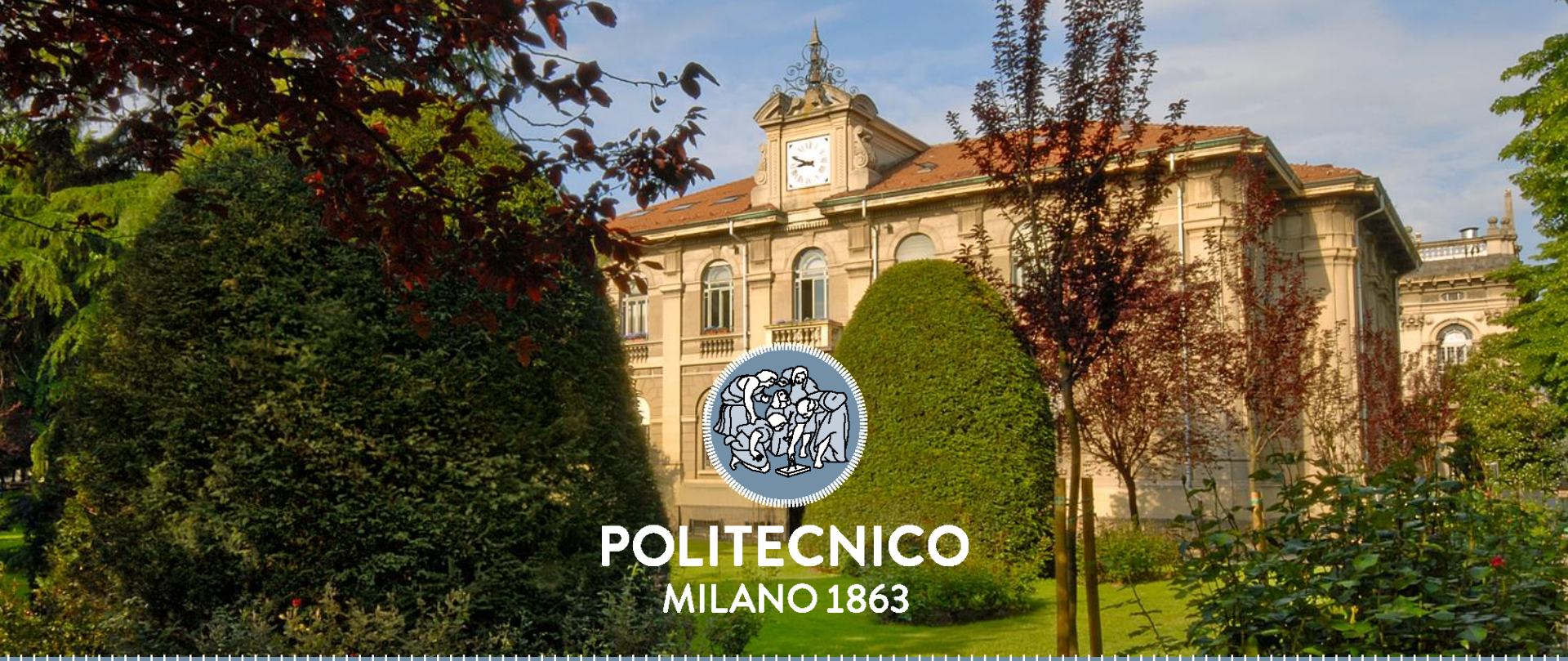

INFORMATICA A

A.A. 2017-18

Laboratorio n°2

Ing. Gian Enrico Conti

Dott. Michele Zanella

HEAP LAB

Info Logistiche

- Contatti:
  - gianenrico.conti@mail.polimi.it
  - michele.zanella@polimi.it
    - HEAP Lab – Campus Leonardo, via Golgi 39, Edificio 21, Piano 1, Ufficio 4, +39 02 2399 **9613**
      (mandatemi una mail per accordarci su giorno e ora)
- Tutor: Niccolò Izzo: niccolò.izzo@mail.polimi.it
- Sito web del corso:
  - https://beep.metid.polimi.it
- Repository con i sorgenti delle soluzioni:
  - https://github.com/Fixfire/Info--A
- **Nota per le mail:**

  Oggetto: *[INFO A] il vostro oggetto*

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	
03/11/2017	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	

Cicli

- **For**

```
for(init-expr; test-expr; increment-expr) {
    statement
}
```

  - *Init-expr:* dichiara ed inizializza uno o più contatori
  - *Test-expr:* se viene soddisfatta (true) allora viene eseguita la *statement*, altrimenti il ciclo termina
  - *Increment-expr:* viene eseguita alla fine di un'iterazione e incrementa uno o più contatori.

```
for(int i=0; i<2; i++)
    printf('%d\t',i);
```

  - **Risultato: *0 1***

Cicli (cont'd)

- **While**

```
while(condition) {
    statement
};
```

La *statement* viene eseguita finchè la condizione è verificata

- **Do….while**

```
do {
    statement
}while(condition);
```

La condizione viene verificata alla fine, la *statement* è eseguita almeno una volta.

Condizioni

- **if**

```
if(boolean_expression){
    statement
}
```

Lo *statement* viene eseguito se l'espressione è verificata

- **If…else**

```
if(boolean_expression) {
    statement1
} else {
    statement2
}
```

Lo *statement1* viene eseguito se l'espressione è verificata **altrimenti** viene eseguito lo *statement2*

Condizioni

- **Switch**

```
switch(variable) {
    case value: statement1;
    …
    default: default_statement;
}
```

La variabile viene verificata nei diversi casi, se per uno è vero ne viene eseguito il relativo *statement*. Altrimenti viene eseguito il caso di *default*.

Stringhe

- Le stringhe sono una sequenza di caratteri. In C esse sono definite come array di *char.*

```
char stringa[50];
```

- Per operare sulle stringhe esiste una libreria *string.h*
- Per stampare una stringa si utilizza lo specificatore di tipo *%s*.
- Per accedere ad un carattere della stringa basta accedere alla posizione nell'array -> `stringa[0]` indica il primo carattere della stringa.
- La stringa viene terminata con il *carattere terminatore \0*

Stringhe: string.h

- *strlen(stringa)*: ritorna la lunghezza della stringa (escluso il carattere terminatore)
- *strcmp(stringa1, stringa2)*: compara le due stringhe, r
  - *< 0 : se stringa1* è alfabeticamente precedente alla *stringa2*
  - *= 0 :* se *stringa1* e *stringa2* sono uguali
  - *> 0 :* se *stringa2* è alfabeticamente precedente alla *stringa1*
- *strcat(stringa_dest, stringa_src):* appende la *stringa_src* alla fine di *stringa_dest*
- *strcpy(stringa_dest, stringa_src)*: copia la *stringa_src* nella *stringa_dest*

Esercizio 1: Parentesi

Scrivere un programma che richiede all'utente una sequenza di caratteri (stringa) che non contiene spazi, li registra in una struttura dati appropriata e verifica se la sequenza è ben parentesizzata".

- Una sequenza si dice ben "parentesizzata" se le parentesi, per semplicità consideriamo solo quelle tonde, sono aperte e chiuse correttamente.
- Ad esempio sono ben parentesizzate" le sequenze:

  Bla

  (bla)

  (bla(bla))

  (bla()(bla)())

- **Hint:** si tenga conto man mano di quante sono le parentesi già aperte, e si decida di conseguenza. Si arresti la scansione non appena si scopre che la sequenza è illegale, anche prima di arrivare alla fine.

Esercizio 2: Alfabeto farfallino

Scrivere un programma che legge da *stdin* una sequenza di caratteri e stampa su *stdout* una sequenza derivata dalla precedente secondo le regole dell'alfabeto farfallino (i.e., ogni vocale viene raddopiiata inserendovi in mezzo una 'f').

Esempio:

quanto mi piace questo corso!

*qu**fu**a**fa**nto**fo** mi**fi** pia**fi**a**fa**ce**fe** qu**fu**e**fe**sto**fo** co**fo**rso**fo**!*

Hint:

- Considerare solo lettere minuscole
- Scandire un carattere alla volta

Varianti:

- La frase sia trascritta interamente in lettere maiuscole -> vedere funzione *toupper()*.

Esercizio 3: Triangolo di Floyd

Scrivere un programma che legge un numero intero e stampa video le prima *n* righe del triangolo di Floyd (i.e, un triangolo rettangolo che contiene tutti i numeri naturali disposti come segue).

Esempio per *n=10*:

1

2 3

4 5 6

7 8 9 10

…

Hint: la soluzione più immediata utilizza **due cicli annidati**.

Variante: Si progetti anche una soluzione che utilizza **un solo ciclo**.

Esercizio 4: Sottosequenza

Scrivere un programma che legge una sequenza (di lunghezza arbitraria) di numeri interi positivi, temrinata da 0, e indica, alla fine della sequenza, qual è la lunghezza della massima sottosequenza di numeri consecutivi in ordine crescente.

Esempio:

13 3 8 4 5 1 17 0

Lung. Max = 2

21 19 18 14 9 6 4 3

Lung. Max = 1

2 1 3 6 8 0 1 12

Lung. Max = 4

Hint: creare una variable «di stato» che ad ogni iterazione tenga conto della lunghezza massima raggiunta finora ed eventualmente si aggiorni se si incontra una sequenza più lunga.