UNIVERSITÀ
degli STUDI
di CATANIA

Variabili, tipi primitivi, operazioni aritmetiche in C

Corso di programmazione I (A-E / O-Z) AA 2022/23

Corso di Laurea Triennale in Informatica

Fabrizio Messina
fabrizio.messina@unict.it

Dipartimento di Matematica e Informatica

Indice

Variabili, costanti e commenti in C

Variabili in C

Cosa è una variabile?

Uno spazio in memoria adeguato ad ospitare un **dato** di un certo **tipo**.

Lo spazio è identificato da un **nome univoco** all'interno del programma.

La variabile corrisponde ad un certo **indirizzo nella memoria del calcolatore** nel quale si esegue il programma.

Variabili in C

Definizione di una variabile in C

```
int numero_di_ordini = 10;
```

Suggerimenti

Usare un **nome** che "descriva" il contenuto della variabile.

Specificare un valore iniziale? Il C non lo rende obbligatorio ma è altamente consigliato per evitare errori.

Il carattere "**;**" termina la singola istruzione che definisce la variabile.

Variabili in C

Definizione senza inizializzazione:

```
int j; // alloca memoria per un numero intero
```

Definisce una variabile `int` e la inizializza con il valore 10:

```
int numero_di_ordini = 10;
```

Il contenuto della `totale_ordini_euro` viene inizializzato con il valore attuale di alcune variabili:

```
int totale_ordini_euro= prezzo * quantita;
```

Variabili in C

Errore in fase di compilazione!

```
int codice = "AA10";
```

Definisce una o più variabili contemporaneamente:

```
int v1, v2=3, v3;
```

NB: v1 e v3 nessun valore iniziale, v2 inizializzata!

Definisce una variabile di tipo carattere (e la inizializza):

```
char c='a';
```

Variabili in C

Letterali vs variabili

Inizializziamo le variabili con espressioni che usano **letterali e/o altre variabili**.

Un **letterale** è un elemento del programma che rappresenta un valore, e che viene opportunamente rappresentato in memoria.
`1.0` e `40` sono letterali di tipo, rispettivamente **double**, e **int**.

`"acqua"` è un letterale stringa, mentre 'c' è un letterale di tipo **char**.

Variabili in C

Letterale	Tipo	Note
-6	int	int non ha parte frazionaria, può essere negativo
0.5	double	Viene rappresentato in memoria come un double
$0.5f$	float	Viene rappresentato in memoria come un float
1E6	double	Notazione esponenziale. Eqauivale a 1×10^6 oppure 1000000
10,456	#	**Errore** in fase di compilazione! Va usato il punto per i decimali
3 1/2	#	**Errore** in fase di compilazione! Va usata una espressione in forma decimale: 3.5

Variabili in C

Regole per i nomi delle variabili

- nomi debbono **iniziare** con una lettera, oppure underscore ("_");
- i **rimanenti** caratteri possono essere anche numeri, oppure ancora lettere o underscore.
- **NO spazi** nel nome delle variabili!
- ATTENZIONE: C **case-sensitive**!
  - `var` e `Var` non sono la stessa variabile.
- Le parole *riservate* (e.g. `double`) non si possono usare per i nomi di variabile..

Variabili in C. Esempi

```c
int _alpha123; // OK

float 2beta ; // Errore in fase di compilazione

double uno due ; // Errore in fase di compilazione

// uno e Uno variabili distinte!
float uno; // ok
float Uno; // ok

int class = 4; // Errore in fase di compilazione
```

Variabili in C

In che modo può cambiare il valore di una variabile nel tempo?

- Assegnamento: ES: `a=10;`, `b=z-20;`
- Incremento/decremento: (forma postfissa) `a++;` `b--;`, (forma prefissa) `++a;` `--b;`
- Istruzione di input: `scanf("%d", &x);`

Usare la stessa variabile a destra e sinistra di un assegnamento.

Quale sará il valore della variabile var dopo l'esecuzione della seguente istruzione?

$$\texttt{var} = \texttt{var} + 10;$$

1. **Leggi** il contenuto della variabile `var`
2. **Somma**, al valore letto in precedenza, il valore 10
3. **Copia** il valore ottenuto dalla precedente somma nella variabile `var`

Ordine di inizializzazione delle variabili

Il seguente frame di codice è (semanticamente) corretto?

```
1 int a=10;
2 int b;
3 int volume = a * b * altezza;
4 b=15;
```

Attenzione alla inizializzatione (tardiva) di `b`.

Quale sarà il valore di `b` nella valutazione della espressione alla riga 3?

Costanti e variabili costanti

Si vuole assegnare un nome ad uno o più valori **costanti**.

Se il valore è noto a tempo di compilazione:

- (A) Uso della direttiva del **preprocessore** `#define`

Se il valore non é noto a tempo di compilazione:

- (B) Uso della parola chiave `const`

Costanti e variabili costanti

(A) La direttiva del preprocessore `#define`:

```
#define PI 3.14
```

Viene trattata dal preprocessore, ovvero prima della traduzione del codice sorgente in codice macchina.

Tutte le occorrenze del simbolo `PI` vengono sostituite, nel codice sorgente, con il letterale `3.14`.

NB: Generalmente le direttive `#define` vengono raccolte all'interno dei file "header" (file con suffisso .h).

Costanti e variabili costanti

(B) Quando il valore non è noto a tempo di compilazione, è possibile usare la parola chiave `const`, che permette di indicare al compilatore che

- il valore di tale variabile **non può cambiare** rispetto al suo valore iniziale.
- quella variabile const va **inizializzata in fase di creazione**.

```
short c = ...;
// Calcola il valore di π con c cifre dopo la virgola
const double pi = calcolaPiGreco(c);
```

Costanti e variabili costanti

Le costanti migliorano la **leggibilità**, quindi la comprensione del codice e permettono di ridurre gli errori in fase di codifica.

```
int somma_iniziale = num_banconote * 10;
```

VS

```
#define VALORE_BANCONOTA = 10;
int somma_iniziale = num_banconote * VALORE_BANCONOTA;
```

Costanti e variabili costanti

```
#define VALORE_BANCONOTA = 10;
int somma_iniziale = num_banconote * VALORE_BANCONOTA;
```

Se il programmatore avesse la necessità di cambiare il valore corrispondente a `valore_banconota` da 10 a 20...

Può farlo! Cambiando una singola riga di istruzione..

Costanti e variabili costanti

Il compilatore **ignora** il testo che rappresenta un commento

Commento singola linea

```
float alpha = 0.5; // deve essere < beta
```

Commento multi-linea

```
/*
Questo programma calcola il profitto medio:
- mensile
- annuo
*/
int main(){...}
```

Costanti e variabili costanti

Esempi svolti

`7_00_var.c`

`7_04_macro.c`

Tipi in C

Tipi numerici nel C

Le dimensioni in bytes sono valori tipici, dipendono dall'implementazione in una specifica architettura..
(*) Valore approssimato.

Tipo	**Range** (tipico)	**Precisione**	**Dimensione**
int	$[-2^{31}, +2^{31}-1]$	–	4 bytes
unsigned	$[0, 2^{32}-1]$	–	4 bytes
long	$[-2^{63}, 2^{63}-1]$	–	8 bytes
unsigned long	$[0, 2^{64}-1]$	–	8 bytes
short	$[-2^{15}, +2^{15}-1]$	–	2 bytes
unsigned short	$[0, 2^{16}-1]$	–	2 bytes
double	$\pm 10^{308}$ $(*)$	15 cifre	8 bytes
float	$\pm 10^{38}$ $(*)$	6 cifre	4 bytes

Tipi numerici nel C

La specifica dello standard per il C (ISO/IEC 9899:2018) **non definisce in modo completo** il numero di bytes o il range di valori per i tipi numerici.

Tali valori generalmente variano con **l'architettura e l'implementazione** (compilatore + librerie standard).

Tipi numerici nel C

1. Sarà importante **conoscere la dimensione** quando si debbono allocare **grandi porzioni di memoria**. ESEMPIO:
   - allocare un array di miliardi di elementi di tipo `int`
   - il fatto che un tipo int occupi 8 bytes piuttosto che 4 bytes sarà determinante. Nel primo caso la memoria potrebbe non bastare.
2. **Conoscere intervallo di valori rappresentabili** dei tipi in virgola mobile per non incorrere in overflow. ESEMPIO: per gli interi con ordine di grandezza 10^{10} useremo i long.
3. **Conoscere la precisione p** dei tipi in virgola mobile per non incorrere in errori di approssimazione non previsti.

Tipi numerici nel C

Numero di byte corrispondenti ad un determinato tipo

In C esiste **l'operatore** `sizeof()`, che prevede un argomento che è un tipo (es: int) oppure una variabile.

```
 //Stampa la dimensione in byte del tipo int
printf("%lu", sizeof(int));
//Stampa la dimensione in byte del tipo double
printf("%lu", sizeof(double));
```

Tipi numerici nel C

Esempi svolti

`7_01_sizeof.c`

Intervalli numerici e precisione

Gli header `float.h` e `limits.h`.

```
#include <limits.h> // header da includere
INT_MIN, INT_MAX // più piccolo/grande valore intero
LONG_MIN, LONG_MAX // più piccolo/grande valore long
```

```
#include <float.h> //header da includere
// più piccolo/grande valore float positivo
FLT_MIN, FLT_MAX
// no.  di cifre significative rappresentabili
// (precisione!)
FLT_DIG, DBL_DIG
```

Tipi numerici nel C

Esempi svolti

`7_01_float_h.c`

Altri tipi in C

https://en.creference.com/w/c/language/types

Tipo	**Range** (tipico)	**Dimensione**
`char`	[-128,+127]	1 byte
`unsigned char`	[0,255]	1 byte
`void`	#	1 byte

Il tipo `void` sta ad indicare **nessun valore di ritorno**.

In C si usano spesso puntatori void (`void *`) per restituire indirizzi di memoria per i quali il codice "chiamante" eseguirà un **casting** ad un tipo specifico di puntatore.

Codice C

Esempi svolti

`7_01_int.c`

`7_01_char.c`

`7_02_precision.c`

Operatori aritmetici, funzioni matematiche di base, conversioni

Operatori aritmetici

https://en.cppreference.com/w/c/language/expressions

Operatore	Num. argomenti	Significato
*	2	Moltiplicazione
/	2	Divisione
+	2	Somma
-	2	Sottrazione
++	1	Incremento
--	1	Decremento
%	2	Modulo

Operatore modulo restituisce il **resto della divisione** tra due numeri **interi**.

Operatori aritmetici

Esempi svolti

`7_01_unary_op.c`

Funzioni matematiche (libreria `math.h`)

https://en.creference.com/w/c/header/cmath

```
#include <math.h>//inoltre ...
//su GNU/Linux compilare con il flag finale "-lm"
```

Funzione	Argomenti	Significato
`pow()`	(x,y) in virgola mobile	x^y
`sqrt()`	(x,y) in virgola mobile	$\sqrt{x}$
`sin()`	(x) in virgola mobile	$\sin(x)$
`abs()`	(x) in virgola mobile	$\lvert x \rvert$
`log()`	(x) in virgola mobile	$\ln x$
`log10(), log2()`	(x) in virgola mobile	$log_{10}(x)$, $log_2(X)$

Codice C

Uso delle funzioni matematiche nei seguenti file.

Esempi svolti

`7_03_overflow.c`

`7_04_constMath.c`

`7_04_inf.c`

Tipi numerici: conversioni e promozioni

⚡Attenzione ai passaggi da un tipo ad un secondo tipo il quale **contiene meno informazioni** (e.g. long $\leftarrow$ int, double $\leftarrow$ int, etc.).

Da floating point a intero (perdita della frazione!)

```
double d = 2.8965;
int a = d; // No warning.  Si provi il flag -Wconversion
```

Sono dette "narrowing conversions" (narrowing=restringimento).

NB: Si compili con il flag "-Wconversion" (compilatore Gnu GCC).

Tipi numerici: conversioni e promozioni

Divisione / moltiplicazione tra interi

```
float result = 5 / 2; // =2 perdita della frazione!
float result = 5.0 / 2; // =2.5 OK
Quale è la differenza?
```

Conversioni implicite

Per una espressione che contiene interi e numeri in virgola mobile, basta che uno dei membri della espressione sia **esplicitamente floating point**.

Il compilatore opererà una **conversione implicita** in virgola mobile degli altri membri per eseguire moltiplicazioni e divisioni in floating point, **senza alcuna perdita di informazione**.

Operazione di cast (conversione)

Casting o conversione esplicita

Il **casting** é un'operazione con cui si indica al compilatore che **deve convertire** una variabile o il risultato di una espressione ad tipo ben definito.

Si rende necessario quando si vuole una **conversione differente** dalla conversione implicita.

Operazione di cast (conversione)

Esempio di casting

```
 double d = 10.73;
double result = (int) (10 * d + 0.5);
```

Esempi svolti

7_04_casting.c

Riferimenti e approfondimenti

[1] → Capitoli 2 e 3.

---

[1] Paul J. Deitel and Harvey M. Deitel.
C Fondamenti e tecniche di programmazione.
Pearson, 2022.