10 - Argomenti Avanzati

Puntatori a funzione

Il nome di una funzione si comporta come il nome di un array, ovvero rappresenta **l'indirizzo in memoria di partenza del codice** che esegue il compito della funzione.
Le operazioni eseguibili da un puntatore a funzione sono la possibilità di essere passati alle funzioni, restituiti dalle funzioni, memorizzati negli array e assegnati ad altri puntatori a funzione.
Se come argomento si passa una funzione con due parametri di un certo tipo, è possibile utilizzare un puntatore a funzione che abbia la stessa firma.
`int esegui_operazione(int op1, int op2, int (*operazione)(int a, int b))`
possiamo utilizzare come puntatore per *operazione*, ad esempio, la funzione: `int somma(int a,int b)`

Menù a scelta multipla

Un programma richiede ad un utente di selezionare un opzione da un menu, scrivendo il numero dell'elemento associato a quella determinata funzione nel menu; questo può essere un ottimo uso dei puntatori a funzione.
Il programma realizza ciascuna opzione con una diversa funzione e memorizza i puntatori a ciascuna funzione in un array di puntatori a funzione.

```c
#include <stdio.h>

// prototipi
void function1(int a);
void function2(int b);
void function3(int c);

int main(void)
{
    // inizializza un array di 3 puntatori a funzioni che ricevono
    // ognuna un argomento int e restituiscono void
    void (*f[3])(int) = {function1, function2, function3};

    printf("%s", "Inserisci un numero da 0 a 2, 3 per uscire: ");
    int choice = 0;
    scanf("%d", &choice);

    // elabora la scelta dell'utente
    while (choice >= 0 && choice < 3)
    {
        // invoca la funzione alla locazione choice nell'array f e passa
        // choice come argomento
        (*f[choice])(choice);

        printf("%s", "Inserisci un numero da 0 a 2, 3 per uscire: ");
        scanf("%d", &choice);
    }
}
```

Questo è possibile solo se tutte le funzioni hanno lo stesso tipo di parametri. Non è possibile mescolare funzioni con un parametro e funzioni con più parametri nello stesso array.

Uso degli argomenti dalla riga di comando

Uno dei dubbi comuni è perché si debba sempre definire il main e come si passino argomenti a questa.
Nella maggior parte dei casi, la funzione main non riceve parametri e sarà sempre void, ma ci sono dei rarissimi casi in cui si richiede il passaggio di argomenti nel main.
Gli argomenti nel main si passano solo tramite **riga di comando**, il primo argomento stesso sarà il **nome del file da passare al programma**. Per poter passare argomenti da riga di comando al main, quest'ultimo deve includere i parametri `int argc` e `char *argv[]`.
Questi parametri devono essere **obbligatoriamente** chiamati con questo nome e tipo.

- **argc**: riceve il numero di argomenti della riga di comando che l'utente ha inserito
- **argv**: è un array di stringhe in cui sono memorizzati gli argomenti della riga di comando
  Supponendo di avere un programma che copi file in un altro file un carattere alla volta, avendo come nome del file eseguibile *mycopy*, per eseguirlo, dovremmo scrivere nella riga di comando: `mycopy nomefile_in nomefile_out`, in questo caso *argc* avrà valore 3 (dati 3 nomi come argomenti) e quindi l'array *argv* conterà queste 3 stringhe.

Liste di argomenti a lungehzza variabile

Possiamo dichiarare una funzione che possa prendere in input un numero **indefinito di argomenti**. L'esempio lampante di questo tipo di funzione è proprio la `printf()`, questa tecnica ha dei limiti non può essere sempre usata, la sua dichiarazione, continuando ad usare l'esempio tipico della *printf* stessa, è la seguente: `int printf(const char *format, ...)`
Il simbolo dei . . . è definito come **ellissi**, quest'ultimo nel prototipo di una funzione indica che la funzione riceve un numero variabile di argomenti di qualsiasi tipo, inoltre l'ellissi deve essere **l'ultimo parametro** della dichiarazione della funzione, porla al centro di una lista dei parametri è un grave **errore** di sintassi.
Per sfruttare questa lista a lunghezza variabile bisogna inserire una nuova libreria, `<stdarg.h>`:

Identificatore	Spiegazione
`va_list`	Per accedere agli argomenti in una lista di argomenti di lunghezza variabile, deve essere definito una variabile di tipo **va_list**. Questo tipo contiene le informazioni che servono alle macro **va_start**, **va_arg** e **va_end**.
va_start	Una macro che va invocata prima che si possa accedere agli argomenti di una lista di argomenti di lunghezza variabile. La macro inizializza la variabile dichiarata con **va_list**.
va_arg	Una macro che permette l'accesso al valore successivo di **va_list**. Il primo argomento della macro è una varibile **va_list**, il secondo argomento e il tipo di valore che deve essere restituito.
va_end	Una macro da chiamare prima di uscire dalla funzione per ripulire **va_list**.

Vedendo le istruzioni di questo nuovo tipo **va_list**, notiamo che l'unica operazione possibile è ciclare per comprendere ogni argomento presente, ma questo arreca un nuovo problema, quando fermarci e quanti valori dovrà contenere. Infatti non si potrà mai conoscere a priori il numero preciso di argomenti.
Vediamo come risolvere questo problema, ipotizzando di creare una funzione che accetta un

numero indefinito di interi come variabile con output la media dei valori passati come argomento:

La funzione **average**...

```c
double average(int i, ...)
{
    double total = 0; // inizializza total
    va_list ap;       // memorizza gli argomenti

    va_start(ap, i); // inizializza l'oggetto di tipo va_list

    // elabora la lista di argomenti di lunghezza variabile
    for (int j = 1; j <= i; ++j)
    {
        total += va_arg(ap, double);
    }

    va_end(ap);       // pulitura della lista di argomenti di lunghezza variabile
    return total / i; // media calcolata
}
```

E' importante **rispettare i limiti della lista**, si può solo procedere in avanti nella lista e bisogna conoscere in anticipo il tipo degli argomenti trattati, non si può andare in qualsiasi zona della lista con tipi differenti. Non bisogna confondere questa lista come un array, perché questa soluzione sarà adottata in linguaggi diversi dal C.

Esempio dell'uso della funzione average:

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

double average(int i, ...); // ... rappresenta argomenti variabili

int main(void) {
    double w = 37.5;
    double x = 22.5;
    double y = 1.7;
    double z = 10.2;

    printf("%s%.1f; %s%.1f; %s%.1f; %s%.1f\n\n",
           "w = ", w, "x = ", x, "y = ", y, "z = ", z);
    printf("%s%.3f\n%s%.3f\n%s%.3f\n",
           "The average of w and x is ", average(2, w, x),
           "The average of w, x, and y is ", average(3, w, x, y),
           "The average of w, x, y, and z is ", average(4, w, x, y, z));
}
```