

Introduzione : Le basi

Gli Strumenti	1
vim : L' editor di testo.....	1
gcc: Il Compilatore.....	2
gdb : Il debugger.....	2
Ctags Surfing into the code.....	3
I Makefile make it easy!.....	4
Secondo programma.....	4
Costrutti if e cicli.....	5
if : il se.....	5
Cicli : while, for e do while.....	5
I puntatori	6

Gli Strumenti

I requisiti per un buon programma in c non sono molti, sono necessari: un editor,un compilatore ed un debugger. Volendo poi si possono aggiungere

vim : L' editor di testo

Iniziamo con lo scrivere un piccolo programma e poi andando ad analizzare ogni passaggio

*Per prima cosa apriamo un terminale e usando **vim** scriviamo:*

1. vim hello.c

<p>Apriamo un file vuoto e lo chiamiamo hello.c</p>	
---	--

2. E scriviamo questo testo qui:

<pre> /***** hello.c *****/ #include<stdio.h> main(){ printf("Hello World!"); } </pre>	
--	--

3. Usciamo e salviamo :wq (oppure: :x , ZZ)

Questo è il primo programma fatto in c.

Spiego:

- **#include<stdio.h>** : include la libreria standard di input output, contenente le funzioni base, come **printf()** utili per poter stampare sullo schermo **Hello World !**
- **main()** : è la funzione principale, dentro le parentesi graffe c'è il nostro programma .
- **printf()** : serve appunto a stampare il testo contenuto dentro gli apici .

gcc: Il Compilatore

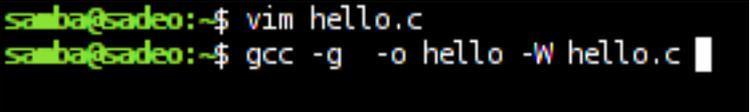
è un compilatore, ovvero quello che trasforma il nostro codice in un eseguibile.

gcc è l'abbreviazione di GNU c compilatore ed è stato scritto da Richard Stallman, per maggiori informazioni www.gnu.org .

*Se gcc non restituisce nessun tipo di **output** significa che è andato tutto bene, altrimenti possono esserci diversi tipi di **errori**, in quel caso, bisogna leggere bene il tipo di errore e la riga relativa; spesso gli errori riscontrati possono essere la mancanza di un ; oppure una parentesi mancante, a volte invece può essere qualcosa di più serio.*

I comandi e le opzioni che servono per compilare il codice sono i seguenti :

gcc -g -o hello -W hello.c



```
samba@sadeo:~$ vim hello.c
samba@sadeo:~$ gcc -g -o hello -W hello.c
```

-g: imposta i flag necessari per il debugging

-o: salva il file eseguibile con il nome specificato

-W: controlla i warning più comuni, per ottimizzare meglio il codice.

hello.c : Il nostro programma che abbiamo scritto e salvato con estensione .c

gdb : Il debugger

*Un debugger è un compagno di navigazione, per navigare nel codice che abbiamo scritto e capirne i significati più interni. Un **debugger** infatti non è altro che uno strumento per analizzare il nostro codice passo per passo per controllare il valore di una variabile o i passaggi più complessi.*

*Esistono tipi di debugger anche grafici come: **xgdb** oppure **ddd**, potete provarli, io sinceramente mi trovo bene così.*

gdb deve essere chiamato **dopo** una compilazione in questo modo:

gdb ./hello

```
samba@sadeo:~$ vim hello.c
samba@sadeo:~$ gcc -g -o hello -W hello.c
samba@sadeo:~$ gdb ./hello
GNU gdb 6.6-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) █
```

Ok, adesso impariamo ad usare gdb, con 4 piccoli comandi:

- **list <number>** - l
 - *stampail codice del nostro programma, senza argomenti stampa le prime 10 righe.*
- **break <n.riga>**: b 3
 - *imposta uno stop , un break nel codice, necessario se si vuole usare gdb e vedere cosa succede in quella riga.*
- **run <args>**: r
 - *parte il programma e si ferma al primo breakpoint trovato. Accetta eventuali argomenti*
- **step <n.passi>** - s
 - *va avanti di un istruzione o **n.passi** nel codice.*
- **refresh**
 - *un piccolo aiuto da parte di gdb per controllare il codice ad ogni aggiornamento, vedremo la riga selezionata ed il codice in alto, mentre in basso possiamo digitare i nostri comandi.*
- **print <variabile>**: p var
 - *serve a stampare il contenuto della variabile in quel preciso istante.*
- **watch <variabile>** : watch var
 - *controlla e visualizza il contenuto di una variabile ad ogni cambiamento*
- **display <variabile>** : display var
 - *visualizza ad ogni passo il contenuto della variabile (differente da watch).*

```
hello.c
#include<stdio.h>
main(){
printf("Hello World!");
}
5
6
7
8
9
10
11
12
13
14
15
child process 21616 In: main Line: 5 PC: 0x8048365
(gdb) █
```

Ctags Surfing into the code

Ctags è un programma che serve per navigare in un codice molto vasto. Per attivarlo basta scrivere in un terminale **ctags <programma>.c** per creare un file chiamato **tags** che conterrà al suo interno degli indici relativi al vostro codice.

Poi utilizzando vim potete navigare nel codice utilizzando la combinazione di tasti:

- **Control +]** - per andare a vedere dove si trova una particolare funzione

- **Control + T** per tornare all'ultima funzione controllata.

I Makefile make it easy!

I Makefile, non sono altro che dei file contenenti delle righe che indicano come compilare il nostro codice, si creano nella cartella corrente e vengono impostati con un etichetta detta **label**.

```

1 target:
2     ctags hello.c
3     gcc -g -o hello -W hello.c
4 debug:
5     gcc -g -o hello -W hello.c
6     gdb ./hello

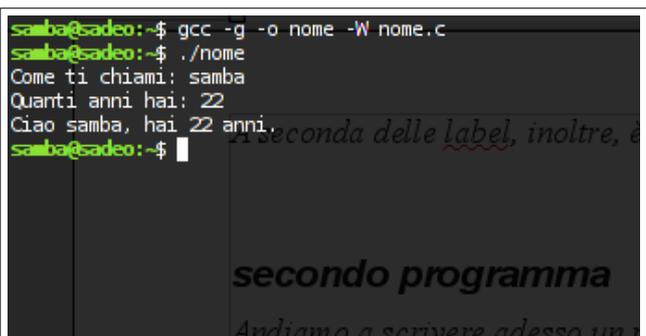
```

Vim supporta i Makefile utilizzando il comando **:make** e ne aiuta la scrittura del codice, andando a compilare e selezionare la riga errata nel codice.

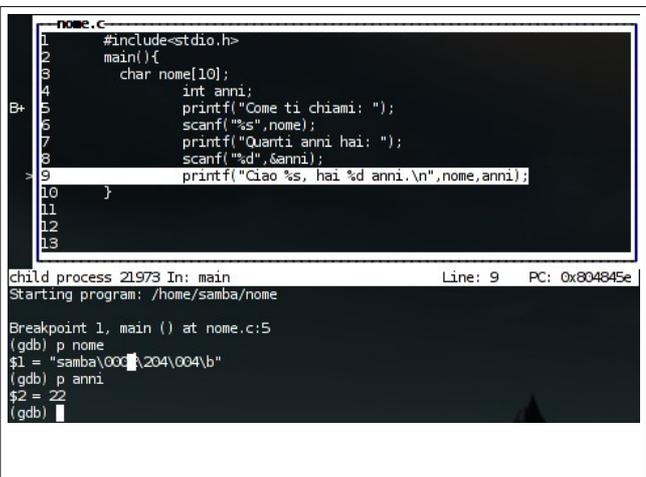
A seconda delle label, inoltre, è possibile utilizzare qualsiasi comando, quindi anche **gdb** e **ctags**.

Secondo programma

Andiamo a scrivere adesso un programma che chiede nome ed età, inserite dall'utente, e ne stampa il valore:

<pre> /***** nome.c *****/ #include<stdio.h> main(){ char nome[10]; int anni; printf("Come ti chiami: "); scanf("%s", nome); printf("Quanti anni hai: "); scanf("%d", &anni); printf("Ciao %s, hai %d anni.\n", nome, anni); } </pre>	
--	---

analizziamone il codice con gdb

<ol style="list-style-type: none"> 1. gdb ./nome 2. list 3. break 5 4. run 5. refresh 6. s 7. p nome 8. s 9. p anni 10. quit 	
--	--

Costrutti if e cicli

Un programma è un processo logico in cui si verificano delle situazioni e noi in base ad una logica costruita riusciamo a decidere cosa fare se si verifica una situazione o un'altra.

if : il se

I costrutti **if** servono esattamente a questo, ovvero a controllare se la condizione che stiamo analizzando risulta vera o falsa .

L'utilizzo è molto semplice :

```
#include<stdio.h>
main(){
    int i;
    printf("inserisci un numero: ");
    scanf("%d", &i);
    if (i > 0){
        printf("Hai inserito un numero positivo");
        printf("\n fine. \n");
    }
}
```

Inserisci un numero: -1
fine.

Inserisci un numero: 4
Hai inserito un numero positivo
fine.

Come vedete se inserisco un **numero negativo** (-1) il programma termina immediatamente poiché ha controllato che il numero inserito non è maggiore di zero, quindi è negativo.

La seconda volta invece inserendo un **numero positivo** (4) appare come output anche una riga che ci informa di aver inserito un numero positivo, poiché la condizione si è verificata e noi siamo entrati dentro il blocco **if** .

Cicli : while, for e do while

Se avete capito il blocco if, allora basta pensare che i cicli non sono altro che dei **blocchi if ripetuti**. Basta pensare che fino a quando una certa condizione non è verificata non si va avanti.

while

```
#include<stdio.h>
main(){
    int i,n;
    i = 0;
    printf("Quanti giri: ");
    scanf("%d",&n);
    while (i < n){
        printf("%d",i++);
    }
    printf("\n fine. \n");
}
```

Quanti giri: 10
0 1 2 3 4 5 6 7 8 9
fine.
Quanti giri: 5
0 1 2 3 4
fine.

Nel costrutto while viene impostata una variabile contatore **i** che all'inizio è uguale a zero e successivamente dentro al ciclo viene incrementata (**i++**) di uno esattamente dopo ogni printf.

Se invece avessi scritto **++i** allora i sarebbe stata incrementata prima di eseguire la printf ed avrei ottenuto come prima variabile : **1** anziché **0**

Successivamente chiedendo **quanti giri** eseguire il ciclo inizia fino a quando la variabile **i** non supera la variabile **n**, scelta dall'utente.

for

<pre>#include<stdio.h> main(){ int i,n; printf("Quanti giri: "); scanf("%d",&n); for(i=0;i<n;i++){ printf("%d",i); } printf("\n fine. \n"); }</pre>	<p>Quanti giri: 10 0 1 2 3 4 5 6 7 8 9 fine. Quanti giri: 5 0 1 2 3 4 fine.</p>
--	---

L'unica differenza tra il while ed il for è la **compattezza**, il for infatti mette insieme **nella stessa riga** le 3 importantissime operazioni necessarie per un ciclo :

- *inizializzazione*
- *condizione*
- *incremento*

do while

<pre>#include<stdio.h> main(){ int n; do{ printf("Inserisci un numero: "); scanf("%d",&n); }while (n <= 0); printf(" Hai inserito: %d",n); printf("\n fine. \n"); }</pre>	<p>Inserisci un numero: -1 Inserisci un numero: -3 Inserisci un numero: 0 Inserisci un numero: 5 Hai inserito: 5 fine.</p>
--	--

Capito?

Il **do while** è un ciclo con la prima iterazione incondizionata, nel senso che il primo giro (quello dopo il do) avviene senza bisogno di superare una condizione, ma se la condizione richiesta dentro il while finale è vera ($n \leq 0$) si rimane dentro il ciclo e viene richiesto di inserire un numero fino a quando non si ottiene un numero maggiore di 0.

I puntatori

immaginate semplicemente che ogni cella di memoria abbia il suo indirizzo e che un puntatore non sia altro che una cosa tipo questa : **0x12345** che indica semplicemente dove si trova quella cella di memoria. Abbiamo già utilizzato i puntatori senza saperlo, quando chiamiamo una funzione scanf.

scanf(%d , &n) qui stiamo mandando alla funzione scanf l'indirizzo di **n** e non il valore che n contiene.

n	10
&n	0x12345

Variabili

- **Per Valore**
 - Per esempio la funzione printf ad esempio usa il valore delle variabili e non deve modificare il valore, quindi non è necessario specificarne l'indirizzo.
- **Per Riferimento:**
 - Una qualsiasi funzione che deve modificare le variabili ricevute e restituirle. La scanf

oppure una funzione creata da noi ad esempio la funzione scambia:

<pre>#include<stdio.h> void scambia(int *a, int *b); main(){ int a , b; printf("Inserisci 2 numeri: "); scanf("%d %d ",&a, &b); scambia(&a,&b); printf("Ho scambiato: %d %d",a,b); printf("\n fine.\n"); } void scambia(int *a, int *b){ int tmp; tmp = *a ; *a = *b; *b = tmp; }</pre>	<p>Inserisci 2 numeri: 4 9 Ho scambiato: 9 4</p> <p><i>consiglio di usare gdb per capire come funziona passo passo il programma.</i></p>
--	--

&a : specifica di prendere l'indirizzo della variabile *a*

quando chiamiamo una funzione e sappiamo che i dati devono essere modificati.

***a** : si usa nelle funzioni che ricevono un indirizzo.

nella funzione **scambio** usiamo `int *a` per specificare che riceviamo un indirizzo di una variabile di tipo intero. Attenzione: il tipo è importante, (**char *carattere** è diverso da **int *numero**).

Introduzione terminata, se siete arrivati fino a questo punto e avete capito quasi tutto, allora potete tranquillamente iniziare a scrivere del codice e vedere cosa succede.

Un linguaggio che non influenza il tuo modo di pensare circa la programmazione non vale la pena di essere imparato.