

Fondamenti di Informatica T-1

Modulo 2

Obiettivi di questa esercitazione

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

Passaggio dei parametri per valore/riferimento

- 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-referenziazione*** *

Esercizio 1

(passaggio parametri per riferimento)

I numeri complessi

- Data la notazione cartesiana di un numero complesso (in parte reale ed immaginaria),
- Realizzare una procedura che ne restituisca la notazione polare (tramite parametri passati per riferimento)
- Si usi opportunamente la funzione `atan2(float im, float re)` della libreria `math.h`

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

La funzione `atan2` gestisce correttamente anche il caso in cui `re==0`. Se così non fosse? Si estenda la funzione di conversione in modo da controllare la correttezza dei parametri: la funzione restituisca un codice di errore se necessario.

```
void converti_complex(float re, float im,  
                    float * modulo, float * argomento)
```

Esercizio 1 - Soluzione

(passaggio parametri per riferimento)

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

void converti_complex(
    float re, float im,
    float * modulo, float * argomento)
{
    *modulo = sqrt(re*re + im*im);
    *argomento = atan2(im, re);
    return;
}

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

Esercizio 1 – Soluzione (variante)

(passaggio parametri per riferimento)

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

int converti_complex( float re, float im,
                    float * modulo, float * argomento) {
    if (re==0) return -1;
    *modulo = sqrt(re*re + im*im);
    *argomento = atan2(im, re);
    return 0;
}

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

Esercizio 2

(passaggio parametri per riferimento)

Somma di due numeri complessi

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

Esercizio 2 - Soluzione

(passaggio parametri per riferimento)

```
void somma_complex(
    float reA, float imA,
    float reB, float imB,
    float * reResult, float * imResult)
{
    *reResult = reA + reB;
    *imResult = imA + imB;
    return;
}

int main() {
    float reResult, imResult;
    somma_complex(1.0, 1.0, 2.0, 2.0, &reResult, &imResult);
    printf("Parte reale: %f\n
           Parte Immaginaria: %f\n", reResult, imResult);
}
```


Esercizio 3

(passaggio parametri per riferimento)

- Un sistema di cronometraggio per la Formula 1 registra i tempi in millisecondi. Tuttavia tali tempi devono essere presentati in termini di minuti, secondi e millisecc.
- Creare una procedura che, ricevuti in ingresso un tempo dato in millisecondi, restituisca l'equivalente in termini di minuti, secondi, millisecc. (tramite eventuali parametri passati per riferimento)
- Si realizzi un main che invochi la funzione e che, dopo aver chiesto all'utente un valore indicante una durata in millisecondi, stampi a video il tempo nel formato min:sec.millisecc

Esercizio 3 - Soluzione

(passaggio parametri per riferimento)

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

void fromMillisec(int millisec, int * mm, int * sec, int * min) {
    *mm = millisec % 1000;
    *sec = millisec / 1000;
    *min = *sec / 60;
    *sec = *sec % 60;
    return;
}

int main(void) {
    int millisec, mm, sec, min;

    printf("Inserisci un tempo in millisec.: ");
    scanf("%d", &millisec);
    fromMillisec(millisec, &mm, &sec, &min);
    printf("Tempo: %d:%d.%d\n", min, sec, mm);

    system("PAUSE");
    return (0);
}
```

Esercizio 4

(passaggio parametri per riferimento)

- Un sistema di gestione mp3 permette di calcolare in anticipo la durata di una compilation di brani.
- Creare una procedura che, ricevuti in ingresso la durata di due pezzi musicali, in termini di ore, minuti e secondi, restituisca la durata risultante dalla somma dei due brani in termini di ore, minuti e secondi.
- Si realizzi un main che chieda all'utente di inserire la durata di diversi brani musicali, e si stampi a video la durata totale (l'utente segnala il termine dei brani da inserire con un brano speciale di lunghezza 0:00.00).

Esercizio 4 - Soluzione

(passaggio parametri per riferimento)

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

void sommaTempi(int h1, int m1, int s1, int h2, int m2, int s2, int * hr,
                int * mr, int * sr) {
    *sr = s1 + s2;
    *mr = *sr / 60;
    *sr = *sr % 60;
    *mr = *mr + m1 + m2;
    *hr = *mr / 60;
    *mr = *mr % 60;
    *hr = *hr + h1 + h2;
}

...
```

Esercizio 4 - Soluzione

(passaggio parametri per riferimento)

...

```
int main(void)
{
    int h1, h2=0;
    int m1, m2=0;
    int s1, s2=0;
    int i=1;

    do {
        printf("inserisci la durata della canzone numero %d (hh:mm:ss): ", i);
        scanf("%d%d%d", &h1, &m1, &s1);
        if (! (h1==0 && m1==0 && s1==0))
            sommaTempi(h1, m1, s1, h2, m2, s2, &h2, &m2, &s2);
        i++;
    } while ( ! (h1==0 && m1==0 && s1==0));

    printf("Durata totale: %dh:%dm:%ds\n", h2, m2, s2);

    system("PAUSE");
    return (0);
}
```

Trattamento degli errori

- 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

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: Trattamento degli Errori

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 – Soluzione

(trattamento degli errori)

```
#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 – Soluzione

(trattamento degli errori)

```
#define BINOMIALE_RET_TYPE int
#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 5

(trattamento degli errori)

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 5 - Soluzione

(trattamento degli errori)

```
#define CALCULATE_RET_TYPE int
#define REGULAR_TRIANGLE 0
#define INVALID_TRIANGLE 1
#define LIMIT_TRIANGLE 2
CALCULATE_RET_TYPE calculate(float a, float b, float c, float* area,
    float* perimeter)
{
    float s;
    if(a < 0 || b < 0 || c < 0 || a > b+c || b > a+c || c > a+b)
        return INVALID_TRIANGLE;
    if(a == 0 || b == 0 || c == 0 || a == b + c || b == a + c
        || c == a + b)
        return LIMIT_TRIANGLE;
    *perimeter = a + b + c;
    s = *perimeter / 2;
    *area = sqrt( s * (s-a) * (s-b) * (s-c));
    return REGULAR_TRIANGLE;
}
```

Esercizio 5 - Soluzione

(trattamento degli errori)

```
void printCalculation(float a, float b, float c)
{
    float area, perimeter;
    CALCULATE_RET_TYPE result;
    result = calculate(a, b, c, &area, &perimeter);
    switch(result)
    {
        case INVALID_TRIANGLE:
            printf("Triangolo non valido\n");
            break;
        case LIMIT_TRIANGLE:
            printf("Triangolo limite\n");
            break;
        case REGULAR_TRIANGLE:
            printf("Perimetro: %f, Area: %f\n", perimeter, area);
    }
}
```

Esercizio 6

(funzioni varie)

- Si supponga l'esistenza di una funzione:
`double f(double x)`
- Il candidato realizzi una funzione `rettangoli(...)` che calcoli l'integrale della funzione `f(x)` utilizzando il metodo di approssimazione dei rettangoli
- Si realizzi sia la versione iterativa, sia la versione ricorsiva

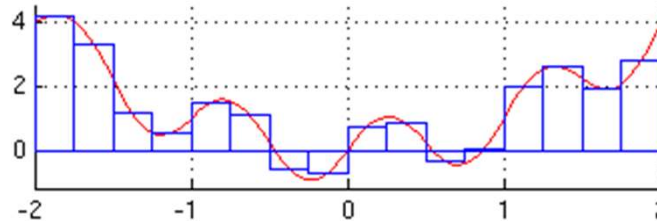
Esercizio 6

(funzioni varie)

- Il metodo dei "rettangoli" calcola l'integrale di una funzione $f(x)$ nell'intervallo $[a,b]$ nel seguente modo:
 - suddivide l'intervallo $[a,b]$ in N intervalli identici
 - per ognuno di questi intervalli, calcola l'area del rettangolo sottostante la funzione. Area = $(b'-a')*f(a')$
 - l'integrale totale è dato dalla somma di tali rettangoli

- Parametri:

- a, b
- N



Esercizio 6 - Soluzione

(funzioni varie)

```
double f(double x) {  
    return sin(x);  
}
```

```
double rettangoliIt(double a, double b, int n) {  
    double delta;  
    double result = 0;  
  
    delta = (b-a)/n; // possibile errore di rappresentazione  
    while (a+delta<=b) {  
        result = result + (delta*f(a));  
        a = a + delta;  
    }  
    return result;  
}
```


Esercizio 6 - Soluzione

(funzioni varie)

```
double f(double x) {
    return sin(x);
}

double rettangoliIt2(double a, double b, int n) {
    double delta;
    double result = 0;

    delta = (b-a)/n;
    while (n>0) {
        if (n>1)
            result = result + (delta*f(a));
        else
            result = result + ((b-a)*f(a));
        a = a + delta;
        n = n-1;
    }
    return result;
}
```

Esercizio 6 - Soluzione

(funzioni varie)

```
double f(double x) {  
    return sin(x);  
}
```

```
double rettangoliRic(double a, double b, int n) {  
    double delta;  
  
    if (n<=0)  
        return 0;  
    else {  
        delta = (b-a)/n;  
        return delta*f(a) + rettangoliRic(a+delta, b, n-1);  
    }  
}
```

Esercizio 7

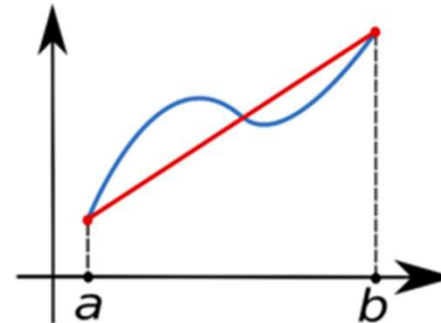
(funzioni varie)

- Si supponga l'esistenza di una funzione:
`double f(double x)`
- Il candidato realizzi una funzione `integrale(...)` che calcoli l'integrale della funzione `f(x)` utilizzando il metodo di approssimazione dei trapezi
- Si realizzi sia la versione iterativa, sia la versione ricorsiva

Esercizio 7

(funzioni varie)

- Il metodo dei "trapezi" calcola l'integrale di una funzione $f(x)$ nell'intervallo $[a,b]$ nel seguente modo:
 - suddivide l'intervallo $[a,b]$ in N intervalli identici
 - per ognuno di questi intervalli, calcola l'area del trapezio sottostante la funzione: $\text{Area} = (f(a') + f(b')) \cdot (b' - a') / 2$
 - l'integrale totale è dato dalla somma di tali trapezi
- Parametri:
 - a, b
 - N



Esercizio 7 - Soluzione

(funzioni varie)

```
double f(double x) {
    return sin(x);
}

double trapeziIt(double a, double b, int n) {
    double delta;
    double result = 0;

    delta = (b-a)/n;
    while (n>0) {
        if (n>1)
            result = result + (f(a)+f(a+delta))*delta/2;
        else
            result = result + (f(a)+f(b))*(b-a)/2;
        a = a + delta;
        n = n - 1;
    }
    return result;
}
```

Esercizio 7 - Soluzione

(funzioni varie)

```
double f(double x) {  
    return sin(x);  
}
```

```
double trapeziRic(double a, double b, int n) {  
    double delta;  
  
    if (n<=0)  
        return 0;  
    else {  
        delta = (b-a)/n;  
        return (f(a)+f(a+delta))*delta/2 + trapeziRic(a+delta, b, n-1);  
    }  
}
```

Esercizio 8

(funzioni varie)

- Si realizzi una funzione RICORSIVA che calcoli la media di una sequenza di interi inseriti dall'utente.
- Non è nota a priori la lunghezza della sequenza.
- L'utente segnala il termine della sequenza tramite l'inserimento del valore intero 0 (zero).

Esercizio 8 - Soluzione

(funzioni varie)

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

double mediaRic(int sommaParziale, int n) {
    int num;
    printf("inserisci nuovo numero (zero per terminare): ");
    scanf("%d", &num);
    if (num==0)
        return ((double) sommaParziale)/n;
    else
        return mediaRic(sommaParziale+num, n+1);
}

int main() {
    printf("La media e': %lf\n", mediaRic(0, 0));
    return 0;
}
```


Esercizio 9

(funzioni varie)

- Si realizzi due funzioni RICORSIVE che calcolino, rispettivamente, il minimo e il massimo di una sequenza di interi inseriti dall'utente.
- Non è nota a priori la lunghezza della sequenza.
- L'utente segnala il termine della sequenza tramite l'inserimento del valore intero 0 (zero).
- Si implementino poi entrambe le funzioni come una unica procedura ricorsiva che restituisca i risultati tramite parametri passati "per riferimento".

Esercizio 9 - Soluzione

(funzioni varie)

```
int minRic2(int currentRecord) {
    int num;
    printf("Minimo, inserisci nuovo numero (zero per terminare): ");
    scanf("%d", &num);
    if (num == 0)
        return currentRecord;
    else {
        if (num < currentRecord)
            return minRic2(num);
        else
            return minRic2(currentRecord);
    }
}
```

```
int minRic() {
    int num;
    printf("Minimo, inserisci nuovo numero (zero per terminare): ");
    scanf("%d", &num);
    if (num != 0)
        return minRic2(num);
    else return 0;
}
```

Esercizio 9 - Soluzione

(funzioni varie)

```
int maxRic2(int currentRecord) {
    int num;
    printf("Massimo, inserisci nuovo numero (zero per terminare): ");
    scanf("%d", &num);
    if (num == 0)
        return currentRecord;
    else {
        if (num > currentRecord)
            return maxRic2(num);
        else
            return maxRic2(currentRecord);
    }
}
```

```
int maxRic() {
    int num;
    printf("Massimo, inserisci nuovo numero (zero per terminare): ");
    scanf("%d", &num);
    if (num != 0)
        return maxRic2(num);
    else return 0;
}
```

Esercizio 9 - Soluzione

(funzioni varie)

```
void minMaxRic2(int * minRecord, int * maxRecord) {
    int num;
    printf("Inserisci nuovo numero (zero per terminare): ");
    scanf("%d", &num);
    if (num==0)
        return;
    else {
        if (num>*maxRecord)
            *maxRecord = num;
        if (num<*minRecord)
            *minRecord = num;
        minMaxRic2(minRecord, maxRecord);
        return;
    }
}
```

Esercizio 9 - Soluzione

(funzioni varie)

```
void minMaxRic(int * minRecord, int * maxRecord) {
    int num;
    printf("Inserisci nuovo numero (zero per terminare): ");
    scanf("%d", &num);
    *maxRecord = num;
    *minRecord = num;
    if (num==0)
        return;
    else {
        minMaxRic2(minRecord, maxRecord);
        return;
    }
}

int main() {
    int min, max;
    printf("Il minimo e': %d\n", minRic());
    printf("Il massimo e': %d\n", maxRic());
    minMaxRic(&min, &max);
    printf("Il minimo e il massimo sono: %d %d\n", min, max);

    return 0;
}
```

Esercizio 10

(funzioni varie)

- Il candidato realizzi una procedura ricorsiva `convertiBin(...)` che ricevuto in ingresso un valore intero positivo, stampi a video la sua rappresentazione in binario.
- Si presti particolare attenzione all'ordine di stampa dei valori binari: i bit meno significativi devono ovviamente essere a destra.

Esercizio 10 - Soluzione

(funzioni varie)

```
#include <stdio.h>

void convertiBin(int n) {
    if (n==0)
        return;
    else {
        convertiBin(n/2);
        printf("%d", n%2);
        return;
    }
}

int main() {
    convertiBin(5);

    return 0;
}
```

Esercizio 11

(funzioni varie)

- Il candidato realizzi una funzione **secondoGrado (. . .)** che ricevuto in ingresso tre valori interi a, b, c, coefficienti del polinomio:

$$\mathbf{ax^2 + bx + c = 0}$$

restituisca (tramite parametri passati per riferimento) le radici del polinomio.

- Tramite il valore di ritorno, la funzione restituisca eventuali codici di errore o di successo.

Esercizio 11 - Soluzione

(funzioni varie)

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

#define RETURN_TYPE int
#define SUCCESS 0
#define DELTA_NEGATIVE -1

RETURN_TYPE secondoGrado(int a, int b, int c, double *rad1, double *rad2) {
    int delta;

    delta = b*b -4*a*c;
    if (delta<0)
        return DELTA_NEGATIVE;
    else {
        *rad1 = (-b + sqrt(delta))/(2*a);
        *rad2 = (-b - sqrt(delta))/(2*a);
        return SUCCESS;
    }
}
```

Esercizio 11 - Soluzione

(funzioni varie)

```
int main() {

    int a, b, c;
    double rad1, rad2;
    RETURN_TYPE funOk;

    printf("Inseriscicoefficienti: ");
    scanf("%d%d%d", &a, &b, &c);

    funOk = secondoGrado(a,b,c, &rad1, &rad2);
    if (funOk==SUCCESS)
        printf("Radici del polinomio: %lf %lf\n", rad1, rad2);
    else
        printf("Il polinomio specificato non ha radici reali.\n");

    return 0;
}
```