POLITECNICO
MILANO 1863

Informatica A

Array e Stringhe
Struct

Info A: P. Perego

ARRAY

Gli array

- Gruppi di celle consecutive
  - Rappresentano gruppi di variabili
  - Con lo stesso nome e lo stesso tipo
- Per riferirsi a un elemento, si specificano
  - Il nome dell'array
  - La posizione dell'elemento (indice)
- Sintassi: `<nomearray>[<posizione>]`
  - Il primo elemento ha indice 0
  - L'n-esimo elemento dell'array `v` è `v[n-1]`

Dichiarazione: int v[12];

v[0]	-46
v[1]	6
v[2]	0
v[3]	72
v[4]	1542
v[5]	-86
v[6]	0
v[7]	62
v[8]	-2
v[9]	1
v[10]	6452
v[11]	78

Gli array

- Array: vettori di elementi
  - Indicizzati
  - Omogenei
  - Memorizzati in celle di memoria consecutive
- Dichiarazione di un array:
  - `char parola[12];`
  - È un array di 12 elementi di tipo char, vale a dire un vettore di 12 caratteri
- La lunghezza dell'array è comunque decisa *durante la compilazione* del programma
  - Nella dichiarazione, *non* usiamo variabili per specificare la dimensione degli array

Gli array

- Sequenza di elementi consecutivi dello stesso tipo in numero predeterminato e costante
  - Noto a tempo di compilazione

- Ogni elemento della sequenza è individuato da un indice
  - La sua posizione nella sequenza
  - Indice con valore da 0 a N-1
    - Dove N è la dimensione dell'array

```
int v[100];
. . .
v[3] = 0;
```

Se i >= 100, v[i] *è un errore*!

Il comportamento è indefinito

Gli array

- Gli elementi di un array sono normali variabili

```
vett[0] = 3;
printf("%d", vett[0]);
> 3
scanf("%d", &vett[1]);
> 17      vett[1] assume valore 17
```

- Si possono usare espressioni come indici

```
x = 3
vett[5-2]
```

è equivalente a `vett[3]`

ed è equivalente a `vett[x]`

POLITECNICO
MILANO 1863

Come opera il calcolatore?

- int v[100];
  - Alloca memoria per 100 elementi interi, a partire da un certo indirizzo di memoria
    - La dimensione deve essere nota al compilatore
    - Deve essere un'espressione senza variabili

- Per accedere all' i-esimo elemento di v[...]
  - Valuta l'indice i
    - Può essere un'espressione
  - All'indirizzo della prima cella di v[...] somma il numero di celle pari allo spazio occupato da i elementi
    - Ottiene così l'indirizzo dell'elemento cercato
    - È possibile perché gli elementi sono tutti dello stesso tipo, e il tipo determina la dimensione in memoria

Inizializzazione di un array

- Sintassi compatta

```
int n[5] = {1, 2, 3, 4, 5};
int n[5] = {13};
```

  - Tutti gli altri elementi sono posti a 0
  - Specificare troppi elementi tra le graffe è un errore di sintassi

- Se la lunghezza dell'array è omessa, gli inizializzatori la determinano

```
int n[] = {5, 47, –2, 0, 24};
```

Equivalente a

```
int n[5] = {5, 47, – 2, 0, 24};
```

– In tal caso la dimensione è inferita automaticamente
  - 5 elementi con indici da 0 a 4

POLITECNICO
MILANO 1863

Operazioni sugli array

- Si opera sui singoli elementi, uno per volta
- Non è possibile operare sull'intero array, agendo su tutti gli elementi simultaneamente

```
/* come ricopiare array1 in array2 */
int array1[10], array2[10];
int i;
…
array2 = array1;                    /* ERRATO */
…
for (i = 0; i < 10; i++) {
        array2[i] = array1[i];      /* CORRETTO */
}
```

Esempi sugli array

- Dichiarazione del vettore

```
int a[20];
```

- Inizializzazione del vettore (omogenea)

```
for (i = 0; i <= 19; i++)
   a[i] = 0;
```

(alternativa alla dichiarazione, valida solo per lo zero: int a[20] = {0})

- Inizializzazione "da terminale"

```
for (i = 0; i <= 19; i++) {
   printf("\n Scrivi un intero: ");
   scanf("%d", &a[i]);
}
```

POLITECNICO
MILANO 1863

Esempi sugli array

- Ricerca del massimo

```
int max = a[0];
for (i = 1; i <= 19; i++)
   if (a[i] > max)
      max = a[i];
```

- Calcolo della media

```
float media = a[0];
for (i = 1; i <= 19; i++)
   media = media + a[i];
media = media / 20;
```

Esempi sugli array

Calcolo di massimo, minimo e media di un vettore
È sufficiente *una sola scansione* del vettore (un solo ciclo)

```
int a[20];
int max, min, i;
float media;
for (i = 0; i <= 19; i++) {
    printf("\n Scrivi un intero: ");
    scanf("%d", &a[i]);
}
max = min = media = a[0];
for (i = 1; i <= 19; i++) {
    media = media + a[i];
    if (a[i] > max)
        max = a[i];
    if (a[i] < min)
        min = a[i];
}
media = media / 20;
```

Un nuovo problema

- Mostrare una sequenza di 100 interi *nell'ordine inverso* rispetto a quello con cui è stata introdotta dall'utente (stdin)
  - Con un array?
  - Senza array?

```
/* Programma InvertiSequenza */
int main() {
        int i, a[100];
        i = 0;
        while (i < 100) {
                printf(" \n fornisci un valore intero: ");
                scanf("%d", &a[i]);
                i++;
        }
        i--;
        while (i >= 0) {
                printf("%d\n", a[i]);
                i--;
        }
        return 0;
}
```

Funziona *solo* per un array di 100 elementi

- Che cosa possiamo fare se sono di meno?
- E se (peggio) sono di più?

Generalizziamo con la direttiva #define

- In testa al programma

```
#define LUNG_SEQ 100
```

- Così possiamo adattare la lunghezza del vettore alle eventuali mutate esigenze senza riscrivere la costante 100 in molti punti del programma
  - Il preprocessore sostituisce nel codice LUNG_SEQ con 100 prima della compilazione

- La lunghezza dell'array, quindi, anche in questo caso è decisa al momento della compilazione del programma

- Nella dichiarazione degli array non usiamo *mai* variabili per specificarne la dimensione

```c
/* Programma InvertiSequenza */
#define LUNG_SEQ 100
int main( ) {
        int i, a[LUNG_SEQ];
        i = 0;
        while (i < LUNG_SEQ) {
                printf("fornisci un valore intero");
                scanf("%d", &a[i]);   i++;
        }
        i--;
        while (i >= 0) {
                printf("%d\n", a[i]);
                i--;
        }
        return 0;
}
```

Parametrizzazione (maggiore astrazione del codice)

```c
#define LUNG_SEQ 100

// Programma InvertiSequenza di lunghezza <= a un valore dato
int main() {
        int lunghezza, i, a[LUNG_SEQ];
        printf("\n lunghezza sequenza: ");
        scanf("%d", &lunghezza);
        if (lunghezza <= LUNG_SEQ) {
                i = 0;
                while (i < lunghezza) {
                        printf("\n fornisci un valore intero  ");
                        scanf("%d", &a[i]);
                        i++;
                }
                i--;
                while (i >= 0) {
                        printf("%d\n", a[i]);
                        i--;
                }
        }
        return 0;
}
```

Trattare anche il caso opposto

Soluzione "a sentinella": legge una sequenza di naturali, terminata da -1, e la stampa in sequenza invertita.
Si ipotizza che la sequenza abbia lunghezza <= 100.

```
int a[LUNG_SEQ], i=0, temp;
scanf ("%d", &temp);
while (temp != -1) {
      a[i] = temp;
      i++;
      /* oppure a[i++] = temp; */
      scanf ("%d", &temp);
}
while (i > 0) {
      i--;
      printf("%d\n", a[i]);
      /* oppure printf ("%d\n", a[--i]); */
}
```

N.B. Si è trascurato il dialogo di input output

La soluzione precedente non evitava di superare il limite fisico del vettore. Con una semplice modifica riusciamo a non generare errori nel caso in cui l'utente immetta più di LUN_SEQ valori.

```
int a[LUNG_SEQ], i=0, temp;
printf("Inserire una sequenza di interi terminata da -1\n");
scanf ("%d", &temp);
while (temp != -1 && i < LUNG_SEQ) {
        a[i] = temp;
        i++;
        scanf ("%d", &temp);
}
if (temp != -1 && i == LUNG_SEQ)
        printf("Raggiunto il limite di %d valori\n\n",
LUN_SEQ);
while (i > 0) {
        i--;
        printf("%d\n", a[i]);
}
```

POLITECNICO
MILANO 1863

```
/* Output Strutturato:
Stampa di un istogramma */
```

```c
/* Output Strutturato: Stampa di un istogramma */
#include <stdio.h>
#define SIZE 10
int main () {
    int n[SIZE] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
    int i, j;
    printf("%s%13s%17s\n\n","Element","Value","Histogram");
    for (i = 0; i < SIZE; i++) {
        printf("%7d%13d", i, n[i]);
        for (j = 1; j <= n[i] ; j++) /* una riga di '*' */
            printf("*");
    printf("\n");
    }
    return 0;
}
```

POLITECNICO
MILANO 1863

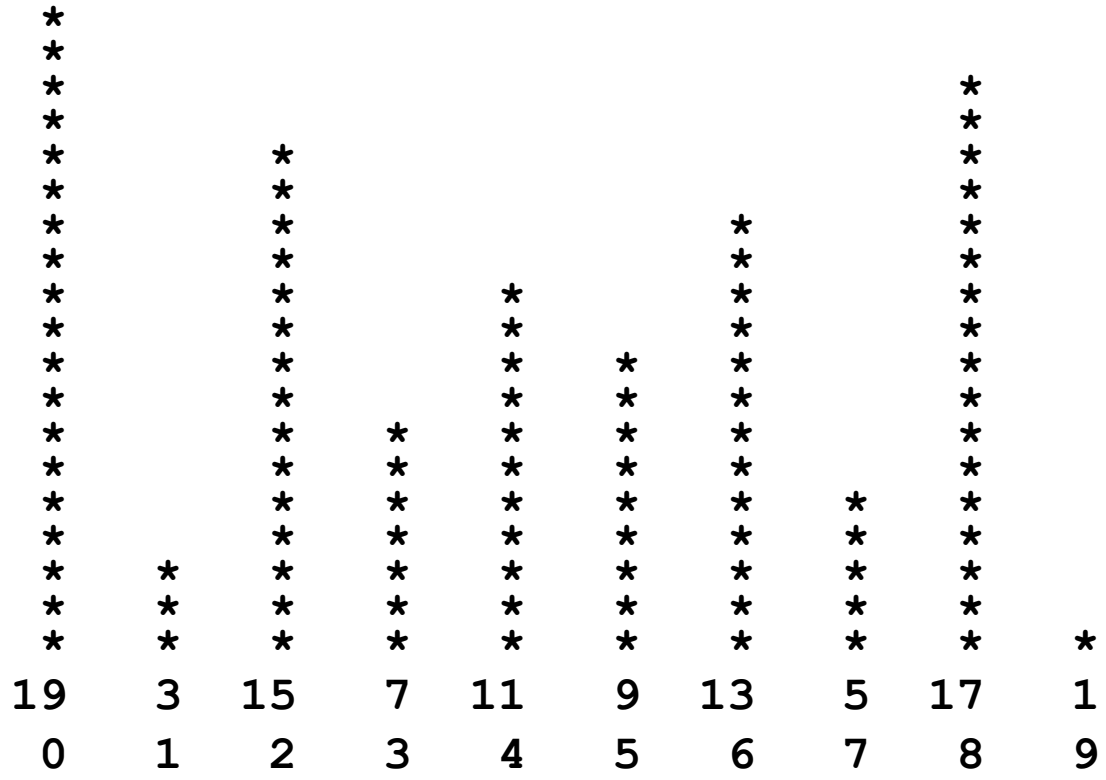
Output del programma

```
Element         Value         Histogram

      0            19         ********************
      1             3         ***
      2            15         ***************
      3             7         *******
      4            11         ***********
      5             9         *********
      6            13         *************
      7             5         *****
      8            17         *****************
      9             1         *
```

Esercizio

- Modificare il programma per visualizzare istogrammi verticali



Array a più dimensioni

- Gli array a 1D realizzano i vettori, quelli a 2D realizzano le matrici, … e così via

- Dichiarazione:

```
int A[20][30];
```

  - A è una matrice di 20 × 30 elementi interi (600 variabili distinte)

```
float F[20][20][30];
```

  - F è una matrice 3D di 20×20×30 variabili di tipo float (12.000!)

- Oppure a quattro dimensioni, ecc …

STRINGHE

Le stringhe

- Array di caratteri
  - Rappresentano "caratteri da leggersi in fila"

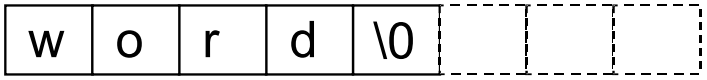
- Dichiarazione e inizializzazione di una stringa

```
char stringa[] = "word";
```

- Il carattere nullo '\0' termina le stringhe
  - Perciò l'array `stringa` ha 5 elementi (non 4):

- Dichiarazione equivalente

```
char stringa[] = {'w','o','r','d','\0'};
```



Stringhe e caratteri

- Qual è la differenza tra 'x' e "x"?
  - 'x' è una costante di tipo char
    - Rappresentata in memoria occupando 1 byte in codifica ASCII
  - "x" è una stringa costante
    - Rappresentata in memoria con un array di char contente i caratteri 'x' e '\0'

- *Attenzione*: le stringhe *non* sono propriamente un tipo di dato
  - Non sono un tipo di base
  - Non hanno operatori nativi
  - Una serie di funzioni della libreria `string.h` permette di manipolarle

POLITECNICO
MILANO 1863

Operazioni su stringhe

```
char str1[32]; /* str1 ha spazio per 32 char. */
char str2[64]; /* str2 ha spazio per 64 char. */

/* inizializza str1 con la stringa "alfa" */
strcpy(str1, "alfa");     /* str1 contiene "alfa"  */

/* copia str1 in str2 */
strcpy(str2, str1);       /* str2 contiene "alfa"  */

/* lunghezza di str1 */
x = strlen(str1);         /* x assume valore 4     */

/* scrivi str1 su standard output */
printf("%s", str1);       /* scrive str1 su stdout */

/* leggi str1 da standard input */
scanf("%s", str1);        /* str1 “riceve” da stdin */
```

Operazioni su stringhe

```
char str1[32];
char str2[64];

scanf("%s", str1);
> ciao↵   /* ora str1 contiene "ciao" */

strcpy(str2, str1);  /* str2 riceve "ciao"*/
val = strlen(str2);  /* val  = 4          */

printf("%s\n", str2);
> ciao /* stampa "ciao" */

Attenzione:     strlen("")   vale 0 !
```

Particolarità delle stringhe

- Inizializzazione e accesso ai singoli caratteri:

  ```
  char stringa[] = "word"  {'w','o','r','d','\0'};
  ```

  `stringa[3]` è un'espressione di valore 'd'

- Il nome dell'array rappresenta l'indirizzo del suo primo elemento
  - Quando ci si vuole riferire all'intero array nella scanf non si mette il simbolo &!
  - `scanf("%s", stringa);` -> `scanf("%s", &stringa[0]);`
  - Questa scanf legge in input i caratteri fino a quando trova il carattere "blank" (lo spazio), o l'invio

- Se il buffer contiene una stringa "troppo lunga"
  - La stringa è memorizzata oltre la fine dell'array!
  - *È un errore grave!*

```
/* Stringhe e array di caratteri */

#include <stdio.h>
#include <string.h>

int main () {
      char str1[20], str2[] = "string literal";
      int i;
      printf("\n Enter a string: ");
      scanf("%s", str1);
      printf("str1: %s\n str2: %s\n", str1, str2);
      printf("str1 with spaces is: \n");
      i = 0;
      while( str1[i] != '\0' ) {
            printf ("%c ", str1[i]);
            i++;
      }
      printf ("\n");
      return 0;
}
```

scanf interrompe la scansione quando incontra uno spazio

```
> Enter a string: Hello guys
> str1: Hello
> str2: string literal
> str1 with spaces is:
> H e l l o
```

strcpy(s1, s2)

- E se non ci fosse la funzione strcpy()?
  - Assegneremmo sempre un carattere alla volta!

```
char s1[N], s2[M];
/* Assegnamento di s2, omesso */
int i = 0;
while (i <= strlen(s2) && i < N) {
      s1[i] = s2[i];
      ++i;
}
```

N.B. Funziona correttamente se s2 è una stringa ben formata (cioè terminata da '\0') e se s1 è sufficientemente grande da contenere i caratteri di s2 (N >= strlen(s2))

Quiz

- Che cosa stampano le seguenti printf()?

```
char ciao[6] = “ciao”;
ciao[strlen(ciao)] = ciao[2];

printf(“%s\n”, ciao);
printf(“%d\n”, strlen(ciao));
```

Morale: mai dimenticare che c'è anche il carattere '\0'

Confrontare due stringhe

- Una funzione apposita: `strcmp(s1, s2)`
  - Restituisce un intero
    - 0 se le stringhe sono uguali
  - Confronta le due stringhe fino al '\0'

```
char s1[32], s2[64];
int diverse;
/* Acquisizione di valori per le stringhe
   (codice omesso)*/
diverse = strcmp(s1, s2);

if (diverse == 0)
       printf("UGUALI\n");
else if (diverse < 0)
       printf("%s PRECEDE %s\n", s1, s2);
else
       printf("%s SEGUE %s\n", s1, s2);
```

strcmp(s1, s2)

- E se non ci fosse?
  - Controlliamo un carattere alla volta
  - Interrompiamo il controllo appena sono diverse

```
char s1[N], s2[M];
int i = 0;
…
while (i<strlen(s1) && i<strlen(s2)
                    && s1[i] == s2[i]) i++;
if (s1[i] == s2[i]) /* s1[i] == '\0'*/
     … Le stringhe sono uguali …
```

STRUCT

Aggregazione di variabili

- Gruppi di variabili *omogenee*
  - Array (vettori)

- Gruppi di variabili *eterogenee*
  - Struct (record)

Dichiarazione dei dati: dati complessi o strutturati

- Record (o struct): memorizzano aggregazioni di dati (ciascun dato è chiamato "campo") di diversa natura

```
struct {
   char via[20];      /* 1o campo:  stringa */
   int numero;        /* 2o campo:  intero  */
   int CAP;           /* 3o campo:  intero  */
   char citta[20];    /* 4o campo:  stringa */
} indirizzo;
```

Nome della variabile di tipo record

- Indirizzo: un record con 4 campi di vario tipo

POLITECNICO
MILANO 1863

Uso dei record

- Il record (o struct) è una sorta di "contenitore" di campi *di tipo eterogeneo*
- Il record raggruppa dati più semplici
  - Ne rende più ordinata la gestione, evitando confusioni
- I campi del record non sono visibili direttamente
  - Il loro nome deve essere preceduto da quello del record a cui appartengono
    - Interponendo . come separatore
  - Sono identificatori *locali* all'interno di una variabile di tipo strutturato, da usarsi come suffissi

Operazioni su record

- Assegnamento ai campi del record

```
strcpy(indirizzo.via, "Ponzio");
indirizzo.numero = 34;
indirizzo.CAP = 20133;
strcpy(indirizzo.citta, "Milano");
```

- Accesso (leggere, scrivere…)

```
printf("%d\n", indirizzo.numero);
> 34
printf("%d\n", strlen(indirizzo.citta));
> 6
printf("%s\n", indirizzo.citta);
> Milano
scanf("%s", indirizzo.via);
> Ponzio↵
scanf("%d", &indirizzo.CAP);
> 20133↵
```

Ancora operazioni su record

- Dati due record identici (cioè dichiarati insieme)
- È lecito assegnare globalmente il primo al secondo

```
struct { … /* campi */ } rec1, rec2;
```

- È lecito scrivere:

```
rec2 = rec1;
```

  - Tutti i campi di rec1 sono ordinatamente copiati nei campi corrispondenti di rec2.
  - Se i due record sono diversi (anche solo per l'ordine dei campi) l'assegnamento è privo di senso !
  - Memento: l'assegnamento *diretto* tra array è vietato
    - Deve avvenire elemento per elemento