

# Laboratorio di Informatica Generale I UD

## Seconda esercitazione

Danilo Severina

2 marzo 2006

### **Il Compilatore**

Il linguaggio C ha le seguenti caratteristiche:

- è molto efficiente, cioè permette di scrivere programmi veloci e compatti;
- consente di programmare ad alto livello, pensando ai problemi logici e non alla macchina fisica, ma permette anche di programmare a basso livello: offre quindi i vantaggi di entrambe le categorie;
- è un linguaggio apparentemente povero: non possiede istruzioni di entrata/uscita, nè istruzioni per operazioni matematiche; ogni funzione diversa dai costrutti di controllo o dalle operazioni elementari sui tipi dati è affidata ad un insieme di librerie esterne (in questo modo, mantenendo il linguaggio compatto, è possibile estendere le funzionalità o modificare quelle esistenti semplicemente aggiungendo o modificando librerie);
- è portabile: i programmi scritti secondo le regole dello standard ANSI possono essere compilati ed eseguiti su ogni tipo di calcolatore, sono cioè portabili. Naturalmente, un programma che usa caratteristiche particolari di un certo computer non è portatile.

Un programma C non può essere eseguito immediatamente. Deve prima essere tradotto in codice macchina, cioè compilato in una forma eseguibile: il testo scritto dal programmatore è detto “programma sorgente” o “codice sorgente” (source) e viene tradotto nel prodotto finale che è detto “codice eseguibile”.

I compilatori sono fondamentali per la moderna informatica. Essi agiscono come traduttori, trasformando i linguaggi di programmazione orientati all'uomo (human oriented) nei linguaggi di programmazione orientati alla macchina (machine oriented). Per la maggior parte degli utenti, un compilatore può quindi essere visto come una scatola nera, che compie una trasformazione tra due diversi linguaggi.

Il **GCC** è il compilatore C standard per la piattaforma Linux, in quanto supporta tutti i moderni standard C, come l'ANSI C, ed ha molte estensioni proprie, che possono essere usate per l'ottimizzazione del codice. Essendo un compilatore multi-piattaforma, consente anche di compilare codice per macchine non-Intel. Il GCC ha moltissime opzioni, ma se ne useranno solo un limitato sottoinsieme.

Se si usa Linux, il compilatore classico è **gcc** (normalmente presente in ogni installazione), se invece si usa MS-DOS, si può usare l'apposito porting di GCC, **DJGPP**, che dovrebbe funzionare anche sotto la maggior parte delle versioni di Windows. La versione DOS del compilatore suddetto si può scaricare da <http://www.delorie.com/djgpp>, mentre la versione Linux è presente in tutte le distribuzioni del sistema operativo. Si possono avere approfondite informazioni consultando il sito <http://gcc.gnu.org/>.

## La compilazione

Le fasi di compilazione sono **due**:

- Generazione di un file oggetto (*xyz.o*)
- Generazione del file eseguibile (*xyz.exe* oppure *xyz*)

Entrambe le fasi si eseguono invocando il comando *gcc* seguito dagli opportuni parametri.

- La generazione del file oggetto si ottiene con il comando  
*gcc -c xyz.c*  
(Solo compilazione, no linking)  
Il risultato di tale comando è il file *xyz.o*.
- La generazione del file eseguibile si ottiene con il comando  
*gcc -o xyz xyz.o*  
(Linking)  
Il risultato di tale comando è il file *xyz*.

Per semplificare ulteriormente la procedura di generazione di un file eseguibile si può utilizzare il comando

```
gcc -o xyz xyz.c
```

Il secondo parametro non è più un file *.o*, ma è il sorgente *.c* contenente il codice scritto dall'utente. (Questo è il comando che si userà nella successiva parte del corso.)

**NB:** se si utilizza il comando *gcc xyz.c* senza il parametro “-o”, il compilatore genera un file eseguibile chiamato “*a.out*”.

Nelle versioni di SO Linux esistono anche altri compilatori, ognuno dei quali supporta un diverso linguaggio di programmazione. Seguono alcuni esempi:

- linguaggio C, compilatore **gcc**
- linguaggio C++, compilatore **g++**

- linguaggio **Fortran**, compilatore **gcc-g77**
- linguaggio **Java**, compilatore **gcc-java**

## Le variabili

Il nome di una variabile deve soddisfare i seguenti vincoli:

- deve iniziare con una lettera ;
- deve contenere solo lettere o cifre;
- è ammesso anche il carattere di sottolineato (\_).

Esempi di nomi di variabile validi e non validi:

|                     |                                   |
|---------------------|-----------------------------------|
| <b>linguaggio c</b> | non valido: contiene uno spazio   |
| <b>linguaggioC</b>  | valido                            |
| <b>ANSIC_123</b>    | valido                            |
| <b>3parole</b>      | non valido: inizia con una cifra. |

Ogni variabile definita in un programma C deve essere caratterizzata da un tipo: possono essere tipi standard oppure tipi non-standard definiti dall'utente stesso. Per ora ci si occuperà solamente del primo genere.

## Tipi di variabili

I tipi standard di variabili previste dal linguaggio C sono brevemente descritti in seguito. Si noti che il numero di byte utilizzati per la rappresentazione di ogni singolo tipo può variare da una piattaforma ad un'altra: esistono solo alcune relazioni che compilatore garantisce (come si vedrà in seguito un tipo `int` non può occupare più byte della rappresentazione di un `long int`).

**char** Sono usati tipicamente per rappresentare un carattere ASCII o un byte di memoria.

Il tipo *char* per default è equivalente a *signed char*.

In ogni caso la sintassi ammessa è:

**[signed — unsigned] char .**

(Nel caso che il tipo venga specificato come `signed char` o `unsigned char`, questo prevale sul default assunto durante la compilazione.)

**int** Rappresentazione tipica per numeri interi con o senza segno. Per default i numeri interi sono con segno.

Sintassi ammessa:

**[ signed – unsigned ] int**

**[ signed – unsigned ] short [ int ]**

[ signed – unsigned ] long [ int ]

In ogni caso il compilatore deve garantire la relazione:

**sizeof(short int) ≤ sizeof(int) ≤ sizeof(long int)**

## float

**double** Il loro impiego è per la rappresentazione di numeri reali in singola precisione (float) o in doppia precisione (double). Molti compilatori supportano anche il tipo **long double** per la rappresentazione di un numero reale in quadrupla precisione.

Sintassi ammessa: **float**

[ long ] double

In ogni caso il compilatore deve garantire la relazione:

**sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)**

**void** Questo tipo specifica un insieme vuoto di valori. Viene utilizzato nella dichiarazione e definizione di funzioni come le funzioni che non ritornano valori (equivalenti alle procedure del Pascal), oppure che non accettano parametri. Altro impiego frequente è nel dichiarare un puntatore generico, senza cioè specificare il tipo a cui esso punta.

---

## Primi programmi

### Primo esempio

Creare un file `es2-primo.c` con il seguente contenuto:

```
#include <stdio.h>

int main()
{
    printf("Ciao mondo!\n");
    return 0;
}
```

Compilazione:

```
gcc -o primo es2-primo.c
```

Esecuzione:

```
./primo
```

**#include** Direttiva per includere una libreria che contiene le definizioni di funzioni. Esistono diverse librerie: *stdio.h*, *stdlib.h*, *math.h*, ...

**int main()** Il corpo del programma. Tutte le istruzioni che il programma deve eseguire devono essere contenute in questa funzione **tra le parentesi graffe**.

**printf(...);** Istruzione per la stampa a video.

**return 0;** (numero **ZERO**) Notifica la fine del programma.

### Secondo esempio

```
#include <stdio.h>

int main()
{
    char car;
    int n1, n2, totint;
    float f3, totfloat;
    printf("Inserisci un carattere: ");
    scanf("%c", &car);
    printf("\nInserisci un numero intero n1: ");
    scanf("%i", &n1);
    printf("\nInserisci un secondo numero intero n2: ");
    scanf("%i", &n2);
    printf("\nInserisci un numero decimale n3: ");
    scanf("%f", &f3);

    printf("\n\n%c%c %c\n%c\n", car, car, car, car);
    totint=n1+n2;
    printf("n1+n2= %d\n\n", totint);
    totint=n1*n2;
    printf("n1*n2= %d\n\n", totint);
    totfloat=n1/n2;
    printf("n1/n2= %f\n\n", totfloat);
    totfloat=(float)n1/(float)n2;
    printf("(float)n1/(float)n2= %f\n\n", totfloat);
    totfloat=(float)n1/n2;
    printf("(float)n1/n2= %f\n\n", totfloat);
    totfloat=n1+f3;
    printf("n1+f3= %f\n\n", totfloat);

    return 0;
}
```

## Terzo esempio

```
#include <stdio.h>

int main()
{
    printf("\n%f\n", .1+.1+.1+.1+.1+.1+.1+.1+.1+.1);
    printf("\n%e\n", .1+.1+.1+.1+.1+.1+.1+.1+.1+.1);
    printf("\n%f\n", .1+.1+.1+.1+.1+.1+.1+.1+.1+.1-1);
    printf("\n%e\n", .1+.1+.1+.1+.1+.1+.1+.1+.1+.1-1);

    return 0;
}
```

Perché il programma fornisce risultati diversi?

## Funzioni

### printf()

La funzione **printf** è una funzione di output e permette di stampare a video sequenze di caratteri. È utilizzata per informare l'utente riguardo alle operazioni da compiere (output) oppure per informarlo dei risultati ottenuti dall'esecuzione del programma. La struttura della funzione prevede l'utilizzo di specificatori di formato, i quali definiscono il formato dei dati, cioè come si deve interpretarli e come si deve scriverli.

Per utilizzare la funzione **printf** è necessario includere il file header **stdio.h**.

Formato:

```
printf(stringa, ...);
```

*stringa* è un parametro obbligatorio: può contenere una sequenza di caratteri (una frase) che devono essere inseriti tra virgolette (“...”) e alcuni specificatori di formato. Per ogni specificatore di formato si deve definire quale dato deve essere scritto a video. I dati da formattare e scrivere possono essere: costanti, variabili o espressioni.

Esempi:

```
printf("Ciao Mondo!");
printf("Ciao %c%c%c%c%c!", 'M', 'o', 'n', 'd', 'o');
printf("Prova %d", 16);
printf("Prova %d", 16+21);
printf("Prova %d %d", 16, 22);
printf("%d", 33);
```

Specificatori di formato:

`%d %i` valori interi relativi in base decimale  
`%f %e` valori con virgola in notazione normale ed esponenziale, rispettivamente  
`%g` come `%f` o `%e`, ma sceglie in modo automatico il modo di scrittura migliore  
`%c` carattere  
`%s` stringa (si vedrà poi!!)  
`%u` interi assoluti in base ottale  
`%x` interi assoluti in base esadecimale

Sequenze di escape:

`\n` new line  
`\t` tabulazione orizzontale  
`\"` carattere virgolette

### **scanf()**

La funzione **scanf** è una funzione di input e permettere all'utente di inserire dei dati. Ha un formato molto simile alla **printf**: un primo parametro obbligatorio che può comprendere una sequenza di caratteri inseriti tra virgolette e specificatori di formato (questa volta almeno uno è obbligatorio!!).

Gli specificatori di formato sono gli stessi visti per la funzione **printf**. La sintassi della funzione prevede che come destinazione dei dati forniti in input dall'utente siano definiti gli indirizzi delle variabili.

Esempi:

```
scanf("%c", &car);  
scanf("%d", &intero);  
scanf("%f", &floatingpoint);  
scanf("%c%d", &car, &intero);  
scanf("%c %d", &car, &intero);
```

**NB:** le parentesi graffe si ottengono con la combinazione di tasti

```
{ ==> AltGr + Shift + [  
} ==> AltGr + Shift + ]  
{ ==> AltGr + 7  
} ==> AltGr + 0
```