UNIVERSITÀ
DI TORINO

Laboratorio di Programmazione I

Lezione n. 6: Funzioni

Alessandro Mazzei

Slides: prof. Elvio Amparore

Outline

- Qualche quiz sulle funzioni
- primi.c
- funzioni_iterative.c
- monete.c
- Lab06-Es1 Statistiche sequenze
- Lab06-Es2 Funzioni su arrays
- vocali.c
- Lab06-Es3 Palindroma
- Extra: Lab06-Es4 Scale e serpi

Come definire le funzioni in C

Permettono di dividere il programma in blocchi di codice separati, che svolgono compiti specifici.

Per ogni sotto-compito di un programma, ci serve:

- definire un **nome** chiaro e conciso
- definire quali **parametri in ingresso** sono necessari per svolgere il compito (*opzionali*)
- definire quali dati **restituisce** (*opzionale*)

```
tipo_di_ritorno nome_funzione(param1, param2, …) {
    <codice della funzione>
}
```

Esercizio 1: saluta l'utente

Vogliamo scrivere una funzione che scrive sul terminale un messaggio di saluto: "Buongiorno!"

- Quale **nome** usiamo?
- Quali **parametri in ingresso**?
- Quale tipo di **ritorno**?

Soluzione: saluta l'utente

Vogliamo scrivere una funzione che scrive sul terminale un messaggio di saluto: "Buongiorno!"

- Quale **nome** usiamo?
  saluta **stampa_saluto** **scrivi_saluto**
  ...
- Quali **parametri in ingresso**?
  Nessuno! → **void**
- Quale tipo di **ritorno**?
  Nessuno! → **void**
  L'output è sul terminale, non serve restituire nulla

```
void saluta(void) {
    printf("Buongiorno!\n");
}
```

Esercizio 2: determina se un numero è primo

Vogliamo scrivere una funzione che determina se un numero positivo **n** è un numero primo.

- Quale **nome** usiamo?
- Quali **parametri in ingresso**?
- Quale tipo di **ritorno**?

Soluzione: determina se un numero è primo

Vogliamo scrivere una funzione che determina se un numero positivo **n** è un numero primo.

- Quale **nome** usiamo?
  verifica_primalita **is_prime**
  test_primalita
- Quali **parametri in ingresso**?
  Il numero **n** → **int**
- Quale tipo di **ritorno**?
  Se è primo o no → **bool**

```
bool verifica_primalita(int n) {
    <determina la primalità di n>
    return <true se n è primo, false altrimenti>
}
```

Esercizio 3: cerca nell'array

Vogliamo scrivere una funzione che conta le occorrenze del numero **n** in un array di interi **arr**.

- Quale **nome** usiamo?
- Quali **parametri in ingresso**?
- Quale tipo di **ritorno**?

Soluzione: cerca nell'array

Vogliamo scrivere una funzione che conta le occorrenze del numero **n** in un array di interi **arr**.

- Quale **nome** usiamo?
  conta_occorrenze ~~**conta**~~
  numero_occorrenze
- Quali **parametri in ingresso**?
  Il numero **n** e l'array (quindi dati e dimensione)
- Quale tipo di **ritorno**?
  Il numero di occorrenze → **size_t** oppure **int**

```
size_t conta_occorrenze(int n, int dati[], size_t len) {
    <conta le occorrenze di n nell'array dati di dimensione len>
    return <il numero di occorrenze trovate>
}
```

Esercizio 4: indici di una sequenza in input

Vogliamo scrivere una funzione che legge dall'input una sequenza di numeri interi positivi, e se non ci sono errori restituisce la **media** e la **varianza**.

- Quale **nome** usiamo?
- Quali **parametri in ingresso**?
- Quale tipo di **ritorno**?

Soluzione: indici di una sequenza in input

Vogliamo scrivere una funzione che legge dall'input una sequenza di numeri interi positivi, e se non ci sono errori ritorna la **media** e la **varianza**.

- Quale **nome** usiamo?
  calcola_indici_sequenza **seq_media_varianza**
- Quali **parametri in ingresso**?
  → i puntatori ai due indici da calcolare
- Quale tipo di **ritorno**?
  Successo o errore → **bool** (oppure **int**)

```
bool calcola_indici_sequenza(int* media, int* varianza) {
    <leggi da standard input la sequenza e calcola media e varianza>
    return <non ho riscontrato errori leggendo l'input>
}
```

Verifica (esaustiva) di primalità

Come determinare se un numero naturale n>1 è primo?

Un numero **n** è primo se è divisibile soltanto da se stesso e da 1.

⇒ cerchiamo se esiste un divisore **d** di **n**, con $2 \leq d < n$

7 è primo?
diventa:

- **2** è un divisore di **7**? No.
- **3** è un divisore di **7**? No.
- **4** è un divisore di **7**? No.
- **5** è un divisore di **7**? No.
- **6** è un divisore di **7**? No.

Allora **7** è primo.

9 è primo?
diventa:

- **2** è un divisore di **9**? No.
- **3** è un divisore di **9**? **Si.**
- **4** è un divisore di **9**? -
- **5** è un divisore di **9**? -
- **6** è un divisore di **9**? -
- **7** è un divisore di **9**? -
- **8** è un divisore di **9**? -

Allora **9** non è primo.

Numeri primi

Scrivere un programma **primi.c** che implementa un metodo **verifica_primalita** che deve prendere come argomento un numero intero maggiore di 1 e restituisce un valore appropriato per indicare se è primo (qual è il tipo appropriato da ritornare?).

Usare la funzione **verifica_primalita** nel main per trovare e stampare tutti i numeri primi minori di 100:

2 3 5 7 11 13 17 19 23 29 31 37 41
43 47 53 59 61 67 71 73 79 83 89 97

Funzioni iterative

Aprire il file **funzioni_iterative.c** fornito nel codice iniziale, leggere il *main* (già completato, non da modificare) ed implementare le funzioni richieste:

- Scrivere una funzione **esponenziale** che, dati in ingresso una base **b** e un esponente **e** (come parametri), restituisce il valore di **b** elevato ad **e**. Assumere **e**≥0.
- Scrivere una funzione **prodotto_multipli** che prende come parametri tre interi non-negativi **n**, **m** e **q** restituisce il prodotto di tutti gli interi compresi tra **n** e **m** (estremi inclusi) che sono multipli di **q**.
- Scrivere una funzione **stampa_al_rovescio** che prende in ingresso un intero **n**≥0 stampa a video i valori da **n** a **0**.

Cambio monete

Scrivere un programma **monete.c** che chiede all'utente un numero naturale **cent** (interpretato come in centesimi), e successivamente stampa a video il più piccolo cambio in monete. I tagli delle monete sono:

```
#define NUM_TAGLI 6
const int tagli_monete[NUM_TAGLI] = {1,2,5,10,20,50};
```

Esempio: se l'utente chiede il cambio di **cent**=95 centesimi, il programma scriverà in output:

```
1 da 50 cent
2 da 20 cent
1 da 5 cent
```

Suggerimento: scrivere un metodo **cambio_taglio** che prende in input un **ammontare** in centesimi ed un **taglio** di una moneta, calcola quante volte può sottrarre il taglio di quella moneta dall'ammontare, stampa "*x* da *t* cent" in output, e ritorna il **resto** (l'ammontare iniziale meno x*t). Nel main chiamare **cambio_taglio** più volte, aggiornando man mano la somma rimanente.

Cambio monete

Esempio esecuzione 1:

```
Scrivere l'ammontare da convertire in monete: 120
2 da 50 cent
1 da 20 cent
```

Esempio esecuzione 2:

```
Scrivere l'ammontare da convertire in monete: 367
7 da 50 cent
1 da 10 cent
1 da 5 cent
1 da 2 cent
```

Esempio esecuzione 3:

```
Scrivere l'ammontare da convertire in monete: 473
9 da 50 cent
1 da 20 cent
1 da 2 cent
1 da 1 cent
```

Statistiche su sequenze

Sulla pagina Moodle trovate un esercizio con nome

Lab06-Es1 Statistiche sequenze

Completate il programma, assicurandovi che passi tutti i test proposti.

Quali parametri deve avere la funzione **leggi_sequenza**?
Cosa deve restituire al chiamante?

Passaggio di array

Possiamo passare un array ad una funzione approfittando del fatto che un array è (anche) un puntatore.

```
int dati[SZ] = { … };
fn(dati);
⋮
void fn(int *dati) { … }
```

oppure:

```
void fn(int dati[]) { … }
```

DOMANDA: Qual è il problema?

Passaggio di array

Qual è il problema?

La funzione **fn** non sa qual è la dimensione dell'array che riceve. Di conseguenza quando passiamo un array dobbiamo ricordarci di passare anche la sua dimensione.

```
int dati[SZ] = { … };
fn(dati, SZ);
⋮
void fn(int dati[], size_t sz) { … }
```

Passaggio di array

Suggerimento

Se i parametri in ingresso sono solo letti e non vengono scritti, possono essere dichiarati come **const**.

```
int dati[SZ] = { … };
fn(dati, SZ);
⋮
void fn(const int dati[], const size_t sz) {
   …
}
```

Funzioni su arrays

Sulla pagina Moodle trovate un esercizio con nome

Lab06-Es2 Funzioni su arrays

Completate il programma, assicurandovi che passi tutti i test proposti.

Funzioni su stringhe nella libreria del C

Il linguaggio C dispone di molte funzioni per gestire le stringhe, nell'header **<string.h>**. Ne vediamo alcune:

- `size_t strlen(const char* str)`
  Ritorna la lunghezza in caratteri (terminatore escluso) della stringa str.
- `int strcmp(const char* s1, const char* s2)`
  Compara le due stringhe in modo *lessicografico*, e ritorna:
  - un valore negativo se s1 precede s2 (in ordine lessicografico);
  - 0 se s1 è uguale, carattere per carattere, ad s2;
  - un valore positivo se s1 segue s2 (in ordine lessicografico).

  L'ordine lessicografico è l'ordine del dizionario.
- `char* strncpy(char* dest, const char* src, size_t count)`
  Copia al più count caratteri (terminatore incluso) dalla stringa src nella stringa dst.

Vocali

Scrivere un programma **vocali.c** che legge dallo standard input una parola (usare %s con scanf) e svolge le seguenti operazioni:

1. Stampa “1” se ci sono delle vocali, altrimenti stampa “0”. Per fare questo scrivere due funzioni:
   a. **is_vowel** che ritorna un booleano per determinare se un singolo carattere è una vocale. Consideriamo vocali i seguenti caratteri: `AEIOUaeiou`
   b. **esistono_vocali** che prende in input una stringa e ritorna un booleano per indicare se esistono vocali nella stringa.
2. Stampa le sole vocali presenti nella parola in input. Per fare questo, scrivere un metodo **stampa_vocali**.

DOMANDA: serve ancora passare la dimensione come per gli array?

Funzioni su stringhe

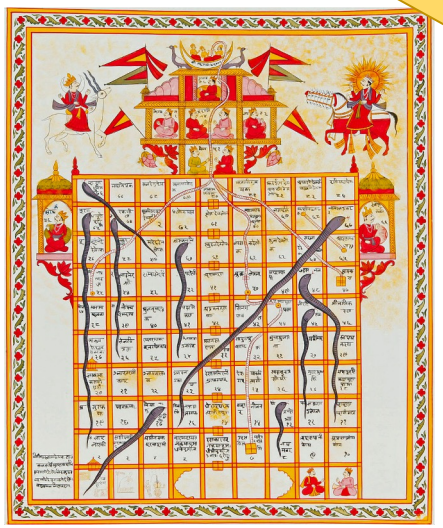
Sulla pagina Moodle trovate un esercizio con nome

Lab06-Es3 Palindroma

Completate il programma, assicurandovi che passi tutti i test proposti.

Scale e Serpi

ESERCIZIO EXTRA



Scale e serpi è un gioco da tavolo tradizionale di origine Indiana, diffuso anche in occidente in versione semplificata. È un gioco di percorso con i dadi, simile al Gioco dell'Oca.

Leggere l'esercizio su Moodle. Il codice del gioco (lancio dei dadi, posizione dei giocatori) è già parzialmente implementato. Scrivere la funzione che determina come muovere un giocatore lungo il percorso del tabellone.

Lab06-Es4 Scale e serpi