

Informatica 1

Corso di Laurea Triennale in Matematica

Gianluca Rossi

`gianluca.rossi@uniroma2.it`

Dipartimento di Matematica
Università di Roma "Tor Vergata"

9: Funzioni



Funzione

Definizione della funzione **f**.

```
tipo f(def argomenti){  
    ...  
    corpo;  
    ...  
    return risultato;  
}
```

Chiamata:

```
f(argomenti);
```



Esempio

```
int somma(int); /* dichiarazione */
```

```
main(){  
    int n, tot;  
    tot = somma(10) + somma(n);  
}
```

```
/* definizione funzione somma */
```

```
int somma(int arg){  
    int s,i;  
    s = 0; i = 0;  
    while(i <= arg){  
        s = s + i;  
        i++;  
    }  
    return s;  
}
```



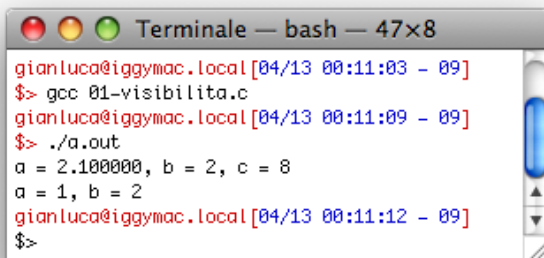
Visibilità (*scope*)

- Le variabili sono dichiarate all'interno di blocchi ($\{\dots\}$).
- La variabile è visibile (con nome e valore) solo dentro il blocco in cui è stata dichiarata. Vale anche nei sotto-blocchi.
- Può essere definita una variabile in un sotto-blocco avente lo stesso nome di una variabile nel super-blocco.
- La variabile nel blocco più interno *nasconde* una eventuale variabile con lo stesso nome nel blocco più esterno.



Visibilità (*scope*)

```
#include<stdio.h>
main(){
    int a = 1, b = 2;
    {
        float a = 2.1;
        short c = 8;
        printf("a = %f, b = %d, c = %d\n", a, b, c);
    }
    printf("a = %d, b = %d\n", a, b);
}
```



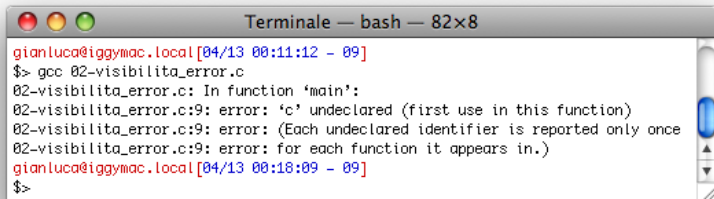
A terminal window titled "Terminale — bash — 47x8" showing the execution of the C program. The user runs `gcc 01-visibilita.c` and `./a.out`. The output shows the values of variables `a`, `b`, and `c` at different points in the program's execution, demonstrating variable scope.

```
gianluca@iggy-mac.local[04/13 00:11:03 - 09]
$> gcc 01-visibilita.c
gianluca@iggy-mac.local[04/13 00:11:09 - 09]
$> ./a.out
a = 2.100000, b = 2, c = 8
a = 1, b = 2
gianluca@iggy-mac.local[04/13 00:11:12 - 09]
$>
```



Visibilità (*scope*)

```
#include<stdio.h>
main(){
    int a = 1, b = 2;
    {
        float a = 2.1;
        short c = 8;
        printf("a = %f, b = %d, c = %d\n", a, b, c);
    }
    printf("a = %d, b = %d, c = %d\n", a, b, c);
}
```

A screenshot of a macOS Terminal window titled "Terminale — bash — 82x8". The window shows the output of compiling a C program. The user enters the command to compile "02-visibilita_error.c". The compiler reports three errors: "In function 'main':", "'c' undeclared (first use in this function)", and "(Each undeclared identifier is reported only once for each function it appears in.)". The terminal prompt is "\$>".

```
gianluca@iggymac.local[04/13 00:11:12 - 09]
$> gcc 02-visibilita_error.c
02-visibilita_error.c: In function 'main':
02-visibilita_error.c:9: error: 'c' undeclared (first use in this function)
02-visibilita_error.c:9: error: (Each undeclared identifier is reported only once
02-visibilita_error.c:9: error: for each function it appears in.)
gianluca@iggymac.local[04/13 00:18:09 - 09]
$>
```



- Una funzione è visibile solo dal momento in cui è dichiarata.

```
float f2(float x){  
    return f1(x)/(1-f1(x));  
}
```

```
float f1(float x){  
    return (3.14*x)/(x-9);  
}
```

```
main(){  
    f2(1.23456);  
}
```

- La funzione **f1** è chiamata prima di essere dichiarata: illegale!!



- Soluzione 1: Disporre le definizioni delle funzioni in base all'ordine in cui vengono chiamate.

```
float f1(float x){  
    return (3.14*x)/(x-9);  
}
```

```
float f2(float x){  
    return f1(x)/(1-f1(x));  
}
```

```
main(){  
    f2(1.23456);  
}
```



Visibilità e funzioni

- Soluzione 2: Far precedere tutte le funzioni (compreso in **main**) dalle dichiarazioni (prototipi).

```
float f1(float);
```

```
float f2(float);
```

```
float f2(float x){  
    return f1(x)/(1-f1(x));  
}
```

```
float f1(float x){  
    return (3.14*x)/(x-9);  
}
```

```
main(){  
    f2(1.23456);  
}
```



- Le variabili dichiarate all'interno di una funzione sono visibili soltanto all'interno della funzione in cui sono dichiarate;
- I parametri delle funzioni vengono passati per valore. La modifica dei parametri non sono visibili all'esterno.
- Possono essere definite **variabili globali** che hanno visibilità in tutto il programma. Da dichiararsi prima delle definizioni di tutte le funzioni.



Visibilità, funzioni e variabili

```
#include<stdio.h>
float f1(float);
float f2(float);

int a = 10; /*variabile globale*/

float f2(float x){
    return f1(x)*a;
}

float f1(float x){
    float a = 2.0;
    return a*x;
}

main(){
    printf("·%f\n", f2(2));
}
```

- Cosa viene stampato?



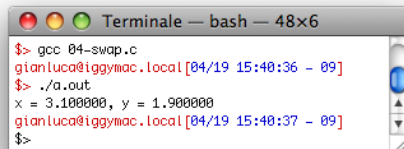
Funzioni che modificano i valori dei parametri

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

```
swap(double, double);
```

```
main(){  
    double x = 3.1, y = 1.9;  
    swap(x, y);  
    printf("x = %f, y = %f\n", x, y);  
}
```

```
swap(double a, double b){  
    double temp;  
    temp = a;  
    a = b;  
    b = temp;  
}
```



```
Terminale — bash — 48x6  
$> gcc 04-swap.c  
gianluca@iggyamac.local[04/19 15:40:36 - 09]  
$> ./a.out  
x = 3.100000, y = 1.900000  
gianluca@iggyamac.local[04/19 15:40:37 - 09]  
$>
```

- L'errore deriva dal fatto che i parametri sono passati per *valore*. Viene creata una copia di *x* e *y* nell'ambiente di **swap**.



Funzioni che modificano i valori dei parametri

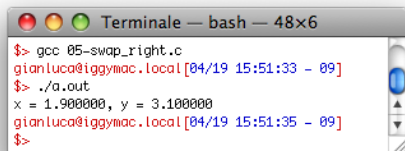
- Il problema può essere aggirato passando alla funzione il *valore* dell'indirizzo della variabile da modificare.

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

```
int swap(double*, double*);
```

```
main(){  
    double x = 3.1, y = 1.9;  
    swap(&x, &y);  
    printf("x = %f, y = %f\n", x, y);  
}
```

```
int swap(double* a, double* b){  
    double temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
}
```



```
Terminale — bash — 48x6  
$> gcc 05-swap_right.c  
gianluca@iggyimac.local[04/19 15:51:33 - 09]  
$> ./a.out  
x = 1.900000, y = 3.100000  
gianluca@iggyimac.local[04/19 15:51:35 - 09]  
$>
```



- Se un parametro di una funzione è un array, la funzione riceve il riferimento (puntatore) alla prima locazione del primo elemento dell'array.
- Le modifiche all'array all'interno della funzione avrà ripercussioni anche all'esterno.

Prototipi e definizioni

```
tipo1 f(tipo2 v[dim]);  
tipo1 f(tipo2 *v);  
tipo1 f(tipo2 v[]);
```

- In ogni caso, l'unica informazione passata alla funzione sul vettore è il suo indirizzo: niente informazioni sulla dimensione



Funzioni ed array: Esempio

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

```
int reverse(int[], int);
```

```
int swap(int*, int*);
```

```
main(){
```

```
    int i, dim, v[] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
```

```
    dim = sizeof(v)/sizeof(int);
```

```
    reverse(v, dim);
```

```
    for(i = 0; i < dim; i++)
```

```
        printf("%d\\n", v[i]);
```

```
}
```

```
int reverse(int a[], int n){
```

```
    int i;
```

```
    for(i = 0; i < n/2; i++){
```

```
        /* scambiamo a[i] con a[n-i-1] */
```

```
        swap(a+i, a+n-i-1);
```

```
        /* oppure swap(&a[i], &a[n-i-1]);*/
```

```
    }
```

```
}
```

```
int swap(int* a, int* b){
```

```
    int temp;
```

```
    temp = *a;
```

```
    *a = *b;
```

```
    *b = temp;
```

```
}
```



Funzioni ed array multidimensionali

```
int f(int[][3], int, int);
```

```
main(){  
    int A[2][3] = {{1,2,3}, {4,5,6}};  
    f(A, 2, 3);  
}
```

```
int f(int X[][3], int n, int m){  
    int i, j;  
    for(i = 0; i < n; i++)  
        for(j = 0; j < m; j++)  
            X[i][j] = X[i][j] + 1;  
}
```

- È necessario indicare nel prototipo e nella definizione della funzione la grandezza del secondo indice (e, per dimensioni maggiori, anche quelli successivi). Perché?
- Il valore `*(A + 1)` si riferisce al puntatore alla seconda riga della matrice. Quindi occorre conoscere quanti elementi compongono le righe della matrice.

