POLITECNICO DI MILANO

Array e Matrici

Informatica B

Daniele Loiacono

Array

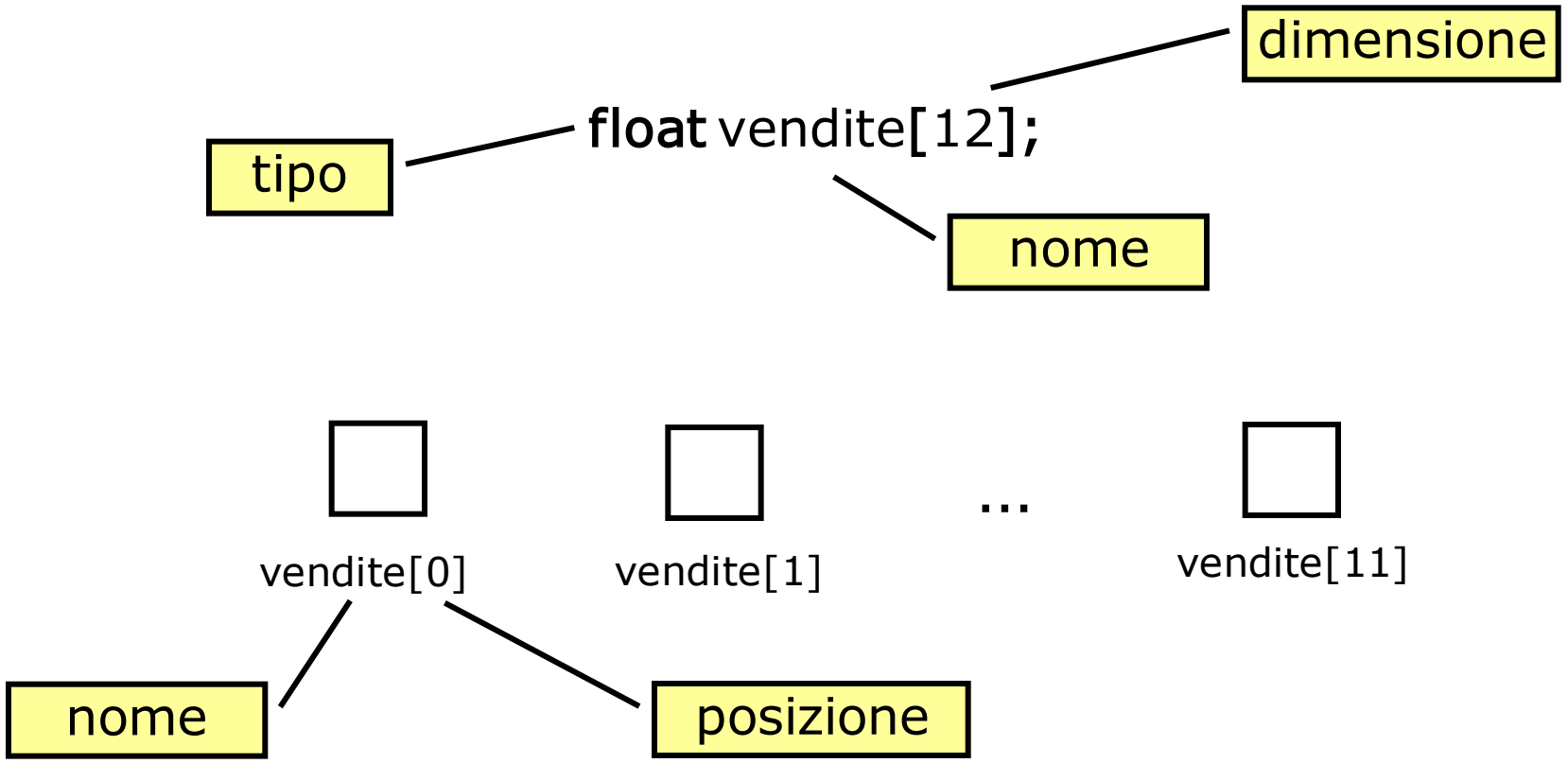
Perchè usare gli array?

```
float venditeGennaio, venditeFebbraio, venditeMarzo,
venditeAprile, venditeMaggio, venditeGiugno, venditeLuglio,
venditeAgosto, venditeSettembre, venditeOttobre,
venditeNovembre, venditeDicembre;
float totale = venditeGennaio + venditeFebbraio +
               venditeMarzo + venditeAprile + venditeMaggio +
               venditeGiugno + venditeLuglio + venditeAgosto +
               venditeSettembre + venditeOttobre +
               venditeNovembre + venditeDicembre;
```

venditeGennaio venditeFebbraio venditeDicembre

Gli array

- Gli array offrono la possibilità di rappresentare in maniera **compatta** una **collezione** di variabili



Elementi di un array

- L'accesso a un elemento avviene attraverso il nome dell'array seguito dalla posizione fra parentesi quadre (**subscript**)
- La posizione deve essere di **tipo intero** (o compatibile) e **parte da 0** fino alla dimensione dell'array meno 1
- Ogni singolo elemento dell'array è del tutto analogo ad una variabile di tipo semplice
- Esempi

```
float vendite[12];
float totale = 0;
int i;
for (i=0; i<12; i++)
{
    totale = totale + vendite[i];
}

scanf("%f",&vendite[0]);
printf("Le vendite di Dicembre sono state: %f",vendite[11]);
printf("L'incremento e' pari a %f", (vendite[1]-vendite[0])/vendite[0]);
```

Array: inizializzazione

- È possibile inizializzare un array in fase di dichiarazione, specificandone tutti gli elementi fra parentesi graffe e separati da virgole:

tipo nome[**N**] = {val1,…,valN};

- Esempio:

```
float prezzo[4] = {13.4,11.10,20.9,30.4};
```

Array: lettura e scrittura

- La lettura e scrittura degli array **avviene un elemento alla volta**
- Per questo scopo è molto conveniente ricorrere all'uso del **ciclo for**
- La lettura e la scrittura di ogni singolo elemento dell'array è del tutto analoga a quella di una variabile di tipo semplice
- Esempio:

```
int i;
float prezzo[4];
for (i=0; i<4; i++)
{
    scanf("%f",&prezzo[i]);
}
for (i=0; i<4; i++)
{
    printf("prezzo[%d] = %f",i,prezzo[i]);
}
```

Array: esempio

- Scrivere un programma che legga da terminale le vendite degli ultimi 6 mesi, le memorizzi in un array e le rappresenti come un istogramma

```
#include<stdio.h>

int main()
{
   float vendite[6];
   int i,j;
   for (i=0; i<6; i++) {
      printf("Vendite di %d mese/i fa: ",i+1);
      scanf("%f",&vendite[i]);
   }
   for (i=5; i>=0; i--) {
      if (i>0)  printf ("%d mesi fa: ",i+1);
      else  printf ("1 mese fa: ");
      for (j=1;j<=vendite[i];j++)
         printf ("*");
      printf("\n");
   }
   system("PAUSE");
   return 0;
}
```

Array: esempio

- Scrivere un programma che legga da terminale le vendite degli ultimi 6 mesi, le memorizzi in un array e le rappresenti come un istogramma

```
#include<stdio.h>

int main()
{
   float vendite[6];
   int i,j;
   for (i=0; i<6; i++
      printf("Vendite
      scanf("%f",&ver
   }
   for (i=5; i>=0; i-
      if (i>0)  printf
      else  printf ("1
      for (j=1;j<=ver
         printf ("*");
      printf("\n");
   }
   system("PAUSE");
   return 0;
}
```

```
C:\Documents and Settings\loiacono\Documenti\Didattica\InfoB\2008-2009\I Part
Vendite di 1 mese/i fa: 4.3
Vendite di 2 mese/i fa: 6.4
Vendite di 3 mese/i fa: 7.5
Vendite di 4 mese/i fa: 9.9
Vendite di 5 mese/i fa: 12.3
Vendite di 6 mese/i fa: 15.6
6 mesi fa: ****************
5 mesi fa: ************
4 mesi fa: *********
3 mesi fa: *******
2 mesi fa: ******
1 mese fa: ****
Premere un tasto per continuare . . .
```

Array: range

- In C è il programmatore a doversi preoccupare di non accedere a elementi dell'array non validi:

```
float prezzo[4];
prezzo[4] = 46;
```

Scrive una zona di memoria non allocata per la variabile prezzo.

Il comportamento del programma diventa impredicibile!

Array: subscript

- È possibile usare enum e char come subscript
- Esempio

```
typedef enum{gen,feb,mar,apr,mag,giu,lug,ago,set,ott,nov,dic}
    mese;
float vendite[12];
mese m;
printf("Vendite di Aprile: %f\n",vendite[apr]);
for (m=gen; m<=dic; m++)
    scanf("%f", &vendite[m]);

float freq[26];
printf("La frequenza della lettera f e': %f\n",freq['f'-'a']);
```

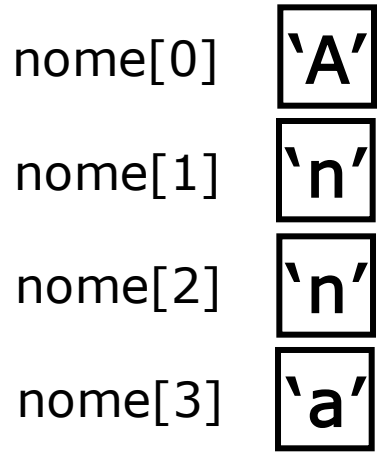
Stringhe

Stringhe

- Gli array di tipo **char** sono detti anche **stringhe**
- Dal momento che sono molto usati, il C mette a disposizione funzioni specifiche per questo tipo di dato

```
char nome[4];

nome[0] = 'A';
nome[1] = 'n';
nome[2] = 'n';
nome[3] = 'a';
```




```
typedef char stringa[30];
stringa messaggio;
```

Stringhe: costanti e inizializzazione

- In C, le costanti di tipo stringa si rappresentano come una sequenza di caratteri racchiusi tra “”
  - E.g. “anna” è una costante di tipo stringa

- L’inizializzazione può avvenire in fase di dichiarazione:

```
typedef char stringa[30];
stringa messaggio="prova";
```

Stringhe: lettura e scrittura

- La lettura e scrittura di stringhe è particolarmente semplice:

```
char nome[30];
printf("Inserisci il tuo nome: ");
scanf("%s",nome);
printf("Ciao %s!\n", nome);
```

Per la lettura delle stringhe NON si deve anteporre **&**

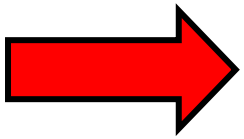
- La scanf legge assume che la stringa non contenga spazi!
- Per leggere stringhe con spazi si utilizza **gets** (messa a disposizione dalle libreria **stdio.h**)

```
printf("Inserisci il tuo nome: ");
gets(nome);
```

Stringhe: carattere terminatore

- In C esiste un carattere speciale che indica la fine di una stringa: il carattere '**\0**'
- Quando la funzione **printf** individua questo carattere speciale smette di stampare a video gli elementi della stringa

```
char msg[30];
msg[0] = 'A';
msg[1] = 'B';
msg[2] = '\0';
printf("%s",msg);
```



Stampa solo "AB" e non 30 caratteri!

- Le funzioni **scanf** e **gets** provvedono ad aggiungere il carattere di terminazione '\0'

Stringhe: funzioni di libreria

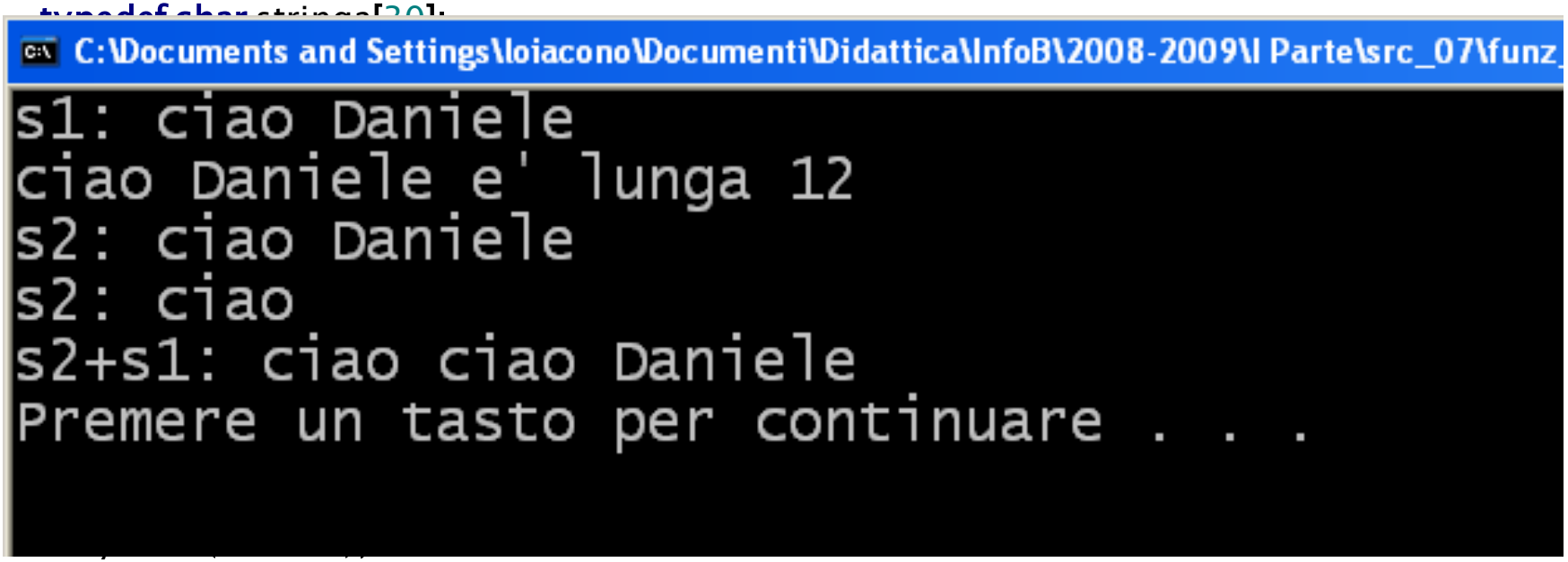
- Esistono diverse funzioni di libreria per la manipolazione delle stringhe e sono definite in **string.h**

```
#include<stdio.h>
#include<string.h>
typedef char stringa[30];
int main()
{
   stringa s1,s2;
   printf("s1: ");
   gets(s1);
   printf("%s e' lunga %d\n",s1,strlen(s1));
   strcpy(s2,s1);
   printf("s2: %s\n",s2);
   strcpy(s2,"ciao ");
   printf("s2: %s\n",s2);
   strcat(s2,s1);
   printf("s2+s1: %s\n",s2);
   system ("PAUSE");
   return 0;
}
```

Stringhe: funzioni di libreria

- Esistono diverse funzioni di libreria per la manipolazione delle stringhe e sono definite in **string.h**

```
#include<stdio.h>
#include<string.h>
typedef char stringa[20];
```




```
    return 0;
}
```

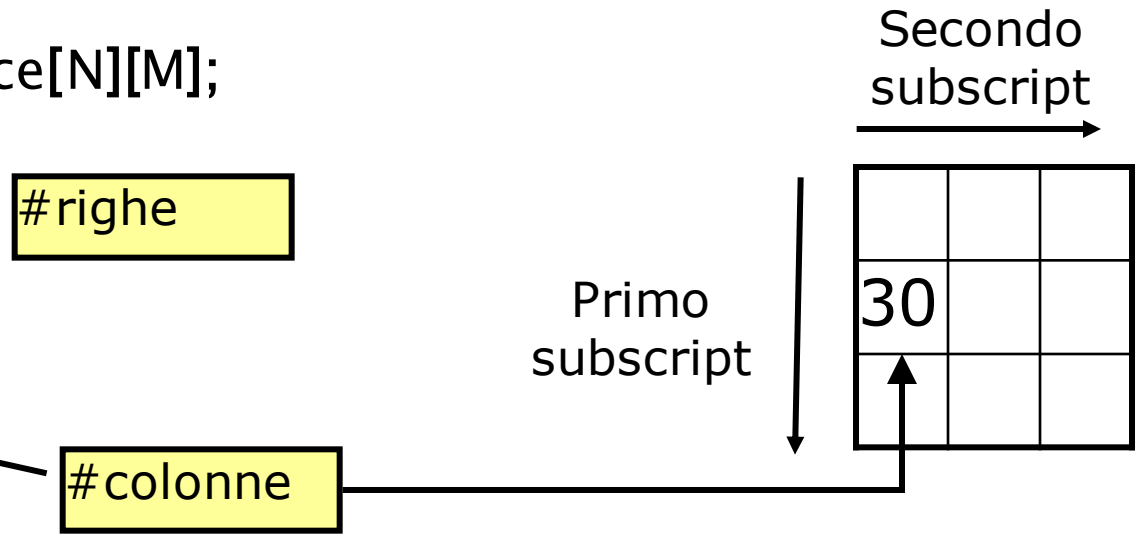
Matrici

Matrici

- Le matrici sono strutture strutture dati bidimensionali
- Vengono rappresentati come array di array
- Esempi:

```
typedef int matrice[N][M];
matrice a;
```

```
float b[3][3];
```



- **Accesso ad un elemento**

```
b[1][0]=30;
```

Lettura di matrici

- Come per gli array, la lettura avviene un elemento alla volta
- Richiede due cicli for innestati:

```
float a[N][M];

for (i=0;i<N;i++)
        for (j=0;j<M;j++)
            scanf("%f",&a[i][j]);
```

Somma fra matrici

```
float a[N][M];
float b[N][M];
float sum[N][M];

 for (i=0;i<N;i++)
        for (j=0;j<M;j++)
            sum[i][j] = a[i][j] + b[i][j];
```

Scrittura di una matrice

- Come per la lettura si scrive un elemento alla volta e si usano due cicli innestati:

```
float sum[N][M];

for (i=0;i<N;i++){
        for (j=0;j<M;j++){
            printf("%f ",sum[i][j]);
        }
        printf("\n");
    }
```

Trasposta di una matrice

```
float a[N][M];
float b[M][N];

 for (i=0;i<N;i++)
        for (j=0;j<M;j++)
            c[j][i] = a[i][j];
```

Moltiplicazione fra matrici

```
float a[N][M];
float b[M][N];

for (i=0;i<N;i++)
{
    for (j=0;j<N;j++)
    {
        prod[i][j] = 0;
        for (k=0; k<M;k++)
        {
            prod[i][j] += a[i][k]*c[k][j];
        }
        printf("%d ",prod[i][j]);
    }
    printf("\n");
}
```