Corso di Laurea triennale in Ingegneria Chimica

in condivisione con

Corso di Laurea triennale in

Ingegneria Navale e Scienze dei Materiali

Elementi di Informatica

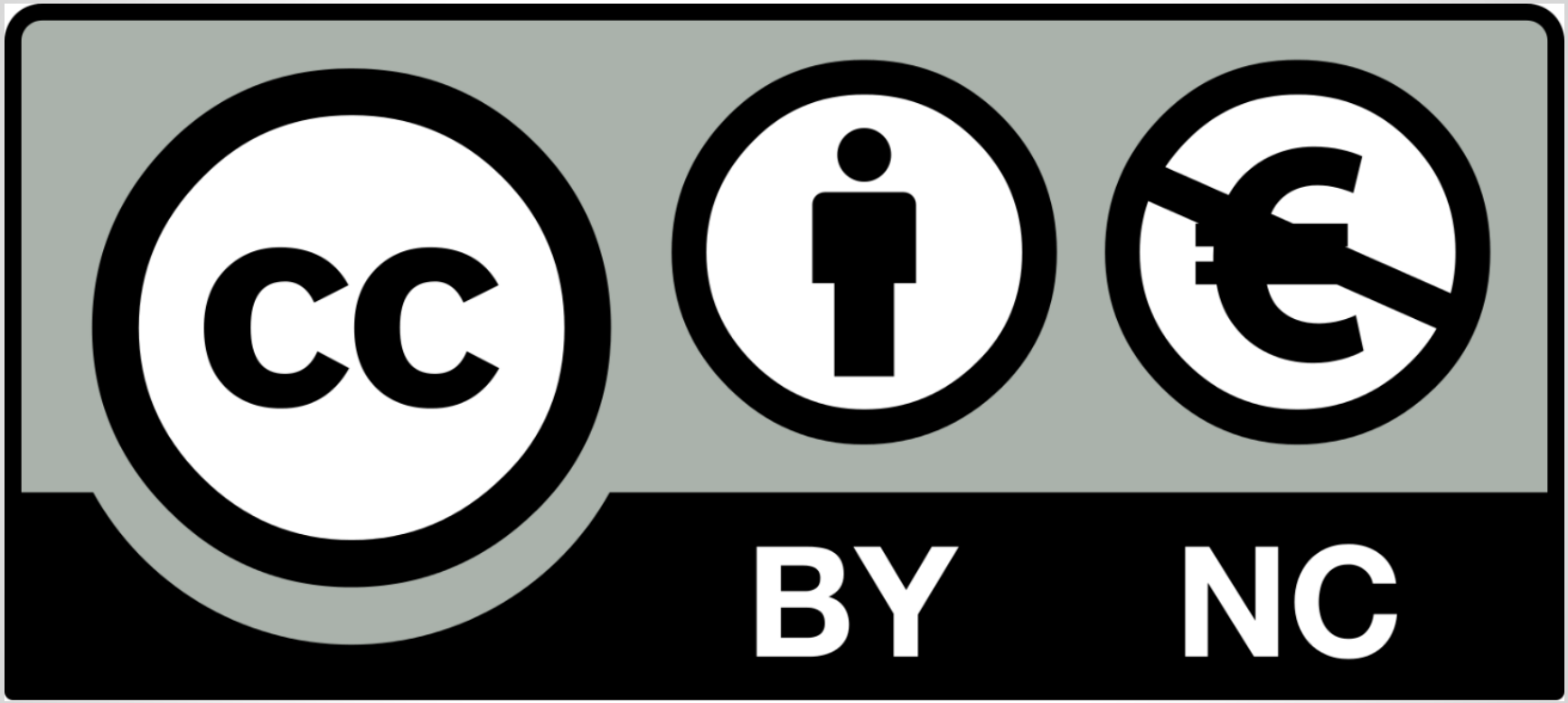
A.A. 2016/17

prof. Mario Barbareschi

Tipi di dati Strutturati

Informazioni di Licenza

- Questo lavoro è licenziato con la licenza Creative Commons BY-NC



- Per consultare una copia della licenza visita: http://creativecommons.org/licenses/by-nc/3.0/legalcode

Gli array: dichiarazione

- Un array è una **struttura composta** da un insieme di **elementi tutti dello stesso *tipo*** e con lo stesso *nome*. Il numero di elementi dell'array è detto **dimensione** ma anche *cardinalità* dell'array.
- Per dichiarare un array si devono specificare il tipo dei dati in esso contenuti, il nome dell'array e la sua cardinalità

```
<Tipo> nome_array[dimensione];
```

- La dimensione dell'array deve essere un **valore costante** per consentire al compilatore di fissarne l'allocazione in memoria prima della esecuzione del programma. Gli elementi vengono allocati in locazioni di memoria consecutive. La dichiarazione è senza inizializzazione, quindi il contenuto degli elementi dell'array v non è definito.
- **Esempio**:

  int vettore[5];

  – Indica un array di 5 locazioni di tipo intero, non inizializzate
  – l'etichetta *vettore* corrisponde all'indirizzo del primo elemento dell'array

Gli array: inizializzazione

- L' inizializzazione di un array si effettua indicando i valori degli elementi tra parentesi graffe separati da virgole
- **Esempio**
  - float v[3] = {0.5, -13.25, 12.2};
  - char lettera[5]={'a', 'b', 'c', 'd', 'e'};
  - int vett[3] = {0}; //inizializza tutti i valori a zero

Gli array: accedere agli elementi

- Per accedere ad uno specifico elemento dell'array si usa il nome dell'array e la posizione dell'elemento fra parentesi quadre
- Gli elementi di un array **partono dalla posizione 0**;
  - se N è la cardinalità, gli elementi hanno indice fra 0 e N-1
- Per indicare la posizione di un elemento dell'array si possono usare valori costanti, espressioni o variabili, ma tutti devono essere di tipo intero

Esempio:

- **vettore[3]** indica l'elemento in posizione 3 (il quarto) dell'array *vettore*
- **vettore[i]** indica l'elemento in posizione i-esima
- **vettore[c+3]** indica l'elemento in posizione c+3

Gli array multidimensionali

- Si possono anche dichiarare array multidimensionali:

```
<Tipo> nome_array[dim1][dim2]…[dim3]
```

- Essi sono strutture in cui sono necessarie più posizioni per identificare un elemento (es. matrici)

- **Esempio**
  - int mat[2][3]; //dichiara una matrice mat di 2 righe e 3 colonne
  - Int elem = mat[i][j] //indica l'elemento di posto (i,j) in mat
- Anche gli array bidimensionali vengono disposti in locazioni di memoria consecutive. In particolare vengono disposte le **righe una dopo l'altra**
- **Esempio**
  - int mat[2][3] = {1,2,3,4,5,6}; //si può scrivere anche
  - int mat[2][3] = {{1,2,3},{4,5,6}};
  - int mat[3][3] = {0}; //inizializza tutti i valori a zero

Definire un nuovo tipo array

- Con **typedef** è possibile definire nuovi tipi a partire da quelli semplici e strutturati
- ***Esempio 1:***
  - **#define N 10**
  - **typedef float arrayDiReali[N];**
- Le due istruzioni precedenti definiscono un **nuovo tipo** che si chiama ***arrayDiReali*** con cui è possibile dichiarare delle variabili:
  - **arrayDiReali vettore;** //vettore è un array di N elementi di tipo float
- ***Esempio 2*** (multidimensionali)
  - **#define N 2**
  - **#define M 3**
  - **typedef int matrice[N][M];**
  - **matrice mat**; //istanzia una variabile di tipo matrice, ossia un array bidimensionale 2x3 di interi

Esercizio Matrice

- Codificare un programma in C/C++ che effettua:
  - Il calcolo del determinante di una matrice quadrata NxN
    - int determinante(int matrice[][], int N);
  - Collaudare la funzione con matrici test di 2x2, 3x3, etc.
- Sviluppare inoltre una funzione per la stampa a video di una matrice NxM
  - void stampaMatrice (int matrice[][], int N, int M)

Le stringhe di caratteri

- Una stringa è una sequenza di caratteri che può essere definita come array di **char**

```
char nome_stringa[dim];
```

- La stringa viene conclusa con il **carattere terminatore '\0'** (ASCII NULL) che occupa la posizione *dim-1;*
  - Sovente, a '\0' ci si riferisce col nome **carattere tappo**;
- Una stringa può essere **inizializzata come un qualsiasi array** elencando i caratteri che devono farne parte, incluso il terminatore
- Esempio:
  - char stringa[10]={'c', 'i', 'a', 'o', '\0'}; //*oppure*
  - char stringa[10]="ciao"; //*in questo caso il terminatore viene aggiunto automaticamente in fondo*

char stringa[]="ciao"; *in questo caso la dimensione viene determinata automaticamente dalla lunghezza della costante stringa più uno*

Attenzione! Se si creano stringhe più lunghe di quanto dichiarato, non ci saranno messaggi di errore ma il programma invade aree di memoria non previste!

Le stringhe di caratteri: input e output

- L'inizializzazione e le procedure di input e output possono essere effettuate sull'intera stringa invece di dover operare su un carattere alla volta:
- **Esempio**
  - **char nome[11]=”Ciao Mondo";**
  - **nome=”Informatica";**
  - **cout<<nome;** *//stampa tutti i caratteri fino al terminatore*
  - **cin>>nome;** *//carica nel vettore nome tutti i caratteri immessi da tastiera e aggiunge automaticamente il terminatore*

La libreria <string.h>

- Il linguaggio mette a disposizione la libreria **string.h** in cui sono predefinite le funzioni di gestione delle stringhe.
- Per usare la libreria, è necessario includerla con la direttiva:

```
#include <string.h>
```

- La libreria mette a disposizione, oltre che alla classe String, molte funzioni per la manipolazione e trattamento di array di caratteri:

Nome Funzione	Utilità
strlen	Calcola la lunghezza della stringa
strcpy	Copia una stringa in un'altra
strncpy	Copia n caratteri da una stringa ad un'altra
strcat	Accoda una stringa ad un'altra
strncat	Accoda n caratteri di una stringa ad un'altra
strcmp	Compara due stringhe
strncmp	Compara i primi n caratteri di due stringhe
strchr	Cerca un carattere in una stringa
strstr	Cerca una stringa in un'altra

Esercizi Stringhe

- Codificare in C/C++ le funzioni (ricorrendo al costrutto while oppure do while):
  - int strlen(char c[]);
  - void strcpy(char c1[], char c2[]);
  - int strcmp(char c1[], char c2[]);
- Convertire le funzioni precedenti in funzioni ricorsive, ricordando che ciascuna chiamata ricorsiva deve specificare l'indice del carattere considerato:
  - int strlen(char c[], int i);
  - void strcpy(char c1[], char c2[], int i);
  - int strcmp(char c1[], char c2[], int i);

I record: dichiarazione

- Il **record**, detto nel **struttura**, è una collezione di elementi a cui viene dato un unico nome;
- Gli elementi della struttura sono detti *campi* od anche *membri*, possono essere tutti di uno stesso tipo, ma anche di tipo diverso;
- La dichiarazione della struttura prevede che ne venga fissato il nome e che ne siano elencati i campi specificando per ognuno di essi il tipo di appartenenza.

```
struct nome_record{
        <tipo_campo1> nome campo1;
        <tipo_campo2> nome campo2;
        …
        <tipo_campoN> nome campoN;
}; //Ricordare punto e virgola!
```

I record: definizione di variabili

- Per la definizione di variabili si usa il nome del tipo seguito dai nomi delle variabili come di consueto:

```
Struct nome_record record1, record2;
```

```
struct nome_record record = {valore 1, valore 2, …, valoreN};
```

- Si può anche usare un'unica dichiarazione di tipo e di variabili come l'esempio seguente mostra.
  - In una tale definizione si può anche omettere il nome della struttura introducendo un tipo struct senza nome non più utilizzabile in seguito.

```
struct nome_record{
        <tipo_campo1> nome campo1;
        <tipo_campo2> nome campo2;
        …
        <tipo_campoN> nome campoN;
} record 1, record2;
```

I record: accedere ai campi

- Una volta definita una variabile di tipo struct, si può solo operare con i suoi campi accedendo ad essi indicandone il nome preceduto dal nome del record adeguatamente separato mediante un punto (***dot notation***).
- Esempio:

```
nome_record.nome_campo
```

```
struct tipo_data{
        int giorno;
        char mese[15];
        int anno;
} data_esame;

data_esame.giorno = 19;
data_esame.mese = “Dicembre”;
data_esame.anno = 2016;
```

Definire i record come tipo

- Così come accade per gli array, anche i record possono essere utilizzati per definire un nuovo tipo da poter utilizzare all'interno del programma;

```
typedef struct tipo_data{
        int giorno;
        char mese[15];
        int anno;
} Data;

Data data_esame = {19, "Dicembre", 2016};
```

- In alternativa:

```
struct tipo_data{
        int giorno;
        char mese[15];
        int anno;
};
typedef struct tipo_data Data;
Data data_esame = {19, "Dicembre", 2016};
```

I record: osservazioni

- Due dichiarazioni struct, anche se comprendenti gli stessi campi, introducono variabili che vengono considerate di tipo **diverso**;
- Sono dello stesso tipo le variabili elencate nella stessa dichiarazione struct;
- Sui record le **uniche operazioni ammesse** sono quelle **definite sui singoli campi** e la ***copia* di un intero record** in un altro dello stesso tipo.
- Esempio:

```
Data data, esameEI, esameAM;
cout<<“Inserisci giorno: ”;
cin>>data.giorno;
cout<<“Inserisci mese: ”;
cin>>data.mese;
cout<<“Inserisci anno: ”;
cin>>data.anno;
esameEI=data;
data.giorno++;
esameAM=data;
```

Esercizio Struct

- Codificare un programma in C/C++ che dichiari un tipo Contatto come record dei seguenti campi
  - char nome[15]; //Nome
  - char cognome[20]; //Cognome
  - char telefono[30]; //Telefono
  - bool preferito; //TRUE se è un contatto preferito
  - int posizione; //Posizione all'interno della rubrica
- Creare due funzioni per la popolazione di un record e per la stampa:
  - void inserisciContatto(Contatto &cont) //N.B. passaggio per riferimento
  - void stampaContatto(Contatto cont)