LABORATORIO DI PROGRAMMAZIONE 1
CORSO DI LAUREA IN MATEMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO

2018–2019

22.V.2019

VINCENZO MARRA

Indice

Parte 1. Input/Output standard da file 3
Esercizio 1 3
Eco di file 3
Tempo: 15 min. 3
Esercizio 2 3
Eco di file parametrico 3
Tempo: 10 min. 3
Esercizio 3 3
Copia di file 3
Tempo: 25 min. 3
Esercizio 4 3
Copia di `stdin` *su file* 3
Tempo: 20 min. 3

Parte 2. Array di puntatori 4
Esercizio 5 4
`Reverse` *dalla riga di comando* 4
Tempo: 15 min. 4
Esercizio 6 4
`Clean` *dalla riga di comando* 4
Tempo: 20 min. 4
Esercizio 7 4
Settimana anglofona 4
Tempo: 30 min. 5

Parte 3. Uso di `typedef`. 6

Parte 4. Strutture 7
Esercizio 8 7
Distanza euclidea nel piano 7
Tempo: 15 min. 7
Esercizio 9 7

Ultima revisione: 22 maggio 2019.

Operazioni sui vettori 7
Tempo: 25 min. 7
Esercizio 10 7
Indipendenza lineare 7
Tempo: 15 min. 8
Esercizio 11 8
Indipendenza affine 8
Tempo: 15 min. 8
Esercizio 12 8
Triangoli 8
Tempo: 10 min. 8
Esercizio 13 8
Area di un triangolo 8
Tempo: 25 min. 9
Esercizio 14 9
L'area è invariante per rotazioni 9
Tempo: 30 min. 9

Parte 1. Input/Output standard da file

Esercizio 1

Eco di file.

Tempo: 15 min.

Si scriva un programma che visualizzi il suo stesso codice sorgente sulla console, e termini. (*Suggerimento.* Il programma apre il file sorgente in lettura con `fopen`, e legge il contenuto del file una riga per volta con `fgets`.)

Esercizio 2

Eco di file parametrico.

Tempo: 10 min.

Si modifichi il programma dell'Esercizio 1 di modo che esso riceva come argomento dalla riga di comando un nome di file, e visualizzi il contenuto del file sulla console. Si gestiscano gli errori: cosa succede se il nome non corrisponde a un file esistente? E se manca l'argomento sulla riga di comando?

Esercizio 3

Copia di file.

Tempo: 25 min.

Si scriva un programma che riceva due nomi di file dalla riga di comando, e che copi il primo file sul secondo (carattere per carattere). Se il primo file non può essere aperto, visualizzate un messaggio appropriato. Cosa succede se il file di destinazione della copia è già esistente e contiene dei dati? Migliorate il programma facendo sì che esso chieda all'utente conferma di voler sovrascrivere il file di destinazione prima di procedere. (*Suggerimento.* Per controllare se il file esiste, potete tentare di aprirlo in lettura e testare il valore restituito da `fopen`.)

Esercizio 4

Copia di `stdin` *su file.*

Tempo: 20 min.

Si scriva un programma che riceva un nome di file dalla riga di comando, e che poi copi (riga per riga) tutto quanto l'utente digita da tastiera sul file. Il programma termina quando l'utente invia il segnale di `EOF` da tastiera, ossia quando digita CTRL+D. Si noti che se invece l'utente digita invio, sul file occorre scrivere una riga vuota costituita dal solo ritorno a capo. Gestite gli errori.

Parte 2. Array di puntatori

Esercizio 5

`Reverse` *dalla riga di comando*.

Tempo: 15 min.

Modificate la vostra soluzione all'Esercizio 1 della Lezione del 18.IV.2018 in modo che l'utente possa inserire la stringa di cui calcolare la stringa riflessa dalla riga di comando. Per esempio, se avete chiamato il codice eseguibile del vostro programma `rev`, allora digitando dalla shell

```
./rev Ailatiditalia
```

dovreste ottenere in uscita:

```
ailatiditaliA
```

Osservate che non è necessario memorizzare la stringa riflessa — basta visualizzarla.

Modificate la vostra soluzione all'esercizio in modo che il programma visualizzi un messaggio d'errore appropriato nel caso un cui non vi sia l'argomento necessario sulla riga di comando.

Esercizio 6

`Clean` *dalla riga di comando*.

Tempo: 20 min.

Modificate la vostra soluzione all'Esercizio 5 della Lezione del 18.IV.2018 in modo che l'utente possa inserire la stringa da ripulire e la stringa che indica i caratteri da eliminare dalla riga di comando. Per esempio, se avete chiamato il codice eseguibile del vostro programma `clean`, allora digitando dalla shell

```
./clean Teleologicamente eTo
```

dovreste ottenere in uscita:

```
llgicamnt
```

Osservate che per poter dimensionare correttamente, nel `main`, l'array che conterrà la stringa ripulita, dovrete calcolare la lunghezza della stringa in ingresso dalla riga di comando. Provate anche a scrivere una versione del programma che non memorizza la stringa ripulita, ma stampa semplicemente il risultato desiderato.

Modificate la vostra soluzione all'esercizio in modo che il programma visualizzi messaggi d'errore appropriati nel caso un cui vi siano troppi argomenti sulla riga di comando, o ve ne siano troppo pochi.

Esercizio 7

Settimana anglofona.

Tempo: 30 min.

Scrivete un programma che accetti in ingresso dalla riga di comando il nome di un giorno della settimana, e produca in uscita il corrispondente nome in inglese.[1]

Il programma userà una funzione di prototipo

```
int indice(char *s)
```

che accetta in ingresso una stringa, e restituisce l'indice corrispondente del giorno della settimana, da 0 a 6, oppure -1 se la stringa non è interpretabile come giorno della settimana. La funzione deve comportarsi in modo da non far differenza fra maiuscole e minuscole: se il parametro in ingresso è `lUnedi`, il valore restituito è 0.[2] (*Suggerimenti.* Convertite l'intera stringa passata in argomento in maiuscola, o in minuscola, usando le funzioni della libreria standard del C. Poi confrontate la stringa con gli elementi di un opportuno array di stringhe, inizializzato coi nomi dei giorni della settimana. Usate la funzione

```
int strcmp(char *s, char *t)
```

che restituisce 0 se, e solo se, le due stringhe in ingresso `s` e `t` sono uguali. La funzione è definita nel file di intestazione `string.h`.) Il programma userà poi un array di puntatori a carattere per produrre in uscita il nome del giorno specificato dall'utente, in inglese. Gestite gli errori in modo appropriato.

[1] I giorni della settimana in inglese sono, a partire da Lunedì: `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday` e `Sunday`.

[2] Trascurate per semplicità gli accenti finali: l'utente dovrà scrivere, per esempio, `Lunedi` e non `Lunedì` o `Lunedi'`.

Parte 3. Uso di `typedef`.

La parola chiave `typedef`

Abbiamo visto come definire nuovi tipi in C tramite l'uso delle strutture. È possibile abbreviare il nome dei nuovi tipi definiti in questo modo usando la parola chiave `typedef`. La possibilità di rinominare tipi esistenti con `typedef` non è limitata alle sole strutture, anche se è in questo contesto che il suo uso è più frequente. In generale, assumiamo che il tipo `T` sia già stato definito (o sia primitivo). Allora l'istruzione:

```
typedef T mio-nome;
```
$(*)$

stabilisce che `mio-nome` è sinomimo di `T`.[a] Il nome di tipo `T` continua a essere disponibile anche dopo l'esecuzione di $(*)$. Applicando questi fatti generali all'Esercizio 8, le righe di codice:

```
struct punto
{
    double x;
    double y;
};
```

definiscono un tipo strutturato (con etichetta opzionale `punto`) atto a rappresentare un punto nel piano. Questo nuovo tipo, ossia:

```
struct punto
```

può ora essere rinominato `Vett`, come chiede l'esercizio, in questo modo, cfr. $(*)$:

```
typedef struct punto Vett;
```

Se però volessimo usare nel resto del programma solo il sinonimo `Vett`, e mai il nome originario `struct punto`, allora potremmo più concisamente definire `Vett` in questo modo, senza bisogno dell'etichetta `punto`, e sempre in accordo con $(*)$:

```
typedef struct
        {
            double x;
            double y;
        } Vett;
```

In ogni caso, dopo il `typedef` è possibile dichiarare una variabile di tipo `Vett` al solito modo:

```
Vett v;
```

[a] Si noti quindi bene che, a rigore, `typedef` *non* definisce un nuovo tipo, ma più semplicemente introduce un sinonimo per un tipo già esistente.

Parte 4. Strutture

Definire i tipi `Vett` e `Tri` con `typedef`

Negli Esercizi 8 e 12 definirete i tipi `Vett` e `Tri` i cui valori rappresentano punti e triangoli nel piano euclideo $\mathbb{R}^2$, rispettivamente. Nei rimanenti esercizi i due tipi sono usati senza altre spiegazioni.

Esercizio 8

Distanza euclidea nel piano.

Tempo: 15 min.

Si scriva un programma che definisca (tramite `typedef`) il tipo `Vett` come una struttura di due campi, ciascuno di tipo `double`, che rappresentano le coordinate di un punto nel piano. Chiamate i due campi `x` e `y`, rispettivamente.

Il programma chiede all'utente di inserire le coordinate di due punti del piano, ne calcola la distanza invocando una funzione di prototipo `double dist(Vett, Vett)`, visualizza il risultato e termina.

Esercizio 9

Operazioni sui vettori.

Tempo: 25 min.

Si scrivano funzioni di prototipo:

```
Vett somma(Vett,Vett)
Vett invadd(Vett)
Vett sott(Vett,Vett)
Vett pscal(Vett)
double pint(Vett,Vett)
```

che implementino, rispettivamente, le operazioni di somma, inverso additivo ($\vec{v} \mapsto -\vec{v}$), sottrazione, prodotto per uno scalare e prodotto interno standard. Scrivete una funzione `main` che permetta di testare le vostre implementazioni.

Esercizio 10

Indipendenza lineare.

Tempo: 15 min.

Si scriva una funzione di nome `linind` che accetti in ingresso due puntatori a `Vett` e restituisca in uscita l'intero 1 se i due vettori puntati dai parametri in ingresso sono linearmente indipendenti, e 0 altrimenti. Si scriva poi una funzione `main` che accetti dalla riga di comando le coordinate di due vettori nel piano, e produca in uscita sulla console uno dei due messaggi `Linearmente indipendenti` o `Linearmente dipendenti`, a seconda di quale caso si verifichi. Si gestiscano gli errori legati alla lettura degli argomenti dalla riga di comando. Per convertire una stringa sulla riga di comando in `double` si usi la funzione `double atof(char *s)` della libreria standard dichiarata nel file di intestazione `stdlib.h`.

Esercizio 11

Indipendenza affine.

Tempo: 15 min.

Si scriva una funzione di nome `affind` che accetti in ingresso tre puntatori a `Vett` e restituisca in uscita l'intero 1 se i tre vettori puntati dai parametri in ingresso sono affinemente indipendenti,[3] e 0 altrimenti. Si scriva poi una funzione `main` che accetti dalla riga di comando le coordinate di tre vettori nel piano, e produca in uscita sulla console uno dei due messaggi `Affinemente indipendenti` o `Affinemente dipendenti`, a seconda di quale caso si verifichi. Si gestiscano gli errori legati alla lettura degli argomenti dalla riga di comando.

Esercizio 12

Triangoli.

Tempo: 10 min.

Definite (tramite `typedef`) il tipo `Tri`, i cui valori rappresentano triangoli nel piano euclideo. Il tipo è definito come una struttura di tre campi, ciascuno di tipo `Vett`, che rappresentano le coordinate dei tre vertici del triangolo. Chiamate i tre campi `v1`, `v2` e `v3`, rispettivamente.

Si scriva una funzione di prototipo:

```
int degenere(Tri *)
```

che restituisca 1 se il triangolo puntato dall'argomento è degenere — ossia, se i suoi vertici sono affinemente dipendenti — e 0 altrimenti. Scrivete una funzione `main` che permetta di testare la vostra implementazione.

Esercizio 13

Area di un triangolo.

[3] Si ricordi che, per definizione, i tre vettori $v_0, v_1, v_2 \in \mathbb{R}^2$ sono affinemente indipendenti se i due vettori $v_1 - v_0, v_2 - v_0 \in \mathbb{R}^2$ sono linearmente indipendenti.

Tempo: 25 min.

Si scriva una funzione di prototipo:

```
double area(Tri *)
```

che restituisca l'area del triangolo puntato dall'argomento. Per calcolare l'area del triangolo potete usare la *formula di Erone*: detta $\mu(T)$ l'area del triangolo T, si ha

$$\mu(T) = \sqrt{s(s-a)(s-b)(s-c)},$$

dove a, b e c sono le lunghezze dei tre lati di T, ed $s := \frac{1}{2}(a+b+c)$ ne è il semiperimetro. Scrivete una funzione `main` che permetta di testare la vostra implementazione.

Esercizio 14

L'area è invariante per rotazioni.

Tempo: 30 min.

Si ricorda che, dato un angolo θ, la matrice

$$R(\theta) := \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

è detta *matrice di rotazione* (di angolo θ). Dato un vettore $\vec{v} \in \mathbb{R}^2$, il prodotto $R(\theta)\vec{v}$ è l'immagine del vettore $\vec{v}$ dopo la rotazione in senso antiorario del piano di un angolo θ attorno all'origine.

Scrivete una funzione di prototipo:

```
void rot(Vett *, double a)
```

che applichi al vettore passato come primo parametro la rotazione attorno all'origine di un angolo `a`. (*Suggerimento.* Per calcolare seni e coseni, usate le funzioni `double sin(double)` e `double cos(double)` definite nel file di intestazione `math.h` della libreria standard.) Scrivete poi una funzione di prototipo:

```
void rot(Tri *, double a)
```

cha applichi la rotazione al triangolo passato come argomento.

Scrivete un programma che permetta all'utente di specificare un triangolo nel piano tramite le coordinate dei suoi vertici, assieme a un angolo θ (espresso in radianti e codificato come tipo `double`). Il programma calcola e visualizza l'area del triangolo — si usi la funzione scritta per risolvere l'Esercizio 13 — applica al triangolo la rotazione antioraria di angolo θ, calcola e visualizza nuovamente l'area del triangolo dopo la rotazione, e termina.

(V. Marra) Dipartimento di Matematica *Federigo Enriques*, Università degli Studi di Milano, via Cesare Saldini, 50, I-20133 Milano

Email address: vincenzo.marra@unimi.it