Informatica

procedure e funzioni

- dichiarazione e definizione
  - definizione del comportamento della funzione
  - utilizzo di ***parametri formali***
  - prototipo (solo dichiarazione)
- esecuzione
  - esecuzione della funzione (chiamata)
  - utilizzo di ***parametri attuali***

procedure in C

- in linguaggio C / C++ non esistono procedure ma solo funzioni (funzioni **`void`**)
- la ***dichiarazione*** di una procedura deve essere inserita ***prima*** della sua esecuzione
- utilizzando i ***prototipi*** è possibile definire le procedure dopo il programma principale
- l'esecuzione di una procedura termina quando si raggiunge la sua ultima istruzione o si incontra l'istruzione **`return`**

Esempio in linguaggio C

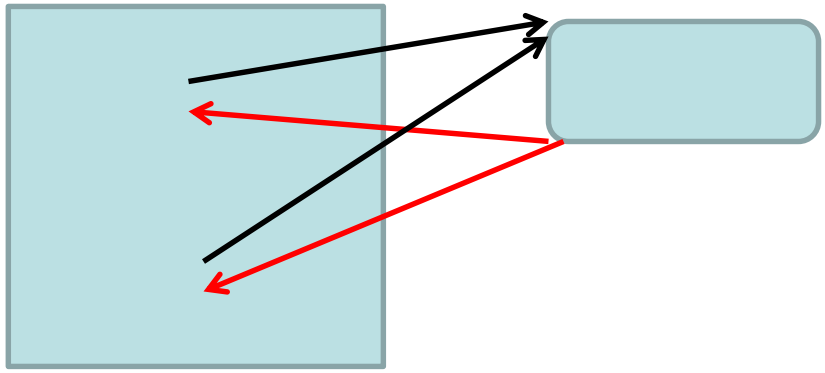

```c
/*Esempio funzioni*/
#include<stdio.h>
void stampaLogo()           // dichiarazione e definizione della procedura
{
    printf("\n");
    printf("**********************************\n");
    printf("*** Classe III A Informatica ***\n");
    printf("***  Itis Leonardo da Vinci  ***\n");
    printf("**********************************\n");
}
int main()
{
    stampaLogo();           // chiamata della procedura
    printf("\nProgramma di prova\n");
    stampaLogo();           // chiamata della procedura
}
```

```c
/*Esempio funzioni*/
#include<stdio.h>
void stampaLogo();              // dichiarazione (prototipo)

int main()
{
    stampaLogo();               // chiamata della procedura
    printf("\nProgramma di prova\n");
    stampaLogo();               // chiamata della procedura
}
void stampaLogo()               // definizione della procedura
{
    printf("\n");
    printf("*********************************\n");
    printf("*** Classe III A Informatica ***\n");
    printf("***  Itis Leonardo da Vinci  ***\n");
    printf("*********************************\n");
}
```

esecuzione del codice

- la chiamata di una funzione provoca
  - l'interruzione momentanea dell'esecuzione del codice
  - l'esecuzione del codice della funzione
  - al termine dell'esecuzione di questa, la ripresa del codice inizialmente sospeso



variabili globali

A. Ferrari

```c
/*Variabili globali*/
#include<stdio.h>

int x;                          // variabile globale visibile sia dalla procedura che dal main
int cub;                        // variabile globale visibile sia dalla procedura che dal main
int n;                          // variabile globale visibile sia dalla procedura che dal main

void stampaCubo()               // dichiarazione e definizione della procedura
{
    cub=x*x*x;
    printf("Il cubo di %d vale %d \n",x,cub);
}

int main()
{
    printf("Inserisci un valore ");
    scanf("%d",&n);
    for (x=1;x<=n;x++)
        stampaCubo();           // chiamata della procedura
}
```

variabili globali e locali

A. Ferrari

```c
/*Variabili globali e locali*/
#include<stdio.h>
int x;                          // variabile globale visibile sia dalla procedura che dal main
void stampaCubo()               // dichiarazione e definizione della procedura
{
    int cub;                    // variabile locale visibile solo dalla procedura
    cub=x*x*x;
    printf("Il cubo di %d vale %d \n",x,cub);
}
int main()
{
    int n;                      // variabile locale visibile solo dal main
    printf("Inserisci un valore ");
    scanf("%d",&n);
    for (x=1;x<=n;x++)
        stampaCubo();           // chiamata della procedura
}
```

- la funzione ha un tipo (il tipo dl valore restituito) e una terminazione esplicita (**return** seguito da una espressione che rappresenta il valore della funzione)
- la funzione opera formalmente su variabili (***parametri formali***) che vengono associate a valori specifici al momento della chiamata (***parametri attuali***)

```c
/*Funzioni*/
#include<stdio.h>
#include <time.h>
// restituisce un valore casuale compreso fra 1 e 6 (rappresenta il lancio di un dado)
int lancio()
{
    int faccia;                         // faccia del dado
    faccia = (rand() % 6) + 1;          // numero casuale compreso fra 1 e 6
    return faccia;
}
int main()
{
    srand(time(0));      // inizializzazione numeri casuali
    int numeroLanci;     // numero totale dei lanci da effettuare
    int nl;              // numero del lancio effettuato
    printf("Numero di lanci da effettuare: ");
    scanf("%d",&numeroLanci);
    for (nl=1;nl<=numeroLanci;nl++)
        printf("Lancio N.ro %d e' uscito il numero %d \n",nl,lancio());
}
```

passaggio per valore


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

float media(int a,int b)// calcola e restituisce la media fra i valori di a e di b
{
  // a e b sono parametri formali al momento dell'esecuzione avviene l'assegnamento
  // fra il valore dei parametri attuali e i parametri formali
  float calcolo;
  calcolo=(float)(a+b)/2;
  return calcolo;
}
int main()
{
    int x,y;            // saranno utilizzati come parametri attuali
    float m;
    printf("Inserisci due valori interi \n");
    scanf("%d",&x);
    scanf("%d",&y);
    m = media(x,y);     // chiamata della funzione e passaggio dei parametri
    printf("la media fra %d e %d e' %f \n",x,y,m);
    // il parametri attuale può essere una qualsiasi espressione (controllo sui tipi)
    m = media(x-1,3);
    printf("la media fra %d e %d e' %f \n",x-1,3,m);
}
```

passaggio per valore

- al momento dell'esecuzione della funzione il valore del parametro attuale viene copiato nella variabile che rappresenta il parametro formale
- Il passaggio per valore non permette però di "ricordare" le modifiche apportate ai parametri da parte della funzione
- La memoria locale della funzione (che contiene i parametri formali) scompare nel momento in cui la funzione termina la sua esecuzione.

esempio


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

void scambia(int a,int b)// // scambia i valori di a e b
{
    int app;
    app = a;
    a = b;
    b = app;
}
int main()
{
    int x,y;            // saranno utilizzati come parametri attuali
    x = 3;
    y = 5;
    printf("il valore di x e' %d e il valore di y e' %d \n",x,y);
    scambia(x,y);
    printf("il valore di x e' %d e il valore di y e' %d \n",x,y);
}
```

passaggio per indirizzo

A. Ferrari

- al momento dell'esecuzione della funzione l'***indirizzo*** del parametro attuale (&) viene copiato nel puntatore (*) che rappresenta il parametro formale

```c
/*Funzioni: Passaggio per indirizzo*/
#include<stdio.h>

void scambia(int *a,int *b)// // scambia i valori di a e b
{
    int app;
    app = *a;
    *a = *b;
    *b = app;
}
int main()
{
    int x,y;            // saranno utilizzati come parametri attuali
    x = 3;
    y = 5;
    printf("il valore di x e' %d e il valore di y e' %d \n",x,y);
    scambia(&x,&y);
    printf("il valore di x e' %d e il valore di y e' %d \n",x,y);
}
```

array come parametro

A. Ferrari

- è necessario passare anche il numero di elementi dell'array

```c
/*Funzioni: Array come parametro*/
#include<stdio.h>
#include <time.h>
void stampaArray(int v[], int n)            // stampa gli elementi dell'array v di n elementi
{
    int i;
    for (i=0; i<n; i++)
        printf("%d ",v[i]);
    printf("\n");
}
int main()
{
    int valori[10];     // conterrà 10 valori casuali compre si fra 0 e 99
    srand(time(0));
    int i;
    for (i = 0; i<10 ; i++)
        valori[i]=rand()%100;
    stampaArray(valori,10);                  // necessario passare anche la dimensione
}
```

gli array sono sempre passati per riferimento

A. Ferrari

```c
#include<stdio.h>
#include <time.h>
void stampaArray(int v[], int n)           // stampa gli elementi dell'array v di n elementi
{
    int i;
    for (i=0; i<n; i++)
        printf("%d ",v[i]);
    printf("\n");
}
void raddoppiaValori(int v[], int n)        // raddoppia il valore degli elementi dell'array
{
    int i;
    for (i=0; i<n; i++)
        v[i] = v[i] * 2;
}
int main()
{
    int valori[10];     // conterrà 10 valori casuali compre si fra 0 e 99
    int i;
    srand(time(0));
    for (i = 0; i<10 ; i++)
        valori[i]=rand()%100;
    stampaArray(valori,10);                 // necessario passare anche la dimensione
    raddoppiaValori(valori,10);
    stampaArray(valori,10);
}
```

procedure e funzioni perché?

- evitare duplicazioni del codice
  - con le procedure si evita di duplicare parti del codice sorgente, quando si chiama o invoca una procedura si esegue il codice corrispondente
  - a ogni nuova chiamata il suo codice è eseguito nuovamente
  - la duplicazione pone due tipi di problemi:
    - aumento della lunghezza del codice e quindi minore leggibilità
    - difficoltà nell'apportare modifiche che devono essere effettuate in tutte le copie del codice
- … ma questa non è la motivazione più importante

progettazione modulare

- per affrontare problemi ***complessi*** si ricorre alla tecnica dei ***raffinamenti*** successivi che suggerisce di ***scomporre*** il problema in problemi più semplici (***sottoproblemi***)
- … e di applicare anche a questi sottoproblemi la stessa tecnica fino ad ottenere problemi facilmente risolvibili
- questa tecnica è definita ***top-down***:
  - si parte da una visione globale del problema (alto livello di astrazione) [***top***]
  - poi si scende nel dettaglio dei sottoproblemi diminuendo il livello di astrazione [***down***]
  - viene fornita inizialmente una soluzione del problema che non si basa però su operazioni elementari, ma sulla soluzione di sottoproblemi

metodologia top down

- se il sottoproblema è semplice allora viene risolto, viene cioè scritto l'algoritmo di risoluzione
- se il sottoproblema è complesso viene riapplicato lo stesso procedimento scomponendolo in sottoproblemi più semplici
- diminuisce il livello di astrazione
  (si affrontano problemi sempre più concreti)
- diminuisce il livello di complessità
  (i sottoproblemi devono essere più semplici del problema che li ha originati)
- fino ad arrivare alla stesura di tutti gli algoritmi necessari

top down e bottom up

- i modelli top-down e bottom-up (dall'alto verso il basso e dal basso verso l'alto) sono ***strategie*** di elaborazione dell'informazione e di gestione delle conoscenze, riguardanti principalmente il software …
- nel modello **top-down** è formulata una visione generale del sistema senza scendere nel dettaglio di alcuna delle sue parti
  - ogni parte del sistema è successivamente rifinita aggiungendo maggiori dettagli dalla progettazione
- nella progettazione **bottom-up** parti individuali del sistema sono specificate in dettaglio
  - queste parti vengono poi connesse tra loro in modo da formare componenti più grandi, che vengono a loro volta interconnessi fino a realizzare un sistema completo

Wikipedia