INFORMATICA (PER AEROSPAZIALI)

A.A. 2018-19

Laboratorio n°3

Dott. Michele Zanella

HEAP LAB

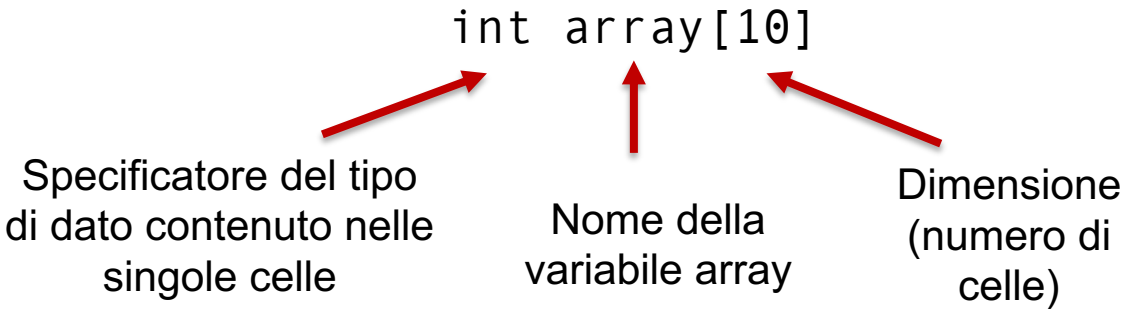
Info Logistiche

- Contatti:
  - 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)
- Sito web del laboratorio:
  - Pagina del corso BeeP (Cartella Laboratori)
  - http://zanella.faculty.polimi.it/teaching/informatica-aero
- **Nota per le mail:**

  Oggetto: *[INFO-AERO] il vostro oggetto*

Array

- Sono dei contenitori formati da una serie di celle in numero fissato contenenti valori di un certo tipo (int, long, char, …).



- Accesso agli elementi degli array, se la sua dimensione è 10:
  - Primo elemento: `array[0];`
  - *i*-esimo elemento: `array[i];`
  - Ultimo elemento: `array[9];` ← **ATTENZIONE!**

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 seguente 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*

Puntatori

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

```
int* nome_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

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

pointer= &stringa[0]; //pointer punta a stringa[0]
pointer=stringa;      //Uguale alla riga sopra
pointer++;            //Ora pointer punta a stringa[1]
```

- Nel dettaglio

```
int a[20]; // a è puntatore alla prima cella dell'array => a[0]
           // a[1] = *(a+1)
           // &[1] = (a+1)

scanf("%d", (a+1))     // &a[1] = &*(a+1)=(a+1)

for(int i=0; i<20; i++){
    printf("%d", a[i]); // Le due printf stampano la stessa cosa
    printf("%d", i[a]); // a[i] = *(a+i) = *(i+a) = i[a]!!!
}
```

Puntatori e tipi

- Puntatori e struct

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

struct punto* pointer;

pointer = &puntoA; //pointer punta a
puntoA

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

Funzioni

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

```
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`

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

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

Funzioni

- Nel codice le funzioni per essere utilizzate devono essere **invocate** attraverso il loro nome e passando i parametri richiesti

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

int main(){
    int ris;

    ris = somma(1,2);

    return 0;
}
```

Funzioni e puntatori

- Posso passare i parametri ad una funzione in due modi:
  - Per *copia*: copio il valore del parametro passato nella variabile locale (parametro)

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

  - Per *referenza*: passo il puntatore (indirizzo) del parametro passato. **Se modifico il puntatore deferenziato modifico anche il valore del parametro all'esterno della funzione!!**

```
void somma(int a, int b, int* ris){
    *ris = a + b; //modifico direttamente il valore di ris
}
```

Approccio per la soluzione dei problemi

1) Analisi dei requisiti, definizione del modello e dei tipi necessari
2) Stesura del Flow chart (o degli step) ad alto livello della soluzione
3) Definizione delle funzioni necessarie e organizzazione del codice
4) Implementazione e compilazione passo-passo (non aspetto alla fine di tutto!)
5) Testing

Esercizio 3.1: Riflessione stringa

Scrivere un programma che, data una stringa, calcoli la sua riflessa.

Esempio:
"roma" -> "amor"

Esercizio 3.2: 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 3.3: Rubrica del telefono

Si scriva un programma che implementi una semplice rubrica del telefono.

Requisiti:

- Possibilità di inserire/cancellare contatti
- Possibilità di stampare elenco contatti con informazioni
- Possibilità di ricercare un contatto dato il cognome

Hints:

- Riutilizzare eventuali funzioni contenuti in librerie (es. `string.h`)
- Settare un numero massimo `CONT_MAX` di contatti

Esercizio 3.3: Rubrica del telefono (cont'd)

Contatto:

Definiamo un tipo di dato personalizzato *contatto*. Il tipo deve includere le informazioni relative a:

- Cognome
- Nome
- Età
- Numero di telefono

Esercizio 3.4: Somma vettori con puntatori

Scrivere una funzione che calcoli la somma di due vettori passati come puntatori e salvi il risultato in una variabile passata come puntatore.

```
void sommavp(int* v1, int* v2, int* ris);
```

Hint:

Considerate la dimensione dei vettori come prefissata

Esercizio 3.5: Mastermind

Scrivere un programma per giocare a **Mastermind**.

http://www.webgamesonline.com/mastermind/

http://www.webgamesonline.com/mastermind/rules.php

Regole di base:

- 2 giocatori (CPU e utente)
- CPU sceglie una sequenza di colori non nota all'utente
- L'utente deve indovinare l'esatta sequenza (posizione e colore)
- Ad ogni round l'utente inserisce una possibile sequenza
- CPU risponde in questo modo:
    - Per ogni posizione e colore indovinato nella sequenza, mostra un colore nero a destra (=black peg)
    - Per ogni colore indovinato (ma posizione sbagliata) nella sequenza, mostra una colore bianco (=white peg)

Esercizio 3.5: Mastermind (cont'd)

Altre regole:

- L'utente definisce la lunghezza della sequenza da indovinare
- Nella sequenza i colori si possono ripetere
- L'utente definisce il numero massimo di round in cui indovinare la sequenza
- L'utente vince se indovina l'esatta sequenza entro il numero massimo di round, altrimenti vince CPU.

Esercizio 3.5: Mastermind (cont'd)

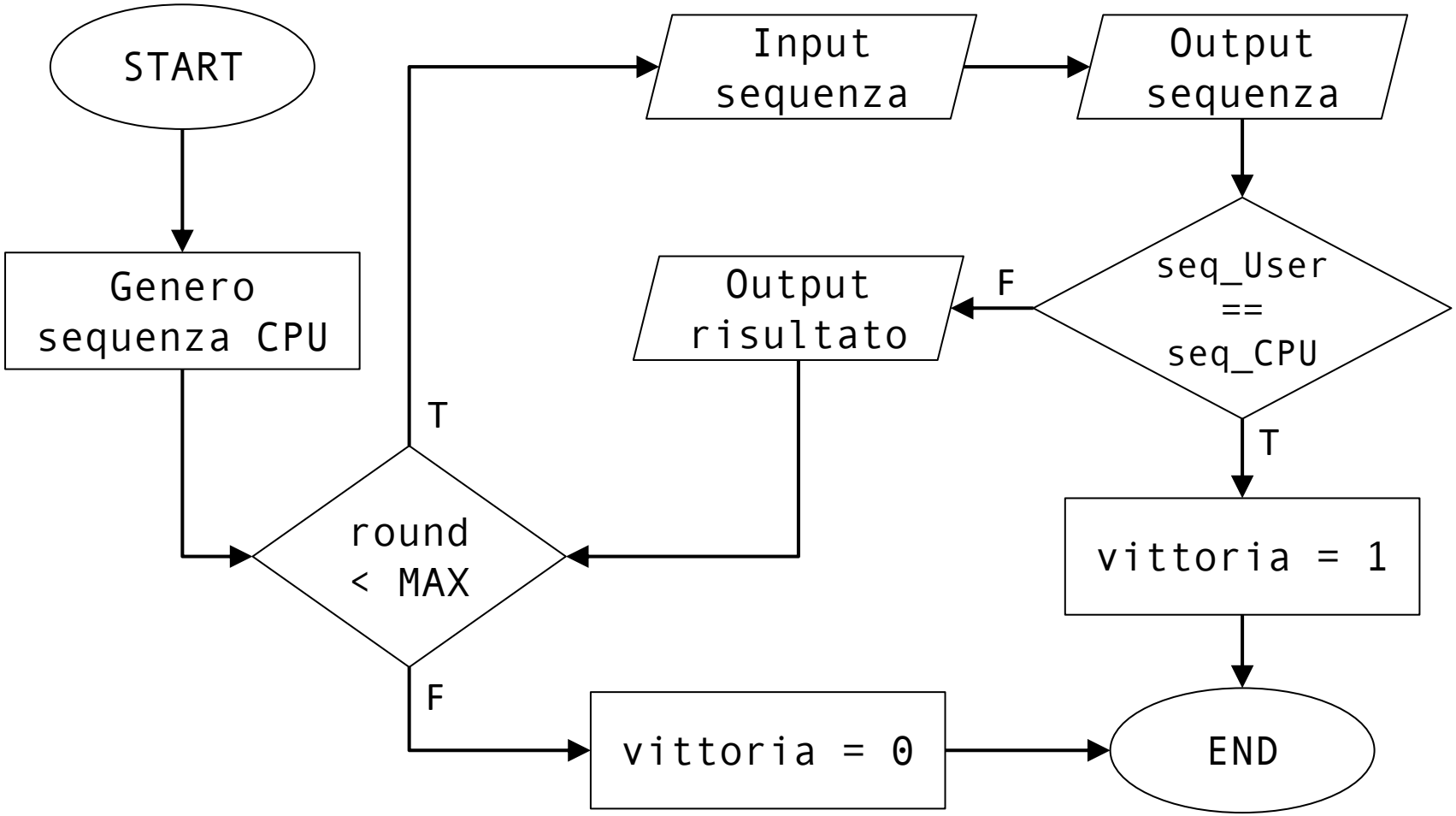
Requisiti:

- L'utente deve definire le impostazioni di gioco (lunghezza sequenza, numero massimo di round)
- CPU deve generare una sequenza casuale di colori
- L'utente deve poter inserire una possibile sequenza ad ogni round
- CPU deve mostrare la qualità del risultato del round (colori bianchi e/o neri)

Hints:

- Utilizzare solo testo: i colori vengono scritti come caratteri testuali (e.g., w = "pallino bianco")
- Definire un carattere per ogni colore per l'inserimento (e.g., 'y' = "giallo")
- 6 colori: rosso, giallo, verde, arancione, nero e bianco

Esercizio 3.5: Mastermind (Flowchart generale)



Esercizio 3.5: Mastermind (cont'd)

Altri hints:

- Utilizzare la funzione `rand()` della librearia `stdlib.h` per generare il codice CPU casuale.
- Definire ed implementare le seguenti funzioni (oltre a quelle per stampare risultato e codice):

```
void genera_codice(char* secret_code);
```

```
void indovina_codice(char* guess_code);
```

```
int controlla_codice(char* secret_code,
                     char* guess_code,
                     int* white_pegs,
                     int* black_pegs);
```

```
void stampa_codice(char* code);
```

Esercizio a casa 3.1: Sottosequenza

Scrivere un programma che legge una sequenza (di lunghezza arbitraria) di numeri interi positivi, terminata 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.

Esercizio a casa 3.2: Confronto tra stringhe

Scrivere un programma che, date due stringhe verifichi se sono uguali o in caso negativo quale viene prima in ordine alfabetico.

Il programma deve stampare:

- 0: le stringhe sono uguali
- -1: la `stringa1` è minore della `stringa2`
- 1: la `stringa2` è maggiore della `stringa2`

Esercizio a casa 3.3: Stringa palindroma

Scrivere un programma che, data una stringa (senza spazi), verifica se è palindroma.

Esempio:

"anna" -> è palindroma

"pippo" -> non è palindroma

Esercizio a casa 3.4: 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 a casa 3.5: Gara di danza

Scrivere un programma che gestisca i punteggi di una gara di danza. calcoli il punteggio dell'esecuzione per ogni atleta di una gara di danza.

Requisiti:

- Ogni atleta è caratterizzato da un codice alfanumerico e dal suo punteggio finale
- Possibilità di inserire il codice degli atleti da input
- Possibilità di inserire i punteggi dei singoli giudici da input e calcolare il risultato finale
- Possibilità di stampare tutti gli atleti con i relativi punteggi finali

Hints:

- Utilizzare le funzioni della `stdio`
- Definire numero di giudici e atleti
- Definire i tipi e le funzioni necessarie