Fondamenti di Informatica T-1 modulo 2

Laboratorio 03

Obiettivi di questa esercitazione

1. ***Passaggio dei parametri per valore/riferimento***
2. Trattamento degli errori: funzioni che restituiscono anche codici di errore
3. ***Vettori***

Passaggio dei parametri per valore/riferimento (1)

- Formalmente, tutti i parametri sono ***passati per valore***
- In C, possibilità di passare come parametro l'indirizzo di memoria di una variabile
  - Passaggio di una ***variabile di tipo puntatore***
  - Passaggio ***dell'indirizzo di una variabile*** tramite ***l'operatore &***
- Si accede al valore contenuto a tale indirizzo tramite l'operatore di ***de-referenziamento ****

Esempio 1: Passaggio per Valore/Riferimento

I numeri complessi

- Data la notazione cartesiana di un numero complesso (in parte reale e immaginaria), realizzare una funzione che ne restituisca la ***notazione polare***
- Si usi opportunamente la funzione atan2(float im, float re) della libreria math.h

$$r = \sqrt{re^2 + im^2} \qquad \varphi = \arctan\left(\frac{im}{re}\right)$$

Esempio 1 - Soluzione

```
#include <math.h>
#include <stdio.h>

void convertiComplex(
              float re, float im,
              float * modulo, float * argomento)
{
   *modulo = sqrt(re*re + im*im);
   *argomento = (re == 0 ? 0: atan2(im, re));
   return;
}

int main() {
   float modulo, argomento;
   convertiComplex(1.0, 1.0, &modulo, &argomento);
   printf("Modulo: %f\n
              Argomento: %f\n", modulo, argomento);
}
```


Esercizio 1

Somma di numeri complessi

- Realizzare una funzione che riceva in ingresso due numeri complessi
  - Un numero complesso è dato da una coppia rappresentante la parte reale e la parte immaginaria
- La funzione deve restituire la somma di tali valori (ancora una coppia)
- Realizzare anche un **`main()`** di esempio

Trattamento degli errori (1)

- È ottima abitudine di programmazione che ogni funzione restituisca, oltre ad uno o più risultati, anche un ***codice identificativo per eventuali errori***
- Quali informazioni? Si deve comunicare il successo, il fallimento e/o il motivo di tale fallimento
- Tipicamente si usa un intero: il significato è stato deciso dal programmatore
- Quindi, tipicamente si devono aggiungere anche informazioni/commenti che spieghino tale significato… altrimenti???

Trattamento degli errori (2)

Esistono diversi modi per comunicare, oltre ai risultati, eventuali codici di errore:

- Una funzione può restituire direttamente il codice di errore e, tramite passaggio per riferimento, i risultati
- Tramite una variabile globale (**NO**)
- Tramite una opportuna variabile (passata per riferimento anch'essa)
- Se la funzione restituisce un intero all'interno di un certo dominio, si possono usare valori esterni al dominio per indicare eventuali errori

Esempio 2: Trattamento degli Errori

Esempio 2 – Calcolo dei coefficienti binomiali

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- Due funzioni: una che calcola il fattoriale, una che calcola il coefficiente binomiale
  - int fact(int x) ha senso se e solo se x è non negativo
  - int binomiale(int n, int k) ha senso se e solo se n>=k
- In entrambi i casi, gli errori possono essere causati da parametri non corretti! Esistono però anche altri tipi di errore

Esempio 2 - Soluzione (1)

```
#define FATTORIALE_RET_TYPE int
#define SUCCESS 0
#define PARAM_NEGATIVE -1

FATTORIALE_RET_TYPE fattoriale(int n, int * result)
{
   int fact = 1, index;

   if (n < 0) // CONTROLLO DEI PARAMETRI!!!
   {
        return PARAM_NEGATIVE;
   }
   else
   {
        for(index = n; index > 0; index--)
                fact = fact * index;
        *result = fact;
        return SUCCESS;
   }
}
```

Esempio 2 - Soluzione (2)

```
#define BINOMIALE_RET_TYPE
#define SUCCESS 0
#define PARAM_NEGATIVE -1
#define BINOMIALE_INCORRECT_PARAMS -5
FATTORIALE_RET_TYPE fattoriale(int n, int * result) {…}
BINOMIALE_RET_TYPE binomiale(int n, int k, int * result)
{
   int numeratore, denominatore1, denominatore2, funOK;
   funOK = fattoriale(n, &numeratore);
   if (funOK == SUCCESS) {
        funOK = fattoriale(k, &denominatore1);
        if (funOK == SUCCESS) {
               funOK = fattoriale(n-k, &denominatore2);
               if (funOK == SUCCESS) {
                      *result=numeratore/(denominatore1*denominatore2);
                      return SUCCESS;
               }
               else return BINOMIALE_INCORRECT_PARAMS;
        }
        else return funOK;
   }
   else return funOK;
}
```


Esercizio 2

Area e perimetro di un triangolo

- Realizzare una funzione che, date le lunghezze dei tre lati di un triangolo
  - Restituisca uno fra tre codici
    - PRIMO CASO: triangolo non valido
      - Un triangolo è invalido se uno dei tre lati è più lungo della somma degli altri due, oppure se uno dei tre lati è negativo
    - SECONDO CASO: triangolo degenere
      - Un triangolo è degenere se uno dei tre lati è nullo, oppure uno dei tre lati è uguale alla somma degli altri due
    - TERZO CASO: triangolo valido
  - Nel caso di triangolo valido
    - La funzione deve anche restituire il perimetro e l'area del triangolo
    - Per l'area, si utilizzi la formula (con s semiperimetro)

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

- Realizzare una procedura per la gestione del risultato

Esercizio 3: Vettori

Test di uguaglianza fra vettori, elemento per elemento

- Creare una funzione che, dati in input due vettori e le rispettive lunghezze, determini se i due vettori sono uguali
- IPOTESI: l'uguaglianza va testata in maniera "ordinata", ovvero elemento per elemento

Esercizio 3 - Note

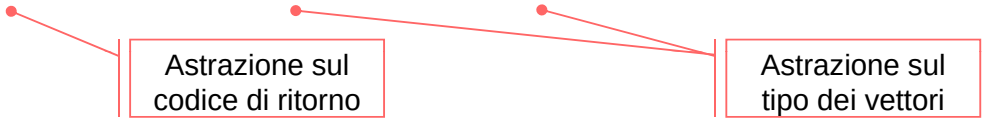
Linee guida per la soluzione

- Cerchiamo di astrarre il più possibile sul tipo dei vettori
  - Uso di costanti simboliche
  - Incapsulamento del test di uguaglianza fra due elementi in una funzione a parte
    - Questa è l'unica funzione che deve conoscere quali sono i tipi!
- Restituizione di codici differenziati
  - Uso di costanti simboliche
- Un occhio all'efficienza
  - ***Cerchiamo di effettuare il minor numero di cicli possibile***

Esercizio 3 - Suggerimenti

- PRIMO PASSO: dichiarazione delle funzioni

```
RESULT compareTo(TYPE v1[], TYPE v2[], int dim1, int dim2)
```




```
BOOLEAN equals(TYPE el1, TYPE el2)
```

- SECONDO PASSO: pseudocodice

```
se dim1 != dim2 i vettori sono DIFFERENTI
per i da 0 alla lunghezza dei vettori
{
    se i-mo elemento di v1 diverso
    rispetto a i-mo elemento di v2,
          i vettori sono DIFFERENTI
}
i vettori sono UGUALI
```


Esercizio 4

Test di uguaglianza fra vettori con elementi non ripetuti

- Questa volta il test deve verificare che i vettori contengano gli stessi elementi, NON NECESSARIAMENTE NELLO STESSO ORDINE
- Ipotesi semplificativa: i vettori non hanno elementi ripetuti

Esercizio 4 - Note

- Che cosa cambia rispetto al test precedente?
  - Non dobbiamo più controllare semplicemente se tutti gli elementi di indice uguale sono uguali
  - Dobbiamo piuttosto verificare che ogni elemento del primo vettore sia contenuto nel secondo
  - Questo è sufficiente?
    - Ricordarsi delle ipotesi!

Esercizio 5

Test di uguaglianza fra vettori con elementi ripetuti

- Ora rimuoviamo anche l'ipotesi sulla possibilità di avere elementi ripetuti
- Che cosa dobbiamo modificare della precedente funzione?
  - Cerchiamo, quando è sensato ☹, di riutilizzare ciò che abbiamo già realizzato

Esercizio 5 - Esempi

1	1	3	5	4
3	4	1	5	1

SI

1	1	3	5	1
3	5	1	5	1

NO

- *Occorre controllare che ci sia una corrispondenza uno a uno fra gli elementi*
- *È necessaria una struttura dati di appoggio!*