Compilatori GNU in Linux: gcc e g++

Un compilatore integrato C/C++

Per Linux e' disponibile un compilatore integrato C/C++: si tratta dei comandi GNU **gcc** e **g++,** rispettivamente.
In realta g++ e' uno script che chiama gcc con opzioni specifiche per riconoscere il C++.

Il progetto GNU

Il comando gcc, GNU Compiler Collection, fa parte del progetto GNU (web server www.gnu.org). Il progetto GNU fu lanciato nel 1984 da Richard Stallman con lo scopo di sviluppare un sistema operativo di tipo Unix che fosse completamente "free" software.

Cosa è GNU/Linux?
Gnu Non è Unix!
"*GNU, che sta per "Gnu's Not Unix" (Gnu Non è Unix), è il nome del sistema software completo e Unix-compatibile che sto scrivendo per distribuirlo liberamente a chiunque lo possa utilizzare. Molti altri volontari mi stanno aiutando. Abbiamo gran necessità di contributi in tempo, denaro, programmi e macchine.*"
[Richard Stallman, Dal manifesto GNU, http://www.gnu.org/gnu/manifesto.html

Quale versione di gcc sto usando?

Si può determinare la versione del compilatore invocando:

```
gcc -v
gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
```

I passi della compilazione
Sia gcc che g++ processano file di input attraverso uno o piu' dei seguenti passi:

1. preprocessing
2. rimozione dei commenti
3. interpretazioni di speciali direttive per il preprocessore denotate da "#" come:
   - #include include il contenuto di un determinato file, Es. #include <math.h>
   - #define definisce un nome simbolico o una variabile, Es. #define MAX_ARRAY_SIZE 100

3. compilation
   - traduzione del codice sorgente ricevuto dal preprocessore in codice assembly
4. assembly
   - creazione del codice oggetto
5. linking
   - combinazione delle funzioni definite in altri file sorgenti o definite in librerie con la funzione main() per creare il file eseguibile.

Estensioni

Alcuni suffissi di moduli implicati nel processo di compilazione:

.c modulo sorgente C; da preprocessare, compilare e assemblare
.cc modulo sorgente C++; da preprocessare, compilare e assemblare
.cpp modulo sorgente C++; da preprocessare, compilare e assemblare
.h modulo per il preprocessore; di solito non nominato nella riga di commando
.o modulo oggetto; da passare linker
.a sono librerie statiche
.so sono librerie dinamiche

Compilatori GNU in Linux: gcc e g++

L' input/output di gcc

gcc accetta in input ed effettua la compilazione di codice C o C++ in un solo colpo. Consideriamo il seguente codice sorgente C:

```
/* il codice C pippo.c */
#include <stdio.h>

int main()
{
 puts("ciao pippo!");
 return 0;
}
```

Per effettuare la compilazione

```
gcc pippo.c
```

In questo caso l'output di default è direttamente l'eseguibile **a.out**

Di solito si specifica il nome del file di output utilizzando l' opzione **-o** :

```
gcc -o prova pippo.c
```

L'eseguibile può essere lanciato usando semplicemente

```
./prova
```

ciao pippo!

Nota. Usare "**./**" puo' sembrare superfluo. In realta' si dimostra molto utile per evitare di lanciare involontariamente un programma omonimo, per esempio il comando "test"!

Consideriamo ora un codice sorgente C++ analogo:

```
// Il codice C++ pippo.cpp
#include<iostream>
using namespace std;

int main()
{
 cout<<"ciao pippo!"<<"\n";
 return 0;
}
```

Questa volta compiliamo usando

```
g++ -o prova pippo.cpp
```

Il valore restituito al sistema

Per verificare il valore restituito dal programma al sistema tramite l'istruzione di **return** usiamo

```
./prova
ciao pippo!

echo $?
0
```

Compilatori GNU in Linux: gcc e g++

Passaggi intermedi di compilazione

Per compilare senza effettuare il link usare

```
g++ -c pippo.cpp
```

In questo caso viene creato il file oggetto **pippo.o**. Per effettuare il link usiamo

```
g++ -o prova pippo.o
```

I messaggi del compilatore

Il compilatore invia spesso dei messaggi all'utente. Questi messaggi si possono classificare in due famiglie:

- messaggi di avvertimento (**warning messagges**)
- messaggi di errore (**error messagges**)

I messaggi di avvertimento indicano la presenza di parti di codice presumibilmente mal scritte o di problemi che potrebbero avvenire in seguito, durante l'esecuzione del programma. I messaggi di avvertimento non interrompono comunque la compilazione.
I messaggi di errore invece indicano qualcosa che deve essere necessariamente corretto e causano l'interruzione della compilazione.

Esempio di un codice C++ che genera un warning:

```
// example1.cpp
#include<iostream>
using namespace std;

float multi(int a, int b)
{
 return a*b;
};

int main()
{
 float a=2.5;
 int b=1;

 cout<<"a="<<a<<", b="<<b<<'\n';
 cout<<"a*b="<<multi(a,b)<<'\n';

 return 0;
}
```

In fase di compilazione apparirà il seguente warning:

```
example1.cpp: In function `int main ()':
example1.cpp:12: warning: passing `float' for argument passing 1 of
'multi (int, int)'
example1.cpp:12: warning: argument to `int' from `float'
```

Il messaggio ci avvisa che alla linea 12 del **main()** e' stato passato alla funzione **multi** un float invece che un int.

Esempio di un codice che genera un messaggio di errore:

```
// example1.cpp
#include<iostream>
using namespace std;

float multi(int a, int b)
{
 return a*b
};
```

```
int main()
{
 int a=2;
 int b=1;
 cout<<"a="<<a<<", b="<<b<<'\n';
 cout<<"a/b="<<multi(a,b)<<'\n';
 return 0;
}
```

Si noti che l' istruzione di return all'interno della funzione multi non termina con il punto e virgola (**;**). A causa di questo grave errore la compilazione non puo' essere portata a termine:

```
example1.cpp: In function `float multi (int, int)':
example1.cpp:5: parse error before `}'
example1.cpp:5: warning: no return statement in function returning
non-void
```

Controlliamo i livelli di warning

Per inibire tutti i messaggi di warinig usare l' opzione **-w**

```
g++ -w -o prova example1.cpp
```

Per usare il massimo livello di warning usare l' opzione **-Wall**

```
g++ -Wall -o prova example1.cpp
```

Compilare per effetture il debug

Se siete intenzionati ad effettuare il debug di un programma, utilizzate sempre l'opzione **-g**:

```
g++ -Wall -g -o pippo example1.cpp
```

L' opzione **-g** fa in modo che il programma eseguibile contenga informazioni supplementari che permettono al debugger di collegare le istruzioni in linguaggio macchina che si trovano nell'eseguibile alle righe del codice corrispondenti nei sorgenti C/C++.

Ottimizzazione

Il compilatore gcc consente di utilizzare diverse opzioni per ottenere un risultato più o meno ottimizzato. L'ottimizzazione richiede una potenza elaborativa maggiore, al crescere del livello di ottimizzazione richiesto.
L' opzione **-O*n*** ottimizza il codice, dove ***n*** é il livello di ottimizzazione. Il massimo livello di ottimizzazione allo stato attuale é il **3,** quello generalmente più usato è **2**.
Quando non si deve eseguire il debug é consigliato ottimizzare il codice.

Opzione	Descrizione
-O, -O1	Ottimizzazione minima
-O2	Ottimizzazione media
-O3	Ottimizzazione massima
-O0	Nessuna ottimizzazione

Esempio di un codice chiaramente inefficiente

```
int main()
{
 int a=10;
 int b=1;
 int c;
 for (int i=0; i<1e9; i++)
 {
```

```
  c=i+a*b-a/b;
 }
return 0;
}
```

Confronto dei tempi di esecuzione in funzione di livelli di ottimizzazione crescente

Livello	Tempo di esecuzione (secondi)
O0	32.02.00
O1	05.04.00
O2	05.02.00
O3	05.02.00

Compilazione di un programma modulare

Un programma modulare e' un programma spezzettato in componenti più' piccole con funzioni specifiche. La programmazione modulare è più facile da comprendere e da correggere.

Nel seguito abbiamo un programma C++ composto da tre moduli: **main.cpp**, **myfunc.cpp** e **myfunc.h** .

```
// main.cpp
#include<iostream>
#include"myfunc.h"
using namespace std;

int main()
{
 int a=6;
 int b=3;

 cout<<"a="<<a<<", b="<<b<<'\n';
 cout<<"a/b="<<div(a,b)<<'\n';
 cout<<"a*b="<<mul(a,b)<<'\n';

 return 0;
}

// myfunc.h
int div(int a, int b);
int mul(int a, int b);

// myfunc.cpp
int div(int a, int b) {
 return a/b;
};

int mul(int a, int b) {
 return a*b;
};
```

Per compilare usiamo

```
g++ -Wall -g -o prova main.cpp myfunc.cpp
```

Si noti che il file **myfunc.h** non appare nella riga di comando, verra' incluso dal gcc in fase di precompilazione.

Inclusione di librerie in fase di compilazione

L'opzione -**lnome_libreria** compila utilizzando la libreria indicata, tenendo presente che, per questo, verrà cercato un file che inizia per **lib**, continua con il nome indicato e termina con **.a** oppure **.so**.
Modifichiamo i moduli myfunc.h e myfunc.cpp aggiungendo la funzione **pot**:

```
// myfunc.h
int div(int a, int b);
int mul(int a, int b);
float pot(float a, float b);

// myfunc.cpp
int div(int a, int b)
{
 return a/b;
};

int mul(int a, int b)
{
 return a*b;
};

float pot(float a, float b)
{
 return pow(a,b);
}
```

La compilazione però si interrompe

g++ -Wall -g -o prova main.cpp myfunc.cpp

```
myfunc.cpp: In function `float pot (float, float)':
myfunc.cpp:11: `pow' undeclared (first use this function)
myfunc.cpp:11: (Each undeclared identifier is reported only once for
each function it appears in.)
```

La funzione **pow** e' contenuta nella libreria matematica **math,** dobbiamo allora aggiungere l'istruzione include nel modulo :

```
// myfunc.cpp
#include<math.h>
using namespace std;

int div(int a, int b) {
 return a/b;
};

int mul(int a, int b) {
 return a*b;
};

float pot(float a, float b) {
 return pow(a,b);
}
```

e compilare con un link alla libreria **libm.so**

g++ -Wall -g -o prova main.cpp myfunc.cpp -lm

Di default il compilatore esegue la ricerca della libreria nel direttorio standard **/usr/lib/**.
Tramite l' opzione **-L/nome_dir**, è possibile aggiunge la directory **/nome_dir** alla lista di direttori in cui gcc cerca le librerie in fase di linking.