

LABORATORIO FONDAMENTI DI INFORMATICA

12 NOVEMBRE 2019 – Incontro 6 di 8 – Funzioni

Esercizio 1

Realizzare un programma che chieda una temperatura in gradi Fahrenheit all'utente e la converta in gradi Celsius tramite la chiamata a una funzione denominata “conversione”. La funzione riceve come unico parametro un numero che rappresenta la temperatura in gradi Fahrenheit e restituisce l'equivalente in gradi Celsius.

- Formula di conversione: $t_C = (t_F - 32) * 5.0 / 9.0$

Temperatura in gradi Fahrenheit? 100

Conversione: 100.000 Fahrenheit = 37.778 Celsius

Esercizio 2

Scrivere una funzione denominata “genera_matrice” che riceva come parametro una matrice quadrata 2 x 2 di numeri interi. La funzione deve riempire in modo casuale la matrice con numeri positivi < 10 e “ritornare” il determinante della matrice e la somma degli elementi contenuti.

- il determinante di una matrice 2 x 2 si calcola con la formula
 $\text{determinante} = \text{mat}[0][0] * \text{mat}[1][1] - \text{mat}[0][1] * \text{mat}[1][0]$
- dato che la funzione deve “ritornare” due valori, occorre che almeno uno dei due venga inserito in una variabile passata per indirizzo
- ossevare che la matrice viene per sua natura passata per indirizzo (la variabile che rappresenta un vettore è anche il puntatore al suo indirizzo in memoria) e quindi le modifiche fatte alla matrice all'interno della funzione permangono anche dopo il return

Matrice 2 x 2:

4 8

1 6

Determinante = 16

Somma elementi = 19

Esercizio 3

Realizzare un programma che chieda all'utente una stringa e un carattere delimitatore e stampi a schermo una sua sottostringa (se esiste) che inizi e termini con il carattere delimitatore.

A tal fine realizzare la funzione “estrazione” che presi come parametri una stringa “orig” (array di char – massimo 50) e un carattere “carat” estragga, se esiste, una sottostringa di “orig” che parta dal carattere “carat” e termini con lo stesso carattere “carat” oppure arrivi in fondo alla stringa. Questa sottostringa (eventualmente vuota) deve essere copiata in un altro array “risult” che viene anch'esso passato come parametro alla funzione “estrazione”.

- per chiedere la stringa all'utente utilizzare la funzione `gets(stringa)` poiché `scanf("%s", stringa)` si fermerebbe al primo spazio.
- Ricordare di inserire come ultimo carattere della stringa “risult” il carattere terminatore `'\0'`
- Utilizzare la funzione `strlen(stringa)` per calcolare la lunghezza in caratteri di una stringa.

stringa in ingresso? stampa pubblica

carattere? p

sottostringa: pa p

carattere? l

sottostringa: lica

Esercizio 4

Definire il tipo di dato

```
typedef float t_matrice[DIM][DIM];
```

e implementare la funzione

```
void calcola(t_matrice matr_media, int matr_orig[DIM][DIM])
```

Tale funzione analizza la matrice di numeri interi “matr_orig” passata come parametro e calcola e inserisce dei valori reali nella matrice “matr_media” anch'essa passata come parametro. In tale matrice ciascun elemento [i][j] dovrà essere un numero reale uguale alla media aritmetica (somma degli elementi diviso numero degli elementi) dei valori presenti nella matrice “matr_orig” nelle caselle adiacenti all'elemento [i][j] (senza coinvolgere l'elemento [i][j]). Le caselle “adiacenti” a una determinata casella [i][j] sono al massimo 8 ma sono di meno quando [i][j] è su uno dei bordi della matrice.

Per verificare se una casella adiacente esiste, ovvero se le sue coordinate [i+h][j+k] con $-1 \leq k \leq 1$ e $-1 \leq h \leq 1$ sono interne alla matrice, realizzare la seguente funzione:

```
int dentro(int x, int y)
```

Tale funzione ritorna 1 se le coordinate x e y sono interne alla matrice (cioè se x e y sono indici validi ≥ 0 e $< \text{DIM}$), altrimenti ritorna 0.

Per calcolare la media relativa a ciascun elemento [i][j], occorre richiamare l'ulteriore funzione da implementare

```
float media(int matr_orig[DIM][DIM], int i, int j)
```

Per gli elementi sui bordi, la media va fatta utilizzando solo gli elementi adiacenti che appartengono alla matrice (interni alla matrice, le cui coordinate fanno ritornare 1 alla funzione “dentro”).

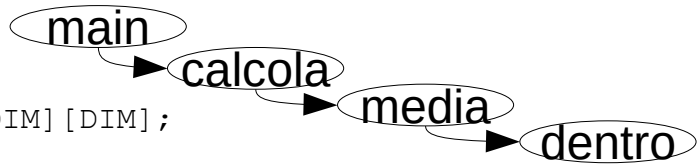
Nella funzione main del programma occorre dichiarare le due matrici “matr_media” e “matr_orig”, riempire a piacere “matr_orig” e richiamare la funzione “calcola” per poi stampare a video le due matrici

Matrice:

```
1  2  3
4  5  6
7  8  9
```

Matrice delle medie:

```
3.67  3.80  4.33
4.60  5.00  5.40
5.67  6.20  6.33
```



Esercizio 5

Realizzare un programma che dopo aver generato un vettore di $\text{DIM} = 10$ numeri interi casuali ne analizzi il contenuto contando quanti numeri sono $< \text{SOGLIA} = 10$; i restanti numeri (quelli $\geq \text{SOGLIA}$) vanno smistati in base alla parità: i numeri pari in un vettore e i dispari in un altro.

Il programma deve essere suddiviso nelle seguenti funzioni da implementare:

- funzione “stampavet”. Riceve come parametri un array “vett” di numeri interi di dimensione non specificata e un numero intero “n” che indica quanti elementi significativi sono presenti nel vettore. La funzione stampa a video tutti gli n numeri significativi presenti nell'array separandoli con uno spazio e infine stampa un carattere di “a capo” ‘\n’;
- funzione “inserisci”. Riceve come parametri un vettore di numeri interi “vett”, un indice “i” e un numero intero “valore”. La funzione deve inserire nel vettore “vett” il valore “valore” in posizione “i”
- funzione “analisi”. Riceve come primo parametro un array “vett” di DIM numeri interi e come secondo parametro un numero intero “sotto_s” che deve essere “modificato” dalla funzione (attenzione alla modalità di passaggio del parametro). La funzione deve analizzare il vettore passato come parametro, inserire in “sotto_s” quanti numeri in questo vettore sono $< \text{SOGLIA}$ e popolare tramite la funzione “inserisci” una variabile del tipo struct specificato sotto che va restituita dalla funzione (attenzione: i numeri $< \text{SOGLIA}$ non vanno inseriti nei vettori di numeri pari o dispari):

```
typedef struct {
    int v_pari[DIM]; //vettore di numeri pari
    int n_pari; //quantita' di numeri pari
    int v_dispari[DIM]; //vettore di numeri dispari
    int n_dispari; //quantita' di numeri dispari
} s_risultato;
```

Nella parte principale del programma occorre dichiarare un array “vett” di massimo DIM = 10 numeri interi e poi popolarlo tramite il generatore di numeri pseudocasuali

```
vett[i] = rand() % 100;
```

Il vettore deve essere stampato a video tramite la funzione “stampavet” e passato come primo parametro alla funzione “analisi” per poi stampare a video quanti numeri sono < SOGLIA e il contenuto dei vettori “v_pari” e “v_dispari” della struct (sempre usando la funzione “stampavet”).

Vettore da analizzare: 1 10 5 28 87 58 77 44 32 59

Quantita' di numeri sotto soglia (< 10): 2

Numeri pari sopra soglia (>= 10): 10 28 58 44 32

Numeri dispari sopra soglia (>= 10): 87 77 59

Esercizio 6

Realizzare un programma che inserisca in un vettore di numeri interi un numero casuale alla volta mantenendo l'univocità dei numeri: se si tenta di inserire un numero già presente nel vettore le sue occorrenze inserite in precedenza vanno “eliminate”.

A tal fine implementare la funzione “memorizza_univ” che riceve come unico parametro un numero intero “quantita” < DIMV = 100. La funzione genera continuamente numeri pseudocasuali tramite una riga di codice simile alla seguente fino a quando ne ha generati un numero pari a “quantita”:

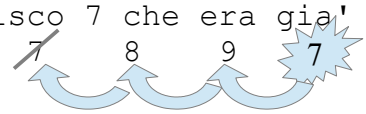
```
num = rand() % RANGE; //RANGE = 10
```

I numeri devono essere memorizzati uno a uno in un vettore di numeri interi “vett” (massimo RANGE + 1 elementi) a partire dalla cella 0 in poi.

Per mantenere l'univocità dei numeri, se il numero da inserire compare già nel vettore è necessario eliminare la sua precedente occorrenza sovrascrivendolo, “spostando indietro” ovvero facendo “shift” di una cella tutti gli eventuali numeri che lo seguono. A ogni passo occorre stampare il contenuto del vettore tramite la funzione “stampavet” realizzata nell'esercizio 3.

...Inserisco 7 che era già presente!

3 5 ~~7~~ 8 9 ~~7~~ ==> 3 5 8 9 7



Ogni volta che un numero viene eliminato, occorre aumentare una variabile intera “eliminati” che conta quanti numeri sono stati eliminati. Tale variabile viene restituita dalla funzione “memorizza_univ”.

Nella parte principale del programma occorre chiedere all'utente il valore della variabile “quantita” e passarlo come parametro alla funzione “memorizza_univ” e quindi stampare a video quanti numeri sono stati eliminati.

Quantita' di numeri da generare (1-100)? 50

...

Passo 8/50: inserisco 0 che NON era già presente.

4 8 5 2 3 7 9 0

Passo 9/50: inserisco 2 che era già presente!

4 8 5 3 7 9 0 2

...

Ho eliminato 40 numeri.

Esercizio 7 (variante dell'esercizio 6)

Realizzare un programma simile al precedente con le seguenti varianti:

- la funzione da implementare va chiamata “memorizza_ord” e riceve come parametro un numero intero denominato “sommafinale”
- il vettore `vett` deve avere dimensione `DIMV` (invece di `RANGE + 1`)
- l'elaborazione termina quando la somma dei numeri ricevuti è $>$ del parametro “sommafinale” (oppure quando il vettore `vett` è “pieno”)
- invece che il mantenimento dell'univocità si richiede il mantenimento dell'ordinamento degli elementi del vettore (dal più piccolo al più grande). Per mantenere l'ordinamento occorre “fare spazio” per il numero da inserire “spostando avanti” di una cella tutti gli eventuali numeri maggiori di esso
- invece di ritornare il numero di elementi eliminati, la funzione “memorizza_ord” ritorna il numero di elementi inseriti

Somma finale (> 0)? 100

...

Inserimento 4: inserisco 0 in posizione 0 - somma attuale = 12/100.

0 1 4 7

Inserimento 5: inserisco 9 in posizione 4 - somma attuale = 21/100.

0 1 4 7 9

Inserimento 6: inserisco 4 in posizione 3 - somma attuale = 25/100.

0 1 4 4 7 9

...

Inserimento 26: inserisco 2 in posizione 10 - somma attuale = 101/100.

0 1 1 1 1 1 2 2 2 2 2 3 4 4 4 4 5 5 5 6 7 7 7 8 8 9

Ho inserito 26 numeri.