Vettori e matrici

Problema. Dato un testo, si determini la frequenza assoluta (il numero di comparizioni) di ogni carattere.

- Possibile soluzione
  - *inizializza le variabili di conteggio a zero*
  - *finché ci sono linee di testo da trattare*
    finché ci sono caratteri sulla riga
    - *considera il carattere corrente e incrementa la variabile corrispondente*
    - *scarica la riga*
  - *stampa i valori di tutte le variabili di conteggio*
- Inizializzazione e stampa comprendono 2*N istruzioni uguali, che differiscono solo per la variabile
- Due istruzioni ripetitive svolgerebbero lo stesso compito, se fosse possibile raccogliere le variabili sotto un nome unico e trattare ciascuna variabile in una ripetizione

Vettori e matrici

- Matrice
  - Corrispondenza tra un insieme di *indici* e un insieme di valori di un dominio DV
- Definizione del *tipo di dato matrice*
  - $indice$: un tipo numerabile ordinale, prodotto cartesiano di n tipi semplici numerabili
  - DV: un insieme di valori di tipo qualunque
  - Operazioni
    - $accedi: matrice \times indice \rightarrow V \in DV$ Data una coppia $(M, I) \in matrice \times indice$ restituisce il valore $V \in DV$ corrispondente a (M, I)
    - $memorizza: matrice \times indice \times DV \rightarrow matrice$ Data una terna $(M, I, V) \in matrice \times indice \times DV$ produce una matrice M' che ha il valore V nella posizione I, ed è identica a M altrove
  - Per $n = 1$ la matrice è chiamata *vettore*

Vettori e matrici

- Rivisitiamo il problema iniziale

  Problema. Dato un testo, si determini la frequenza assoluta (il numero di comparizioni) di ogni carattere alfabetico

  - *per ogni indice I*
    - *poni la variabile di indice I a zero*
  - *finché ci sono righe da trattare*
    - *finché ci sono caratteri sulla riga*
      - *leggi un carattere in C*
      - *se C è un carattere alfabetico minuscolo allora*
        - *determina l'indice corrispondente al valore di C*
        - *incrementa il valore corrispondente di uno*
  - *per ogni indice I*
    - *visualizza la variabile di indice I*

Il costruttore array [ ]

- Vettori e matrici si dichiarano con il *costruttore array* [ ]
- Sintassi

$$\langle \text{var-array} \rangle \longrightarrow \langle \text{tipo} \rangle \, \langle \text{identif} \rangle \, \langle \text{array} \rangle \; ;$$

$$\langle \text{tipo-array} \rangle \longrightarrow \texttt{typedef} \, \langle \text{tipo} \rangle \, \langle \text{identif} \rangle \, \langle \text{array} \rangle \; ;$$

$$\langle \text{array} \rangle \longrightarrow \langle \text{costr-array} \rangle \mid \langle \text{costr-array} \rangle \, \langle \text{array} \rangle$$

$$\langle \text{costr-array} \rangle \longrightarrow [\, \langle \text{espress-costante-intera} \rangle \,]$$

  - $\langle \text{espress-costante-intera} \rangle$ stabilisce il numero di elementi del vettore ed è chiamato *dimensione* del vettore
  - Se n è la dimensione, l'indice può assumere valori tra 0 e $n-1$
- Operazioni sul tipo di dato array

$memorizza(M, I, V)$	`M[I]=V`
$accedi(M, I)$ e memorizza in VV	`VV=M[I]`

Il costruttore array [ ]

- L'inizializzazione si realizza elencando i valori tra graffe

```
int V[5] = {3,1,2,4,6};
int V[5] = {3,1};           /*primi due elem. inizializzati*/
int V[5] = {3,1,2,4,6,8}; /*non corretta,troppi elementi*/
int V[]  = {3,1,2};         /*array di tre elementi*/
```

- Non si può manipolare un array nel suo insieme; in particolare la operazione di copia deve essere programmata esplicitamente:

```
for (i=0; i<dimensione; i++)
  V2[i] = V1[i];
```

Il costruttore array [ ]

Esempio. Acquisire da input un vettore di N interi e visualizzare il massimo

- *considerare il primo elemento come massimo*
- *per le posizioni da 1 a N-1*
  - *se l'elemento corrente è maggiore del massimo*
    - *allora il massimo è l'elemento corrente*

```
#include <stdio.h>
#define N 10
main() {
  int V[N];
  int Max;
  int i;
  for(i=0; i<N; i++) {
    printf("Digitare l'elemento %d: ",i);
    scanf("%d",&V[i]);
  }
  Max=V[0];

  for(i=1; i<N; i++)
    if (V[i]>Max)

      Max = V[i];

  printf("Il valore massimo è %d",Max);
}
```

Il costruttore array [ ]

- La dichiarazione di una matrice si ottiene applicando 2 volte il costruttore array

  ```
  int M[2][4]
  ```

  dichiara una matrice di 2 righe e 4 colonne
- Una matrice è considerata come un array di array—un array in cui ogni elemento è un array a sua volta:

  ```
  typedef int TV[4];
  TV M[2]
  ```
- Memorizzazione e accesso sono analoghi al vettore

$memorizza(M,(I,J),V)$	`M[I][J]=V`
$accedi(M,(I,J))$ e memorizza in VV	`VV=M[I][J]`

- Inizializzazione

```
int M[2][4] = { {1,2,3,4},
                {5,6,7,8} };
int M[2][4] = { 1,2,3,4,5,6,7,8 }; /*equivalente*/
```

Il costruttore array [ ]

Esempio. Date le matrici

```
int M1[2][3] = {1,2,3,4,5,6};
int M2[3][4] = {12,11,10,9,8,7,6,5,4,3,2,1};
```

calcolare e visualizzare il loro prodotto.

- *per ogni riga di M1*
  - *per ogni colonna di M2*
    - *inizializza M1XM2 a zero*
    - *per ogni colonna di M1*
      - *accumula comb. lin.*
- *visualizza matrice prodotto*

```
#include <stdio.h>
#define NR1 2
#define NC1 3
#define NR2 3
#define NC2 4
main() {
  int M1[NR1][NC1] = {1,2,3,4,5,6};
  int M2[NR2][NC2] = {12,11,10,9,8,7,6,5,4,3,2,1};
  int M1XM2[NR1][NC2];
  int i,j,k;
  for(i=0; i<NR1; i++)
    for(j=0; j<NC2; j++) {
      M1XM2[i][j]=0;
      for(k=0; k<NC1; k++)
        M1XM2[i][j]=M1XM2[i][j]+M1[i][k]*M2[k][j];
    }
  for(i=0; i<NR1; i++) {
    for(j=0; j<NC2; j++)
      printf("%6d",M1XM2[i][j]); printf("\n");
  }
}
```

Il costruttore array [ ]

- Rivisitiamo il problema del conteggio dei caratteri

  Problema. Dato un testo, si determini la frequenza assoluta di ogni carattere alfabetico.

```
- inizializza il vettore a zero

- finché ci sono caratteri
  su una riga
  - leggi carattere in Ch
  - se Ch è alfabetico minuscolo
    - incrementa il valore
      Conteggio[Ch-'a']

- visualizza vettore
```

```
#include <stdio.h>
#define NL 'z'-'a'+1
main() {
  int Conteggio[NL];
  char Ch;
  int i;
  for (Ch=0; Ch<NL; Ch++)
    Conteggio[Ch]=0;
  do {

    scanf("%c",&Ch);
    if ((Ch >= 'a') && (Ch <= 'z'))
      Conteggio[Ch-'a']=Conteggio[Ch-'a']+1;
  }
  while (Ch!='\n');
  for(Ch=0; Ch<NL; Ch++)
    if (Conteggio[Ch]!=0)
      printf("%c = %d\n",Ch+'a',Conteggio[Ch]);
}
```

Il costruttore array [ ]

Problema. Dato un vettore V di N elementi, stabilire se l'elemento D è memorizzato in una delle posizioni del vettore.

- *inizializza la pos. corrente a 0*
- *finché la pos. corrente non supera N e l'elemento in pos. corrente è diverso da D*
  - *incrementa pos. corrente*
- *se l'elemento corrente è uguale a D*
  - *visualizza messaggio di successo*

  altrimenti
  - *visualizza messaggio di fallimento*

```
#include <stdio.h>
#define N 5
main() {
  int V[N] = {1,6,7,3,21};
  int D;
  int i;

  printf("Digitare l'elemento da cercare: ");
  scanf("%d",&D);
  /* Ricerca lineare */
  i=0;
  while((i<N) && (V[i]!=D))


    i++;
  if (V[i]==D)
    printf("%d è nel vettore.\n",D);
  else
    printf("%d non è nel vettore.\n",D);
}
```

Il costruttore array [ ]

- Soluzione efficiente: ricerca *dicotomica*

```
variabili:
- vettore inizializzato
- elemento da cercare
- estremi di ricerca, pos. media
- esito della ricerca
- indice per iterazioni, iniz. a 0
algoritmo:

- acquisisci elemento da cercare


- iniz. estremi

- finché Inizio <= Fine e Trovato è 0
  - calcola la pos. centrale C
- se l'elemento di pos. C è uguale a D
  - poni Trovato uguale a 1
  altrimenti
  - se l'elemento di pos. C è  < di D
    - poni Inizio uguale a C+1
    altrimenti
    - poni Fine uguale a C-1

- se Trovato è 1
  - visualizza messaggio di successo
  altrimenti
  - visualizza messaggio di fallimento
```

```
#include <stdio.h>
#define N 5
main() {
  int V[N] = {1,6,7,3,21};
  int D;
  int Inizio,Fine,C;
  int Trovato=0;
  int i;

  printf("Digitare l'elemento da cercare: ");
  scanf("%d",&D);
  /* Ricerca dicotomica */
  Inizio=0;
  Fine=N-1;
  while((Inizio<=Fine) && (Trovato==0)) {
    C=(Inizio+Fine)/2;
    if (D==V[C])
      Trovato=1;
    else
      if (D>V[C])
        Inizio=C+1;
      else
        Fine=C-1;
  }
  if (Trovato==1)
    printf("%d è nel vettore.\n",D);
  else
    printf("%d non è nel vettore.\n",D);
}
```

Stringhe di caratteri

- Non esiste in C un tipo predefinito per rappresentare stringhe; i caratteri di una stringa si memorizzano in una variabile di tipo array di `char`, utilizzata secondo appropriate regole
- Una variabile `S` è una *stringa di lunghezza massima* N se è dichiarata come vettore di caratteri di lunghezza $N+1$:

```
char S[N+1];
```

  ed è utilizzata nel rispetto delle seguenti regole:

  - Esiste una posizione L $(0 \leq L \leq N)$ in cui è memorizzato il carattere speciale `'\0'`, che ha codice ASCII pari a zero, chiamato *terminatore* della stringa
  - Le posizioni da 0 a $L-1$ sono occupate da tutti e soli gli L caratteri della stringa
  - Le posizioni da $L+1$ a N hanno contenuto indefinito

Stringhe di caratteri

- Per la stringa di lunghezza zero (stringa *vuota*) si ha `S[0]='\0'`
- Una costante di tipo stringa di N caratteri si rappresenta con la successione dei suoi caratteri racchiusa tra una coppia di ":

  `"Linguaggio C"`

  ed è un vettore di $N+1$ caratteri

L	i	n	g	u	a	g	g	i	o		C	\0

- L'inizializzazione si realizza seguendo la sintassi dell'inizializzazione di un array, oppure con una costante stringa (anche in assenza di lunghezza dell'array)

```
char S1[8]={'s','t','r','i','n','g','a','\0'};
char S2[8]="stringa";
char S3[] ="stringa";
```

Stringhe di caratteri

- Ogni frammento di codice che tratta stringhe deve essere progettato in accordo alle regole di memorizzazione
  - Calcolo della lunghezza di una stringa

```
char S[] = "stringa";
int lunghezza = 0;
while(S[lunghezza]!='\0')
  lunghezza++;
printf("La stringa \"%s\" è lunga %d\n",S,lunghezza);
```

  - Copia di una stringa in un'altra

```
char S1[] = "stringa";
char S2[10];
int i;
i=0;
while(S1[i]!='\0') {
  S2[i]=S1[i];
  i++;
}
S2[i]='\0';
```

Record

- Record
  - Corrispondenza tra un insieme di *etichette* e un insieme di valori, in modo che ad ogni etichetta e_i corrisponda un valore di uno specifico dominio DV_i
- Definizione del *tipo di dato record*
  - $etichette = \{e_1, \ldots, e_n\}$: insieme finito di simboli, di cardinalità n
  - $DV = \{DV_1, \ldots, DV_n\}$: un insieme di insiemi di valori di tipo qualunque
  - Operazioni
    - $accedi$: $record \times etichette \rightarrow \bigcup DV$ Data una coppia $(R, e_i) \in record \times etichette$ restituisce il valore $v \in DV_i$ corrispondente a (R, e_i)
    - $memorizza$: $record \times etichette \times \bigcup DV \rightarrow record$ Data una terna $(R, e_i, v) \in record \times etichette \times \bigcup DV$ produce un record R' che ha il valore v nella etichetta e_i, ed è identico a R altrove

Il costruttore `struct`

- In C variabili di tipo record si dichiarano con il costruttore `struct`
- Sintassi semplificata

$$\langle \text{var-record} \rangle \longrightarrow \langle \text{costr-struct} \rangle \langle \text{identif} \rangle \texttt{ ;}$$

$$\langle \text{tipo-record} \rangle \longrightarrow \texttt{typedef } \langle \text{costr-struct} \rangle \langle \text{identif} \rangle \texttt{ ;}$$

$$\langle \text{costr-struct} \rangle \longrightarrow \texttt{struct\{ } \langle \text{seq-campi} \rangle \texttt{ \}}$$

$$\langle \text{seq-campi} \rangle \longrightarrow \langle \text{seq-campi-omog} \rangle \mid \langle \text{seq-campi-omog} \rangle \texttt{ ; } \langle \text{seq-campi} \rangle$$

$$\langle \text{seq-campi-omog} \rangle \longrightarrow \langle \text{tipo} \rangle \langle \text{seq-nomi-campi} \rangle$$

$$\langle \text{seq-nomi-campi} \rangle \longrightarrow \langle \text{identif} \rangle \mid \langle \text{identif} \rangle \texttt{ , } \langle \text{seq-nomi-campi} \rangle$$

- Esempi

```
struct {
         int Giorno;
         int Mese;
         int Anno;
       } Data;
```

```
typedef struct {
                  float X,Y;
                } PuntoNelPiano;
```

Il costruttore `struct`

- Operazioni sul tipo di dato record

$memorizza(R, e, V)$	`R.e=V`
$accedi(R, e)$ e memorizza in VV	`VV=R.e`

- L'inizializzazione può essere effettuata nella definizione, specificando i valori tra graffe, nell'ordine in cui le corrispondenti etichette sono dichiarate

```
struct {
          int Giorno, Mese, Anno;
} Data = {29,6,1942};
```

- È consentito l'accesso alla variabile nel suo insieme, oltre che etichetta per etichetta

```
typedef struct {
  float X,Y;
} PtoNelPiano;
```

```
PtoNelPiano A,B;
PtoNelPiano PtoMax = {3.14,3.141};
A.X=3.14;
B.X=3.141;
A=PtoMax;
```

Il costruttore `struct`

Esempio. Calcolo del punto medio di un segmento

```
#include <stdio.h>
typedef struct {
  float X,Y;
} PtoNelPiano;

PtoNelPiano PtoMedio(PtoNelPiano P1, PtoNelPiano P2) {
  PtoNelPiano P;
  P.X=(P1.X+P2.X)/2;
  P.Y=(P1.Y+P2.Y)/2;
  return P;
}
main(){
  PtoNelPiano A,B,PtoMedioAB;
  printf("Coordinate del primo punto: ");
  scanf("%f%f",&A.X,&A.Y);
  printf("Coordinate del secondo punto: ");
  scanf("%f%f",&B.X,&B.Y);
  PtoMedioAB=PtoMedio(A,B);
  printf("Punto medio: %f,%f\n", PtoMedioAB.X,PtoMedioAB.Y);
}
```

Il tipo puntatore

- I tipi finora visti danno luogo a dichiarazioni di variabili cosiddette *statiche* per le quali cioè
  - Il nome della variabile viene fatto corrispondere in fase di compilazione con un indirizzo di memoria
  - Il codice oggetto compilato contiene un riferimento a tale indirizzo dove il sorgente contiene un riferimento al nome
  - L'indirizzo è quello della prima di una serie di celle di memoria *allocate*, cioè riservate a quella variabile
- È possibile creare variabili *dinamiche*, chiamando in fase di *esecuzione* una opportuna procedura che
  - alloca lo spazio in memoria per la variabile
  - restituisce l'indirizzo di quello spazio
- Per utilizzare la variabile dinamica piú volte, l'indirizzo restituito deve essere memorizzato in una variabile di tipo particolare, il tipo *puntatore*

Il costruttore *

- Variabili di tipo puntatore si dichiarano con il costruttore `*` ;
- Sintassi semplificata

$$\langle \text{var-puntat} \rangle \longrightarrow \langle \text{costr-puntat} \rangle \langle \text{identif} \rangle \ ;$$

$$\langle \text{tipo-puntat} \rangle \longrightarrow \texttt{typedef} \ \langle \text{costr-puntat} \rangle \langle \text{identif} \rangle \ ;$$

$$\langle \text{costr-puntat} \rangle \longrightarrow \langle \text{tipo} \rangle \ \texttt{*}$$

$$\langle \text{oper-dereferenziazione} \rangle \longrightarrow \texttt{*} \ \langle \text{nome-variabile} \rangle$$

- Esempi

```
int *P;
struct {
  int Giorno,
      Mese,
      Anno; } *PData;
```

```
typedef int *TipoPI;
TipoPI P1,P2;
```

Il costruttore *

- Con lo stesso simbolo si denota l'operatore di *dereferenziazione*, che
  - applicato a una variabile di tipo puntatore rappresenta la variabile dinamica il cui indirizzo è memorizzato nel puntatore
- Sintassi semplificata

$$\langle \text{oper-dereferenziazione} \rangle \longrightarrow * \langle \text{nome-variabile} \rangle$$

Allocazione/deallocazione dinamica

- La funzione di allocazione dichiarata in `stdlib.h` con intestazione

```
void *malloc(int NumByte)
```

  - alloca `NumByte` byte di memoria
  - restituisce un puntatore alla memoria allocata

  - Il calcolo del numero di byte sufficiente a contenere un valore di un tipo assegnato non è sempre agevole; perciò nella pratica si fa uso della funzione `sizeof` per calcolare il numero di byte
  - Il tipo della funzione è un puntatore a `void`; per assegnare il valore della funzione a un puntatore a un tipo qualunque, si fa uso di un type cast

```
int *P;
P = (int *) malloc(sizeof(int));
```

Allocazione/deallocazione dinamica

- Quando una variabile dinamica non è piú necessaria, occorre **sempre** *deallocarla*, cioè liberare la memoria ad essa riservata (da `malloc`)
- La funzione di deallocazione dichiarata in `stdlib.h` con intestazione

```
void free(void *P)
```

  - dealloca tutti i byte di memoria allocati alla variabile dinamica puntata dall'argomento `P`
  - lascia indefinito il valore di `P`
  - Se l'indirizzo di una variabile dinamica viene perso, **non è piú possibile deallocare** la memoria

Allocazione/deallocazione dinamica

Esempio. Allocazione e deallocazione corretta di una variabile dinamica di tipo `int`

```
#include <stdio.h>
#include <stdlib.h>
main() {
  typedef int TI;  /* dichiara tipo per la var. dinamica */
  typedef TI *PTI; /* dichiara il tipo puntatore */
  PTI P;           /* dichiara la var. statica puntatore */
  P=(int *)malloc(sizeof(int)); /* crea la var. dinamica */
  *P=3;                         /* assegna alla var. dinamica */
  printf("%d\n",*P);            /* accede  alla var. dinamica */
  free(P);                      /* rimuove la var.  dinamica */
}
```

Operatore &

- L'operatore di *estrazione di indirizzo* `&`, applicato a una varibile statica, ne restituisce l'indirizzo
- Sintassi

$$\langle \text{oper-estrazione-indirizzo} \rangle \longrightarrow \texttt{\&}\, \langle \text{nome-variabile} \rangle$$

- L'indirizzo può essere memorizzato in una variabile puntatore

```
int *P
int X=0, Y;
P = &Y;
Y = *P;
*P = Y+1;
```

- L'applicazione piú significativa è però il passaggio dei parametri **per riferimento** nella chiamata di funzione

Operatore &

- Il passaggio dei parametri in C per riferimento si realizza utilizzando parametri formali di tipo puntatore e passando, come parametri attuali, gli **indirizzi** delle variabili, estratti con l'operatore &

Esempio. Programmare una funzione `Scambia` che scambia il contenuto di due variabili in tipo intero **passate come parametri**

```
void Scambia(int *X, int *Y){
  int Temp;

  Temp = *X;
  *X = *Y;
  *Y = Temp;
}
```

```
main() {
  void Scambia(int *X, int *Y);
  int A=39,B=88;
  Scambia(&A,&B);
  printf("%d %d\n",A,B);
}
```

Operazioni su puntatori

- In generale, somma, sottrazione, incremento, decremento e operatori di confronto sono applicabili alle variabili di tipo puntatore, purché dello stesso tipo
  - Le effettive operazioni permesse dipendono dal compilatore
- Le operazioni aritmetiche considerano il *valore logico* del puntatore, non fisico
  - ad esempio il frammento di codice a lato incrementa `P` di 4 unità e incrementa `Q` di 8 unità (GNU C i386)

```
int *P;
double *Q;
P++;
Q++;
```

  - generalmente tali operazioni sono significative solo quando applicate a puntatori a elementi di un array

Operazioni su puntatori

- Ogni variabile `V` di tipo `array` ha un valore pari a `&V[0]`, l'indirizzo del suo primo elemento — cioè `V` è un **puntatore costante**
- Quindi si possono effettuare operazioni aritmetiche
  - `*(V+i)` equivale a `V[i]`
  - `&V[i]` equivale a `(V+i)`
  - `*(p+i)` equivale a `p[i]`

Operazioni su puntatori

- È essenziale, nell'utilizzo dell'aritmetica dei puntatori, prestare la **massima attenzione** all'intervallo di variazione degli indirizzi calcolati, che devono ricadere sempre nello spazio allocato a una variabile
  - risultati errati
  - interruzione forzata del programma da parte del sistema operativo con un errore a tempo di esecuzione

file sorgente erresec.c

```
main() {
  char *p, V[]="stringa";
  p = V;
  printf("%s\n",p);
  printf("%s\n",p+200);
  printf("%s\n",p+5000);
}
```

compilazione/esecuzione a terminale

```
> cc erresec.c -o erresec
> erresec
stringa
ÿÿ¿_ÿÿ¿®ÿÿ¿Âÿÿ¿Îÿÿ¿
Segmentation fault
>
```