Unità G1

Dati strutturati

Obiettivi

- Conoscere il significato dei dati strutturati di tipo array
- Sapere che cos'è una stringa
- Conoscere il significato di dati strutturati di tipo struct
- Essere in grado di realizzare programmi che gestiscono array a una o più dimensioni
- Essere in grado di realizzare programmi complessi con iterazioni e selezioni
- Essere in grado di gestire le stringhe di caratteri
- Essere in grado di lavorare con aggregazioni di dati di tipo eterogeneo
- Essere in grado di realizzare programmi con dati strutturati composti da array e strutture

Aggregazione di dati omogenei: gli array

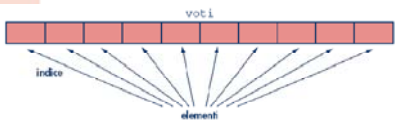
- Spesso gli algoritmi lavorano su insiemi di *dati omogenei*.
  - l'insieme dei valori reali che rappresentano le temperature medie delle capitali europee di un giorno dell'anno;
  - i caratteri che rappresentano le note degli accordi di una canzone nel formato anglosassone;
  - i valori interi che rappresentano i voti ottenuti dagli studenti di una classe in una prova di esame.
- Per risolvere problemi che trattano dati di questo tipo si dovrebbero utilizzare tante variabili dello stesso tipo (int, double o float, char) quanti sono gli elementi dell'insieme d'interesse.

Esempi

- Per i voti ottenuti dagli studenti potremmo utilizzare le variabili voto1, voto2, voto3... per memorizzare i voti ottenuti rispettivamente del primo studente, dal secondo, dal terzo ecc.

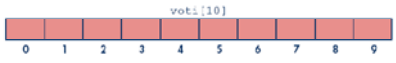
Array

- Un **array** è una **collezione omogenea** di elementi individuati da un **indice**.
- L'**indice** è un valore che consente di individuare ogni singolo elemento.
- Molti linguaggi utilizzano come indici valori interi che partono da 1; alcuni consentono di scegliere il rango degli indici, altri consentono di utilizzare per gli indici anche valori non numerici.

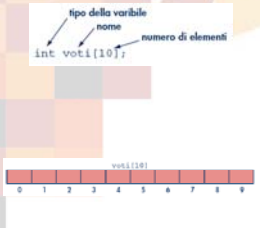


Dichiarazione di un array

- Nella **fase di dichiarazione** si deve specificare il **nome** della variabile, il **numero** degli elementi e il **tipo** di ogni elemento.
- Si tratta di una collezione di elementi **omogenei**, quindi tutti dello stesso tipo.
- In C l'indice del primo elemento è individuato dal valore 0, i seguenti elementi dai successivi valori interi positivi: 1, 2, 3...



Dichiarazione di un array in C



- Es. dichiarazione una variabile array di tipo int, di nome **`voti`**, che contiene dieci valori.
- Ciascuno dei contenitori che compongono l'array è un **elemento**, che si distingue dagli altri per avere un **indice univoco** (posizione dell'elemento all'interno dell'array).
- Il numero degli elementi è detto **lunghezza** dell'array.
- voti[10] ha un indice che assume valori 0, 1, 2... 9 e ha lunghezza 10.

Dichiarazione

- Dichiarazione di un array in C:
  `<tipo> <variabile> [<lunghezza>]`
- Es. **`float temperature[30]`**
- Per selezionare un elemento di un array si utilizza il nome seguito dall'indice dell'elemento tra parentesi quadre
- **`temperature[2]`** seleziona il terzo elemento dell'array **`temperature`**

Selezione di un elemento

- Non è possibile operare su tutto l'array:
  - Non è possibile stampare con una sola istruzione l'intero contenuto (~~cout<<voti~~)
  - Non è possibile ricevere in input con una sola istruzione l'intero array (~~cin>>voti~~)
- È necessario sempre operare su un singolo elemento dell'array
- Per selezionare un elemento di un array si utilizza il nome seguito dall'indice dell'elemento tra parentesi quadre
- Esempio voti[2] indica il terzo elemento dell'array voti

Operare con gli array

- Dovendo operare su ogni singolo elemento dell'array è consigliabile utilizzare un ciclo
- Il ciclo più indicato è il ciclo for
- Esempio di lettura dell'array voti:

```
for (i=0;i<10;i++)
{
  cout<<"inserisci valore dell'el. di indice"<<i;
  cin>>voti[i];
}
```


Dimensione dell'array

- Il numero degli elementi dell'array deve essere definito in fase di dichiarazione e non può variare nel corso dell'esecuzione del programma.
- In fase di dichiarazione la lunghezza dell'array deve essere una **costante**
- **`int voti[10]`** è una dichiarazione corretta
- **`int voti[num_stud]`** è corretta se **`num_stud`** è dichiarato come costante ma errato se **`num_stud`** è dichiarato come variabile

Controlli sui limiti degli indici

- Se si utilizza un indice maggiore o uguale alla lunghezza del vettore, si fa riferimento a spazi di memoria non riservati dalla dichiarazione e l'effetto non sarà prevedibile: molto probabilmente si provocheranno errori.
- Conviene **dimensionare adeguatamente** il vettore e, in caso di dubbio, sovradimensionarlo.
- In altri linguaggi di programmazione il controllo sul valore dell'indice è effettuato dal linguaggio stesso; il C offre la massima libertà al progettista software, che però deve utilizzarla con molta **attenzione**.
- Un eccessivo **sovradimensionamento** causa però uno spreco non giustificato di memoria.

I menu

- Nell'interazione con l'utente sono utilizzati molto spesso i **menu** da cui viene scelta un'**opzione**.
- Esempio

```
do
{
   cout<<"MENU"<<endl;
   cout<<"1. Maggiore"<<endl;
   cout<<"2. Minore"<<endl;
   cout<<"3. Media"<<endl;
   cout<<"Scelta: ";
   cin>>sceltaUtente;
} while (sceltaUtente<1 || sceltaUtente >3);
```


Problema

- **Problema**: *richiedere all'utente il numero di studenti che compongono la classe e i loro voti. Successivamente presentare all'utente un menu con tre opzioni per visualizzare il maggiore, il minore o la media dei voti ottenuti dagli studenti in un compito. Infine visualizzare i voti ottenuti.*
- **Input**: numero degli studenti della classe, voti ottenuti dagli studenti, scelta utente (1 Maggiore, 2 Minore, 3 Media).
- **Output:** voti ottenuti.

Programma

- Riferimento al listato: g1-01.c

Problema

- **Problema**: *determinare il punteggio totale che otto team di Formula 1 hanno conseguito nei Granpremi di Monza e Barcellona.*
- **Input**: i punteggi ottenuti dagli otto team a Monza e quelli ottenuti a Barcellona.
- **Ouput**: il punteggio totale degli otto team ottenuti nei due Granpremi.

Programma

Somma di due array

indici	Monza		Barcellona		punteggioTotale
1	0	+	2	→	2
2	10	+	4	→	14
3	4	+	0	→	4
4	8	+	12	→	20
5	2	+	6	→	8
6	6	+	8	→	14
7	0	+	0	→	0
8	12	+	10	→	22

Dichiarazione e inizializzazione

- È possibile inizializzare un array mentre lo si dichiara.

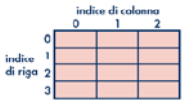
```
int voti[10] = {8, 7, 9, 4, 2};
```

- L'esempio dichiara e definisce la variabile strutturata intera **voti**, inoltre assegna ai suoi elementi con indice 0, 1, 2, 3 e 4 rispettivamente i valori: 8, 7, 9, 4, 2.
- Nel caso in cui si utilizzi l'inizializzazione non importa specificare il numero di elementi.

```
int voti[] = {8, 7, 9, 4, 2};
```


Matrici: array a due dimensioni

- Negli array a due dimensioni, o **matrici**, i dati sono organizzati per righe e per colonne, proprio come in una **tabella**.
- In fase di dichiarazione si dovrà specificare il numero di righe e di colonne che formano la tabella

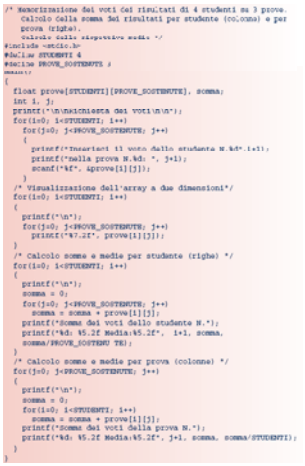


Matrici in C

- In fase di dichiarazione viene specificato il **tipo** degli elementi, il **nome** della matrice, il numero di **righe** e di **colonne**
- L'indice di riga parte da 0 e assume valori interi progressivi: 0, 1, 2... *n* – 1 se *n* sono le righe dell'array.
- L'indice di colonna parte da 0 e assume valori interi progressivi: 0, 1, 2... *m* – 1 se *m* sono le colonne dell'array.

Esempio

- Problema*:*
  - *memorizzare i risultati ottenuti da* n *studenti in* m *prove, con* n *uguale a 4 e* m *uguale a 3.*
  - *Calcolare la somma dei voti che ogni studente ha ottenuto nelle differenti prove e la media dello studente.*
  - *Calcolare la somma dei voti degli studenti per ogni prova e quella relativa per ogni prova.*
- **Input**: voti ottenuti dagli studenti per ogni prova.
- **Output**: visualizza la tabella dei risultati.
  - Per ogni studente la somma dei voti delle differenti prove e la relativa media.
  - Per ogni prova la somma dei voti degli studenti e la relativa media.
- **Analisi:** per memorizzare i voti ottenuti dagli studenti in più prove è utilizzata una variabile strutturata: un array di reali a due dimensioni.



```
/* Memorizzazione dei voti dei risultati di 4 studenti su 3 prove.
   Calcolo della somma dei risultati per studente (colonna) e per
   prova (riga).
   Calcolo della rispettiva media */
#include <stdio.h>
#define STUDENTI 4
#define PROVE_SOSTENUTE 3
main()
{
  float prove[STUDENTI][PROVE_SOSTENUTE], somma;
  int i, j;
  printf("\n\nAcquisizione dei voti\n\n");
  for(i=0; i<STUDENTI; i++)
    for(j=0; j<PROVE_SOSTENUTE; j++)
    {
      printf("Inserisci il voto dello studente N.%d",i+1);
      printf("nella prova N.%d: ", j+1);
      scanf("%f", &prove[i][j]);
    }
  /* Visualizzazione dell'array a due dimensioni*/
  for(i=0; i<STUDENTI; i++)
  {
    printf("\n");
    for(j=0; j<PROVE_SOSTENUTE; j++)
      printf("%7.2f", prove[i][j]);
  }
  /* Calcolo somme e medie per studente (righe) */
  for(i=0; i<STUDENTI; i++)
  {
    printf("\n");
    somma = 0;
    for(j=0; j<PROVE_SOSTENUTE; j++)
      somma = somma + prove[i][j];
    printf("Somma dei voti dello studente N.");
    printf("%d: %5.2f Media:%5.2f",  i+1, somma,
    somma/PROVE_SOSTENU TE);
  }
  /* Calcolo somme e medie per prova (colonne) */
  for(j=0; j<PROVE_SOSTENUTE; j++)
  {
    printf("\n");
    somma = 0;
    for(i=0; i<STUDENTI; i++)
      somma = somma + prove[i][j];
    printf("Somma dei voti della prova N.");
    printf("%d: %5.2f Media:%5.2f", j+1, somma, somma/STUDENTI);
  }
}
```

Array multidimensionali

- In genere i linguaggi forniscono anche array con più di due dimensioni; la sintassi della dichiarazione e della selezione di un elemento dell'array è analoga a quella presentata per gli array mono o bidimensionali.
- In fase di dichiarazione viene definito il numero di elementi per ogni dimensione
- Per esempio, il codice

```
float valori [10][5][8]
```

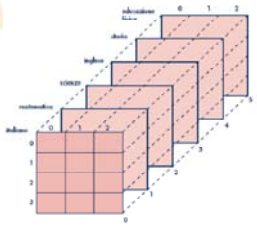
dichiara un array di nome valori a tre dimensioni composto da 10*5*8, cioè 400 elementi.

Esempio

- Se vogliamo, per esempio, gestire le prove sostenute durante l'anno dagli studenti in diverse materie, potremmo definire un array in tre dimensioni:

```
int voti
   [STUDENTI][PROVE_SOST
   ENUTE][MATERIE]
```

- Dove STUDENTI è il numero di studenti, PROVE_SOSTENUTE il numero massimo di prove per materia e MATERIE è il numero di materie.

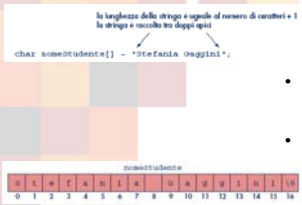


Stringhe

- Un insieme di caratteri in sequenza forma una **stringa**.
- Alcuni linguaggi definiscono per le stringhe uno specifico tipo di dato e offrono metodi specifici per gestire valori di quel tipo.
- Il linguaggio C, tranne che in alcuni suoi dialetti, non fornisce un tipo di dato specifico per le stringhe che vengono trattate come array di caratteri.

```
char nomeStudente[10];
```


Le stringhe in C



- Nell'esempio viene dichiarato l'array di caratteri **`nomeStudente`** e inizializzato con il valore **`"Stefania Gaggini"`.**
- Il numero di elementi dell'array è determinato dalla **lunghezza** della stringa più 1.
- L'elemento in più è necessario al linguaggio per memorizzare il carattere **terminatore di stringa**: \0, conosciuto come il carattere **null**.

Stringhe e … caratteri

- Attenzione alla differenza tra le seguenti dichiarazioni con inizializzazione.

```
char nota = 'C';
```

- che assegna alla variabile nota di tipo char il valore **`C`**

```
char nota[] = "C";
```

- che assegna all'array nota[] la stringa di due caratteri **`C\0`**
- Un carattere inserito tra apici singoli è trattato come un singolo carattere.

Selezione di un carattere

```
…
char nomeStudente[] = "Stefania Gaggini";
nomeStudente[9] = 'P';
```

- L'assegnamento al nono elemento di nomeStudente del carattere P modifica la stringa che diviene **`"Stefania Paggini".`**
- Il carattere terminatore \0 consente di gestire le stringhe senza conoscerne a priori la dimensione.

Caratteri e codici ASCII

In linguaggio C

```
char nomeStudente[] = "Stefania Gaggini";
int i=0;
while(nomeStudente[i]!='\0'){
  printf("carattere %c ASCII %d\n", nomeStudente[i], nomeStudente[i]);
  i++;
}
…
```

Il codice visualizza ogni elemento sia sotto forma di carattere che di codice ASCII

In linguaggio C++

```
char nomeStudente[] = "Stefania Gaggini";
int i=0;
while(nomeStudente[i]!='\0'){
  cout<<"carattere "<<nomeStudente[i]<<" ASCII
   "<<(int)nomeStudente[i]);
  i++;
}
…
```

Il codice visualizza ogni elemento sia sotto forma di carattere che di codice ASCII

Problema

- *Problema: concatenare due parole immesse dall'utente, aggiungendo la seconda alla prima e inserendo tra le due un carattere spazio.*
- **Input**: due parole.
- **Output**: visualizzare la stringa frutto della concatenazione.

Soluzione

```
/* Concatena due stringhe immesse dall'utente inserendo un carattere
spazio tra le due */
#include <stdio.h>

main()
{
  char parola1[50], parola2[15];
  int i, j;
  printf("Immetti una parola: ");
  scanf("%s", &parola1);
  printf("Immetti un'altra parola: ");
  scanf("%s", &parola2);
  for(i=0; (parola1[i])!='\0'; i++)
    ;
  parola1[i++]=' ';
  for(j=0; (parola1[i]=parola2[j])!='\0'; i++,j++)
    ;
  printf("%s\n", parola1);
}
```


Libreria string.h

- La libreria string.h rende disponibili molte funzioni standard per gestire le stringhe.:
- strcpy(stringa1, stringa2) copia stringa2 su stringa1;
- strncpy(stringa1, stringa2, n) copia i primi n caratteri di stringa2 in stringa1;
- strcat(stringa1, stringa2) concatena stringa2 a stringa1;
- strcmp(stringa1, stringa2) confronta stringa2 con stringa1, se sono uguali restituisce 0, se stringa1 è maggiore di stringa2 un valore positivo, altrimenti un valore negativo;
- intero = atoi(stringa) converte una stringa in un intero;
- reale = atof(stringa) converte una stringa in un valore in virgola mobile double;
- intero = strlen(stringa) conta il numero di caratteri di una stringa;

Strutture

- Spesso gli algoritmi lavorano su insiemi di dati eterogenei
- Per esempio, per descrivere le caratteristiche di uno studente potremmo utilizzare le variabili *classe*, *sezione*, *cognome*, *nome* e *votoFinale*, rispettivamente di tipo *int*, *char*, *array di char (stringa)* e *float*.
- Sarebbe più naturale utilizzare una sola variabile in cui sia possibile memorizzare tutti gli elementi dell'aggregazione.
- E' quindi necessario definire un nuovo **tipo dato**, che è un'**aggregazione di tipi dati eterogenei**.
- In informatica spesso si fa riferimento a un'aggregazione di dati di tipo eterogeneo con il termine **record** (registrazione) e ai suoi elementi con il termine **campo del record**.
- Nel linguaggio C/C++ un'aggregazione di dati di tipo eterogeneo è una **struttura** e ogni suo elemento un **membro** della struttura.

Dichiarazione di un nuovo tipo di dato

```
studente = Record
  classe Di Tipo Intero
  sezione Di Tipo Carattere
  cognome Di Tipo Stringa
  nome Di Tipo Stringa
  votoFinale Di Tipo Reale
FineRecord
```

```
struct  studente
{
  int   classe;
  char  sezione;
  char  cognome[15];
  char  nome[15];
  float votoFinale;
};
```

- La dichiarazione di un tipo di dato è simile nei vari linguaggi (tranne per piccole differenze sintattiche)
- La dichiarazione non alloca spazio in memoria e non definisce alcuna variabile.
- Una volta dichiarato il tipo studente si possono definire variabili di quel tipo:
  `struct studente iscritto;`
- *iscritto* è una **variabile strutturata**, composta da cinque parti: una di tipo int, classe, una di tipo char, sezione, due di tipo stringa (array di char), cognome e nome, una di tipo float, votoFinale.
- La dichiarazione di una variabile di tipo struttura alloca la memoria necessaria a contenere i suoi elementi, ciascuno in base al proprio tipo di dato.

Linguaggio C - sintassi

- Dichiarazione di un nuovo tipo di dato:

```
struct struttura
{
tipo1 membro1;
tipo2 membro2;
...
tipo3 membroN;
};
```

- Dichiarazione di variabili

```
struct struttura variabile1, variabile2.. variabileN;
```

- Esempio

```
struct studente iscritto, a, b;
```

Linguaggio C – sintassi alternativa

- Dichiarazione di un nuovo tipo di dato e di variabili

```
struct struttura
{
tipo1 membro1;
tipo2 membro2;
...
tipo3 membroN;
} variabile1, variabile2.. variabileN;
```


Uso delle strutture

- Per fare riferimento ai campi di una variabile di tipo struttura si utilizza l'**operatore punto**.
- Sintassi:
  `variabileStruttura.membro`
- Esempi:

```
iscritto.classe = 3;
iscritto.sezione = 'E';
iscritto.cognome = "Benvenuti";
iscritto.nome = "Francesca";
iscritto.votoFinale = 9;
```


typedef

- La parola chiave **typedef** (*type definition*: definizione di tipo) consente di creare **sinonimi** (alias) dei tipi dati.
- Esempio

```
typedef char carattere;
typedef int intero;
```

- creano gli alias `carattere` del tipo `char` e `intero` del tipo `int` che possono essere utilizzati nelle dichiarazioni:

```
carattere pausa;
intero i;
```

- typedef può essere utilizzato con le strutture:

```
typedef struct studente
{
     int classe;
     char sezione;
     char cognome[15];
     char nome[15];
     float votoFinale;
} tipoStudente;
```

- che permette poi di utilizzare la dichiarazione:

```
tipoStudente iscritto;
```


Strutture annidate

```
struct residenza
{
     char indirizzo[35];
     char citta'[30];
     char prov[2];
     char telefono[15];
};

struct studente
{
     int classe;
     char sezione;
     char cognome[15];
     char nome[15];
     struct residenza recapito;
     float votoFinale;
};
```

- La struttura `studente` ha il membro `recapito` di tipo struttura `residenza`.
- Se `qualificato` è una variabile di tipo `studente`, per fare riferimento al numero di telefono dello studente si scrive
  `qualificato.recapito.telefono`

Array di struct

- Un **array di struct** rappresenta un insieme di entità dello stesso tipo, per esempio tutti gli studenti di un istituto.
- Si ottengono oggetti che rappresentano aggregazioni omogenee (gli array) di oggetti eterogenei (le strutture.

```
struct studente istituto[100];
```

- L'esempio definisce un array composto da 100 elementi omogenei di tipo studente