



Programmazione I

Il Linguaggio C

Esercizi

Daniel Riccio

Università di Napoli, Federico II

30 novembre 2021



Sommario



- Argomenti
 - Progetti multifile
 - Il preprocessore
 - Esercizi

Il Preprocessore



Modifica il codice C prima che venga eseguita la traduzione vera e propria

Le direttive al preprocessore riguardano:

- inclusione di file (**#include**)
- definizione di simboli (**#define**)
- sostituzione di simboli (**#define**)
- compilazione condizionale (**#if**)
- macroistruzioni con parametri (**#define**)

Non essendo istruzioni C non richiedono il ';' finale

Il Preprocessore



La riga con `#include` viene sostituita dal contenuto testuale del file indicato

`#include <stdio.h>`

Il nome del file può essere completo di percorso

Ha due forme:

- `#include <file.h>` Il file viene cercato nelle directory del compilatore
- `#include "file.h"` Il file viene cercato prima nella directory dove si trova il file C e poi, se non trovato, nelle directory del compilatore

I file inclusi possono a loro volta contenere altre direttive `#include`

La direttiva `#include` viene in genere collocata in testa al file sorgente C, prima della prima funzione

Generalmente, i file inclusi non contengono codice eseguibile, ma solo dichiarazioni (es. prototipi e variabili `extern`) e altre direttive

Il Preprocessore



#define nome

definisce il simbolo denominato **nome**

#define DEBUG

I simboli vengono utilizzati dalle altre direttive del preprocessore (ad es. si può verificare se un simbolo è stato definito o no con **#if**)

Lo scope di **nome** si estende dalla riga con la definizione fino alla fine di quel file sorgente e non tiene conto dei blocchi di codice

nome ha la stessa sintassi di un identificatore (nome di variabile), non può contenere spazi e viene per convenzione scritto in maiuscolo

#undef nome annulla una **#define** precedente

Inclusione condizionale

Permette di include o escludere parte del codice dalla compilazione e dal preprocessing stesso

```
#if espressione_1
    istruzioni
#elif espressione_2
    istruzioni
...
#else
    istruzioni
#endif
```

Solo uno dei gruppi di `istruzioni` sarà elaborato dal preprocessore e poi compilato

Inclusione condizionale



Le **espressioni** devono essere costanti intere (non possono contenere `sizeof()`, `cast`), sono considerate vere se `!=0`

L'espressione `defined(nome)` produce 1 se `nome` è stato definito (con `#define`), 0 altrimenti

`#ifdef nome` equivale a:

`#if defined(nome)` e verifica che `nome` sia definito

`#ifndef nome` equivale a:

`#if !defined(nome)` e verifica che `nome` non sia definito

Inclusione condizionale



Nel caso in cui un file incluso ne includa a sua volta altri, per evitare di includere più volte lo stesso file, si può usare lo schema seguente (quello che segue è il file `hdr.h`):

```
#ifndef HDR
#define HDR
...
contenuto di <hdr.h>
#endif
```

Se venisse incluso una seconda volta, il simbolo `HDR` sarebbe già definito e il contenuto non verrebbe nuovamente incluso nella compilazione

Per escludere dalla compilazione un grosso blocco di codice (anche con commenti):

```
#if 0
codice da non eseguire
#endif
```

Per isolare istruzioni da usare solo per il debug:

```
#ifdef DEBUG
printf("Valore di x: %d\n", x);
#endif
```


Esercizio – Parola palindroma



Realizzare un programma che prenda una parola da linea di comando e:

- Dica se la parola è palindroma
- Se la parola è palindroma stampi la metà che poi si ripete in modo speculare.

Esercizio – Parola palindroma



Scriviamo l'header file **palindroma.h**

```
#ifndef PALINDROMA_H
#define PALINDROMA_H

    char *Palindroma(char *parola);

#endif
```

Esercizio – Parola palindroma



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <palindroma.h>

int main(int argc, char *argv[])
{
    char *parola;
    char *semiparola;

    // Controlliamo la correttezza della linea di comando
    if(argc != 2){
        printf("Il numero dei parametri non è corretto\n");
        printf("Utilizzo:>palindroma.exe parola\n");
        return 0;
    }
```

Esercizio – Parola palindroma



```
parola = argv[1];
```

```
semiparola = Palindroma(parola);
```

```
if(semiparola){  
    printf("La parola è palindroma\n");  
    printf("La radice è: %s\n", semiparola);  
  
    free(semiparola);  
}  
  
return EXIT_SUCCESS;  
}
```

Esercizio – Parola palindroma



```
char *Palindroma(char *parola)
{
    char *p, *q;
    char *semiparola = NULL;
    int palin=1;
    int semiplen = 0;

    p=parola;                /* punta al primo carattere */
    q=parola+strlen(parola)-1; /* punta all'ultimo carattere */
    while (p < q){
        if (*p++ != *q--){
            palin = 0;
            break;
        }
    }
}
```

Esercizio – Parola palindroma



```
if (palin){  
    printf("La parola è palindroma\n");  
    semiplen = strlen(parola)/2;  
    semiparola = (char *)malloc((sempiplen+1)*sizeof(char));  
    semiparola[sempiplen+1]='\0';  
  
    while(sempiplen>=0)  
        semiparola[sempiplen] = parola[sempiplen--];  
}else  
    printf("Non palindroma\n");  
  
printf("La radice è: %s\n", semiparola);  
  
return semiparola;  
}
```

Esercizio – Parola palindroma

Se avessimo voluto utilizzare un puntatore a funzione?

Definizione e assegnazione

```
char *(*fp)(char *) = NULL;
```

```
fp = Palindroma;
```

Chiamata della funzione

```
semiparola = fp(parola);
```

Esercizi



1. Zero di una funzione v.1.0

Analizziamo ora il problema di "risolvere", in campo reale, un'equazione ad una incognita, cioè di trovare il passaggio per lo zero di una funzione qualunque.

Teorema dello zero

Una funzione continua che assume valori di segno opposto agli estremi di un intervallo ammette almeno uno zero nell'intervallo.

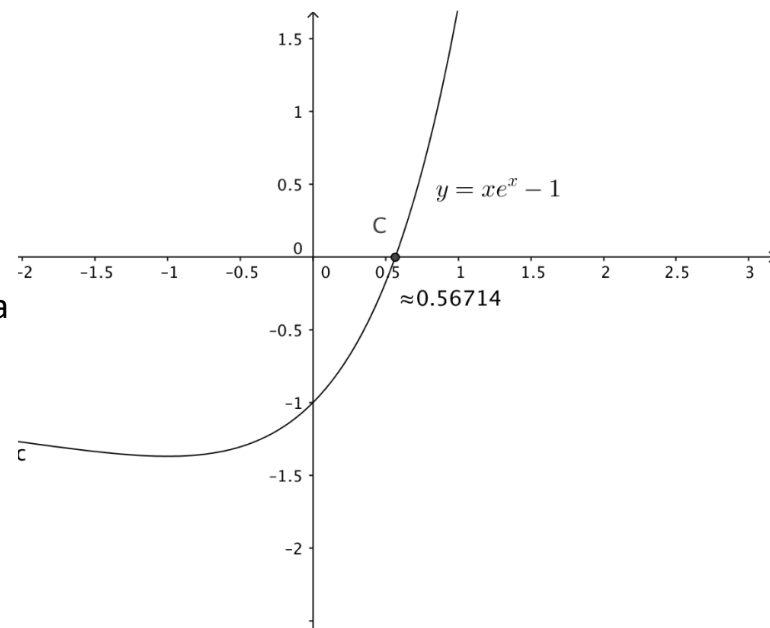
Definizione dell'intervallo

Due sono i criteri utilizzati normalmente:

- 1) Espansione di un intervallo "piccolo" fino a trovare che la funzione cambia di segno.
- 2) Partizionamento di un intervallo "grande" in intervalli più piccoli, fino a trovare uno (o più) intervalli agli estremi dei quali la funzione cambia di segno.

Tolleranza

La precisione richiesta nella determinazione dello zero stesso. Gli algoritmi di ricerca degli zeri si aspettano di ricevere dall'utente l'errore di misura tollerato.



Esercizi - Zero di una funzione v.1.0



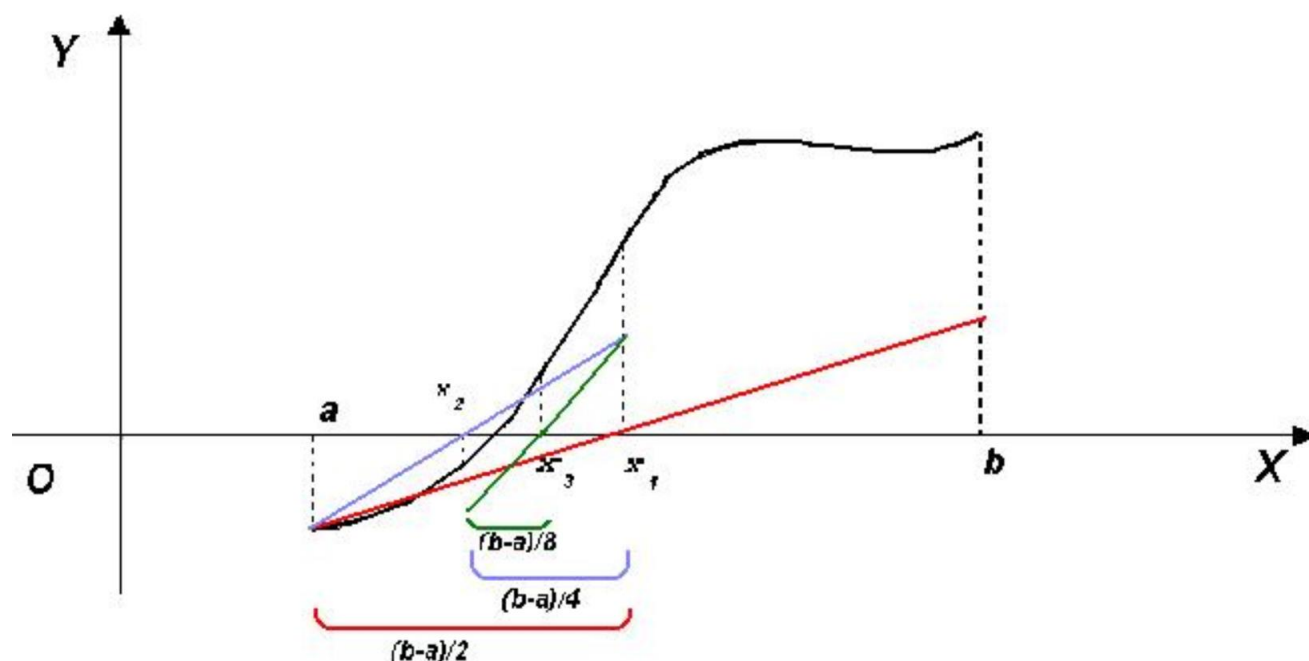
Il metodo di bisezione

Il criterio di ricerca degli zeri più semplice consiste nel continuare a dividere in due l'intervallo di ricerca iniziato, e continuare a scegliere l'intervallo agli estremi del quale la funzione cambia segno.



Zero di una funzione v.1.0

Non è un criterio molto efficiente, ma sicuramente converge, anche se la funzione è discontinua o ha punti di singolarità. Se nell'intervallo ci sono più zeri, convergerà ad uno di questi.



Esercizi - Zero di una funzione v.1.0



Algoritmo

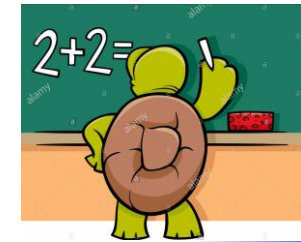
- 1) Impostiamo un **intervallo iniziale** $\{x_1, x_2\} = \{-0.1, 0.1\}$
- 2) Espandiamo l'intervallo moltiplicando gli estremi per un **fattore di espansione**:
 - a) finché la funzione non assume segno opposto ai suoi estremi
 - b) finché non è raggiunto il **numero massimo di espansioni**
- 3) finché il valore assoluto della funzione calcolato nel punto medio x_m dell'intervallo è maggiore di una **tolleranza prefissata**

se $f(x_m) * f(x_1) \geq 0$

$x_1 = x_m;$

altrimenti

$x_2 = x_m;$



Zero di una funzione v.1.0

```
#define ESPANSIONE_FATTORE 1.5
#define ESPANSIONE_MAX_ITERAZIONI 50
#define TOLLERANZA 0.000001
```

```
typedef struct {
    double x1;
    double x2;
} Intervallo;
```

```
void cerca_intervallo(Intervallo *I);
void stampa_intervallo(Intervallo I);
double cerca_zero(Intervallo *I);
```

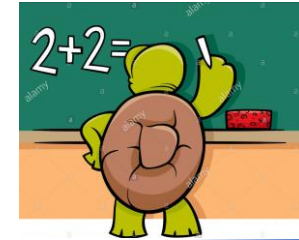
Esercizi - Zero di una funzione v.1.0

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

```
#define ESPANSIONE_FATTORE 1.5
#define ESPANSIONE_MAX_ITERAZIONI 50
#define TOLLERANZA 0.000001
```

```
typedef struct {
    double x1;
    double x2;
} Intervallo;
```

```
void cerca_intervallo(Intervallo *I);
void stampa_intervallo(Intervallo I);
double cerca_zero(Intervallo *I);
```



Zero di una funzione v.1.0

Esercizi - Zero di una funzione v.1.0

```
// funzione  $f(x) = x \cdot \exp(x) - 1$ 
```

```
int main()
```

```
{
```

```
    double zero;
```

```
    Intervallo I={-0.1,0.1};
```

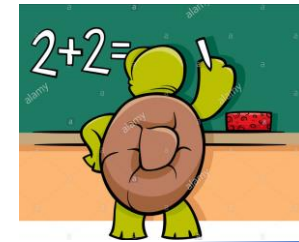
```
    cerca_intervallo(&I);
```

```
    zero = cerca_zero(&I);
```

```
    printf("La funzione ha uno zero in %f\n", zero);
```

```
    return 0;
```

```
}
```



Zero di una funzione v.1.0

Esercizi - Zero di una funzione v.1.0

```
// funzione f(x) = x*exp(x)-1
int main()
{
    double zero;

    Intervallo I={-0.1,0.1};

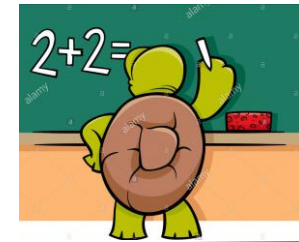
    cerca_intervallo(&I);

    zero = cerca_zero(&I);

    printf("La funzione ha uno zero in %f\n", zero);

    return 0;
}
void stampa_intervallo(Intervallo I)
{
    printf("Intervallo={%f, %f}\n", I.x1, I.x2);

    return;
}
```



Zero di una funzione v.1.0

Esercizi - Zero di una funzione v.1.0

```
void cerca_intervallo(Intervallo *I)
{
    int i=0;
    double y1,y2;

    y1 = I->x1 * exp(I->x1) - 1;
    y2 = I->x2 * exp(I->x2) - 1;

    while(y1*y2>=0 && i<ESPANSIONE_MAX_ITERAZIONI){

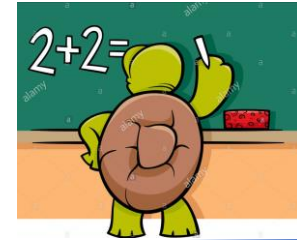
        I->x1 = ESPANSIONE_FATTORE * I->x1;
        I->x2 = ESPANSIONE_FATTORE * I->x2;

        y1 = I->x1 * exp(I->x1) - 1;
        y2 = I->x2 * exp(I->x2) - 1;

        stampa_intervallo(*I);

        ++i;
    }

    return;
}
```



Zero di una funzione v.1.0

Esercizi - Zero di una funzione v.1.0

```
double cerca_zero(Intervallo *I)
{
    double y1,y2;
    double valore=1;
    double xm;

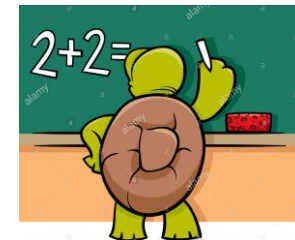
    do{
        xm = (I->x1 + I->x2)/2;
        valore = xm * exp(xm) - 1;

        y1 = I->x1 * exp(I->x1) - 1;
        y2 = I->x2 * exp(I->x2) - 1;

        if(valore * y1 >=0)
            I->x1 = xm;
        else
            I->x2 = xm;

        stampa_intervallo(*I);
        printf("Valore: f(%f)=%f\n", xm, valore);
    }while(fabs(valore) > TOLLERANZA);


    return xm;
}
```



Zero di una funzione v.1.0

Esercizi - Zero di una funzione v.1.0

Intervallo $I = \{-0.1, 0.1\}$;

 Run terminal

```
D:\Lezioni\ProgrammazioneI\Esercizi>Esercizio_23_02.exe
```

```
Intervallo={-0.150000, 0.150000}
```

```
Intervallo={-0.225000, 0.225000}
```

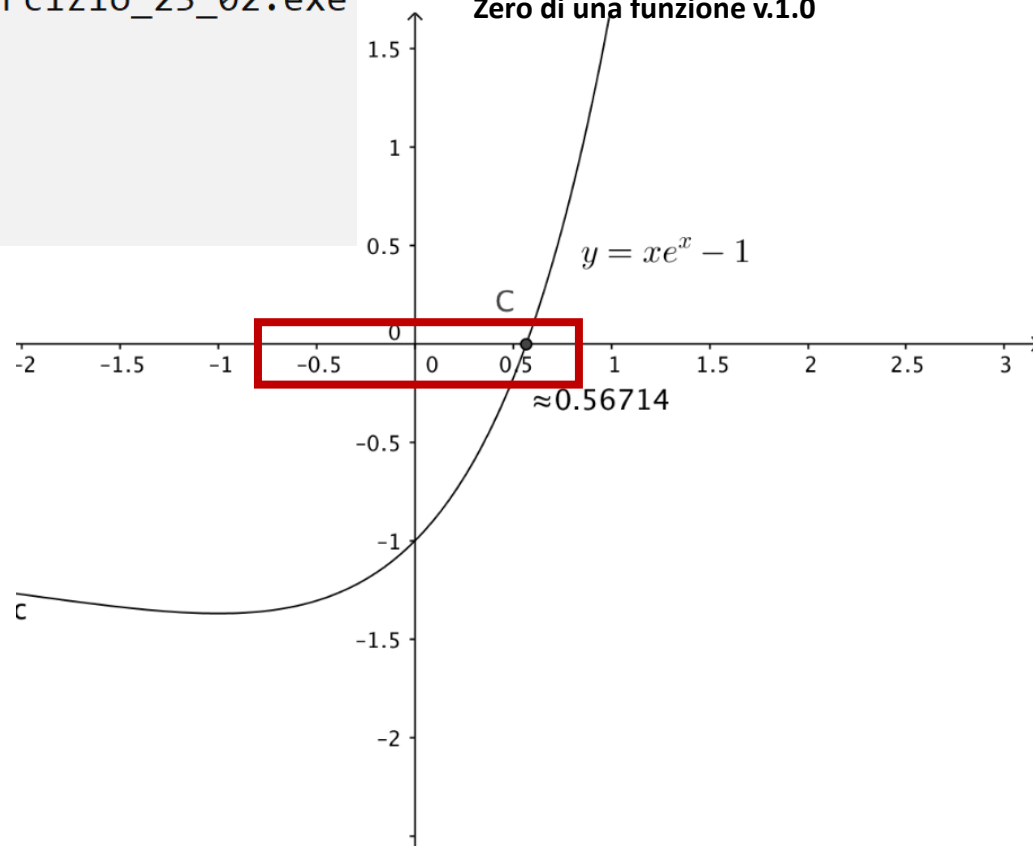
```
Intervallo={-0.337500, 0.337500}
```

```
Intervallo={-0.506250, 0.506250}
```

```
Intervallo={-0.759375, 0.759375}
```



Zero di una funzione v.1.0



Esercizi - Zero di una funzione v.1.0



```
zero = cerca_zero(&I);
```

Run terminal - Esercizio_23_02.exe

```
D:\Lezioni\ProgrammazioneI\Esercizi>Esercizio_23_02.exe
```

```
Intervallo={-0.150000, 0.150000}  
Intervallo={-0.225000, 0.225000}  
Intervallo={-0.337500, 0.337500}  
Intervallo={-0.506250, 0.506250}  
Intervallo={-0.759375, 0.759375}  
Intervallo={0.000000, 0.759375}  
Valore: f(0.000000)=-1.000000
```

```
Intervallo={0.379688, 0.759375}  
Valore: f(0.379688)=-0.444962
```

```
Intervallo={0.379688, 0.569531}  
Valore: f(0.569531)=0.006611
```

```
Intervallo={0.474609, 0.569531}  
Valore: f(0.474609)=-0.237119
```

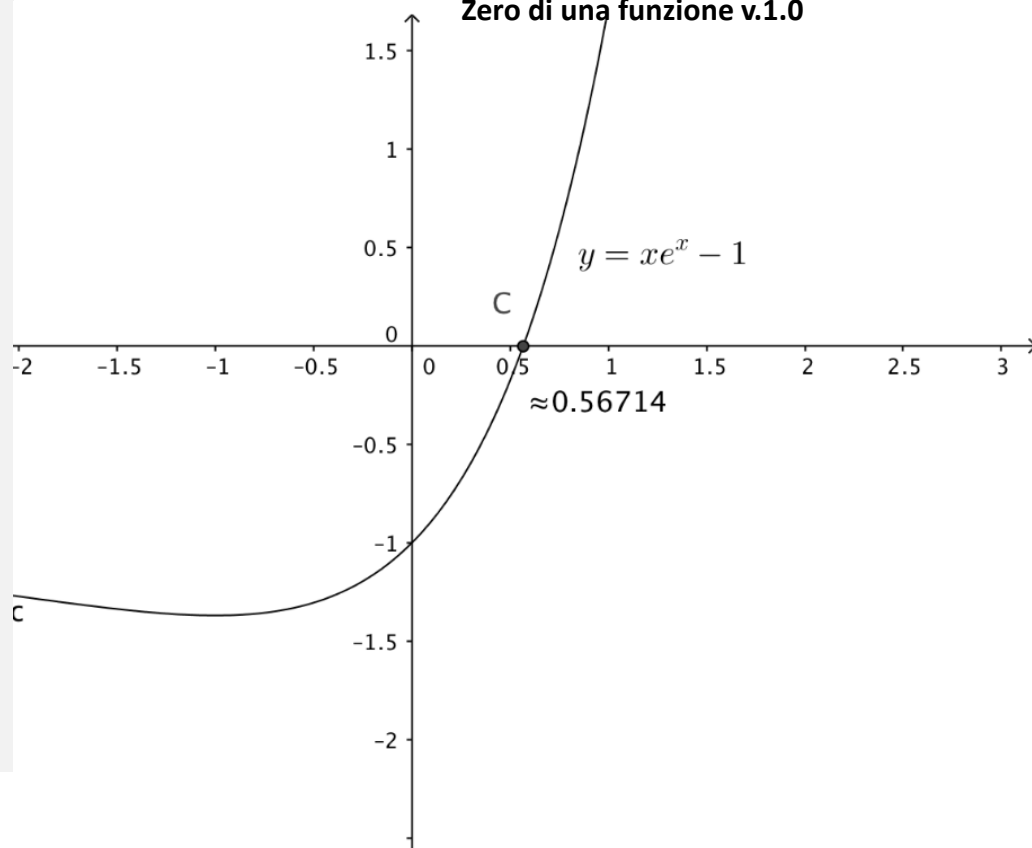
```
Intervallo={0.522070, 0.569531}  
Valore: f(0.522070)=-0.120043
```

```
Intervallo={0.545801, 0.569531}  
Valore: f(0.545801)=-0.057953
```

```
Intervallo={0.557666, 0.569531}  
Valore: f(0.557666)=-0.025985
```



Zero di una funzione v.1.0



Esercizi - Zero di una funzione v.1.0

`zero = cerca_zero(&I);`

Intervallo={0.379688, 0.759375}
Valore: $f(0.379688)=-0.444962$

Intervallo={0.379688, 0.569531}
Valore: $f(0.569531)=0.006611$

Intervallo={0.474609, 0.569531}
Valore: $f(0.474609)=-0.237119$

Intervallo={0.522070, 0.569531}
Valore: $f(0.522070)=-0.120043$

Intervallo={0.545801, 0.569531}
Valore: $f(0.545801)=-0.057953$

Intervallo={0.557666, 0.569531}
Valore: $f(0.557666)=-0.025985$

Intervallo={0.563599, 0.569531}
Valore: $f(0.563599)=-0.009766$

Intervallo={0.566565, 0.569531}
Valore: $f(0.566565)=-0.001597$

Intervallo={0.566565, 0.568048}
Valore: $f(0.568048)=0.002502$

Intervallo={0.566565, 0.567307}
Valore: $f(0.567307)=0.000451$

Intervallo={0.566936, 0.567307}
Valore: $f(0.566936)=-0.000573$

Intervallo={0.567121, 0.567307}
Valore: $f(0.567121)=-0.000061$

Intervallo={0.567121, 0.567214}
Valore: $f(0.567214)=0.000195$

Intervallo={0.567121, 0.567167}
Valore: $f(0.567167)=0.000067$

Intervallo={0.567121, 0.567144}
Valore: $f(0.567144)=0.000003$

Intervallo={0.567133, 0.567144}
Valore: $f(0.567133)=-0.000029$

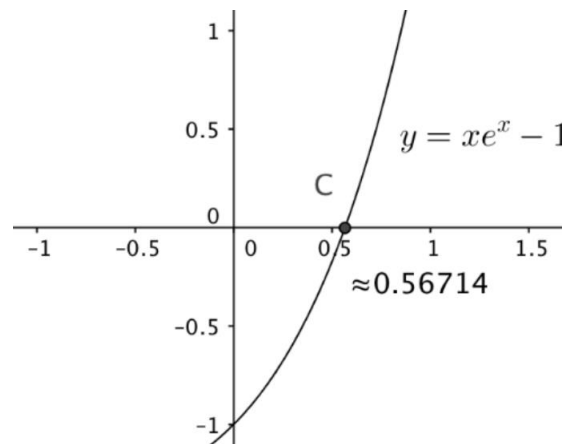
Intervallo={0.567139, 0.567144}
Valore: $f(0.567139)=-0.000013$

Intervallo={0.567141, 0.567144}
Valore: $f(0.567141)=-0.000005$

Intervallo={0.567143, 0.567144}
Valore: $f(0.567143)=-0.000001$

Intervallo={0.567143, 0.567144}
Valore: $f(0.567144)=0.000001$

La funzione ha uno zero in 0.567144



Esercizi - Zero di una funzione v.2.0



Limiti

- 1) I parametri sono fissati in fase di compilazione
- 2) Cambiare la funzione implica variazioni in tutto il codice



Zero di una funzione v.2.0

Varianti

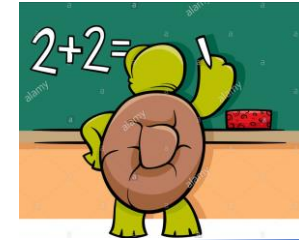
- 1) Dividiamo il programma in più file
- 2) Prendiamo i parametri da riga di comando
- 3) Introduciamo una funzione **double valuta_funzione(double x)** alla quale poi facciamo riferimento con un puntatore a funzione

Esercizi - Zero di una funzione v.2.0



Varianti

- 1) Dividiamo il programma in più file



Zero di una funzione v.2.0

Scriviamo tre file:

- 1) un header file **ZeroFunc.h**
- 2) un source file **ZeroFunc.c**
- 3) un source file **Esercizio_23_02.c**

La compilazione è effettuata mediante la seguente linea di comando:

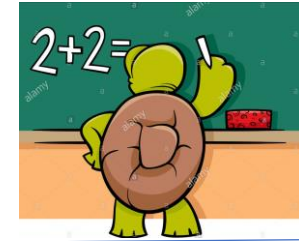
g++ Esercizio_23_02.c ZeroFunc.c **-o** Esercizio_23_02.exe

Esercizi - Zero di una funzione v.2.0



Varianti

2) Prendiamo i parametri da riga di comando



Zero di una funzione v.2.0

Il programma viene chiamato nel modo seguente:

Esercizio_23_02.exe x1 x2 tolleranza fattore_expansione max_iterazioni

All'interno della funzione `int main(int argc, char *argv[])` i parametri vanno controllati

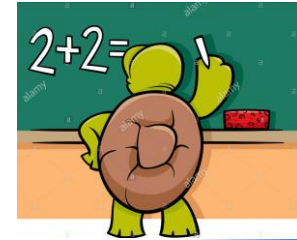
```
if(argc>=3){
    I.x1 = atof(argv[1]);
    I.x2 = atof(argv[2]);
}
if(argc>=4){
    tolleranza = atof(argv[3]);
}
if(argc>=5){
    fattore_expansione = atof(argv[4]);
}
if(argc>=6){
    max_iterazioni = atof(argv[5]);
}
```

Esercizi - Zero di una funzione v.2.0



Varianti

3) Introduciamo una funzione **double valuta_funzione(double x)** alla quale poi facciamo riferimento con un puntatore a funzione



Zero di una funzione v.2.0

Definiamo la variabile:

```
double (*fp)(double x)
```

Effettuiamo l'assegnazione:

```
fp = valuta_funzione;
```

La funzione main deve passare i parametri alle diverse funzioni, compreso il puntatore alla funzione da valutare. I prototipi delle funzioni devono essere modificati come segue:

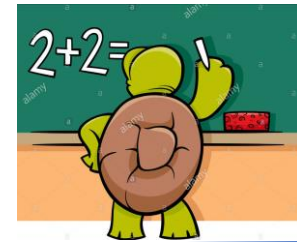
```
void cerca_intervallo(Intervallo *I, double (*fp)(double), double fattore_espansione,  
int max_iterazioni);
```

```
void stampa_intervallo(Intervallo I);
```

```
double cerca_zero(Intervallo *I, double (*fp)(double), double tolleranza);
```

```
double valuta_funzione(double x);
```

Esercizi - Zero di una funzione v.2.0



Zero di una funzione v.2.0

```
#ifndef _ZERO_FUNC_H_
#define _ZERO_FUNC_H_
```

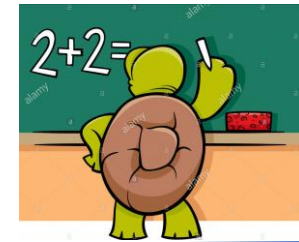
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
typedef struct {
    double x1;
    double x2;
} Intervallo;
```

```
void cerca_intervallo(Intervallo *I, double (*fp)(double), double
fattore_expansione, int max_iterazioni);
void stampa_intervallo(Intervallo I);
double cerca_zero(Intervallo *I, double (*fp)(double), double tolleranza);
double valuta_funzione(double x);
```

```
#endif
```


Esercizi - Zero di una funzione v.2.0



Zero di una funzione v.2.0

```
...
#include "ZeroFunc.h"

int main(int argc, char *argv[])
{
    int max_iterazioni = 50;
    double fattore_expansione = 1.5;
    double tolleranza = 0.000001;
    double zero;
    double (*fp)(double);
    Intervallo I={-0.1,0.1};

    printf("Utilizzo:\n");
    printf(":>zero_func.exe x1 x2 tolleranza fattore_expansione max_iterazioni\n");

    <Controllo dei parametri>

    fp = valuta_funzione;
    cerca_intervallo(&I, fp, fattore_expansione, max_iterazioni);

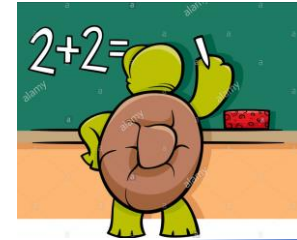
    zero = cerca_zero(&I, fp, tolleranza);
    printf("La funzione ha uno zero in %f\n", zero);

    return 0;
}
```


Esercizi - Zero di una funzione v.2.0



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "ZeroFunc.h"
```



Zero di una funzione v.2.0

```
void stampa_intervallo(Intervallo I)
{
    printf("Intervallo={%f, %f}\n", I.x1, I.x2);

    return;
}
```

```
double valuta_funzione(double x)
{
    return x*exp(x) - 1;
}
```

Esercizi - Zero di una funzione v.2.0



```
void cerca_intervallo(Intervallo *I, double (*fp)(double), double fattore_espansione, int
max_iterazioni)
{
    int i=0;
    double y1,y2;

    y1 = fp(I->x1);
    y2 = fp(I->x2);

    while(y1*y2>=0 && i<max_iterazioni){

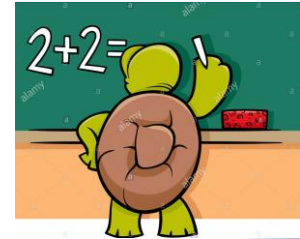
        I->x1 = fattore_espansione * I->x1;
        I->x2 = fattore_espansione * I->x2;

        y1 = fp(I->x1);
        y2 = fp(I->x2);

        stampa_intervallo(*I);

        ++i;
    }

    return;
}
```



Zero di una funzione v.2.0

Esercizi - Zero di una funzione v.2.0

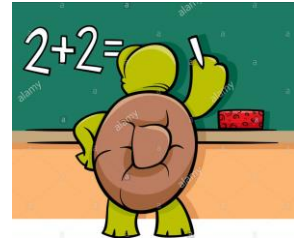
```
double cerca_zero(Intervallo *I, double (*fp)(double), double tolleranza)
{
    double y1,y2;
    double valore=1;
    double xm;
    do{
        xm = (I->x1 + I->x2)/2;
        valore = fp(xm);

        y1 = fp(I->x1);
        y2 = fp(I->x2);
        if(valore * y1 >=0)
            I->x1 = xm;
        else
            I->x2 = xm;

        stampa_intervallo(*I);
        printf("Valore: f(%f)=%f\n", xm, valore);
        getc(stdin);

    }while(fabs(valore) > tolleranza);

    return xm;
}
```



Zero di una funzione v.2.0

Esercizi - Zero di una funzione v.2.0



Run terminal

```
D:\Lezioni\ProgrammazioneI\Esercizi>Esercizio_23_02.exe -0.01 0.01 0.0001 2 100
Utilizzo:
```

```
>:zero_func.exe x1 x2 tolleranza fattore_espansione max_iterazioni
```

```
Intervallo={-0.020000, 0.020000}
```

```
Intervallo={-0.040000, 0.040000}
```

```
Intervallo={-0.080000, 0.080000}
```

```
Intervallo={-0.160000, 0.160000}
```

```
Intervallo={-0.320000, 0.320000}
```

```
Intervallo={-0.640000, 0.640000}
```

```
Intervallo={0.000000, 0.640000}
```

```
Valore: f(0.000000)=-1.000000
```

```
Intervallo={0.320000, 0.640000}
```

```
Valore: f(0.320000)=-0.559319
```

```
Intervallo={0.480000, 0.640000}
```

```
Valore: f(0.480000)=-0.224284
```

```
Intervallo={0.560000, 0.640000}
```

```
Valore: f(0.560000)=-0.019623
```

```
Intervallo={0.560000, 0.600000}
```

```
Valore: f(0.600000)=0.093271
```

```
Intervallo={0.560000, 0.580000}
```

```
Valore: f(0.580000)=0.035902
```

```
Intervallo={0.560000, 0.570000}
```

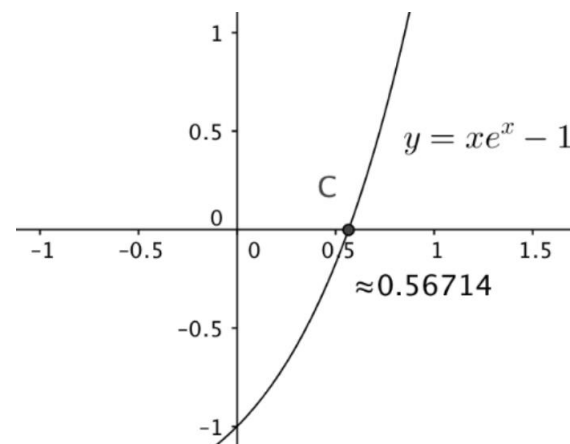
```
Valore: f(0.570000)=0.007912
```

```
Intervallo={0.565000, 0.570000}
```

```
Valore: f(0.565000)=-0.005912
```

```
Intervallo={0.565000, 0.567500}
```

```
Valore: f(0.567500)=0.000986
```



```
Intervallo={0.565000, 0.567500}
```

```
Valore: f(0.567500)=0.000986
```

```
Intervallo={0.566250, 0.567500}
```

```
Valore: f(0.566250)=-0.002467
```

```
Intervallo={0.566875, 0.567500}
```

```
Valore: f(0.566875)=-0.000741
```

```
Intervallo={0.566875, 0.567188}
```

```
Valore: f(0.567188)=0.000122
```

```
Intervallo={0.567031, 0.567188}
```

```
Valore: f(0.567031)=-0.000310
```

```
Intervallo={0.567109, 0.567188}
```

```
Valore: f(0.567109)=-0.000094
```

La funzione ha uno zero in 0.567109

Esercizi - Zero di una funzione v.2.0

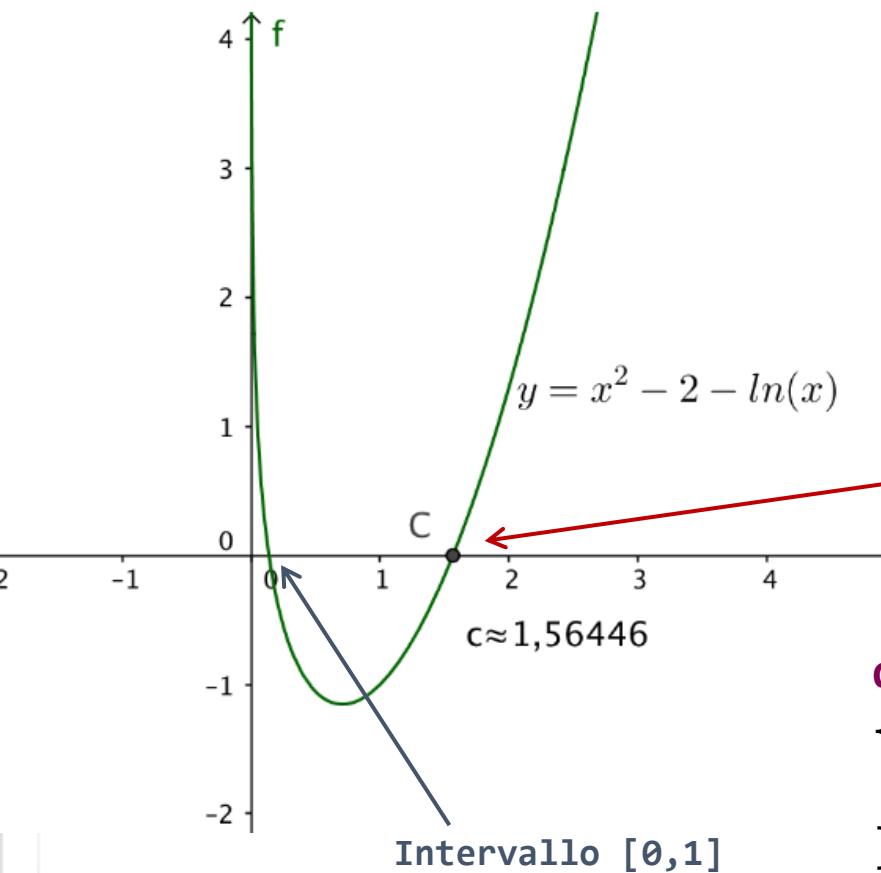


Il metodo di bisezione

Troviamo gli zeri della funzione $f(x) = x^2 - \ln(x) - 2$



Zero di una funzione v.1.0



Intervallo [1,2]

```
double valuta_funzione(double x)
{
    return x*x - log(x) - 2;
}
```

Esercizi - Zero di una funzione v.2.0



Run terminal

```
D:\Lezioni\ProgrammazioneI\Esercizi>Esercizio_23_02.exe 1 2 0.0001 2 100
```

Utilizzo:

```
>zero_func.exe x1 x2 tolleranza fattore_expansione max_iterazioni
```

```
Intervallo={1.500000, 2.000000}
```

```
Valore: f(1.500000)=-0.155465
```

```
Intervallo={1.500000, 1.750000}
```

```
Valore: f(1.750000)=0.502884
```

```
Intervallo={1.500000, 1.625000}
```

```
Valore: f(1.625000)=0.155117
```

```
Intervallo={1.562500, 1.625000}
```

```
Valore: f(1.562500)=-0.004881
```

```
Intervallo={1.562500, 1.593750}
```

```
Valore: f(1.593750)=0.073949
```

```
Intervallo={1.562500, 1.578125}
```

```
Valore: f(1.578125)=0.034241
```

```
Intervallo={1.562500, 1.570313}
```

```
Valore: f(1.570313)=0.014607
```

```
Intervallo={1.562500, 1.566406}
```

```
Valore: f(1.566406)=0.004845
```

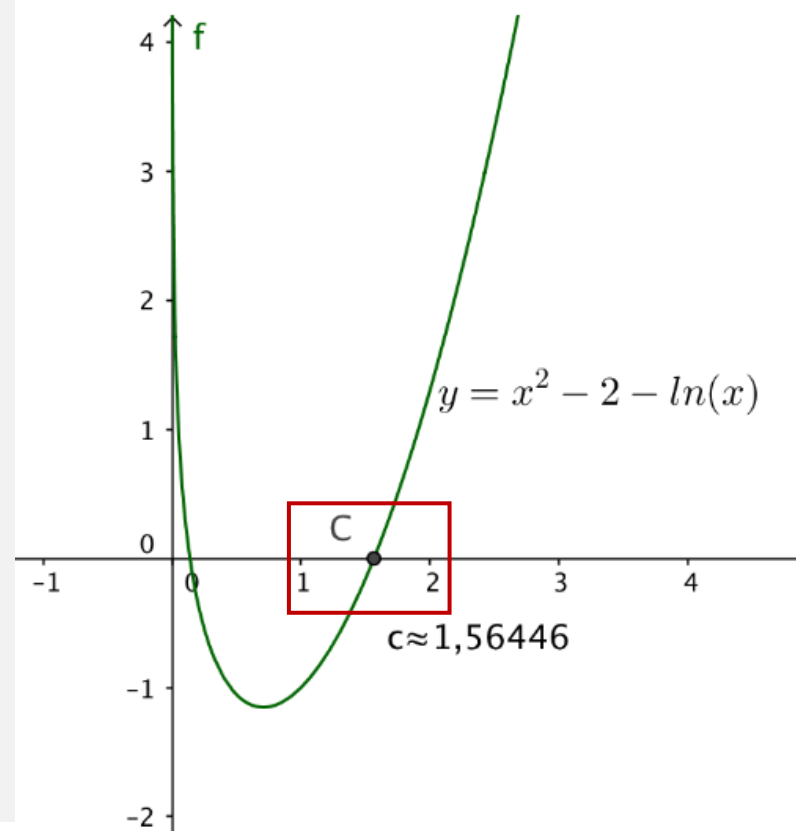
```
Intervallo={1.564453, 1.566406}
```

```
Valore: f(1.564453)=-0.000023
```

La funzione ha uno zero in 1.564453



Zero di una funzione v.1.0



Esercizi - Zero di una funzione v.2.0



Run terminal

```
D:\Lezioni\ProgrammazioneI\Esercizi>Esercizio_23_02.exe 0 1 0.0001 2 100
```

Utilizzo:

```
>zero_func.exe x1 x2 tolleranza fattore_espansione max_iterazioni
```

```
Intervallo={0.000000, 0.500000}
```

```
Valore: f(0.500000)=-1.056853
```

```
Intervallo={0.000000, 0.250000}
```

```
Valore: f(0.250000)=-0.551206
```

```
Intervallo={0.125000, 0.250000}
```

```
Valore: f(0.125000)=0.095067
```

```
Intervallo={0.125000, 0.187500}
```

```
Valore: f(0.187500)=-0.290867
```

```
Intervallo={0.125000, 0.156250}
```

```
Valore: f(0.156250)=-0.119288
```

```
Intervallo={0.125000, 0.140625}
```

```
Valore: f(0.140625)=-0.018566
```

```
Intervallo={0.132813, 0.140625}
```

```
Valore: f(0.132813)=0.036456
```

```
Intervallo={0.136719, 0.140625}
```

```
Valore: f(0.136719)=0.008521
```

```
Intervallo={0.136719, 0.138672}
```

```
Valore: f(0.138672)=-0.005125
```

```
Intervallo={0.137695, 0.138672}
```

```
Valore: f(0.137695)=0.001672
```

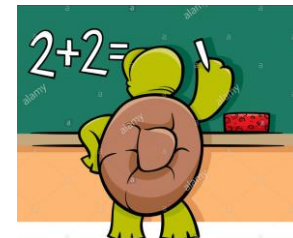
```
Intervallo={0.137695, 0.138184}
```

```
Valore: f(0.138184)=-0.001733
```

```
Intervallo={0.137695, 0.137939}
```

```
Valore: f(0.137939)=-0.000032
```

```
La funzione ha uno zero in 0.137939
```



Zero di una funzione v.1.0

