RIPASSO C
Parte 4

(Slides cortesemente concesse in uso dal Prof. Fraschini)

Funzioni

Parte 1

Funzioni: intro

- Le funzioni rappresentano:
  - dei *sottoprogrammi* costituiti da un insieme di istruzioni
  - un meccanismo che permette quindi di realizzare programmi più chiari da leggere, da capire e da modificare

- Attraverso l'uso delle funzioni si evita di dover riscrivere più volte del codice. E' possibile infatti chiamare una funzione più volte in un programma.
- Programmazione *modulare*
- Creazione *librerie* (scanf, printf) che nascondono i dettagli implementativi

Funzioni: intro

valori in ingresso -> FUNZIONE -> valore restituito

Funzioni: definizione

- Abbiamo già usato le funzioni!
  - main(), printf(), scanf()

- Importante: scelta del nome della funzione!

void nome_funzione(void); **<---- DICHIARAZIONE**

- void può essere sostituito con i parametri da passare alla funzione
- void può essere sostituito con il tipo di valore che la funzione restituisce

Primo esempio

```c
#include <stdio.h>

void stampa_messaggio();

int main()
{
  stampa_messaggio();
  return 0;
}

void stampa_messaggio()
{
    printf("La mia prima funzione!\n");
}
```

Dichiarazione della funzione stampa_messaggio()

Chiamata della funzione stampa_messaggio()

Definizione della funzione stampa_messaggio()

NB: L'esecuzione del programma inizia sempre dal main()!!!

Altri esempi

```c
/* pari o dispari */
#include <stdio.h>

void pari_dispari(int);

int main()
{
    int num;

    printf("Inserisci un numero:\n");
    scanf("%d",&num);
    pari_dispari(num);
    return 0;
}

void pari_dispari(int n)
{
    if (n % 2 == 0) printf("%d e' pari\n", n); else printf("%d e' dispari\n", n);
}
```

Altri esempi

- **n** è il nome del **parametro formale**
- **num** è il nome del **parametro attuale**

```c
/* pari o dispari */
#include <stdio.h>

void pari_dispari(int);

int main()
{
    int num;

    printf("Inserisci un numero:\n");
    scanf("%d",&num);
    pari_dispari(num);
    return 0;
}

void pari_dispari(int n)
{
    if (n % 2 == 0) printf("%d e' pari\n", n); else printf("%d e' dispari\n", n);
}
```

Altri esempi

```c
//Tabelline N x 10
#include <stdio.h>

void tabellina(int);

int main()
{
    int num;

    printf("Inserisci un numero (tabellina da calcolare): \n");
    scanf("%d",&num);
    tabellina(num);
    return 0;
}

void tabellina(int n)
{
    int j;
    for(j=1;j<=10;j++)
        printf("%d X %d = %d \n",n,j,n*j);
}
```

n e j sono **variabili locali (automatiche)**

Altri esempi

Passaggio di più parametri

- **Numero**
- **Tipo**
- **Ordine**

```c
/* BMI*/
#include <stdio.h>

void calcolo_bmi(float, float);

int main()
{
    float altezza,peso;

    printf("Inserisci altezza in m:\n");
    scanf("%f",&altezza);

    printf("Inserisci peso in Kg:\n");
    scanf("%f",&peso);

    calcolo_bmi(altezza, peso);

    return 0;
}

void calcolo_bmi(float a, float p)
{
    float bmi;
    bmi=p/(a*a);
    printf("Il tuo BMI vale:\n %.2f\n",bmi);
}
```

Altri esempi

Restituire un valore

```c
/* BMI*/
#include <stdio.h>

float calcolo_bmi(float, float);

int main()
{
    float altezza,peso,res;

    printf("Inserisci altezza in m:\n");
    scanf("%f",&altezza);

    printf("Inserisci peso in Kg:\n");
    scanf("%f",&peso);

    res=calcolo_bmi(altezza, peso);
    printf("Il tuo BMI vale:\n %.2f\n",res);
    //printf("Il tuo BMI vale:\n %.2f\n",calcolo_bmi(altezza, peso));

    return 0;
}

float calcolo_bmi(float a, float p)
{
    float bmi;
    bmi=p/(a*a);
    return bmi;
}
```

Altri esempi

Una funzione non modifica i valori dei parametri che vengono passati

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato(int);

int main()
{
    int num,res;

    printf("Inserisci numero intero:\n");
    scanf("%d",&num);

    res=calcolo_quadrato(num);
    printf("Il quadrato di %d vale: %d\n",num,res);

    return 0;
}

int calcolo_quadrato(int n)
{
    n=n*n;
    return n;
}
```

```
Inserisci un numero intero:
5
Il quadrato di 5 vale 25.
```

Scope (visibilità)

- La dichiarazione di una variabile introduce un nome simbolico visibile all'interno di una specifica parte (*blocco*) del programma
- Nome locale – **variabile locale**
  - visibilità solo all'interno del blocco delle istruzioni della funzione: dal punto della dichiarazione alla fine del blocco
- Nome globale – **variabile globale**
  - definita fuori dalla funzione: dal punto della dichiarazione alla fine del file

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato(int);
int n=3;

int main()
{
    int n=4; //MASCHERAMENTO

    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato(n));
    return 0;
}

int calcolo_quadrato(int n)
{
    n=n*n;
    return n;
}
```

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato(int);
int n=3;

int main()
{
    int n=4; //MASCHERAMENTO

    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato(n));
    return 0;
}

int calcolo_quadrato(int n)
{
    n=n*n;
    return n;
}
```

```
Il quadrato di 4 vale: 16
```

ATTENZIONE AL MASCHERAMENTO!

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato(int);
int n=3;

int main()
{
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato(n));
    return 0;
}

int calcolo_quadrato(int n)
{
    n=n*n;
    return n;
}
```

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato(int);
int n=3;

int main()
{
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato(n));
    return 0;
}

int calcolo_quadrato(int n)
{
    n=n*n;
    return n;
}
```

```
Il quadrato di 3 vale: 9
```

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato();
int n=3;

int main()
{
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato());
    printf("Ora n vale: %d\n",n);
    return 0;
}

int calcolo_quadrato()
{
    n=n*n;
    return n;
}
```

ATTENZIONE!!!

Ora n vale: **?**

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato();
int n=3;

int main()
{
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato());
    printf("Ora n vale: %d\n",n);
    return 0;
}

int calcolo_quadrato()
{
    n=n*n;
    return n;
}
```

ATTENZIONE!!!

```
Il quadrato di 3 vale: 9
Ora n vale: 9
```

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato();
int n=3;

int main()
{
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato());
    int n=5;
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato());
    return 0;
}

int calcolo_quadrato()
{
    n=n*n;
    return n;
}
```

```
Il quadrato di 3 vale: 9
Il quadrato di 5 vale: 81
```

```c
/* quadrato*/
#include <stdio.h>

int calcolo_quadrato();
int n=3;

int main()
{
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato());
    n=5;
    printf("Il quadrato di %d vale: %d\n",n,calcolo_quadrato());
    return 0;
}

int calcolo_quadrato()
{
    n=n*n;
    return n;
}
```

```
Il quadrato di 3 vale: 9
Il quadrato di 5 vale: 25
```

Scope

'global' (internal linkage)	int n;	int n;	int n;	-	-
main()	int n;	int n;	(r/w n)	int n;	int n;
f()	int n;	(r/w n)	(r/w n)	int n;	(r/w n)
	- *Different* variables - 'local' hides 'global'	f() can read and write *'global'*	f() and main() can read and write *'global'*	- *Different* variables	*Error: f() cannor r/w n*
	scope01.c	scope02.c	scope03.c	scope04.c	scope05.c

Vedere i relativi file scopeX.c

Scope

```c
#include <stdio.h>
void f(void);
int n=10; //global
int main() {
  int n=5; // declare n
  printf("n in main: %d \n",n);
  f();
  printf("n in main: %d \n",n);
  getchar();
  return 0;
}
```

```c
void f(){
  int n=1; // declare n
  printf("n in f(): %d \n",n);
  return;
}
```

```
n in main: 5
n in f(): 1
n in main: 5
```

Scope

```c
#include <stdio.h>
void f(void);
int n=10; //global
int main() {
  int n=5; // declare n
  printf("n in main: %d \n",n);
  f();
  printf("n in main: %d \n",n);
  f();
  getchar();
  return 0;
}
```

```c
void f(){
  printf("n in f(): %d \n",n);
  n=n+1;
  printf("n in f(): %d \n",n);
  return;
}
```

```
n in main: 5
n in f(): 10
n in f(): 11
n in main: 5
n in f(): 11
n in f(): 12
```

Scope

```c
#include <stdio.h>
void f(void);
int n=10; //global
int main() {
  printf("n in main: %d \n",n);
  n=n+1;
  f();
  printf("n in main: %d \n",n);
  f();
  getchar();
  return 0;
}
```

```c
void f(){
  printf("n in f(): %d \n",n);
  n=n+1;
  printf("n in f(): %d \n",n);
  return;
}
```

```
n in main: 10
n in f(): 11
n in f(): 12
n in main: 12
n in f(): 12
n in f(): 13
```

Scope

```c
#include <stdio.h>
void f(void);
int main() {
  int n=5;
  printf("n in main: %d \n",n);
  f();
  printf("n in main: %d \n",n);
  getchar();
  return 0;
}
```

```c
void f(){
  int n=10;
  printf("n in f(): %d \n",n);
  return;
}
```

```
n in main: 5
n in f(): 10
n in main: 5
```

Scope

```c
#include <stdio.h>
void f(void);
int main() {
  int n=5;
  printf("n in main: %d \n",n);
  f();
  getchar();
  return 0;
}
```

```c
void f(){
  printf("n in f(): %d \n",n);
  return;
}
```

error: use of undeclared identifier 'n'
printf("n in f(): %d \n",n);

Passaggio dei parametri

- Parametri **formali**: definizione della funzione (*tipo, numero e ordine*)
- Parametri **attuali**: passati alla funzione quando viene chiamata

- In C il passaggio dei parametri avviene solo ***per valore***!
  - Durante la chiamata di una funzione ogni parametro formale viene inizializzato con il valore del corrispondente parametro attuale

```c
#include <stdio.h>
void scambia(int, int);

int main()
{
    int x=0,y=1;

    printf("Prima dello scambio x = %d e y = %d\n",x,y);
    scambia(x,y);
    printf("Dopo lo scambio x = %d e y = %d\n",x,y);
    return 0;
}

void scambia(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
```

```
Prima dello scambio x = 0 e y = 1
Dopo lo scambio x = 0 e y = 1
```

```c
#include <stdio.h>
void scambia(int *, int *);

int main()
{
    int x=0,y=1;

    printf("Prima dello scambio x = %d e y = %d\n",x,y);
    scambia(&x,&y);
    printf("Dopo lo scambio x = %d e y = %d\n",x,y);
    return 0;
}

void scambia(int *x, int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

```
Prima dello scambio x = 0 e y = 1
Dopo lo scambio x = 1 e y = 0
```

Funzioni e array

- E’ possibile passare ad una funzione il valore di un elemento di un array o un *array**
- **Nel primo caso**

  nome_funzione(nome_array[i]);
- **Nel secondo caso**

  nome_funzione(nome_array);

Si dice che il nome dell'array 'decade' al puntatore al suo primo elemento

*attenzione a questo caso!

Esempio

Quando un array viene passato ad una funzione viene passato l'indirizzo del primo elemento dell'array!

Un parametro definito come <u>nome di un array</u> è un puntatore!

SCRITTURE EQUIVALENTI:

```c
int calcola_minimo(int val[], int n_el);

int calcola_minimo(int *val, int n_el);
```

```c
#include <stdio.h>
#define DIM 100

int calcola_minimo(int [], int);

int main()
{
    int valori[DIM],min,n;

    printf("Quanti valori vuoi inserire?\n");
    scanf("%d",&n);
    printf("Inserisci i %d valori:\n",n);
    for(int i=0;i<n;++i)
    {
        printf("Inserisci il valore %d: ",i+1);
        scanf("%d",&valori[i]);
    }
    min=calcola_minimo(valori,n);
    printf("Il valore più basso e': %d\n",min);
    return 0;
}

int calcola_minimo(int val[], int n_el)
{
    int min;
    min=val[0];
    for(int i=0;i<n_el;++i)
    {
        if(val[i]<min)
            min=val[i];
    }
    return min;
}
```

Esempio

```c
#include <stdio.h>
#define DIM 100

int calcola_minimo(int *, int);

int main()
{
    int valori[DIM],min,n;

    printf("Quanti valori vuoi inserire?\n");
    scanf("%d",&n);
    printf("Inserisci i %d valori:\n",n);
    for(int i=0;i<n;++i)
    {
        printf("Inserisci il valore %d: ",i+1);
        scanf("%d",&valori[i]);
    }
    min=calcola_minimo(valori,n);
    printf("Il valore più basso e': %d\n",min);
    return 0;
}
```

```c
int calcola_minimo(int *val, int n_el)
{
    int min;
    min=*val;
    for(int i=0;i<n_el;++i)
    {
        if(*(val+i)<min)
            min=*(val+i);
    }
    return min;
}
```

```c
/* str_len passaggio array*/
#include <stdio.h>

int str_len(char *);

int main()
{
    char str[]="test";
    printf("La stringa %s e' lunga %d\n",str,str_len(str));

    return 0;
}

int str_len(char *p)
{
    int i=0;
    while(*p++ != '\0') i++;

    return i;
}
```