Tipi e Valori

Moreno Marzolla
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna
http://www.moreno.marzolla.name/

Copyright © Mirko Viroli
h ://m v .a ce.un b . /
© 2008 S efan zza
://use s.d m .un ud. /~s efan .m zza /d da/P 0708/
© 2016–2018 en a z a
://www.m en .ma z a.name/ eac n /FINFA/

T s w s censed unde e ea ve mm ns A bu n-N n mme c a 2.0 (BY-N 2.0) L cense. T v ew a c f s cense, v s
://c ea vec mm ns. / censes/b -nc/2.0/ send a e e ea ve mm ns, 43 H wa d S ee , F , San F anc sc , a f n a, 9410 , USA.

Ringraziamenti

- prof. Mirko Viroli, Università di Bologna
  - http://mirkoviroli.apice.unibo.it/
- prof. Stefano Mizzaro, Università di Udine
  - http://users.dimi.uniud.it/~stefano.mizzaro/

Identificatori

- Sono i "nomi" di alcune parti di programmi, tra cui:
  - variabili
  - costanti
  - funzioni

Regole per gli identificatori

- È vietato usare parole riservate del linguaggio
  - Es: non possono esistere variabili chiamate `int`, `float`, `for`, `while`, ...
- Il carattere iniziale deve essere uno fra:
  - `A` … `Z`, `a` … `z`, `_`
- I caratteri seguenti devono essere scelti fra:
  - `A` … `Z`, `a` … `z`, `_` , `0` … `9`
- Lunghezza massima 31 caratteri (ANSI C) oppure 63 caratteri (C99)
- Maiuscole diverse dalle minuscole
  - `area` e `Area` sono due identificatori diversi

Esempi di identificatori

- Validi
  - `AreaRettangolo`
  - `area_rettangolo`
  - `A`
  - `A1`
  - `a1`
  - `_x` (meglio evitare...)
- Non validi
  - `Area Rettangolo`
  - `Area-Rettangolo`
  - `Area:Rettangolo`
  - `1a`
  - `Area$`
  - `int`
- Il compilatore ci aiuta
  - Se tentiamo di usare un identificatore non valido, segnala un errore in fase di compilazione

Variabili e costanti

- Una variabile è "*un nome a cui è associato un valore [di un certo tipo]*"
- Una variabile può assumere valori diversi durante l'esecuzione del programma, una costante no

```
/* area-cerchio.c - calcola l'area di un cerchio */
#include <stdio.h>

int main( )
{
    const float PI_GRECO = 3.14;
    float raggio;
    float area;

    raggio = 10.0;
    area = raggio * raggio * PI_GRECO;
    printf("%f\n", area);
    return 0;
}
```

PI_GRECO è una costante

raggio e area sono variabili

Variabili e costanti

- Il compilatore segnala errore se si prova a riassegnare un valore ad una costante

```
/* Calcola l'area di un cerchio */
#include <stdio.h>

int main( void )
{
    const float PI_GRECO;
    float raggio;
    float area;

    PI_GRECO = 3.14; /* ERRORE */
    raggio = 10.0;
    area = raggio * raggio * PI_GRECO;
    printf("%f\n", area);
    return 0;
}
```

```
area-cerchio.c: In function ‘main’:
area-cerchio.c:10:5: error: assignment of read-only variable ‘PI_GRECO’
     PI_GRECO = 3.14;
     ^
```

Inizializzazione delle variabili

- Quando si dichiara una variabile o una costante, è possibile specificare il suo valore iniziale
- Se non viene specificato, il valore iniziale è indefinito

```
int inizio, fine;  /* valori iniziali indefiniti */
int mesi = 12;     /* valore iniziale 12 */

/* radice ha valore iniziale indefinito, tasso ha valore iniziale 12.3 */
float radice, tasso = 12.3;
```

Attenzione

- Le variabili non inizializzate hanno valore iniziale arbitrario
  - Accedere ad una variabile non inizializzata è un errore grave

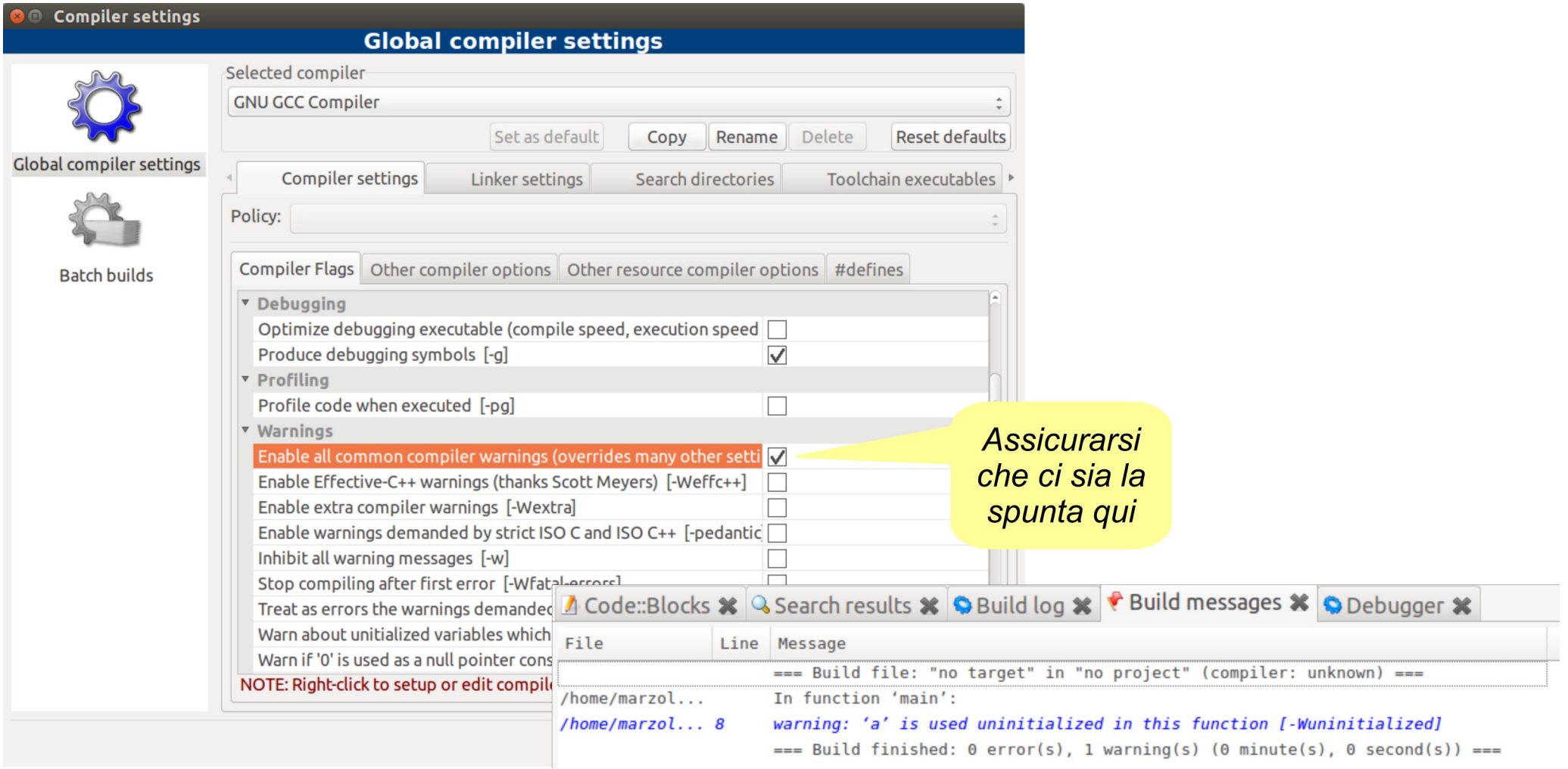
```
/* undefined.c: non si puo' prevedere cosa stampera' questo programma */

#include <stdio.h>

int main( void )
{
    int a;
    printf("a=%d\n", a); /* puo' stampare qualunque valore */
    return 0;
}
```

Suggerimento

- In alcuni casi il compilatore ci può aiutare segnalando un warning quando tentiamo di usare una variabile non inizializzata
- Andare in *Settings* → *Compiler…* → *Compiler Settings* → *Compiler Flags* e assicurarsi che "*Enable all common compiler warnings*" sia abilitato



Definizione di variabili

- Una variabile è “visibile” (quindi utilizzabile) solo dopo che è stata dichiarata
- ANSI C consente dichiarazioni di variabili solo all'inizio di un blocco
  - Cioè subito dopo {
- C99 consente dichiarazioni di variabili in qualunque punto del codice
- Una variabile è visibile solo all'interno del blocco in cui è stata dichiarata
  - Vedremo...

```
#include <stdio.h>

int main( void )
{
    int a = 3;
    printf("Hello, world!\n");
    printf("a=%d\n", a);
    if ( a>0 ) {
        int b = 10 + a;
        printf("b=%d\n", b);
    } else {
        int b = 17 - a;
        printf("b=%d\n", b);
    }
    return 0;
}
```

I tipi primitivi del C

- Ci sono solo quattro tipi primitivi in C

`char`

  - Un singolo byte (8 bit), in grado di contenere il codice ASCII di un carattere

`int`

  - Un intero (positivo o negativo) di lunghezza fissata, che normalmente riflette la lunghezza "naturale" degli interi sulla macchina (32 o 64 bit)

`float`

  - Un valore in virgola mobile a singola precisione

`double`

  - Un valore in virgola mobile a doppia precisione

`short`, `long`, `unsigned`

- Al tipo **`int`** si può preporre **`short`** o **`long`**
  - es: **`short int`**, **`long int`**
  - Consente di avere versioni codificate con più o meno bit
- Al tipo **`char`** e **`int`** si può preporre **`signed`** o **`unsigned`**
  - es: **`signed int`**, **`signed char`**, **`unsigned int`**, **`unsigned char`**
  - Consente di avere versioni con valori solo positivi (**`unsigned`**) o sia positivi che negativi (**`signed`**)
  - **`int`** si assume sempre **`signed`** (non occorre specificarlo)
  - **`char`** potrebbe essere sia *signed* che *unsigned*
  - se vogliamo essere sicuri, dobbiamo specificarlo (**`signed char`** oppure **`unsigned char`**)
- short/long e signed/unsigned possono essere combinati
  - Es: **`unsigned long int`**, **`unsigned short int`**

Su una macchina moderna

Type	Keyword	Byte	Valori rappresentabili
character	`char`	1	-128 .. 127
unsigned character	`unsigned char`	1	0 .. 255
integer	`int`	4	-2147483648 .. 2147438647
unsigned integer	`unsigned int`	4	0 .. 4294967295
short integer	`short`	2	-32768 .. 32767
unsigned short integer	`unsigned short`	2	0 .. 65535
long integer	`long int`	8	-2^{63} .. $2^{63}-1$
unsigned long integer	`unsigned long int`	8	0 .. $2^{64} - 1$
single-precision	`float`	4	1.2×10^{-38} .. 3.4×10^{-38}
double-precision	`double`	8	2.2×10^{-308} .. 1.8×10^{308}

I tipi primitivi del C

- La specifica del linguaggio non indica quanti bit bisogna usare per rappresentare valori di ciascun tipo
- È possibile scoprire cosa fa il compilatore usando l'operatore **`sizeof(`***`T`***`)`**
  - Restituisce il numero di byte necessari per rappresentare una variabile di tipo *T*
- Il linguaggio C garantisce che

  1 = sizeof(char) ≤ sizeof(int) ≤ sizeof(float) ≤ sizeof(double)

Uso di `sizeof()`

```
/* size.c */
#include <stdio.h>

int main( void )
{
  printf("Size of char is %lu\n",          sizeof(char));
  printf("Size of short int is %lu\n",     sizeof(short int));
  printf("Size of int is %lu\n",           sizeof(int));
  printf("Size of long int is %lu\n",      sizeof(long int));
  printf("Size of float is %lu\n",         sizeof(float));
  printf("Size of double is %lu\n",        sizeof(double));
  return 0;
}
```

%lu indica che l'argomento di printf è di tipo "unsigned long"

Output sulla mia macchina:

```
Size of char is 1
Size of short int is 2
Size of int is 4
Size of long int is 8
Size of float is 4
Size of double is 8
```

Letterali (costanti numeriche)

- `10` ha tipo `int`
- `010` ha tipo `int` e rappresenta (in base 8) il valore decimale 8
- `0x10` ha tipo `int` e rappresenta (in esadecimale, cioè base 16) il valore decimale 16
- `10l` ha tipo `long int`
- `10u` ha tipo `unsigned int`
- `10ul` ha tipo `unsigned long int`
- `10.0f` ha tipo `float`
- `10.0` ha tipo `double`
- `'a'` ha tipo `char`
- `"prova\n"` ha tipo *stringa* (=sequenza di caratteri)

Funzione `printf`

- È usata per stampare a video
- Accetta 1 o più argomenti (parametri)
  - Il primo è una stringa (letterale, cioè racchiusa tra “...”)
  - La stringa contiene caratteri che vengono stampati tali e quali, e altri caratteri di controllo

`printf` : caratteri di controllo

- I seguenti caratteri di controllo stampano un valore di un certo tipo, passato come ulteriore parametro:
  - **`%d`** stampa un valore di tipo **`int`**
  - **`%ld`** stampa un valore di tipo **`long int`**
  - **`%u`** stampa un valore di tipo **`unsigned int`**
  - **`%lu`** stampa un valore di tipo **`long unsigned int`**
  - **`%c`** stampa un carattere di tipo **`char`**
  - **`%s`** stampa un valore di tipo stringa (vedremo...)
  - **`%f`** stampa un valore di tipo **`float o double`**
- Casi particolari:
  - Usare **`%d`** per stampare la rappresentazione numerica di un **`char`** o **`unsigned char`**

Escape characters in `printf`

- Nel primo parametri di printf() si possono inserire caratteri speciali (*escape characters*)
  - `\n` per andare a capo
  - `\r` per tornare a inizio riga
  - `\t` per stampare una tabulazione
  - `\\` per stampare il carattere \
  - `%%` per stampare il carattere %
  - `\"` per stampare il carattere virgolette "

```c
/* test-printf.c */
#include <stdio.h>

int main( void )
{
  char a = 'w';
  int b = -12;
  double c = 3.14;
  float d = 2.54;    /* nota: conversione double → float */
  unsigned int e = 133;  /* nota: conversione int → unsigned int */

  printf("a vale \"%c\" ma anche \"%d\"\n", a, a);
  printf("b vale \"%d\"\n", b);
  printf("c vale \"%f\"\n", c);
  printf("d vale \"%f\"\n", d);
  printf("e vale \"%u\"\n", e);
  return 0;
}
```

```
a vale "w" ma anche "119"
b vale "-12"
c vale "3.140000"
d vale "2.540000"
e vale "133"
```

Uso del tipo `char`

- 'A', 'a', 65 (codice ASCII di 'A')

```
#include <stdio.h>

int main( void )
{
   char c1 = 'a', c2 = 'A', c3 = 65;
   printf("%c %c %c %c\n", c1, c1+1, c2, c3);
   return 0;
}
```

```
a b A A
```

`scanf` : caratteri di controllo

- Gli stessi caratteri di controllo si possono usare con **`scanf()`** per leggere un valore da tastiera e assegnarlo ad una variabile di un certo tipo
  - **`scanf("%d",&x)`** legge un int e lo assegna a x
  - **`scanf("%ld",&x)`** legge un long int e lo assegna a x
  - **`scanf("%u",&x)`** legge un unsigned int e lo assegna a x
  - **`scanf("%lu",&x)`** legge un long unsigned int e lo assegna a x
  - **`scanf("%f",&x)`** legge un float e lo assegna a x
  - **`scanf("%lf",&x)`** legge un double e lo assegna a x

Attenzione: il valore di una variabile di tipo **`double`**
si stampa con **`"%f"`** *ma si legge con* **`"%lf"`** *!!*

Esempio

```
/* test-scanf.c */
#include <stdio.h>

int main( void )
{
  int b;
  double c;
  float d;
  unsigned int e;

  printf("Digita un int\n"); scanf("%d", &b);
  printf("Hai digitato: %d\n", b);
  printf("Digita un double\n"); scanf("%lf", &c);
  printf("Hai digitato: %f\n", c);
  printf("Digita un float\n"); scanf("%f", &d);
  printf("Hai digitato: %f\n", d);
  printf("Digita un unsigned int\n"); scanf("%ud", &e);
  printf("Hai digitato: %ud\n", e);
  return 0;
}
```

I booleani?

- In C non esiste un tipo predefinito per indicare valori booleani (true/false)
- Si usano al loro posto gli interi
  - 0 vuol dire *falso*
  - tutti gli altri valori significano *vero*
  - (si consiglia di usare 1 per indicare vero)
- Quindi:
  - l'espressione `(20 > 10)` ha tipo `int` e valore 1
  - l'espressione `(10 > 20)` ha tipo `int` e valore 0