Programmazione B
Ingegneria e Scienze Informatiche - Cesena
A.A. 2021-2022

Alfabeto ed elementi lessicali del linguaggio C

Catia Prandi - catia.prandi2@unibo.it

Credit: Pietro Di Lena

```
static short legs; int
main() { double sized; bool lover;
long l; return 4; }
   // Haiku: #25
   // Audrius Kovalenko, 2014
```

Introduzione

- Il linguaggio C, così come qualsiasi altro linguaggio di programmazione o linguaggio naturale è definito da:
  - **Alfabeto**. Insieme di simboli ammissibili nel linguaggio
  - **Regole lessicali**. Con i caratteri dell'alfabeto possiamo formare sequenze finite di simboli, dette *parole*. Non tutte le sequenze sono parole ammissibili nel linguaggio. La grammatica del linguaggio definisce le regole lessicali per decidere quali sequenze sono simboli del linguaggio.
  - **Regole sintattiche**. Con le parole appartenenti al linguaggio possiamo definire sequenze di parole, dette *frasi*. Le regole sintattiche ci dicono quali sono le frasi grammaticalmente corrette nel linguaggio.
  - **Regole semantiche**. Solo alcune delle frasi grammaticalmente corrette del linguaggio sono anche *valide* o, in altri termini, hanno un significato. La semantica del linguaggio stabilisce quali frasi corrette sono anche valide e si occupa dell'interpretazione (significato) di tali frasi.
- In queste slide ci occuperemo nel dettaglio dell'**alfabeto** e delle (principali) **regole lessicali** del linguaggio C.

Alfabeto del linguaggio C

- **Alfabeto**: set di simboli usati per rappresentare informazione.
- L'alfabeto di un linguaggio è l'unità minima di informazione che permette di definire costanti, variabili, operatori, parole chiave ed espressioni che formano il codice sorgente di un programma.
- Lo standard ISO richiede che l'alfabeto del linguaggio C comprenda
  - un **alfabeto base** per il codice sorgente (*source character set*).
  - un alfabeto per il codice eseguibile (*execution character set*).
  - trigraphs (sequenze di tre caratteri trattate come un unico carattere)
- Un set esteso di caratteri (ad esempio, lettere accentate) può essere utilizzato (se supportato dalla macchina) solo in commenti, stringhe e costanti di tipo carattere.

Alfabeto base del linguaggio C

Tutti gli standard ISO richiedono che l'alfabeto base del linguaggio C comprenda almeno i seguenti 96 simboli:

- 26 caratteri in minuscolo dell'alfabeto inglese

```
a  b  c  d  e  f  g  h  i  j  k  l  m
n  o  p  q  r  s  t  u  v  w  x  y  z
```

- 26 caratteri in maiuscolo dell'alfabeto inglese

```
A  B  C  D  E  F  G  H  I  J  K  L  M
N  O  P  Q  R  S  T  U  V  W  X  Y  Z
```

- 10 cifre decimail

```
0  1  2  3  4  5  6  7  8  9
```

- 29 caratteri grafici

```
!  "  #  %  &  '  (  )  *  +  ,  -  .  /  :
;  <  =  >  ?  [  \  ]  ^  _  {  |  }  ~
```

- 5 caratteri di spaziatura (*white spaces*)

nuova linea (*newline*) tab orizzontale (*horizontal tab*) spazio (*space*)
nuova pagina (*form feed*) tab verticale (*vertical tab*)

Trigraphs

- I seguenti 9 caratteri dell'alfabeto base del C non sono supportati dallo standard ISO/IEC 646 (standard che specifica la codifica di set di caratteri) e non sono quindi editabili su alcune vecchie tastiere:

  # \ ^ [ ] | { } ~

- I **Trigraph** sono sequenze di tre caratteri, trattate come un singolo carattere, che permettono di editare i simboli non supportati dallo standard ISO/IEC 646:

Carattere	Trigraph
#	??=
\	??/
^	??'
[	??(
]	??)
|	??!
{	??<
}	??>
~	??-

- I Trigraphs sono gestiti dal preprocessore. Molti compilatori moderni non supportano Trigraph o non li gestiscono correttamente. Generalmente, le opzioni di compilazione per la gestione dei Trigraph devono essere attivate esplicitamente dal programmatore.

Esempio: Hello World con trigraph

```
1 ??=include<stdio.h>
2
3 /* Stampa Hello, World! sul terminale */
4
5 int main() ??<
6  printf("Hello, World!??/n");
7  return 0;
8 ??>
```

- **Regole lessicali**: set di regole per poter definire **parole** sull'alfabeto del linguaggio.
- Le parole valide nel linguaggio possono essere utilizzate per la definizione di programmi.
- In C possiamo definire le seguenti principali categorie lessicali:
  - **Parole chiave** (*keywords*)
  - **Identificatori**
  - **Costanti letterali**
  - **Stringhe letterali**
  - Commenti
  - Segni di punteggiatura e operatori
- Formalmente, i *commenti* (già visti), così come *operatori e segni di punteggiatura* (che vedremo più avanti) sono categorie lessicali del linguaggio.

Categorie lessicali: parole chiave

- Il linguaggio C contiene una serie di parole riservate (**keywords**) che non possono essere utilizzate come identificatori di variabili o funzioni.
- Io standard ISO C89 ha definito il seguente set di 32 parole chiave:

```
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
```

- Nello standard successivo ISO C99 sono state aggiunte le seguenti parole chiave:

```
inline   restrict   _Bool   _Complex   _Imaginary
```

Categorie lessicali: identificatori

- Un **identificatore** è una parola composta da una sequenza di caratteri:

```
_  a  b  c  d  e  f  g  h  i  j  k  l  m
   n  o  p  q  r  s  t  u  v  w  x  y  z
   A  B  C  D  E  F  G  H  I  J  K  L  M
   N  O  P  Q  R  S  T  U  V  W  X  Y  Z
```

  e cifre

```
   0  1  2  3  4  5  6  7  8  9
```

- Regole per la creazione di identificatori:
  1. Il primo carattere di un identificatore non può essere una cifra.
  2. Una parola chiave non può essere un identificatore.
  3. Gli identificatori sono *case-sensitive*.
     - Ex., *tmp* è un identificatore diverso da *Tmp*.
  4. Non ci sono limiti specifici alla massima lunghezza di un identificatore.
     - I limiti sono definiti dalla specifica implementazione del compilatore. Un'implementazione deve considerare come *significativi* almeno i primi 31 caratteri di un identificatore.
- Un identificatore può essere utilizzato per denotare:
  - un nome di variabile o funzione
  - un membro o nome (tag) di struttura, union, enumerazione
  - un nome di tipo (definito con typedef)
  - un nome di macro o parametro di una macro

Esempi di identificatori

- Quali dei seguenti sono validi identificatori in C?

`abc`	`123`	`int`	`printf`	`main`	`cioè`	`a+b+c`	`a_b_c`	`i`	`_i`	`2i`	`_`
SI	NO	NO	SI	SI	NO	NO	SI	SI	SI	NO	SI

- 123 inizia con una cifra
- `int` è una parola riservata del linguaggio
- `cioè` contiene un carattere non ammesso (è)
- `a+b+c` contiene due caratteri non ammessi (+)
- `2i` inizia con una cifra

Categorie lessicali: costanti letterali

In C esistono diversi tipi di **costanti letterali**:

- **Costanti intere**. Numeri interi.
- **Costanti decimali o in virgola mobile**. Numeri decimali.
- **Costanti carattere**. Costanti che indicano caratteri alfanumerici.

Costanti intere

- Una **costante intera** è definita come una sequenza di cifre (e lettere per costanti esadecimali).
- Il primo carattere deve essere una cifra.
- Può avere un prefisso che ne specifica la base: decimale (nessun prefisso), ottale (prefisso `'0'`), esadecimale (prefisso `'0x'` oppure `'0X'`).
- Le costanti intere possono essere precedute dal **segno** + o -.
- Esempi:

Base	**Alfabeto**	**Costante**	**Valore**
Decimale	`0, 1, 2, 3, 4, 5, 6, 7, 8, 9`	78 255	78 255
Ottale	`0, 1, 2, 3, 4, 5, 6, 7`	0116 0377	78 255
Esadecimale	`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`	0x4E 0xFF	78 255

Costanti decimali (o in virgola mobile)

- Una **costante numerica decimale** o in **virgola mobile** rappresenta un numero reale che può essere scritto in forma decimale oppure mediante la notazione scientifica:

  `[<Parte Intera>][.<Parte Decimale>][E[<Segno>]<Esponente>]`

- Le costanti in virgola mobile possono essere precedute dal **segno** + o -.
- Esempi:

Costante	**Valore**
0.0	0
0.1	0.1
480E+4	$4800000 = 480 \times 10^4$
.14E-2	$0.0014 = 0.14 \times 10^{-2}$
-3.5E+3	$-3500 = -3.5 \times 10^3$

Costanti carattere

- In C le **costanti carattere** sono singoli simboli racchiusi tra apici ' '

  `'a', 'b', 'c', .., 'A', 'B', 'C', '0', '1', ...`

  - Attenzione: la *costante carattere* `'1'` è diversa dalla *costante intera* `1`.
- E' possibile utilizzare in C caratteri speciali, generalmente non stampabili. Questi caratteri costanti vengono chiamati **sequenze o caratteri di escape**. Le sequenze di escape sono sempre precedute dal simbolo \.

Sequenza di escape	**Descrizione**
`'\a'`	Segnale sonoro (beep)
`'\b'`	Una battuta indietro (backspace)
`'\f'`	Salto pagina (form feed)
`'\n'`	Nuova riga (newline)
`'\b'`	Una battuta indietro (backspace)
`'\r'`	Ritorno a capo della stessa riga (carriage return)
`'\t'`	Tabulazione orizzontale
`'\v'`	Tabulazione verticale
`'\\'`	\(barra rovesciata)
`'\''`	' (apice singolo)
`'\?'`	? (punto di domanda)
`'\"'`	" (doppi apici)
`'\0'`	NULL (Carattere nullo o di fine stringa)

Categorie lessicali: strighe letterali

- Una **stringa letterale** (o costante stringa) è una qualsiasi sequenza di caratteri compresa tra una coppia di doppi apici

  `" .. "`

- I caratteri utilizzabili in una stringa dipendono dal set di caratteri disponibili sul calcolatore. In genere, set di caratteri **Unicode**.
- Ad esempio,

  `"Hello, World!\n"`

  è una stringa letterale terminata dalla sequenza di escape *newline* `'\n'`.
- In C, ogni stringa deve essere terminata con la sequenza di escape `'\0'` (fine stringa).
  - Nelle stringhe costanti, la sequenza dei escape `'\0'` viene inserita automaticamente dal compilatore.
  - In altri casi (che vedremo), deve essere inserita esplicitamente dal programmatore.

Esempio: stringhe letterali e sequenze di escape

```c
// Esempi di utilizzo sequenze di escape.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
  printf("Inserisci il salario mensile desiderato: ");
 // Backspace
  printf("$_______\b\b\b\b\b\b\b");
  getchar(); fflush(stdin);
  printf("Premi invio per ottenere un suono.");
  getchar(); fflush(stdin);
 // Segnale sonoro
  printf("\a");
  printf("Questa riga scomparira' tra 5 secondi.");
  fflush(stdout);
  sleep(5);
 // Ritorno a capo su stessa riga
  printf("\r                                        \n");
  return 0;
}
```

- **La funzione** `getchar()` **legge un carattere da tastiera. La utilizziamo unicamente per "mettere in pausa" le diverse stampe.**
- **La funzione** `fflush(stdin)` **"ripulisce" il *buffer di lettura* della** `getchar()`**. In sistemi non Win si può usare anche (fpurge(stdin). La funzione** `fflush(stdout)` **forza l'output allo standard output.**
- **La funzione** `sleep(t)` **sospende l'esecuzione per** `t` **secondi. Su Win, usare** `Sleep()`**.**