];
		}
	}
	printf("Matrice prodotto:\n");
	for(i=0;i<L;i++)
	{
		for(k=0;k<N;k++)
			printf("%5d",C[i][k]);
		printf("\n");
```

```
      }
}

  //ALGORITMO DICOTOMICO
        #include <stdio.h>
        #include <conio.h>
        void main()
        {
           int
      N,v[10],ele,passi,sx,dx,pos,med,i,j,imin,temp;
           do
           {
                 do
                 {
                       printf("Nm  componenti  da  inserire
      (max 10): ");
                       scanf("%d",&N);
                 }
                 while(N>10);
                 for(i=0;i<N;i++)
                 {
                       v[i] = i;
                 }
                 /*ricerca dicotomica*/
                 printf("\nInserisci il valore da cercare:
      ");
                 scanf("%d",&ele);
                 passi=0;sx=0;dx=N-1;pos=-1;
                 do
                 {
                       med=(dx+sx)/2; passi=passi+1;
                       if(v[med]==ele)
                             pos=med;
                       else if(v[med]<ele)
                             sx=med+1;
                       else
                             dx=med-1;
                 }
                 while(sx<=dx && pos==-1);
                 if(pos!=-1)
                       printf("Nm  passi:%d;posizione
      elemento:%d",passi,pos+1);
                 else
                       printf("L'elemento     non   e'
      presente nel vettore");
```

```
            printf("\nRieseguire     il     programma?
(s/n)\n");
      }
      while(getch()!='n');
  }
```

Ringraziamenti. Il presente capitolo è stato scritto anche grazie al prezioso contributo degli studenti Donato Mancuso, Giuseppe Ragno, Flavio Palmieri, Pasquale Bonasia.

Riferimenti

1. Bevilacqua, V.: Dispense Linguaggio C In: http://www.vitoantoniobevilacqua.it
2. http://it.wikipedia.org/wiki/Diagramma_a_blocchi
3. http://it.wikipedia.org/wiki/Array