

Introduzione al linguaggio C

Primi programmi

Violetta Lonati

Università degli studi di Milano
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati
Corso di laurea in Informatica

28 settembre 2017

Argomenti

Intro

- Uso del compilatore gcc
- Struttura di un programma C

Tipi di base

- Definizione di tipi

Input e output formattati

- Letture e scrittura di caratteri

Uso del compilatore gcc

- ▶ Si usa da linea di comando: `gcc file_da_compilare.c`
- ▶ Per default produce il file eseguibile `a.out`
- ▶ Per eseguire il file basta digitare `./a.out`
- ▶ Opzioni utili:
 - ▶ `-o file_di_output` per salvare l'`output` nel file specificato,
 - ▶ `-c` per arrestare la compilazione prima della fase di `link`,
 - ▶ `-s` per arrestare la compilazione prima della fase di `assemble`,
 - ▶ `-E` per arrestare la compilazione dopo la fase di `preprocessing`,
 - ▶ `-W` , con parametro `all` , per la segnalazione di avvertimenti (warning),
 - ▶ `-l` , con parametro `m` , per linkare le librerie matematiche,
 - ▶ `-ansi` per ignorare le funzionalità di gcc incompatibili con lo standard ANSI,
 - ▶ `-pedantic` per aumentare il livello di pedanteria nel segnalare cose non conformi allo standard ANSI!

Esempio

```
$ gcc -lm -Wall -o cerchio cerchio.c
$ ./cerchio
```

Forma generale di un programma C

Esempio di programma

```
#include <stdio.h>

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

Struttura generale di un programma

```
DIRETTIVE
int main( void ) {
    ISTRUZIONI
}
```

Tipi numerici in C

In C esistono due **tipi numerici built-in** (di base):

- ▶ numeri interi **int**
- ▶ numeri decimali in virgola mobile **float**

La dimensione dei tipi varia a seconda della macchina. L'operatore **sizeof** consente di determinare quanta memoria occupa una variabile di un determinato tipo:

- ▶ **sizeof (int)** rappresenta il numero di byte necessari a memorizzare una variabile di tipo int,
- ▶ **sizeof (a)** rappresenta il numero di byte necessari a memorizzare la variabile a.

Tipo int

- ▶ Possono avere segno oppure no:
 - ▶ `signed int`: il bit più a sinistra vale 0 se il numero è ≥ 0 e 1 se è < 0
 - ▶ `unsigned int`: bisogna specificarlo nella dichiarazione
- ▶ Possono avere diverse dimensioni:
 - ▶ `short int`: nelle dichiarazioni si può abbreviare in `short`
 - ▶ `long int`: nelle dichiarazioni si può abbreviare in `long`
- ▶ Il range di variabilità non è fissato nel C standard ma cambia a seconda della macchina, gli unici vincoli sono i seguenti:
$$\text{short int} \leq \text{int} \leq \text{long int}$$
- ▶ Il file di intestazione `limits.h` fornisce alcune macro che definiscono i valori limite, per l'architettura corrente, dei tipi interi.

Sei possibili dichiarazioni per i tipi interi:

```
short si;           unsigned short usi;
int i;             unsigned int ui;
long li;          unsigned long uli;
```

Tipo float

- ▶ In C ci sono tre tipi di numeri in virgola mobile:
 - ▶ `float`: single-precision
 - ▶ `double`: double-precisione
 - ▶ `long double`: extended-precision
- ▶ Anche per i float il range di variabilità non è fissato
- ▶ Il file di intestazione `float.h` fornisce alcune macro che definiscono, per l'architettura corrente, la precisione dei tipi float.
- ▶ Le costanti possono essere scritte in molti modi, purchè contengano un decimale e/o un'esponente. Ad esempio:

<code>57.0</code>	<code>57.</code>	<code>57.0e0</code>	<code>57E0</code>
<code>5.7e1</code>	<code>5.7e+1</code>	<code>.57e+2</code>	<code>570.e-1</code>

Per default le costanti sono memorizzate come `double`. Se basta la singola precisione, basta usare la lettera `f` o `F` alla fine della costante (es: `57.0f`).

Tipo char

L'ultimo tipo built-in del C è il tipo `char` (carattere)

```
char ch;  
ch = 'a'; /* a minuscola */  
ch = 'A'; /* A maiuscola */  
ch = '0'; /* zero */  
ch = '␣'; /* spazio */
```

- ▶ Il valore di un char può cambiare a seconda della **character set** della macchina.
- ▶ Si può generalmente assumere che le lettere minuscole siano contigue nell'ordine alfabetico, idem per le maiuscole, idem per le cifre.
- ▶ I char sono usati dal C come tipi interi: le variabili char possono essere incrementate o confrontate come interi (l'esito del confronto dipende dall'ordinamento del character set).
- ▶ Il file di intestazione **ctype.h** fornisce alcune funzioni per l'elaborazione di caratteri e la conversione di lettere da maiuscole a minuscole e viceversa.

Tipo char - esempi

Assumendo che il character set sia ASCII:

```
char ch;  
int i;  
i = 'a';           /* ora i vale 97 */  
ch = 65;          /* ora ch vale 'A' */  
ch++;             /* ora ch vale 'B' */
```

Per convertire lettere minuscole in maiuscole:

```
if ( 'a' <= ch && ch <= 'z' )  
    ch = ch - 'a' + 'A';
```

Oppure:

```
#include <ctype.h>  
...  
if ( islower( ch ) )  
    ch = toupper( ch );
```

Definizione di tipi

La parola chiave `typedef` consente di definire nuovi tipi a partire da quelli built-in o dai tipi precedentemente definiti

Esempio

Definiamo un nuovo tipo `Bool`. `Bool` potrà essere usato nelle dichiarazioni di variabili esattamente come gli altri tipi built-in.

```
#define VERO 1
#define FALSO 0

typedef int Bool;    /* Def. del tipo Bool */

int main ( void ) {

    Bool flag;      /* Dich. della var flag */
    flag = VERO;    /* Assegno a flag valore VERO */
    ...
}
```

Input e output formattati

- ▶ `printf` e `scanf` danno la possibilità di stampare output e leggere output formattati.
- ▶ Il primo argomento è dato dalla **stringa di formato** che può contenere
 - ▶ **sequenze di escape** (es `\n`)
 - ▶ **specifiche di formato**:
 - ▶ `%d` per gli interi;
 - ▶ `%f` per i float in notazione decimale;
 - ▶ `%e` per i float in notazione esponenziale;
 - ▶ `%c` per i char;
 - ▶ è possibile specificare anche il numero di decimali, di 0 iniziali, l'allineamento...
- ▶ Funzionamento di `scanf` in presenza di spazi bianchi:
 - ▶ gli spazi bianchi iniziali vengono ignorati nella lettura di int e float, ma non di char;
 - ▶ uno spazio bianco nella stringa di formato significa "salta uno o più caratteri bianchi"; ad esempio `scanf("␣%c", &ch)` salta gli spazi bianchi e poi memorizza `ch`.

Letture e scrittura di caratteri

`getchar` e `putchar` permettono di leggere e stampare un carattere alla volta.

- ▶ `ch = getchar();`
memorizza in `ch` il prossimo carattere da standard input;
- ▶ `putchar(ch);`
stampa il carattere `ch` su standard output.

```
/* Trasforma la riga da minuscole a maiuscole */
char ch;
while ( ( ch = getchar() ) != '\n' ) {
    if ( islower( ch ) )
        putchar( toupper( ch ) );
    else
        putchar( ch );
}
```

```
/* Salta gli spazi bianchi */
while ( ( ch = getchar() ) == ' ' )
    ;
```