Introduzione al linguaggio C

Emilio Di Giacomo

Programmi in C

- In questa lezione vedremo come scrivere programmi in C
  - i programmi che vedremo saranno molto semplici
  - ci permetteranno comunque di illustrare alcuni aspetti fondamentali del linguaggio

Programmi in C

- Come abbiamo visto il C è un linguaggio imperativo
- Il programma è dunque una "sequenza" di istruzioni
- Le istruzioni sono raggruppate in funzioni:
  - ogni funzione ha un "compito" specifico
- Il punto di inizio di un programma è sempre costituito da una funzione speciale chiamata ***main***

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Nel seguito discuteremo le varie linee di codice, introducendo i vari concetti del linguaggio C coinvolti

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Le porzioni di codice evidenziate, introdotte da // sono dei commenti. Un commento è una parte di testo che serve solo a documentare (spiegare) il codice; esso viene ignorato dal compilatore.

I commenti introdotti da // terminano a fine linea (commenti di linea)

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
/* la funzione main inizia l'esecuzione
   del programma */
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Un modo alternativo di introdurre un commento è usando i simboli /* e */. Tutto ciò che compare tra /* e */ è considerato un commento ed ignorato dal compilatore.
Questo tipo di commenti può estendersi su più linee, per questo si parla di commento multilinea

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Le riga evidenziata è una direttiva per il preprocessore.
Essa indica che nel nostro programma useremo delle funzioni definite in una libreria standard del C chiamata *stdio*
Tale libreria contiene funzioni per la gestione dell'input e output

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Le righe che iniziano con il simbolo # vengono elaborate da un preprocessore prima della compilazione vera e propria.
Torneremo più avanti su questi aspetti

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Quella evidenziata è la definizione della funzione *main*.
Come abbiamo detto, ogni programma C inizia dalla funzione *main*, quindi ogni programma dovrà necessariamente avere una funzione *main*

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

La definizione della funzione main (così come quella di qualunque altra fuzione) è costituita da un'intestazione e da un corpo

Intestazione di funzioni

- L'intestazione di una funzione, detta anche prototipo della funzione, contiene obbligatoriamente le seguenti informazioni:
  - nome della funzione
  - parametri formali: "contenitori" usati per passare dati di input al metodo; ogni parametro ha un tipo e un nome
  - tipo di ritorno: il tipo di valore eventualmente restituito dal metodo al termine della sua esecuzione

Intestazione di funzioni

- La sintassi è la seguente:

<tipo ritorno> *<nome funzione>* **(***<elenco parametri formali>***)**

nell'elenco, i parametri formali vanno separati da una virgola; ogni parametro è descritto dal tipo e dal nome (in questo ordine); se il metodo non prevede parametri, l'elenco può essere lasciato vuoto o si può usare la parola chiave void

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Il tipo restituito dalla funzione main è int. Vedremo più avanti che cosa significa restituire un valore.
In questo esempio il main non riceve parametri. Vedremo più avanti funzioni con parametri e come questi vengono usati.

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

La parte evidenziata è il corpo della funzione *main*, cioè l'insieme di istruzioni che verranno svolte quando il metodo sarà eseguito; il corpo di un metodo va sempre racchiuso tra due parentesi graffe; ogni istruzione è terminata dal simbolo “;”, detto terminatore di istruzione

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

In questo esempio il corpo del main consiste di una sola istruzione. Tale istruzione è un'istruzione che mostrerà video il messaggio

Welcome to C!

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Più precisamente l'istruzione è un'invocazione di funzione, cioè la richiesta di esecuzione di una funzione.
La funzione invocata è la funzione printf che fa parte della libreria standard stdio.

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

La funzione printf richiede un argomento: il messaggio da mostrare a video. Tale messaggio deve essere una stringa. Una stringa è una sequenza finita di caratteri.
La stringa da visualizzare può essere passata come argomento alla funzione printf sotto forma di letterale stringa.
Un letterale stringa è una sequenza di caratteri racchiusa tra doppi apici.

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

Quando la funzione *printf* viene invocata, il controllo del programma passa temporaneamente dalla funzione *main* alla funzione *printf.*
Tale funzione è implementata in modo da far apparire a video la stringa passatagli come argomento

Il mio primo programma in C

```
// Un primo programma in C
#include <stdio.h>
// la funzione main inizia l'esecuzione del programma
int main(void){
   printf("Welcome to C!\n");
}// fine della funzione main
```

I caratteri che compaiono nella stringa vengono, di norma, visualizzati così come appaiono. Ci sono però alcune eccezioni.

Sequenze di escape

- I caratteri \n della stringa **`"Welcome to C!\n"`** non vengono visualizzati così come appaiono
- Il simbolo \ è detto carattere di escape ed è usato per creare quelle che si chiamano sequenze di escape
- Le sequenza di escape vengono usate per indicare dei caratteri "speciali"
- Ad esempio la sequenza di \n indica una nuova linea
  - il testo visualizzato andrà a capo in corrispondenza di tale sequenza

Sequenze di escape

- Alcune altre sequenze di escape

\n	Nuova linea. Inserisce un a capo
\t	Tabulazione. Inserisce un carattere di tabulazione
\a	Alert. Produce un suono o altro tipo di allarme
\\	Backslash. Inserisce il backslash
\"	Virgolette. Inserisce le virgolette

Fase operativa

- Vediamo ora come è possibile operativamente effettuare le seguenti attività:
  - *scrittura* del programma
  - *compilazione* del programma
  - *esecuzione* del programma

Scrittura

- Il programma che abbiamo visto va scritto in un file di testo
- Il file può avere qualsiasi nome con estensione .c
  - ad esempio *Prova.c*

Compilazione

- Per compilare il programma che abbiamo visto è necessario dotarsi di un compilatore C
- Esistono molti compilatori per il C sia gratuiti che a pagamento
  - Nel nostro corso noi useremo il compilatore Mingw-w64
- Le modalità di esecuzione del compilatore dipendono dallo specifico compilatore utilizzato
- Quando il programma viene compilato esso viene tradotto in linguaggio macchina
- Il risultato della compilazione è un file eseguibile
  - è possibile indicare il nome del file eseguibile

Errori di compilazione

- Se il codice C contiene errori sintattici, il compilatore non produrrà il file eseguibile, e segnalerà gli errori individuati con degli opportuni messaggi:
  - i messaggi di errore del compilatore *aiutano* a comprendere gli errori e a correggerli

Esecuzione

- Una volta ottenuto il file eseguibile esso può essere eseguito:
  - con un doppio click
  - scrivendone il nome su un prompt dei comandi

Errori di esecuzione

- Un programma può anche contenere o generare errori in fase di esecuzione, di varia natura:
  - errori di robustezza: dovuti a situazioni limite mal controllate dal programmatore
  - errori di natura esterna: un evento esterno imprevisto impedisce la corretta prosecuzione del programma
  - errori di logica: il programma fornisce output inesatti a causa di un errore di logica nell'implementazione di qualche algoritmo da parte del programmatore

Dettagli sulla compilazione

- Abbiamo detto che la compilazione di un programma C consiste nel tradurre le istruzioni del linguaggio C in istruzione del linguaggio macchina
- In realtà quando compiliamo un programma avvengono diverse cose:
  - preprocessamento
  - compilazione
  - linking

Preprocessore

- Il preprocessore esegue le direttive per il preprocessore
- Questi sono comandi che tipicamente consistono in inclusione di file o sostituzione di testo
  - sono riconoscibili perché precedute dal simbolo #
- Ad esempio, l'istruzione *#include <stdio.h>* del nostro programma è una direttiva per il preprocesore il cui effetto è l'inclusione del file *stdio.h* nel nostro codice
  - il file *stdio.h* contiene la dichiarazione della funzione *printf*

Compilatore

- Dopo la fase di preprocessamento c'è la vera e propria compilazione
- Il compilatore traduce le istruzioni C in istruzione del linguaggio macchina
- Il risultato è chiamato codice oggetto

Linker

- Il codice oggetto non è ancora un file eseguibile in quanto tipicamente contiene riferimenti a funzioni definite altrove
  - ad esempio nelle librerie standard o in altre librerie scritte da altri programmatori
- Il linker collega il codice oggetto con il codice delle funzioni mancanti e crea il file eseguibile

Un secondo programma

```
#include <stdio.h>
int main(void){
   int integer1; //primo numero
   int integer2; //secondo numero
   int sum;  // risultato
   printf("Inserisci un intero\n");
   scanf("%d", &integer1); // legge un intero
   printf("Inserisci un altro intero\n");
   scanf("%d", &integer2); // legge un secondo intero
   sum = integer1 + integer2; // calcola la somma
   printf("Il risultato è %d\n", sum); // stampa
}
```

Dichiarazione di variabili

- Le tre istruzioni seguenti

```
int integer1; //primo numero
int integer2; //secondo numero
int sum;  // risultato
```

sono dichiarazioni di variabili

Variabili

- Una variabile è un “contenitore” che può memorizzare un valore di un certo tipo, detto il tipo della variabile
- Il valore di una variabile può cambiare nel tempo, ma il suo tipo rimane sempre lo stesso
- Una variabile si definisce specificando il suo tipo ed il suo nome
- Le variabili devono essere definite prima di essere utilizzate

Variabili

- Nel nostro esempio abbiamo tre variabili i cui nomi sono *integer1*, *integer2* e *sum*
- Tutte e tre queste variabili sono di tipo *int*
  - ciò significa che potranno memorizzare valori numerici interi

Nomi di variabili: regole

- Un nome di variabile può essere un qualunque identificatore valido
- Un identificatore è una sequenza di caratteri consistenti di lettere, cifre e underscore (_) che non cominci con una cifra

Nomi di variabili: convenzioni

- Gli identificatori che iniziano con un underscore sono utilizzati dal compilatore e nelle librerie standard
  - è pertanto sconsigliato utilizzare tali identificatori in quanto si potrebbero avere conflitti
- Per convenzione gli identificatori usati come nomi di variabili iniziano con la lettera minuscola

Nomi di variabili: convenzioni

- Ci sono due convenzioni per i nomi di variabile costituiti da più parole
  - le parole vengono separate da underscore
    - es: *primo_numero*
  - le parole vengono concatenate e ogni parola successiva alla prima inizia con la lettera maiuscola
    - es: *primoNumero*

La funzione scanf

- L'istruzione

```
scanf("%d", &integer1);
```

è usata per leggere un valore da tastiera e assegnarlo alla variabile *integer1*

- La *scanf* è una funzione della libreria *stdio.h* (come la *printf*)
- La funzione *scanf* richiede due argomenti

La funzione scanf

- Il primo argomento (*"%d"*) è una stringa di controllo del formato, cioè un letterale stringa che contiene uno specificatore di conversione (cioè *%d*)
- uno specificatore di conversione indica il tipo di dato che deve essere letto da tastiera
  - in questo caso un intero

La funzione scanf

- Il secondo argomento (*&integer1*) indica la variabile in cui memorizzare il valore che verrà letto da tastiera
- In questo caso la variabile è *integer1*, definita in precedenza
- Il nome della variabile è preceduto dal simbolo *&*, che è chiamato operatore di indirizzo

La funzione scanf

- L'operatore di indirizzo serve per indicare alla *scanf* l'indirizzo della cella di memoria corrispondente alla variabile *integer1*
- Capiremo meglio in futuro il significato dell'operatore di indirizzo
- Per ora basta sapere che le variabile passate come argomento alla *scanf* devono essere precedute dall'operatore di indirizzo
  - se non lo si fa si ha un errore in fase di esecuzione

Ancora sulla scanf

- Con un'invocazione della *scanf* è possibile assegnare più valori letti da tastiera a più variabili
- In questo caso si devono utilizzare più specificatori di conversione
- e fornire più variabili (espressioni) come parametri successivi al primo
  - il numero di specificatori di conversione e di variabili deve essere lo stesso
- Esempio:

```
scanf("%d%d", &integer1, &integer2);
```

Assegnazione

- L'istruzione

```
sum = integer1 + integer2;
```

è un'istruzione di assegnazione

- L'istruzione di assegnazione è utilizzata per assegnare un valore ad una variabile

Assegnazione: sintassi

- La sintassi dell'istruzione di assegnazione è la seguente

  <nome variabile> **=** *<espressione>*

- Il simbolo = è chiamato operatore di assegnazione
- A sinistra dell'operatore di assegnazione c'è un nome di variabile
  - nel nostro esempio la variabile somma
- A destra dell'operatore di assegnazione c'è un espressione
  - nel nostro esempio l'espressione integer1+integer2

Espressioni

- Un'espressione combina uno o più valori (ad esempio costanti o variabili) per mezzo di operatori e/o funzioni
- Le operazioni e le funzioni sono valutate (calcolate) secondo delle regole di precedenza producendo un valore (il risultato dell'espressione)
- Nell'espressione *integer1* + *integer2* i valori delle variabili *integer1* e *integer2* sono combinati per mezzo dell'operatore di somma +

Assegnazione: semantica

- L'espressione a destra del simbolo = viene valutata, cioè calcolata
- Il valore risultante viene memorizzato nella variabile che si trova a sinistra del simbolo =
  - si dice che il valore viene assegnato alla variabile
  - qualunque valore fosse stato assegnato in precedenza alla variabile viene sovrascritto
- Il risultato dell'espressione deve essere un valore il cui tipo è uguale a quello della variabile
  - ci sono alcune eccezioni a questa regole che vedremo più avanti

Assegnazione

- Per eseguire l'istruzione

```
sum = integer1 + integer2;
```

quindi:

  - verrà letto il valore della variabile *integer1*
  - verrà letto il valore della variabile *integer2*
  - i due valori verranno sommati
  - il risultato verrà memorizzato nella variabile *sum*

Ancora sulla printf

- L'istruzione

```
printf("Il risultato è %d\n", sum);
```

è un'invocazione dell'istruzione *printf* il cui obiettivo è stampare il valore della variabile sum

- Questa invocazione dell'istruzione *printf* è un po' diversa da quelle che abbiamo visto finora
  - essa ha infatti due argomenti

Ancora sulla printf

- Il primo argomento è una stringa di controllo del formato; essa contiene:
  - una serie di caratteri che verranno visualizzati così come appaiono (*Il risultato è*)
  - una sequenza di escape (*/n*)
  - uno specificatore di conversione (*%d*)
- In questo caso lo specificatore di conversione indica che in quel punto verrà visualizzato il valore di una variabile
  - in questo caso una variabile di tipo intero

Ancora sulla printf

- Il secondo argomento è una variabile il cui valore verrà visualizzato al posto dello specificatore di conversione *%d*
- Più in generale, il secondo parametro può essere un'espressione il cui risultato sia del tipo indicato dallo specificatore di conversione
  - sarebbe quindi del tutto equivalente scrivere

```
printf("Il risultato è %d\n", integer1+integer2);
```

Ancora sulla printf

- In un'invocazione della *printf* è possibile visualizzare il contenuto di più variabili (o espressioni)
- In questo caso si devono utilizzare più specificatori di conversione
- e fornire più variabili (espressioni) come parametri successivi al primo
  - il numero di specificatori di conversione e di variabili deve essere lo stesso
- Esempio:

```
printf("Primo: %d, Secondo: %d\n", integer1, integer2);
```