

ANSI C – Elementi di Base

Bentornati. Finalmente in questo tutorial inizieremo a mettere mano sul codice, cercando di condire il tutto con un pochino di teoria, non spaventatevi i concetti saranno molto semplici anche per chi è alle prime armi, come ho già detto lo scopo non è quello di creare il perfetto programmatore, ma di dare a tutti le basi per iniziare a programmare in ANSI C, questa serie di tutorial non è un punto di arrivo, bensì un punto di partenza per iniziare uno studio approfondito e sistematico dell'argomento.

Una piccola nota, lungo la stesura del tutorial saranno riportati esempi di codice o parti di esso, il codice, o le parole chiave ad esso appartenenti, inserite all'interno del testo, saranno scritte per convenzione in `courier`, in modo da semplificarne la lettura.

CIAO MONDO!!!

Partiamo dal classico esempio per cercare di vedere come è strutturato un programma in C, quindi vogliamo scrivere del codice che ci permette di stampare a video la scritta “Hello World”, per far questo ci occorre un editor di testo o ancora meglio una IDE che ci consenta di sviluppare del codice, ovviamente ci serve anche il nostro compilatore (Vi rimando al tutorial dedicato al set-up dell'ambiente di sviluppo).

```
#include<stdio.h> // Riga 1

main() // Riga 2
{ // Riga 3
    printf("Hello World"); // Riga 4
} // Riga 5
```

Questo codice sarà poi salvato all'interno di un file che potremo chiamare HelloWorld.c e compilato per avere un eseguibile.

Iniziamo a guardarci intorno chiedendoci che cosa abbiamo scritto, innanzitutto abbiamo cinque righe di codice, come si può notare il linguaggio è molto simile a quello umano, per chi conosce un minimo della lingua inglese potrà capire il significato di alcune parole, questo perchè come detto nell'introduzione, il linguaggio ANSI C è un linguaggio di alto livello che consente all'utente di scrivere codice più vicino al suo modo di esprimersi che non a quello della macchina (ovvero una serie di impulsi elettrici).

Prendiamo in esame tutte le componenti di questo programma, sottolineo che quello che abbiamo scritto sopra è un programma funzionante a tutti gli effetti, una volta compilato e mandato in esecuzione stamperà la scritta “Hello World” sulla vostra console. Innanzitutto esaminiamo la prima riga di codice:

```
#include<stdio.h> // Riga 1
```

La riga inizia con `#include`, è seguita da una cosa tra parentesi angolate `<stdio.h>` per poi concludersi con `// Riga 1`.

`#include` è una qualche cosa che ci permette di inserire all'interno del nostro codice dell'altro codice. In pratica si tratta di una direttiva del preprocessore che dice all'ambiente di prendere il file inserito tra le parentesi, nel nostro caso `stdio.h` e di inserirlo all'inizio del nostro codice, ovviamente noi tutto questo non lo vediamo, per noi il nostro sorgente è formato da solamente quelle cinque righe, ma in realtà è formato da migliaia di righe che ci siamo presi e portati dentro tramite la direttiva `#include`. Ora vi starete domandando, ma perchè dobbiamo fare questo, che cosa ci serve portarci dietro migliaia di righe di codice? Ebbene la risposta è abbastanza semplice ma necessita dell'inserimento di un nuovo concetto chiave, ovvero quello di libreria. Che cosa è una libreria? Non è altro che un insieme di codice scritto da altri (ma ovviamente possiamo anche noi scrivere delle nostre librerie) che viene preso, inserito all'interno del nostro sorgente e utilizzato da noi.

Supponiamo di voler costruire un programma che esegue delle operazioni aritmetiche anche complesse, teoricamente siamo costretti a scrivere centinaia di righe di codice per definire tutte le singole operazioni, supponete che invece di riscrivere tutto vi venga dato un insieme di codice scritto da altri che vi consenta di fare qualsiasi operazione aritmetica complessa senza che voi dobbiate preoccuparvene e che questo codice sia inserito in un file chiamato `Math.h`, a questo punto voi volete usarlo, ma come potete fare? Semplicemente vi basta inserire tutto questo codice all'interno del vostro programma e poi utilizzarlo. Le strade sono due, o prendete `Math.h`, fate copia di tutto, poi andate nel vostro sorgente e fate incolla, oppure lo inserite semplicemente dicendo `includimi Math.h` dentro il mio codice: `#include<Math.h>`.

Allora la domanda che viene in questo momento è: ok, allora cosa c'è dentro `stdio.h`? Anche qui la risposta è abbastanza semplice, esaminando il nome del file è già possibile capire di che cosa si tratta, se prima `Math.h` stava per operazioni matematiche, ora abbiamo `stdio` che può essere vista come una composizione delle parole `std`(standard) + `io`(I/O input-output), in pratica si tratta della libreria standard fornita da ANSI C per la gestione degli ingressi e delle uscite. Dato che noi vogliamo scrivere "Hello World", abbiamo bisogno di andare a scrivere nel dispositivo di uscita "monitor" la scritta, quindi un dispositivo di output, anche in questo caso o ci andiamo a ridefinire tutte le funzioni per utilizzare un monitor, e vi assicuro che non è una cosa così banale, oppure più semplicemente includiamo all'interno del nostro codice una libreria scritta da altri che ci consente di scrivere sul monitor senza sporcarci noi stessi le mani con codice complesso.

Completiamo la riga di codice numero uno spiegando che cosa significa `// Riga 1`, in pratica all'interno del nostro codice possiamo (e vi consiglio vivamente di farlo) mettere dei commenti che vengono ignorati totalmente dal compilatore, ma che sono utili a noi per capire cosa stiamo facendo. Esistono due modi per definire dei commenti, il primo è quello visto nell'esempio, ci basta scrivere `//` e tutto quello che **starà a destra sulla medesima riga** sarà considerato un commento. E' importante sottolineare che è tutto quello che si trova a destra, quindi quello a sinistra è codice sorgente che verrà compilato, e notare anche sulla medesima riga, significa che come vado a capo, il commento non esiste più e quello che scrivo diventa codice valido. Ad esempio:

OK!!!

```
// questo è un commento valido
```

NO!!!

```
// questo è un commento valido fino qui  
ma qui non è più valido
```

Se vogliamo scrivere un commento che va su più linee siamo costretti ad utilizzare un'altra sintassi decisamente molto comoda per commenti lunghi, in pratica ci basterà scrivere il commento all'interno di `/* ... */` ecco un esempio.

```
/* Questo qui è un commento valido
   che va tranquillamente su più righe
   senza problemi di nessun tipo */
```

Ripeto, non abbiate timore di scrivere commenti e commentate il più possibile le righe di codice che non vi sono chiare, anche perchè del codice ben commentato è leggibile sia per voi, sia per chi dovrà utilizzarlo in futuro. Vi assicuro che risulta molto difficile mettere mano su un codice di migliaia di righe scritto mesi prima senza l'aiuto dei commenti che avete messo.

Passiamo ora a prendere in esame la riga due del nostro programma, ovvero:

```
main() // Riga 2
```

Come vedete ritroviamo un altro commento che ci ricorda che stiamo esaminando la seconda riga, da questo momento in poi eviterò di focalizzare ulteriormente l'attenzione sui commenti e continuerò con il resto del codice.

A questo punto troviamo `main()`, di che cosa si tratta? Ebbene il `main` è la funzione fondamentale di tutti i programmi in C (nei prossimi capitoli spiegheremo in dettaglio che cosa sono le funzioni, per ora è superfluo), ogni programma scritto in ANSI C ha bisogno di avere al suo interno questa funzione, altrimenti non può in alcun modo funzionare. Non possono essere definite più funzioni chiamate `main`, in quanto il programma deve aver inizio solo ed esclusivamente in un punto. A questa funzione possono essere passati dei parametri per ricevere degli input, oppure omessi come in questo caso, infatti, come vedete tra le due parentesi tonde affianco al nome non vi è scritto nulla (nel capitolo sulle funzioni spiegheremo che cosa sono i parametri, come possono essere passati e sfruttati).

Dopo aver scritto il nome della funzione, ovvero `main()`, bisogna dire che cosa è contenuto al suo interno, ovvero che cosa deve fare il programma appena viene lanciato. Quindi vogliamo scrivere del codice dentro la funzione, ma per far questo è necessario capire dove questa comincia e dove questa finisce. Per far questo ci vengono incontro le righe 3 e 5:

```
{ // Riga 3
} // Riga 5
```

Come si può notare si tratta solo di due parentesi graffe, ebbene nel linguaggio C tutto quello che è contenuto all'interno di una coppia di parentesi graffe è un blocco di istruzioni che vengono eseguite in maniera sequenziale da quella più in alto a quella più in basso. In pratica la riga 3 ci dice che in quel punto inizia il blocco che nel nostro caso è composto dalla sola riga 4, mentre la riga 5 ci dice che il blocco viene chiuso in quel momento, quindi tutto quello che è contenuto tra la riga 3 e la 5 comprese fa parte della funzione `main()`.

Non ci resta che spiegare il significato della riga 4:

```
printf("Hello World"); // Riga 4
```

Anche qui possiamo notare come la lingua inglese ci venga incontro, `printf(...)`; ebbene `print` ci ricorda molto le stampanti, significa che noi vogliamo “stampare” qualche cosa, come potete notare abbiamo un nome `printf` e subito dopo la parentesi tonda aperta, la scritta `Hello World` messa tra apici e poi la parentesi chiusa, se notate la sintassi è molto simile a `main()`, anche lì avevamo nome e due parentesi, in effetti si tratta anche qui di una funzione, ma a differenza di prima dove andavamo a definire come era fatta la funzione principale del nostro programma, qui invece andiamo ad utilizzare una funzione che è stata definita da altri, voi vi chiederete dove è stata definita. Se ci pensate e rileggete quello scritto in precedenza abbiamo detto che includevamo una libreria standard per la gestione degli input/output, in questa libreria è definito il funzionamento di `printf()`, se non avessimo incluso la `stdio.h` non avremo mai potuto usare quella funzione.

Come si definiscono e usano le funzioni verrà spiegato successivamente per ora basti capire che includo la libreria, prendo la funzione per scrivere sullo schermo, gli dico di scrivermi “Hello World” e poi chiudo la istruzione con “;”

Che cosa rappresenta “;”? Il punto-e-virgola serve per far capire al compilatore che in quel momento desideriamo chiudere l'istruzione, in pratica tutto quello che verrà dopo farà parte di un'altra istruzione. E' importante prestare attenzione a mettere bene i ; altrimenti si corre il rischio di avere degli errori quando andiamo a compilare il codice.

Anche per questo tutorial è tutto, spero di essere stato abbastanza chiaro, in caso aspetto critiche e suggerimenti. Nel prossimo tutorial metteremo ancora più carne al fuoco, toccando un paio di concetti importanti della programmazione, ovvero le variabili e le costanti. A presto.

Manuel (aka Jekrom)

per segnalazioni : manuel.montini@gmail.com

<http://iphonecodes.wordpress.com/>

LICENZA:

Questo tutorial è rilasciato sotto licenza Creative Commons : “Attribuzione-Non commerciale-Non opere derivate 2.5 Italia”,

<http://creativecommons.org/licenses/by-nc-nd/2.5/it/>