Guida alla programmazione (volume 1)

evilsocket@gmail.com
http://www.evilsocket.net/

La volta scorsa abbiamo preso in considerazione alcuni aspetti teorici che purtroppo, vengono troppo spesso messi da parte dagli altri tutorial per dare più spazio alla parte "pratica" .
Quest'oggi invece, inizieremo a vedere come scrivere un semplice programma in C che, a mio parere, è il miglior linguaggio con il quale iniziare e soprattutto è quello che meglio si presta agli scopi di una didattica generica di questa guida .
Iniziamo con il materiale del quale avrete bisogno per "far girare" questi esempi .
Se siete sotto Winzozz (cambiate OS :D) :
1) Un editor di testo decente con l'evidenziazione della sintassi, come il Notepad++ (http://notepadplus.sourceforge.net/) .
2) Un compilatore OpenSource con relative binutils (www.mingw.org) .
3) Un cervello, tanta pazienza e soprattutto tanta curiosità e voglia di imparare ^^ .
Se siete sotto Linux (grandi !!! ^^) :
1) Sempre un editor di testo, io di solito uso Kate perchè è già integrato nel KDE, ma voi scegliete quello che volete ^^ .
2) Il mitico GCC (il compilatore per chi non lo sapesse), che dovrebbe essere installato in tutte le distro decenti .
3) VEDERE IL PUNTO 3 DEGLI UTENTI WINZOZZ :D

Ok, installato tutto il necessario, siamo pronti ad iniziare .
Come avevamo detto nella precedente parte di questa guida, "un programma è un insieme di dati ed istruzioni che hanno lo scopo di assolvere uno o più compiti" ... prendiamo in esame i DATI ...
Come potete ben immaginare, la quantità di dati con la quale un programma deve interagire è enorme ... per esempio, tutte le scritte
che vedete all'interno di un programma, anche questo stesso testo che state leggendo tramite il vostro browser, sono i dati di un programma .
Per capire bene come i dati di un programma vengono gestiti dal computer, vi mostro qui di seguito quali sono i passi nell esecuzione
di un programma :

1) L'utente richiama il programma (con un doppio click o da console, meglio specificare tutto :D) .
2) Il sistema operativo trova il percorso completo del modulo eseguibile chiamato .
3) Sempre il SO crea nella vostra RAM, un blocco di memoria dove metterà il codice compilato del programma e i suoi dati statici .
4) Il SO "mappa" il programma in questo blocco di memoria e salta (esegue un "jump") all indirizzo iniziale del codice .

Nel punto tre, ho parlato di "memoria statica" e già immagino la

faccia di alcuni di voi mentre leggevano questi termini arcaici :D .
Tranquilli ragazzi, vi pare che scrivo qualcosa e non ve la spiego ?!?!? ^^
Allora, a questo punto, mi sembra abbastanza chiaro che i dati di un programma vengono messi nella RAM, ma non è così semplice in realtà, perchè il blocchi di memoria usati da un programma, si differenziano in due categorie principali :

1) Memoria statica .
2) Memoria dinamica .

La memoria statica, sono tutti quei dati di dimensione già nota a priori, che sono già inseriti all interno del programma .
Pensate per esempio a tutte le scritte dei menu di un programma ... il programma già è a conoscenza di questi dati e di quanto occuperanno i memoria ... cavolo è robba sua !!! :D
In questo caso, tutto il contenuto di questi dati viene caricato in memoria insieme al programma stesso al momento dell suo avvio .
A differenza di quella statica, la memoria dinamica è un po più particolare e difficile da spiegare (nemmeno tanto ^^) .
Vediamo un po con uno dei miei fantastici esempi :D .
Immaginate questa situazione ... siete a casa vostra e avete invitato un vostro amico a cena ... per fare i bravi padroni di casa
sistemate la tavola e le sedie prima che lui venga, mettendo la sedia per voi, vostra madre, vostro padre ed il vostro amico .
Quindi, prima del suo arrivo, già avete 4 sedie perchè sapete a priori quanti sarete e di quanto spazio avrete bisogno .
Beh, questo è il concetto della memoria statica, ora passiamo alla dinamica con lo stesso tipo di esempio ...
Chiamate sempre il vostro amico e lui vi dice "Ciao, senti, viene anche qualche altro amico mio così ci divertiamo" e riattacca il telefono .
Il problema è che non sapete quanti amici inviterà, quindi non sapete in anticipo di quante sedie avrete bisogno ... l'unico modo che avete per risolvere il sistema è aspettare che arrivino gli invitati, contarli e mettere al volo le sedie necessarie .
Questa invece è la memoria dinamica !!! :D
Non sapendo a priori quanto spazio in memoria sarà necessario per una determinata serie di dati, il programma lo creerà (in gergo si dice "allocherà") a tempo debito dopo il suo avvio .
Ora introduciamo il concetto di "variabile" ... come abbiamo detto, i dati di un programma si trovano in RAM ... questo significa
che ad ogni dato corrisponderà un determinato indirizzo di memoria .
Ma il programmatore non sa a priori in quale indirizzo verranno caricati i suoi dati perchè è il sistema operativo a sceglierlo in base alla memoria libera e altri fattori analoghi .
Quindi, come si fa ad accedere a queste informazioni senza sapere dove si trovano ?
Proprio per ovviare a questo problema, ad ogni dato si da un nome

... questo nome, chiamato VARIABILE, sarà l'identificativo del dato associato, quindi per accedervi, si userà questo nome e sarà il sistema operativo a preoccuparsi di trovarne il rispettivo indirizzo in memoria .
In termini tecnici, questo procedimento, si chiama DICHIARAZIONE DI UNA VARIABILE, vediamone un piccolo esempio in C

```
int numero;
numero = 1234;
```

Ora vi spiego il significato delle varie porzioni di questo piccolo ritaglio di codice .
int è il TIPO di variabile che stiamo dichiarando, in questo caso un numero intero ... essendo questa una variabile allocata staticamente (ve lo ricordate cosa significa no ?), è necessario specificarne il tipo per far sapere al SO quanta memoria deve riservare per quel determinato dato .
Nel nostro caso, un numero intero (quindi dichiarato come "int") occupa in memoria 4 bytes (quindi 32 bit, 8 bit a byte per i 4 byte del tipo) e può contenere i numeri da 2.147.483.648
a
+2.147.483.647 .

Gli altri tipi base del C sono :

short int	*2 byte (16 bit)*	*da 32.768 a +32.767*
unsigned short int	*2 byte (16 bit)*	*da 0 a +65.535*
unsigned int	*4 byte (32 bit)*	*da 0 a +4.294.967.295*
long int	*4 byte (32 bit)*	*da 2.147.483.648 a +2.147.483.647*
signed char	*1 byte (8 bit)*	*da 128 a +127*
unsigned char	*1 byte (8 bit)*	*da 0 a +255*
float	*4 byte (32 bit)*	*da 3.4e38 a 3.4e38 (7 decimali)*
double	*8 byte (64 bit)*	*da 1.7e308 a 1.7e308 (15 decimali)*
long double	*12 byte (96 bit)*	*(il range varia molto in base al sistema operativo)*
bool	*1 byte (8 bit)*	*può essere "true" o "false"*
void		*è un tipo speciale associato a nessun tipo, vedremo in seguito come si usa*

numero è il nome della variabile, ovvero il nome associato al suo indirizzo in memoria .
numero = 1234 in questo modo scriviamo nell indirizzo in memoria associato alla variabile "numero"
il numero 1234 .
Semplice no ?
Ok, anche se molti di voi rimarrano delusi, per questa parte è tutto ... non perchè io sono pigro e non mi va di scrivere, ma perchè preferisco che voi passiate un po di tempo ad assimilare

bene questi concetti in modo che siate in grado di capire veramente il resto delle altre lezioni .
La prossima volta (se tutto va bene ^^) inizieremo a vedere le "funzioni", come farle interagire con questi dati e inizierò a spiegare il concetto di vettore tanto per completare la spiegazione sulla memoria dinamica .
E' tutto, Ciaoz !

evilsocket