Primi esempi di programmi

Il tipo int
Le variabili
Altri tipi interi
Operatori di assegnazione

Università
degli Studi
di Pavia

Il tipo int

- Un int permette di trattare numeri interi
  - 4 byte (32 bit) con la tecnica del complemento a 2 (ma dipende dal sistema)
- Costanti di tipo intero sono sequenze di caratteri numerici senza il punto decimale
- Se la prima cifra è 0 il numero è interpretato come scritto secondo la notazione ottale (cifre 0-7)
- Se la prima cifra è 0 seguito da x (X) il numero è considerato esadecimale (cifre 0-9 + lettere a-f maiuscole o minuscole)
- 127 può perciò anche essere scritto sia come 0177 che 0x7f

Il tipo int

Espressioni di tipo intero utilizzano:

operatori unari

`+, -` **(simbolo del segno)**

operatori di tipo aritmetico

`+, -, *, /, %` **(resto della divisione)**

```
28 / 5 --> 5
28 % 5 --> 3
```

Tipo intero: operatori bit a bit

& (and) `0xe & 0x3  0x2  (1110 & 0011  10)`

| (or) `0xe | 0x3  0xf  (1110 | 0011  1111)`

^ (exor `0xe ^ 0x3  0xd  (1110 ^ 0011  1101))`

~ (not) (scambia gli 1 con gli 0) `(~ 0…01110  1…10001)`

>>, << (shift) (implementano in modo efficiente moltiplicazioni e divisioni per potenze di 2)

```
0xe << 4  0xe0 (1110 << 4  11100000)
0xe >> 2  0x3  (1110 >> 2  11)
```

>> normalmente **conserva il bit di segno**

```
(010…011100 >> 1  0010…01110)
(110…011100 >> 1  1110…01110)
```

PortaMonete.c

```
/* file PortaMonete.c */
#include <stdio.h>

int main(int argc, char **argv)
{
      // il portamonete contiene
      int uncent = 8;      // otto monete da 1 cent,
      int diecicent = 4;   // quattro da 10 cent,
      int venticent = 3;   // e tre da 20 cent

      // calcola il valore totale delle monete
      int totale = uncent*1 + diecicent*10;
      totale = totale + venticent*20;
      // stampa il risultato
      printf("Valore totale = %d centesimi\n", totale);
      return 0;
}
```

Ho definito 4 variabili di tipo int

```
Valore totale = 108 centesimi
```

PortaMonete.c

user@ubuntu: ~

```
$ gcc PortaMonete.c
$ ./a.out
Valore totale = 108 centesimi
$
```

Commenti

// il resto della riga è un commento

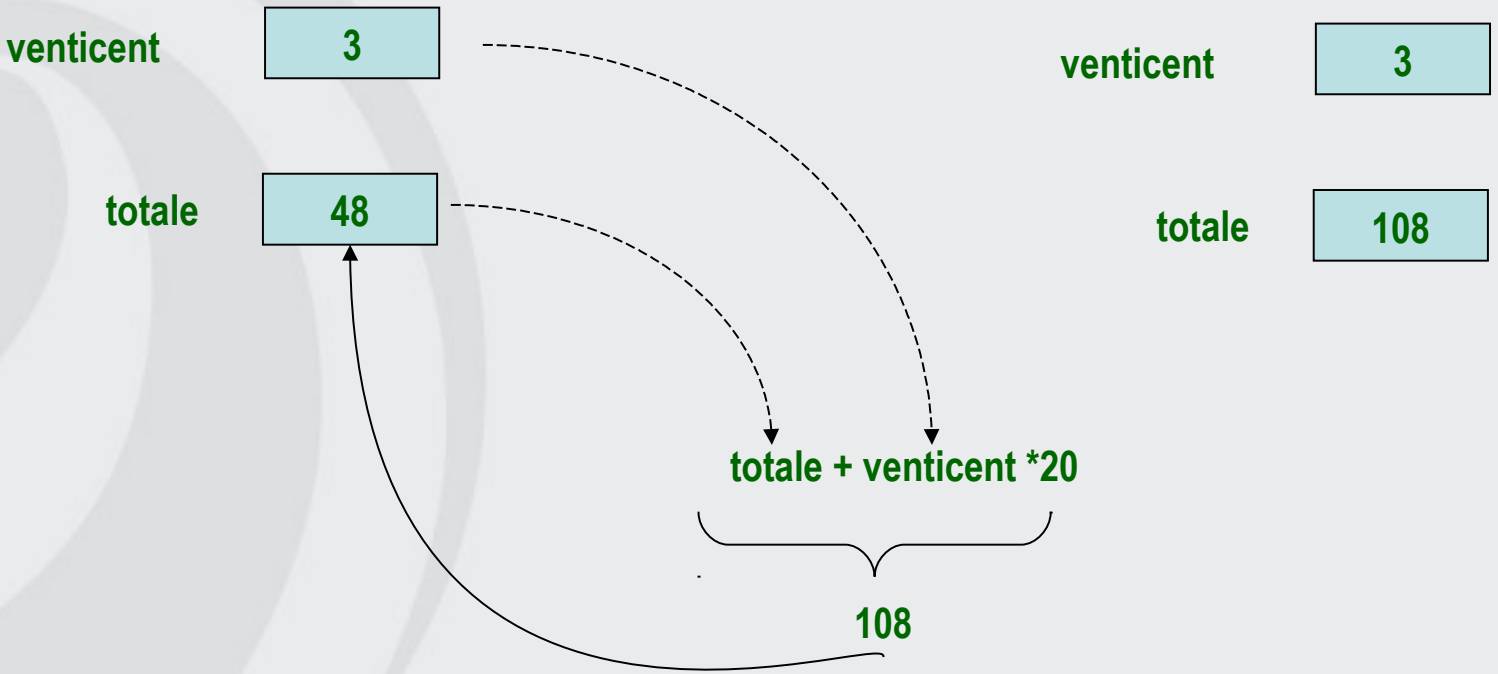
- **I commenti sono ignorati dal compilatore**
- **Servono per documentare e descrivere il codice**

Commenti su più righe: /* ... */

```
/*
Questo è un commento su
più righe.
*/
```

Assegnazione

```
totale = totale + venticent*20;
```



Il nuovo valore ha sostituito irreversibilmente quello vecchio

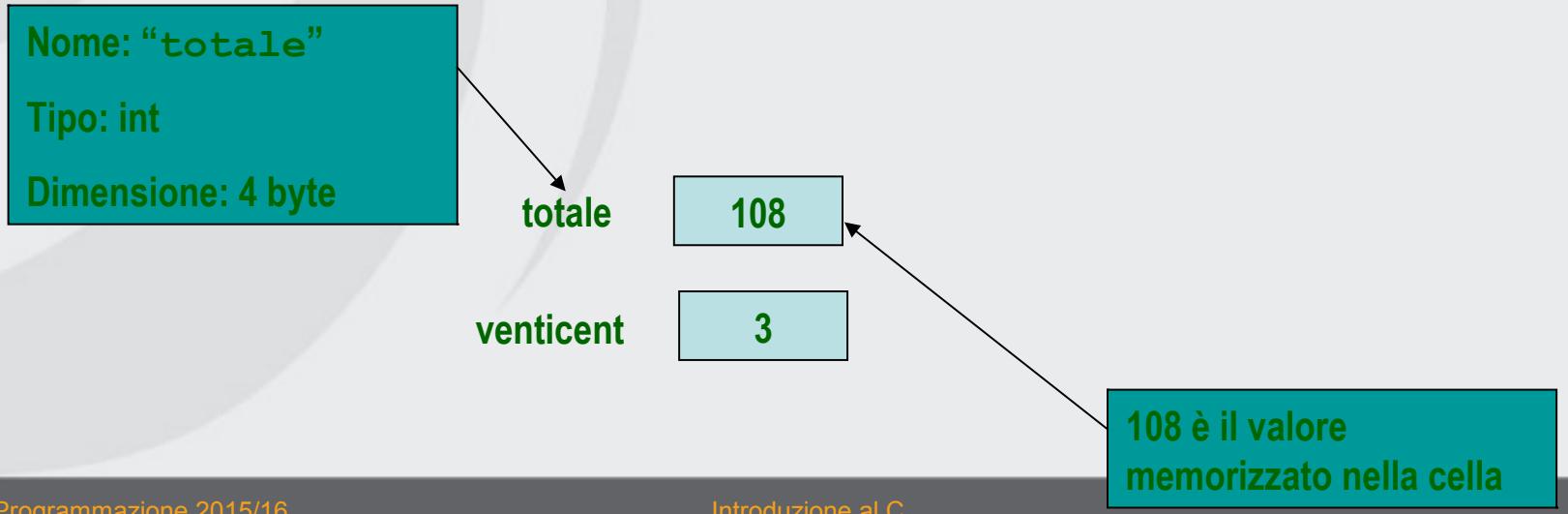
Si ricordi che l'assegnazione non è una equazione (il risultato non è 20)

```
x = 2*x - 20;
```

Il concetto di memoria

Variabili

- **Ogni variabile ha un nome, un tipo, una dimensione e un valore**
  - nome, tipo e dimensione non variano durante l'esecuzione del programma
  - il tipo definisce il modo di interpretare i bit memorizzati
- **Il nome corrisponde a un indirizzo di memoria**
- **Quando un nuovo valore è posto in una variabile, sostituisce (e quindi distrugge) il contenuto precedente**
- **Leggere variabili dalla memoria non altera il loro valore**



PortaMonete2.c

```
/* file PortaMonete2.c */
#include <stdio.h>

int main(int argc, char **argv) {
      // il portamonete contiene
      int uncent = 8;      // otto monete da 1 cent,
      int diecicent = 4;   // quattro da 10 cent,
      int venticent = 3;   // e tre da 20 cent

      // calcolo il valore totale delle monete
      int totale = uncent*1 + diecicent*10;
      totale = totale + venticent*20;

      // stampa il risultato
      printf("Valore totale = %d euro e %d centesimi\n",
          totale / 100, totale % 100);
      return 0;
}
```

```
Valore totale = 1 euro e 8 centesimi
```

Operatori di incremento

++, - - incrementano o decrementano di 1 l'operando

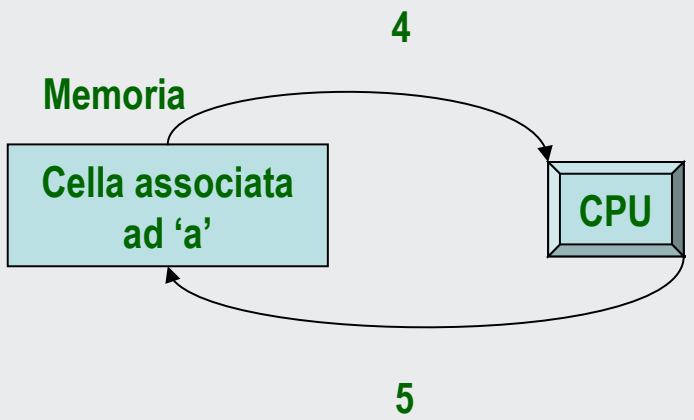
se l'operatore precede l'operando il risultato dell'espressione è quello ottenuto dopo l'applicazione dell'operatore stesso

```
1.        int a = 4
2.        int b = ++a;
3.        printf("%d", a);
4.        Printf("%d", b);
```

```
5
5
```

Esecuzione

1. Prelevo il valore associato alla variabile a
2. Incremento il valore
3. Uso il valore (assegno a b)
4. Salvo nella variabile a il valore aggiornato



Operatori di incremento

++, - - incrementano o decrementano di 1 l'operando

se l'operatore precede l'operando il risultato dell'espressione è quello ottenuto dopo l'applicazione dell'operatore stesso

```
1.        int a = 4
2.        int b = a++;
3.        printf("%d", a);
4.        Printf("%d", b);
```

```
5
4
```

Esecuzione

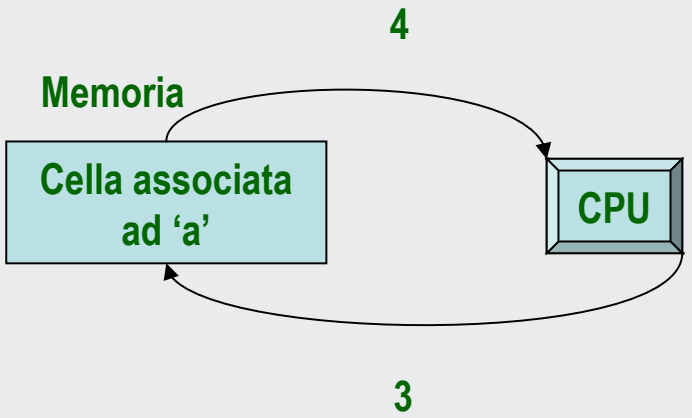
1. Prelevo il valore associato alla variabile a
2. Uso il valore (assegno a b)
3. Incremento il valore
4. Salvo nella variabile a il valore aggiornato

Operatori di decremento

Discorso analogo con l'operatore `--`

```
a = 4;
b = --a; // (a memorizza 3, b 3)
```

```
a = 4;
b = a--; // (a memorizza 3, b 4)
```



++ e -- sono applicabili solo a *variabili* non espressioni

es. ++4 non è corretto (il compilatore segnala un errore di sintassi)

Operatori: priorità

Esiste una priorità fra gli operatori che corrisponde in genere alle normali regole dell'algebra: prima moltiplicazioni e divisioni (e modulo), poi somme e sottrazioni

A parità di priorità le operazioni vengono solitamente eseguite da sinistra a destra, esiste comunque la possibilità di utilizzare le parentesi tonde per forzare un ordine particolare nell'ordine di esecuzione

In caso di dubbi usate sempre le parentesi, in genere i programmi diventano anche più leggibili

In generale la leggibilità aumenta spezzando espressioni complesse in più istruzioni semplici

Operatori: priorità

```
a = b*c + d + e >> 3 + b*c;
```

In realtà volevo scrivere

```
a = ((b*c + d + e) >> 3) + b*c;
```

Spesso è utile spezzare una singola espressione in più istruzioni

```
tmp1 = b*c;
tmp2 = (tmp1 + d + e) >> 3;
a = tmp2 + tmp1;
```

Le parentesi sarebbero inutili ma rendono più chiara l'espressione

Altri tipi interi

costanti di tipo long hanno come ultimo carattere la lettera L (maiuscola o minuscola)

Gli operatori sono ancora quelli visti per il tipo int

Tipo	Memoria	Valore minimo	Valore massimo
char	8 bit	-128	127
unsigned char	8 bit	0	255
short	16 bit	-32768	32767
unsigned short	16 bit	0	65535
long	32 bit	-2147483648	2147483647
unsigned long	32 bit	0	4294967295

I tipi long su alcuni sistemi possono utilizzare 64 bit.

Il tipo char

Il tipo char è comunemente utilizzato per trattare caratteri

```
char c = 'A'; // assegno a c il codice del carattere A
```

Per stampare un carattere con printf uso la notazione %c

```
printf("Codice di '%c': %d\n", c, c);
c++;
printf("Codice di '%c': %d\n", c, c);
```

```
Codice di 'A': 65
Codice di 'B': 66
```

I codici di caratteri consecutivi sono consecutivi

```
printf("%d\n", 'c'-'a');
c='0';
printf("Codice di '%c': %d\n", c, c);
```

```
2
Codice di '0': 48
```

Ovviamente 0 e '0' sono due costanti diverse

Le sequenze di escape

Alcune sequenze di caratteri sono interpretate in maniera particolare, alcuni esempi:

'\"' il carattere ' (apice singolo)

'\\' il carattere \

'\n' “new line” (a capo)

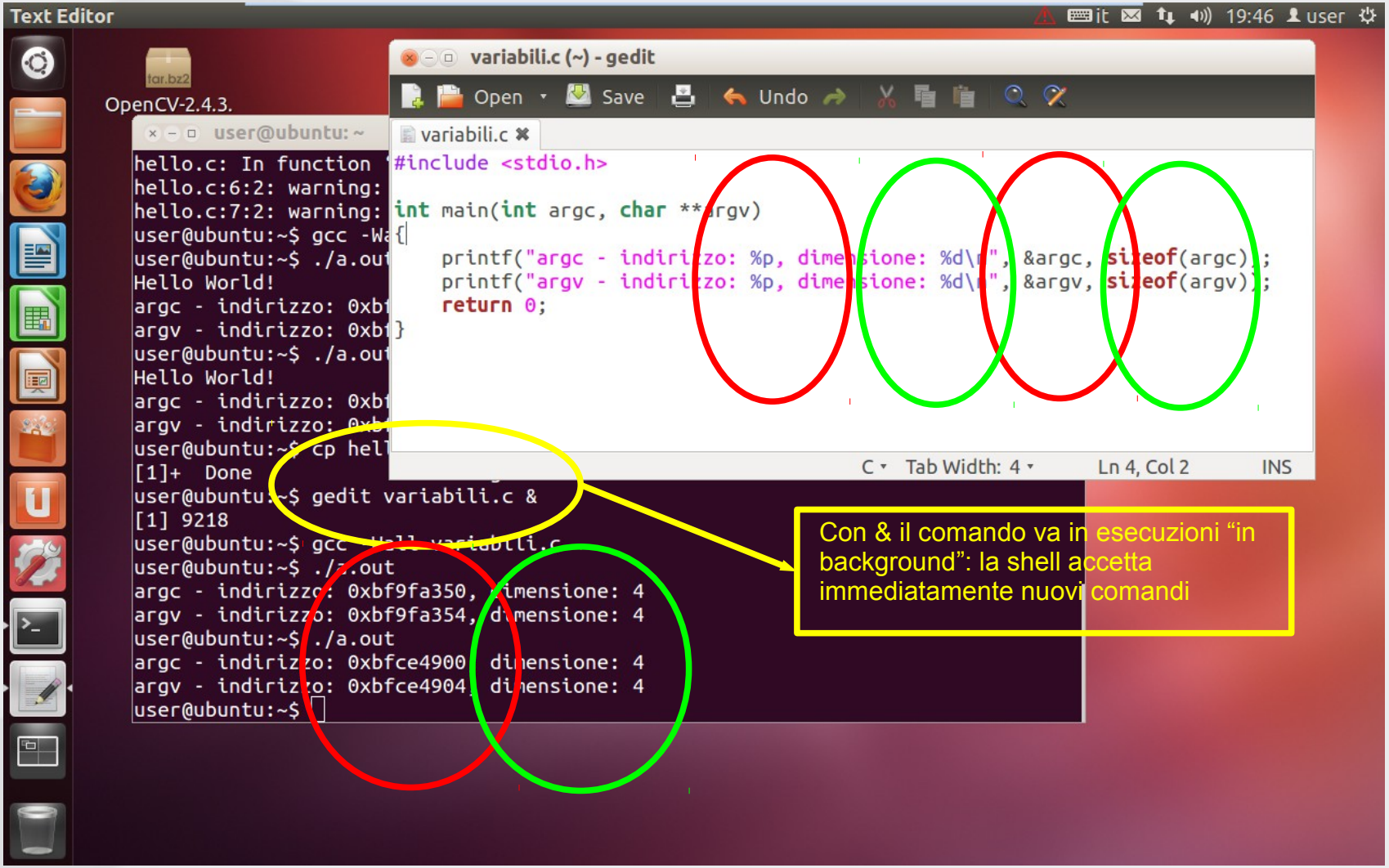
'\r' “carriage return” (torna all'inizio della riga)

'\t' il carattere di tabulazione

```
printf("\ta\rb";
```

```
b       a
```

Informazioni variabili



Informazioni variabili

- "**sizeof**" relativo ad una variabile restituisce il numero di byte utilizzato per memorizzare il valore associato
  - Può dipendere dal sistema
- L'operatore "**&**" applicato ad una variabile restituisce l'indirizzo di memoria corrispondente
  - Con "%p" nella stringa di formattazione di printf stampo un indirizzo di memoria (rappresentato tramite un numero esadecimale)
  - Esecuzioni diverse possono dare risultati diversi

Input da tastiera

variabili.c (~) - gedit

Open Save Undo

variabili.c

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int dato;
    // è bene che appaia un messaggio di richiesta prima di ogni input
    printf("Scrivi un intero: ");
    // funzione duale di printf per la lettura di dati
    // richiede un indizzo di memoria ove memorizzare il valore letto
    scanf("%d", &dato);
    printf("dato=%d\n", dato);
    return 0;
}
```

Permette di definire l'effetto del carattere di tabulazione

C Tab Width: 4 Ln 6, Col 52

```
user@ubuntu:~$ gcc -Wall variabili.c
user@ubuntu:~$ ./a.out
Scrivi un intero: 13
dato=13
user@ubuntu:~$ ./a.out
Scrivi un intero: sss
dato=-1216659468
user@ubuntu:~$
```

Non ho inserito un intero corretto

Input da tastiera

user@ubuntu: ~

```
$ ./a.out <<< ff
Scrivi un intero: dato=-1216806924
$ ./a.out <<< ff
Scrivi un intero: dato=-1217404940
$ ./a.out <<< ff
Scrivi un intero: dato=-1217568780
$ ./a.out <<< ff
Scrivi un intero: dato=-1216712716
$ ./a.out <<< ff
Scrivi un intero: dato=-1216667660
$ ./a.out <<< 16
Scrivi un intero: dato=16
$
```

"<<<" passa come standard input al programma la parola successiva (negli esempi prima "ff" poi "16")

Operatore di assegnazione

- Anche l'operatore = è un operatore lecito all'interno di una espressione restituisce come valore il risultato dell'espressione alla sua destra, gli operatori di assegnazione sono eseguiti da destra a sinistra
  - `a=b=0; // assegna 0 sia a b che ad a`

```
int a, b;
a = b = 4 * 5 / 3;
printf("a=%d b=%d\n", a, b);
a = (b = 4 * 5) / 3;
printf("a=%d b=%d\n", a, b);
```

```
a=6 b=6

a=6 b=20
```

Operatore di assegnazione

Esistono altri operatori di assegnazione:

`+=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=`

il risultato dell'espressione

(var op= espressione) sarà in generale equivalente all'espressione

(var = var op espressione)

Esempio

x += 3 (x = x + 3)

```
totale = totale + venticent*20;
// può essere riscritto come
totale += venticent*20;
```

Tipi floating point

float precisione semplice, double precisione doppia

Costanti
double
133.3
78e-5 **(78 x 10^{-5})**
sono caratterizzate o dal punto decimale o dall'esponente

float
133.3f
78e-5F
sono caratterizzate dal suffisso f(F) (maiuscolo o minuscolo)

Tipo	Memoria	Esponente	Valore massimo
float	32 bit	8 bit	maggiore di 10^{38}
double	64 bit	11 bit	maggiore di 10^{308}

Tipi floating point: operatori

Sono gli stessi operatori visti per i tipi interi

Non sono definiti gli operatori relativi alla gestione dei bit

La divisione ovviamente produce decimali
L'operatore % non è definito

```
28.0 / 5.0 --> 5.6
```

PortaMoneteFloat.c

```
/* file PortaMoneteFloat.c */

#include <stdio.h>

int main(int argc, char **argv)
{
      // il portamonete contiene
      int uncent = 8, diecicent = 4, venticent = 3;
      // provare a gestire l'input da tastiera

      // calcola il valore totale delle monete
      double totale =
          uncent*0.01 + diecicent*0.10 + venticent*0.20;
      // stampa il risultato
      printf("Valore totale = %f euro\n", totale);
      return 0;
}
```

Valore totale = 1.08 euro