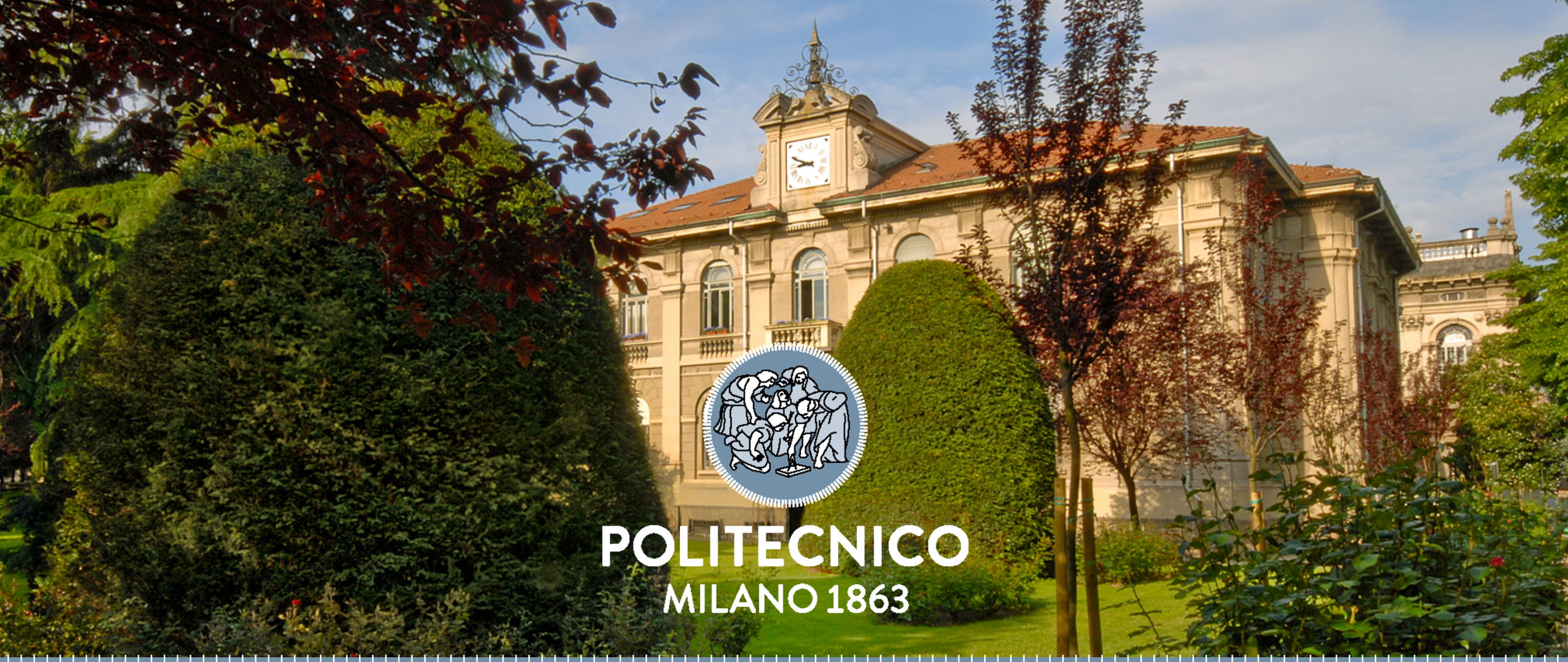

INFORMATICA A

A.A. 2017-18

Laboratorio n°4

Ing. Gian Enrico Conti

Dott. Michele Zanella

Info Logistiche (cont'd)

- Calendario laboratori

Data	Orario	Squadra	Aula	Resp.	Programma
23/10/2017	9:00 - 12:00	A	CS 0.2	Zanella	Lab 1
23/10/2017	9:00 - 12:00	B	CS 1.4	Conti	
26/10/2017	14:15-17:15	C	CS 0.11	Conti	
30/10/2017	9:00 - 12:00	A	CS 0.2	Zanella	Lab 2
30/10/2017	9:00 - 12:00	B	CS 1.4	Conti	
02/11/2017	14:15-17:15	C	CS 0.11	Conti	
06/11/2017	9:00 - 12:00	A	CS 0.2	Zanella	Lab 3
06/11/2017	9:00 - 12:00	B	CS 1.4	Conti	
09/11/2017	14:15-17:15	C	CS 0.11	Conti	
20/11/2017	9:00 - 12:00	A	CS 0.2	Zanella	Lab 4
20/11/2017	9:00 - 12:00	B	CS 1.4	Conti	
23/11/2017	14:15-17:15	C	CS 0.11	Conti	
27/11/2017	9:00 - 12:00	A	CS 0.2	Zanella	Lab 5
27/11/2017	9:00 - 12:00	B	CS 1.4	Conti	
30/11/2017	14:15-17:15	C	CS 0.11	Conti	
04/12/2017	9:00 - 12:00	A	CS 0.2	Zanella	Lab 6
04/12/2017	9:00 - 12:00	B	CS 1.4	Conti	
14/12/2017	14:15-17:15	C	CS 0.11	Conti	

Fasi Progetto

Nella cartella *Step Progetto* di Beep:

1. **Step00 – Strutture e tipi**: Definizione le strutture e i tipi utilizzati
2. **Step00b – Header files:** Spostamento delle definizioni in un file *header* esterno
3. **Step01 – Inserimenti**: Implementazione delle funzioni basiche di inserimento di aeroporti, aerei, passeggeri e voli
4. **Step02 – Struttura menù**: Creazione della struttura generale del menù di selezione
5. **Step03 – Menù:** Implementazione di alcune funzioni del menù
6. **Step04 – Prototipi**: Creazione dei prototipi di funzioni per le operazioni da implementare
7. **Step05 – Funzioni**: Implementazione delle funzioni
8. **Step06 – File I/O**: Implementazione delle funzioni per salvare i dati su .csv o .JSON

Header files

- Vengono utilizzati per spostare le definizioni delle variabili globali e dei tipi in un file esterno:
    - Possibilità di riutilizzare strutture dati in più file (es., librerie)
    - Migliorano la leggibilità del codice (meno righe nei sorgenti .c)
- Vengono salvati con un formato specifico: `my_header.h`
- Vengono inclusi nei file .c tramite la `#include "my_header.h"`

- Per il progetto creiamo un file `define_and_typedefs.h`

- Negli header files posso anche inserire dei **prototipi** delle funzioni, ossia una loro dichiarazione fatta senza specificare il corpo della funzione stessa.

Funzioni

- Hanno una "firma" (*signature*):
  - Tipo dell'output
  - Nome univoco
  - Parametri di input

```c
int my_function(int a)
```

- Generalmente terminano con la restituzione, al *chiamante*, del risultato della computazione, attraverso l'istruzione **`return`**.
  - Esistono anche funzioni che non restituiscono nulla: quelle di tipo **`void`** (*procedure*)

```c
int somma(int a, int b){
    int s = a + b;
    return s;
}
```

- Vengono utilizzate per riutilizzare parti di codice e per migliorarne la leggiblità.
- **In C non è possibile utilizzare una funzione prima che sia dichiarata!**

Puntatori

- I **puntatori** sono delle variabili che contengono l'indirizzo di memoria di un'altra variabile

```
int* puntatore;
```

- "*" è chiamato **operatore di deferenziazione** e restituisce il contenuto dell'oggetto puntato dal puntatore; mentre l'operatore "&" restituisce l'indirizzo della variabile.

```
int var=0, var2, ind;

int* puntatore;

puntatore = &var; //puntatore punta a var
var2 = *puntatore; //var2 == var
ind = puntatore; //ind contiene l'indirizzo di var
*puntatore = 5 //var == 5
```

Operazioni sui puntatori

- È possibile effettuare operazioni sui puntatori.
  - Spostamento in avanti: *p* + *i*, sposta il puntatore di *i* posizioni avanti
  - Spostamento indietro: *p* – *i*, sposta il puntatore di *i* posizioni indietro
- **Un puntatore esiste sempre in funzione del tipo di oggetto puntato.**
  - Un puntatore ad *int* ha un blocco di memoria di 4 byte, a *char* di 1 byte, ecc…
  - Se sommo al puntatore un intero o utilizzo l'operatore **++,** il puntatore si sposterà in avanti di tanti byte quanti ne prevede il tipo di variabile puntata.
    Ad esempio: *p* + *i* equivale a *posizione* + *(i*dimensione tipo puntato)*
- Se ad una funzione passo dei puntatori eventuali operazioni su questi parametri saranno effettuate sulle variabili originali!
- È possibile agire sugli array come se si stesse agendo su un puntatore poiché entrambi sono blocchi contigui di memoria. **Attenzione:** devo comunque assegnare la prima cella dell'array ad un puntatore.
- Posso utilizzare i puntatori per puntare a strutture, in questo caso per accedere ai campi della *struct* utilizzo l'operatore **"->"**

Puntatori e tipi

- Puntatori ed array

```c
char stringa[20];
char* pointer;

pointer= &stringa[0]; //pointer punta a stringa[0]
pointer++; //Ora pointer punta a stringa[1]
```

- Puntatori e struct

```c
struct punto {
     int x;
     int y;
} puntoA;

struct punto* pointer;

pointer = &puntoA;

pointer->x = 6; //Assegno il campo x
pointer->y = 7; //Assegno il campo y
```

File I/O

- I file rappresentano una sequenza di byte, siano essi *file di testo* o *file binari*.
- La fine del file è espressa dal carattere speciale **EOF**.
- Vengono rappresentati tramite una struttura **FILE**
- Nella manipolazione dei file si utilizza il puntatore a tale struttura: `FILE* fp;`
- La libreria <stdlib.h> mette a disposizione una serie di funzioni per operare sui file:
  - `FILE* fopen(const char* filename, const char* mode)`: crea o apre un file, specificando il nome del file e la modalità di apertura.
    - **r:** lettura
    - **w:** scrittura
    - **a:** writing appending mode
  - `int fputc(int c, FILE* fp)`: scrive il caratter *c* nel file puntato da *fp* tramite un *output stream*
  - `int fputs(const char* s, FILE *fp)`: scrive una stringa *s* nel file puntato da *fp*
  - Altre simili a quelle di console: `fwrite`, `fprintf`...

File I/O

- Altre funzioni
  - `int fgetc(FILE *fp);` leggere un carattere dal file puntato da *fp*
  - `char* fgets(char* buf, int n, FILE *fp);` legge fino a *n-1* caratteri dal file puntato da *fp*. La funzione si ferma se incontra un '\n' o la fine del file.
  - Alte simili a quelle di console: `fread`, `fscanf`…
  - `int fclose(FILE *fp);` Chiude un file ed esegue un flush sui buffer di scrittura/lettura.

- La sequenza per operare sui file è:
  1. fopen(…)
  2. freaed/fgets oppure fwrite/fputs
  3. fclose
- In caso di problemi in apertura il puntatore *fp* = NULL

Esercizio 1: Distnza punti

Si implementi la funzione che permetta di stabilire, dati due punti, quali dei due è più distante dall'origine.

- I tipi definiti «a mano» possono essere usati come tutti gli altri tipi nelle funzioni.

Esercizio 2: Fattoriale

Si implementi una funzione *fact()* che, presi in ingresso un intero n, ne restituisca il fattoriale.

- Il fattoriale di un numero n ("n!") è la produttoria di tutti gli interi da 1 ad n (compreso):
- Si utilizzi un semplice ciclo *for*.
- Ad esempio:

  5! = 120

  0! = 1

  1! = 1