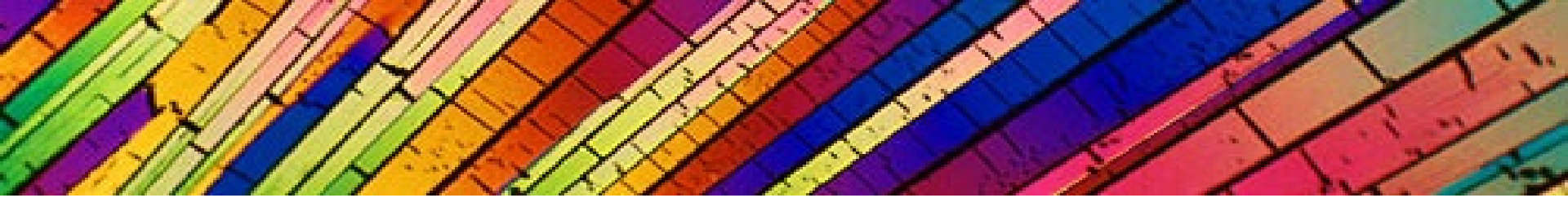


Il linguaggio C

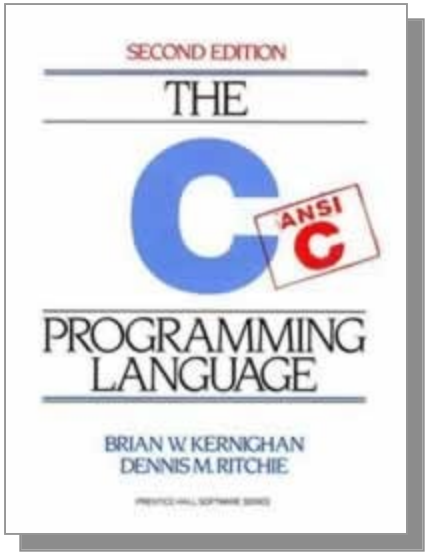
Ver. 3

Storia e versioni

- Sviluppato da Dennis M. Ritchie ai Bell Labs nel 1972 per realizzare il sistema operativo UNIX
  - **K&R C**: 1978 (prima versione, chiamato "K&R" dal nome degli autori del libro che lo ha divulgato: Brian W. Kernighan e Dennis M. Ritchie)
  - **ANSI C**: 1989 (alias: Standard C, C89)
  - **ISO C**: 1990 (è il C89 ratificato dall'ISO, alias: C90)
  - **C99**: 1999
  - **C11**: 2011
  - **C17**: 2018 (Ultimo standard ISO)
- In questo corso: C89

Bibliografia

- "*Il linguaggio C", B. Kernighan, D. Ritchie, 2ª ed.,* 2004, Pearson/Prentice-Hall
  (2 ª ed. originale: 1988)
  "Testo sacro" per generazioni di programmatori, la versione italiana è aggiornata secondo l'errata corrige dagli autori per aderire allo standard ANSI 89



  Ottimo libro *di riferimento* per il C89, conciso ed essenziale, **non adatto** a chi non sa già programmare in qualche linguaggio.

Bibliografia

- "*Programmare in C", K. N. King,* Apogeo, 2009
  Ottimo testo didattico e di riferimento, molto completo e approfondito; tratta completamente sia il C89 sia il C99. Molti esempi, suggerimenti ed esercizi (soluzioni online).
  Con le sue 816 pagine è adatto allo studente esigente che intende conoscere il Linguaggio C a fondo e non solo a livello di corso introduttivo.

Bibliografia

- "*Il linguaggio C. Fondamenti e tecniche di programmazione*", H. Deitel, P. Deitel, case editrici diverse nel tempo.
  Precedentemente il titolo era "*C. Corso completo di programmazione*". Le versioni più recenti includono anche le versioni successive del linguaggio (C99, ecc.).
  Buon testo didattico, sebbene il percorso seguito organizzi gli argomenti in modo diverso da tutti gli altri testi.

Bibliografia

- Riferimenti on-line
  - http://www.eskimo.com/~scs/C-faq/top.html
    "C-FAQ-list", S. Summit, 2004, diverse informazioni sono state incluse nel testo di K. N. King
  - http://www.lysator.liu.se/c/
    Molte risorse e molti documenti, tra cui gli Amendements, i Technical Corrigenda, il Rationale (allegato allo standard per spiegare le scelte adottate), ecc.
  - Newsgroup: comp.lang.c
  - Moltissime pagine su Internet

Caratteristiche del linguaggio

- Un compilatore C è disponibile su tutti i sistemi
- Codice molto efficiente (veloce)
- Caratteristiche di alto livello: adatto per programmi anche complessi
- Caratteristiche di basso livello (accesso ai bit): permette di sfruttare le peculiarità proprie di una macchina o architettura (efficienza)
- Tantissime librerie per aggiungere funzionalità
- Tra i linguaggi più diffusi, il più usato per sviluppare software di sistema
- Interfaccia utente testuale (+libreria grafiche)
- Non è a oggetti e ha gestione manuale della memoria dinamica (nessun *garbage collector*)

La compilazione e gli errori

- Quando un codice di programmazione viene compilato, viene suddiviso (*parsing*) in parti dette *token* dall'*analizzatore lessicale*
- I ***token*** sono elementi sintattici corrispondenti a keyword, identificatori, operatori, punteggiatura, stringhe costanti, ecc.
- Come tutti i linguaggi, anche quelli di programmazione sono descritti da una sintassi che descrive come si possano costruire "frasi" (istruzioni o *statement* nei linguaggi di programmazione) corrette
- Se le istruzioni sono scritte violando le regole della sintassi si ha un *errore sintattico*

La compilazione e gli errori

- Un'istruzione (come anche una frase in linguaggio naturale) sintatticamente corretta non è necessariamente significativa
- Un programma può quindi avere **errori semantici** (ossia di logica) che si evidenziano al *run-time* (ossia quando si esegue il programma): il programma non fa quello che si voleva, il problema può derivare o dalla non correttezza dell'algoritmo o dall'errata implementazione dello stesso

Errori sintattici

- Gli errori sintattici vengono identificati al *compile-time*, ossia dal compilatore durante la compilazione)
- Il compilatore produce due tipi di segnalazioni:
  - gli **Error** sono errori sintattici e impediscono la generazione dell'eseguibile
  - i **Warning** segnalano un possibile (e altamente probabile) problema che il compilatore risolve in base a regole generiche (ma la soluzione generica potrebbe non essere quella corretta in quel caso), non impediscono la generazione dell'eseguibile
- Un codice pulito non deve produrre né errori né warning

Esempi di Warning

- Saranno chiari solo dopo aver studiato i corrispondenti argomenti:
  - `if (a=b)`
    il singolo '=' invece di '==' in una `if` viene normalmente identificato come potenzialmente errato (ma è comunque sintatticamente giusto perché un'assegnazione è lecita anche in una `if`)
  - `for (i=0;i<10;i++);`
    `{ … }`
    non tutti i compilatori segnalano che c'è un ';' sospetto messo alla fine di un `for`, perché ciò è sintatticamente corretto (vedasi "istruzione nulla" nelle slide relative ai cicli), in questo caso il blocco non viene eseguito 10 volte, ma una sola con `i`=10

Identificazione degli errori

- Gli errori al *run-time* possono essere identificati verificando il flusso del programma mentre è in esecuzione mediante:
  - l'aggiunta di istruzioni `printf` in opportuni punti del codice che danno informazioni sulle variabili
  - l'uso di un *debugger*, ossia un programma che esegue il codice in modo controllato e permette di:
    - eseguire le istruzioni fermandosi dopo ciascuna
    - verificare e cambiare il contenuto delle variabili
    - fermarsi automaticamente quando si verificano certe condizioni
  - L'utilizzo del debugger è indispensabile per una programmazione seria

Fasi della compilazione (build)

1. Il **preprocessore** elabora le direttive `#include`, `#define`, ... del file C modificando (temporaneamente) il sorgente da compilare
2. Il **compilatore** traduce il codice C in un file oggetto in linguaggio macchina, due opzioni:
   1. con ottimizzazione (della velocità di esecuzione o della dimensione dell'eseguibile)
   2. senza ottimizzazione (per il debug)
3. Il **linker** assembla in un unico file eseguibile:
   - il file oggetto corrispondente al programma C
   - altri file oggetto (anche compilati da sorgenti scritti in altri linguaggi)
   - le librerie (I/O, matematiche, network, ecc.)

Forma delle istruzioni

- Il linguaggio C è "*case sensitive* ", ossia i caratteri maiuscoli sono diversi dai minuscoli
- Le *istruzioni* sono terminate dal carattere ';' e possono continuare su più righe
- Si può andare a capo in ogni punto dove si può mettere uno spazio, esclusi quelli all'interno delle *stringhe* (sequenze di caratteri delimitate da doppie virgolette, es. "uno due")
- Una sequenza composta da uno o più spazi e/o caratteri di tabulazione (Tab) e/o ritorni a capo e/o commenti è trattata come un unico spazio (tranne che all'interno delle stringhe)

Gli identificatori

- I nomi delle variabili, delle costanti, delle funzioni e di altri elementi sono genericamente detti *identificatori*
- Gli identificatori sono composti da:
  - Lettere (maiuscole e minuscole)
  - Cifre (0-9)
  - Il carattere underscore '_'
- Il 1° carattere deve essere una lettera (o un '_', ma bisogna evitare di usarlo in questo modo in quanto così è usato dal compilatore)
- Non può contenere spazi

Gli identificatori

- Un identificatore deve essere costituito da almeno un carattere
- Lo Standard C89 richiede che gli identificatori *interni* possano essere costituiti da almeno 31 caratteri significativi; se si creano identificatori più lunghi, i caratteri oltre il limite potrebbero essere ignorati dal compilatore, in genere i compilatori ammettono dimensioni maggiori (problemi di portabilità)
- La lunghezza massima è diversa nel caso di *identificatori esterni* definiti in librerie o altri *file oggetto* (es. creati da altri compilatori anche in altri linguaggi), descritto in altre slide

Gli identificatori

- Le lettere maiuscole e quelle minuscole sono considerate diverse: **`V`**`ar`, **`v`**`ar` e **`VAR`** sono identificatori diversi e indipendenti, è considerata cattiva pratica definire variabili con lo stesso nome, salvo casi particolari, alcuni descritti più avanti)
- *È bene usare identificatori con nomi significativi di ciò che rappresentano*
- Per convenzione le variabili si scrivono in minuscolo e le costanti (`#define`, `enum`) in tutto maiuscolo

Gli identificatori

- Per miglior leggibilità, nel caso di variabili composte da parole si può usare l'underscore come separatore o una lettera maiuscola per l'inizio di ciascuna parola tranne la prima:
  Es. `sommaTot`, `somma_tot`, `media7giorni`

Gli identificatori

- Non si possono usare nomi che corrispondono alle ***keyword** del linguaggio:*
  - `auto`, `break`, `case`, `char`, `const`, `continue`, `default`, `do`, `double`, `else`, `enum`, `extern`, `float`, `for`, `goto`, `if`, `int`, `long`, `register`, `return`, `short`, `signed`, `sizeof`, `static`, `struct`, `switch`, `typedef`, `union`, `unsigned`, `void`, `volatile`, `while`
- Non si possono usare i nomi delle funzioni della libreria standard (anche se non viene usata, es. `sin`, `cos`) né i nomi dei simboli definiti negli *header file* associati alle librerie standard

Gli identificatori

- Ogni identificatore a seconda della sua funzionalità appartiene a un ben determinato *spazio dei nomi* (*namespace*)
- I namespace sono *indipendenti* (identificatori uguali in namespace diversi sono *scorrelati*)
- I namespace sono i seguenti:
  - nomi delle etichette (`label`)
  - tag di `struct`, `union` ed `enum` (è uno solo)
  - membri di `struct` e `union` (ogni `struct` e `union` definisce un namespace indipendente)
  - tutti gli altri *identificatori ordinari* (variabili, nomi di funzioni, costanti enumerative, nomi typedef)

I commenti

- I *commenti* sono annotazioni sul codice fatte dal programmatore per appuntare spiegazioni, scelte, informazioni utili per la futura revisione del codice, ecc.
- Vengono ignorati dal compilatore che li considera come un unico carattere spazio, sono di due tipi:
  - quelli *racchiusi* da **/*** e ***/**:

    possono spaziare più righe, ma non possono contenere altri commenti di questo tipo

    ```
    if (b*b-4*a*c <0) /* delta equaz */
    ```
  - quelli *che iniziano* con **//** (solo C99):

    e terminano a fine riga

    ```
    if (b*b-4*a*c <0) // delta equaz
    ```

Istruzioni composte e blocchi

- Più istruzioni possono essere raggruppate per costituire un'unica macro-istruzione detta *istruzione composta*
- Un'istruzione composta può contenere anche istruzioni di definizione di variabili, in questo caso si chiama più propriamente ***blocco***
- Un blocco è individuato da una coppia di parentesi graffe che racchiude le istruzioni
- Le parentesi graffe sono opzionali e normalmente omesse se il blocco non è il corpo di una funzione (compreso il `main`) ed è costituito da una sola istruzione eseguibile

Istruzioni composte e blocchi

- In generale un blocco contiene:
  - una prima sezione *opzionale* con la definizione di tutte le variabili ad uso esclusivo di quel blocco (non si possono definire variabili tra le istruzioni eseguibili come invece permette il C99)
  - una seconda sezione con le istruzioni eseguibili, almeno una

Indentazione del codice

- Le istruzioni di un blocco (NON le eventuali parentesi graffe) si scrivono tutte "*indentate*" (rientrate) di un numero fisso di spazi (es. 4)

```
{
    a = 12;
    b = 23;
    c = a + b;
}
```

- L'indentazione viene ignorata dal compilatore ma *aiuta il programmatore* a comprendere meglio il flusso del programma per cui va fatta *mentre si programma* e non dopo
- La posizione delle graffe e il numero di spazi di indentazione definiscono stili diversi