



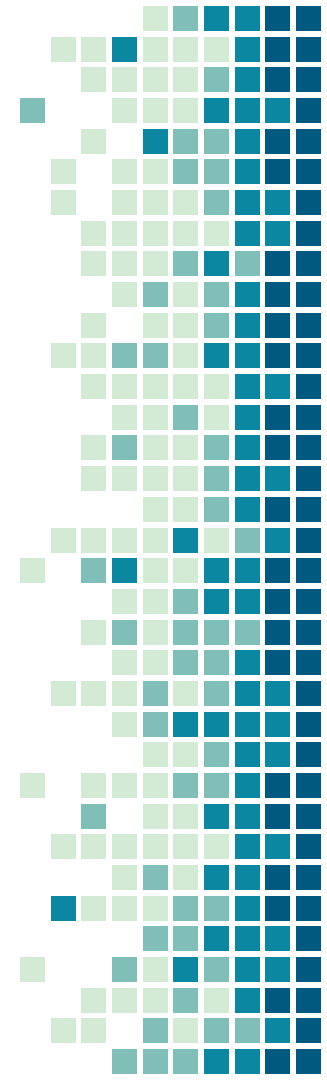
CdL in Informatica – A.A. 2024 – 2025

Programmazione 1 – Modulo 2

Lezione 1
01/10/2024

Andrea Loddo

Federico Meloni - Alessandra Perniciano - Fabio Pili



Argomenti

- Anatomia di un programma in C
- Direttive di preprocessing
- Commenti
- Variabili
- Tipi di dato
- Operatori aritmetici
- Assegnamento
- Stampa sul terminale
- Acquisizione valori da input



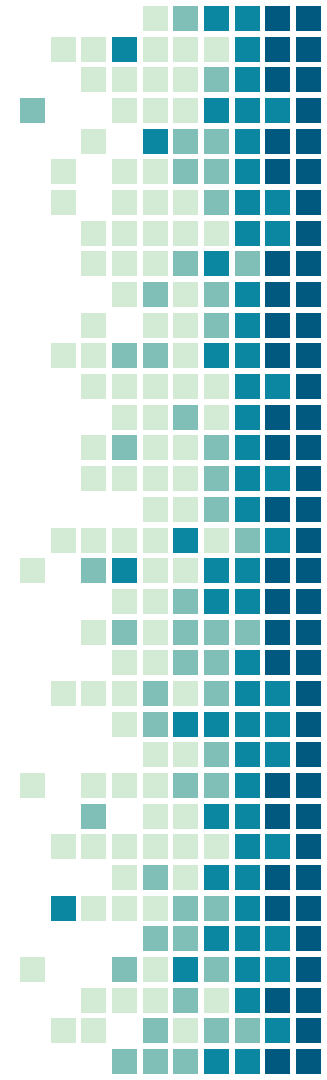
Il punto di partenza

Prerequisiti indispensabili per la scrittura di programmi in un qualsiasi linguaggio:

- comprensione della sintassi
- comprensione della semantica

In altre parole, prima di cimentarsi nella scrittura di programmi al PC sarebbe buona norma:

- essere in grado di comprendere un generico programma
- per esempio, "eseguendolo" con carta e penna



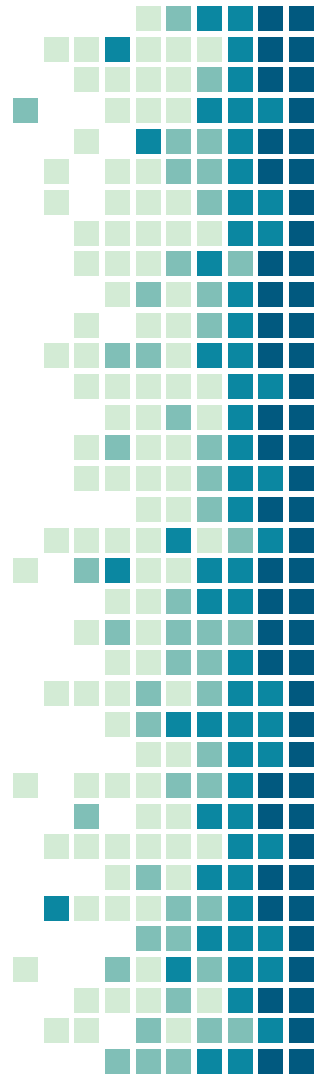
Anatomia di un programma in C

Paradigmi di programmazione del linguaggio C:

- imperativo: istruzioni come "ordini"
- strutturata – procedurale

Caratteristiche **strutturali** di un programma in linguaggio C:

- un **punto di ingresso univoco**
(**entry point**: punto di partenza dell'esecuzione del programma)
- **uno o più** punti di uscita (**exit points**)
- tra l'ingresso e l'uscita del programma sono specificate le **istruzioni** che costituiscono il flusso d'esecuzione del programma stesso
- costrutti **sequenza, selezione e iterazione**



Anatomia di un programma in C

```
#include <stdio.h>

/* Entry point */
int main()
{
    /* Dichiarazione variabili */
    int addendo1;    // Primo addendo
    int addendo2;    // Secondo addendo
    int somma;       // Risultato calcolato della somma

    /* Istruzioni */
    addendo1 = 5;
    addendo2 = 10;
    somma = addendo1 + addendo2;

    /* Istruzioni di output */
    printf("La somma vale %d. \n", somma);    // Somma

    /* Exit point */
    return 0;
}
```

Anatomia di un programma in C

```
#include <stdio.h>
```

Direttive di preprocessing:

- Informazioni utilizzate dal **preprocessore** che trasforma il testo del codice prima della compilazione
- Regole sintattiche:
 - su singola riga di codice
 - nessun carattere delimitatore
 - sempre a inizio codice
 - carattere iniziale: **#**
- **#include**: definisce dove sono situate le **dichiarazioni** delle **funzioni** e delle **macro** che consentono di estendere le potenzialità del linguaggio. La definizione della funzione **printf()** è presente nel file **stdio.h**
- **#define**: consente di definire **macro** utilizzabili all'interno del programma

```
#define NUMERO 5
```

Direttive di preprocessing: Macro

La direttiva `#define` serve per dare un nome simbolico a valori che vengono utilizzati, di solito più volte, all'interno del programma

NON viene istanziata alcuna variabile, in fase di compilazione la macro viene **sostituita** con quello che abbiamo definito per essa.

```
#define PIGRECO 3.1415
...
int raggio = 4;
double area = PIGRECO * raggio * raggio;
```

Definisce una macro, PIGRECO, a cui è associato il valore 3.1415; ogni volta che è necessario il programmatore può usare la parola PIGRECO.

Utilità pratica: se il valore cambia (es. maggior precisione), è sufficiente cambiare il valore associato alla macro.

Utilità stilistica: aiuta la leggibilità e la comprensione del codice, riduce i **magic numbers**

Convenzione e suggerimento: macro **SEMPRE** maiuscole

Anatomia di un programma in C

```
/* Entry point */  
int main()  
{  
    ...  
    /* Exit point */  
    return 0;  
}
```

Entry/exit point: definiamo una funzione `main()` che non ha parametri (parentesi tonde vuote) e restituisce un valore intero (int).

- **Entry point:** keyword `main`
L'esecuzione del programma procede dalla prima **graffa** fino a un exit point esplicito (**return**) o implicito (**graffa** chiusa)
- **Exit point:** keyword `return 0;`
Se il main rende 0, per convenzione, il programma è terminato senza errori. Questa istruzione di pone a fine blocco del main.
- **Blocco:** sequenza di istruzioni all'interno di due parentesi **graffe**

Anatomia di un programma in C

```
/* Dichiarazione variabili */  
int addendo1;    // Primo addendo  
int addendo2;    // Secondo addendo  
int somma;       // Risultato calcolato della somma
```

Cosa è una **variabile**:

- Una variabile è un **contenitore** preposto a contenere dei valori

Dichiarazione delle variabili:

- Specificano quali saranno le **variabili** utilizzate nel programma
- **DOVE**: si scrivono all'inizio del blocco

Anatomia di un programma in C

```
/* Istruzioni */  
addendo1 = 5;  
addendo2 = 10;  
somma = addendo1 + addendo2;  
  
/* Istruzioni di output */  
printf("La somma vale %d. \n", somma);
```

Istruzioni:

- Definiscono le **operazioni** che verranno effettuate
- Le operazioni vengono eseguite secondo l'**ordine** in cui sono state scritte nel nostro programma
- **DOVE:** si scrivono dopo le dichiarazioni di variabili

Le istruzioni terminano **sempre** col simbolo **;** che indica la **conclusione** dell'istruzione attuale e prepara all'esecuzione dell'istruzione successiva.

Anatomia di un programma in C

```
/* Entry point */  
/* Dichiarazione variabili */  
// Addendi  
// Risultato calcolato della somma  
/* Istruzioni  
   di output */  
// Somma  
/* Exit point */
```

Commenti:

- Ignorati dal compilatore
- **Aiutano chi legge** il nostro codice a capire cosa fa un particolare frammento di codice e **aiutano noi** a capire quello che abbiamo scritto la settimana scorsa
- È sempre buona norma commentare il proprio codice
- **DOVE: ovunque!**

 (soprattutto in vista dell'esame e all'esame)

Commenti: formati

Commenti sintetici:

- preceduti da **//** e si possono estendere su singola riga
- forniscono dettaglio su una **porzione limitata** del programma

```
somma = addendo1 + addendo2; // calcolo dalla somma tra due numeri
```

Commenti estesi:

- Racchiusi tra **/*** e ***/** e si possono estendere su una o più righe
- Forniscono informazioni su una **porzione ampia** del programma

```
/* Questo programma effettua la somma di due
   numeri.
   Input: nessuno
   Output: la somma dei due numeri*/
int main()
{
    ...
}
```

Commenti: qualche consiglio

Consigli:

- Non serve ribadire l'ovvio
- Quantità di commenti \simeq quantità di codice
- Prima vs dopo:
 - Ogni programma dovrebbe avere un commento che documenti almeno chi sia l'autore, la data, quali sono i parametri di input e l'output
 - Ogni passaggio di calcolo che **non è palesemente auto-esplicativo**, per questo motivo deve essere commentato

Per la valutazione di questo corso: la **mancanza** o la **scarshezza** di commenti saranno considerati come **errore grave**, al pari di difetti evidenti nel funzionamento del codice.

Variabili: definizione

Variabile: contenitore preposto a contenere dei valori

- Dichiarare una variabile \Rightarrow riservare lo spazio che serve in memoria
- Tipicamente, la dichiarazione si trova prima delle istruzioni

Per ogni variabile è necessario specificare **tipo** e **nome**:

```
tipo_variabile nome_variabile;  
int somma;
```

Variabili di uno stesso tipo possono essere dichiarate consecutivamente:

```
int somma, differenza, moltiplicazione, divisione;
```

Variabili: i tipi di dato

Ogni variabile è caratterizzata da:

- un **tipo**, che indica
 - la **tipologia** di valori che la variabile può assumere;
 - la **quantità di spazio** occupata in memoria;
 - le **operazioni** che possono essere compiute su di essa.
- un **nome** univoco, detto anche **identificatore**, che la identifica all'interno del blocco in cui è dichiarata.

Tipi supportati *nativamente* dal C: **int**, **float**, **double**, **char**

- **INT**: numeri interi (+ e -) \Rightarrow *solitamente occupano 4 byte*
- **FLOAT**: numeri reali (+ e -), single-precision \Rightarrow *solitamente occupano 4 byte*
- **DOUBLE**: numeri reali (+ e -), double-precision \Rightarrow *solitamente occupano 8 byte*
- **CHAR**: caratteri alfanumerici (codice ASCII) \Rightarrow *solitamente occupano 1 byte*

Ogni variabile ha un solo tipo e può contenere solo valori di quel tipo specifico.

Variabili: nomi

Nel linguaggio C, gli identificatori delle variabili devono iniziare:

- con una lettera
- con il carattere underscore _
- ma NON con una cifra

Il nome può contenere **lettere** , **cifre** e **_** ma NON può contenere altri caratteri speciali.

Convenzioni e suggerimenti:

- nomi **mnemonici e significativi**;
- usare notazione **camelCase** o
- usare notazione **snake_case**:

```
int sommaDiDueNumeri; // camelCase  
int somma_di_due_numeri; // snake_case
```

NB #1: una variabile dichiarata all'interno di un blocco è visibile, e quindi utilizzabile, solo all'interno del blocco stesso (**scope locale**)

NB #2: il linguaggio C è **case-sensitive**

Quali di questi nomi sono validi?

- giocatoreUnO
- 2giocatore
- VariabileCasuale
- questaèunavariabile
- nome-cognome
- main
- _underscore_
- get_next_char
- temperatura1
- Temperatura1
- define
- intensita_di_corrente
- Printf



Quali di questi nomi sono validi?

- giocatoreUnO
 - 2giocatore (non deve iniziare con un numero)
 - VariabileCasuale
 - questaèunavariabile (lettera accentata non ammessa)
 - nome-cognome
matematico) (il trattino viene valutato come simbolo)
 - main (keyword)
 - _underscore_
 - get_next_char
 - temperatura1
 - Temperatura1
 - define (keyword)
 - intensita_di_corrente
 - Printf
- NB: **temperatura1** e **Temperatura1** sono due variabili differenti tra loro (il C è un linguaggio case-sensitive)

Operazioni aritmetiche e assegnamenti

Simbolo dell'assegnamento: `=`

Ruolo: assegnare alla variabile a **sinistra** del simbolo (**Lvalue**) la valutazione dell'espressione a **destra** del simbolo (**Rvalue**).

```
/* Esempi di assegnamenti e operazioni aritmetiche */
int a, b, c;
char d;

a = 3;                // valutazione di un valore già definito
c = b = a;            // assegnamento multiplo
a = b + 6;            // espressione aritmetica con +
a = b * c + 5;        // espressione aritmetica con *
a = c - 7 / 5 + a;    // espressione aritmetica con /
a = 10 % 4;           // espressione aritmetica con % (operatore modulo)
d = 'f';              // assegnamento di un carattere
```

Valgono le regole di precedenza **canoniche**.

Per alterare l'ordine, si utilizzano le parentesi tonde:

```
x = b * c + a * e;
y = b * ((c + a) * e);
z = ((b * c) + a) * e;
```

CONSIGLIO: inizializzare SEMPRE una variabile quando la si dichiara.

Funzione di stampa sul terminale

Funzione della libreria stdio: `printf()`

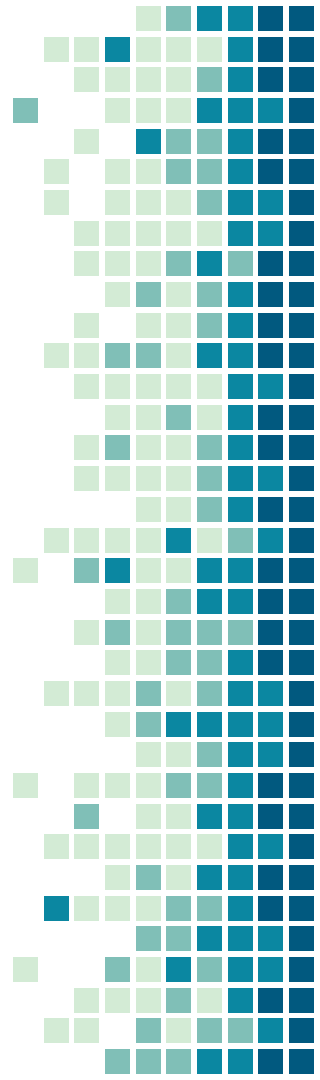
Visualizza sul terminale una sequenza di caratteri e, se desiderato, uno o più valori (o espressioni).

Sintassi: `printf("messaggio", valore1, valore2, ...);`

I **segnaposto**, specifici per ogni tipo di dato, indicano dove sarà visualizzato il valore:

```
int a = 5;
float b = 3.5;
char c = 'z';
double d = 7.9;

printf("Ciao"); // stampa la stringa "Ciao"
printf("%d", a); // stampa un numero intero, cioè 5
printf("b vale: %f", b); // stampa la frase "b vale: 3.500000"
printf("Lettera %c", c); // stampa la frase "Lettera z"
printf("Valore double: %lf", d); // stampa la frase "Valore double: 7.900000"
```



Funzione di stampa sul terminale

È possibile formattare la stampa arricchendola con i seguenti caratteri:

- `\n` Inserimento di un'andata a capo
- `\t` Inserimento di una tabulazione
- `\\` Stampa del carattere backslash
- `\'` Stampa del carattere ' (apice)
- `\"` Stampa del carattere " (doppio apice)
- `%%` Stampa del carattere %

È anche possibile formattare la stampa dei numeri:

```
printf("Valore di b: %.2f", b); // stampa la frase "Valore di b; 3.50"  
  
/* Indicando %.2f si è arrotondato il numero alla seconda cifra  
   decimale, per esempio. */
```

Acquisizione dei dati

Funzione della libreria stdio: `scanf()`

Legge uno o più dati da tastiera e li memorizza nelle variabili corrispondenti indicate.

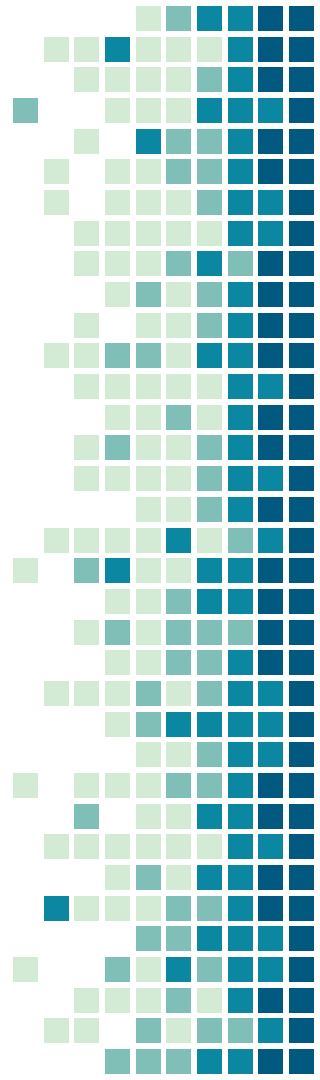
Sintassi:

```
scanf(tipo di dato, indirizzo delle variabili);
```

L'**indirizzo** di una variabile si ottiene antepoendo il simbolo **&** al nome della variabile.

I **segnaposto** indicano il tipo di dato da acquisire, come nella **printf()**:

```
/* %d -> segnaposto di un valore intero  
   &num -> indica di salvare il valore acquisito nella  
   locazione di memoria occupata dalla variabile num */  
  
scanf("%d", &num);
```



Acquisizione dei dati

È possibile acquisire più dati insieme ma è preferibile acquisire **una sola** variabile per **scanf()**.

```
scanf("%d", &a);    // legge un numero intero e lo salva in a
scanf("%f", &b);    // legge un numero float e lo salva in b
scanf("%lf", &c);   // legge un numero double (lf = long float) e lo salva in c
scanf("%c", &c);    // legge un singolo char (carattere) e lo salva in c
```

Esistono funzioni alternative a **scanf()** e **printf()**, utilizzate **SOLO** per l'inserimento di caratteri:

- **getchar()** per l'input
- **putchar()** per l'output

```
char x;
x = getchar();    // legge un carattere e lo memorizza in x
putchar(x);       // stampa a schermo il carattere contenuto in x
```


Meno versatili rispetto a **scanf()** e **printf()**: singolo valore per volta!

Domanda

A cosa serve l'operatore '&'?



Correttezza di un programma



"La compilazione è andata a buon fine! Il programma è sicuramente corretto!"

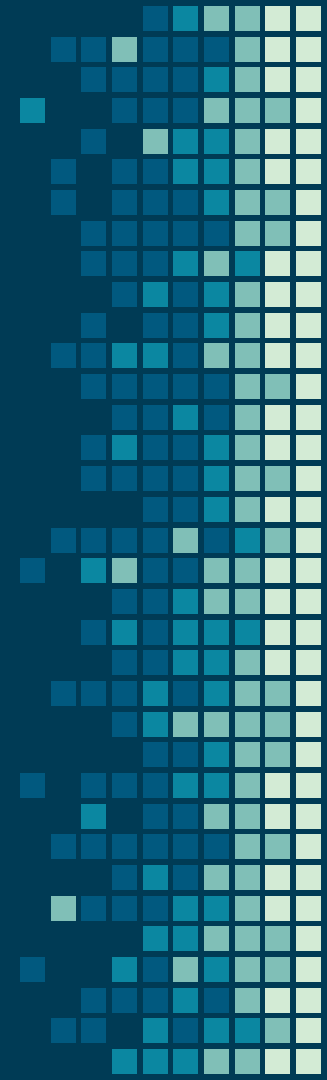
NON È VERO!

Il compilatore verifica soltanto la correttezza **sintattica** del programma, non quella **semantica**

In pratica, non è in grado di predire cosa farà il programma una volta eseguito

FINE!

Domande?



Autovalutazione



Autovalutazione (1/3)

1. Cosa sono un entry point e un exit point?
2. Dove si dichiarano le variabili?
3. Come si scrive un commento su una singola linea? E su più righe?
4. Per cosa è indicato un commento in linea? E su più righe?
5. Cosa caratterizza una variabile?
6. È possibile utilizzare una variabile dichiarata alla riga #25 in un'espressione alla riga #24?
7. Posso avere due variabili con lo stesso nome ma tipo diverso? E con lo stesso tipo ma con nome diverso?
8. Il ';' dopo una qualsiasi istruzione è opzionale?
9. Due variabili di diverso tipo possono occupare uno spazio diverso in memoria?
10. Il nome di una variabile può iniziare con il simbolo '_'? E con '-'?
11. Conosco la differenza tra int, char, float e double?
12. Come si effettua un assegnamento tra interi, float e char?
13. Posso assegnare il valore di una variabile ad un'altra? Come?
14. Sono in grado di effettuare le operazioni di moltiplicazione, divisione, modulo, somma e sottrazione?



Autovalutazione (2/3)

15. A cosa serve una macro e dove deve essere definita? Con quale keyword?
16. Per convenzione, come vanno scritte le macro?
17. Che differenza c'è tra macro e variabile?
18. Posso associare ad una macro una porzione di codice?
19. Cosa sono i magic numbers e perché dovrebbero essere evitati?
20. A cosa serve la funzione 'printf()'?
21. Quali e quanti parametri riceve in ingresso la funzione printf()?
22. So stampare correttamente un float, un char, un double e un int?
23. Come posso andare a capo su terminale? E inserire una tabulazione?
24. Come posso stampare un numero in virgola mobile visualizzando solo le prime tre cifre decimali?
25. La funzione main() restituisce un tipo di dato? Se sì, quale?



Autovalutazione (3/3)

26. Che differenza c'è tra `scanf()` e `printf()`?
27. Quale è il numero minimo di parametri che devono essere passati alla funzione `scanf()`?
28. Sono in grado di acquisire un intero, un float, un double, un char?
29. A cosa serve l'operatore `'&'`?
30. Che differenza c'è tra `getchar()` e `scanf()`?
31. Sono in grado di acquisire due char consecutivi?
32. Per quale motivo si potrebbe creare un problema nella gestione dell'input nel caso in cui acquisisca un carattere?
33. Come viene utilizzata la `getchar()` per gestire al meglio l'input di caratteri?
34. Possiamo affermare che a ogni intero corrisponda un char? In caso affermativo, a cosa corrisponde 65?
35. Posso utilizzare caratteri all'interno delle espressioni? Se sì, perché?
36. Quando vale l'operazione `'a' + 'b' * 2`?
37. Cosa stamperà la seguente operazione `printf("%c", 5 + 'a');` ?

Esercizi



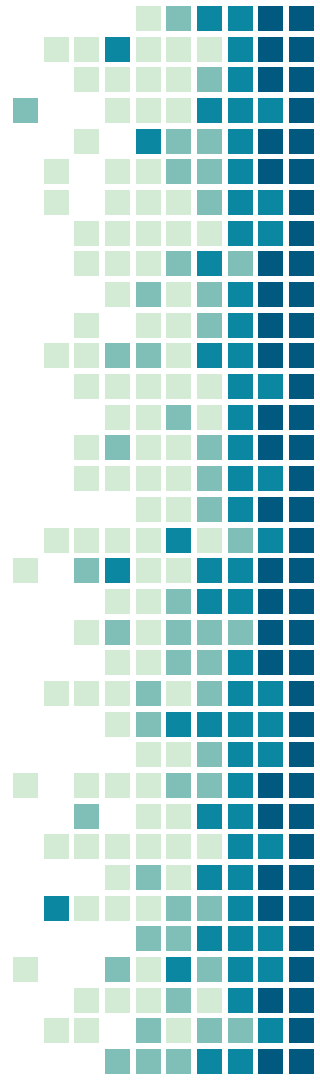
Esercizio 1

Scrivere un programma in cui vengono dichiarate tre variabili intere a, b, c, a cui viene assegnato un valore a piacere.

Il programma deve visualizzare, poi, il risultato dei tre seguenti calcoli:

- $a - b + c$
- $a - b + c * a$
- $(a / b) \% c$

Non devono essere presenti magic numbers e si deve commentare opportunamente tutto il codice.



Esercizio 2

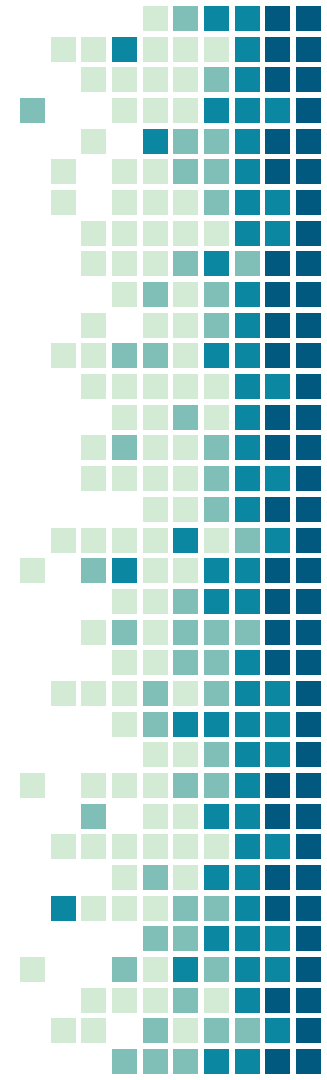
Scrivere un programma in cui si dichiarano due variabili A e B e si assegnino a esse due valori a piacere dello studente.

Il programma deve ora scambiare il contenuto di A e di B.

- Esempio: se inizialmente A contiene 15 e B 7, dopo lo scambio A contiene 7 e B 15.

Stampare i valori delle due variabili **prima** e **dopo** lo scambio.

Non devono essere presenti magic numbers e si deve commentare opportunamente tutto il codice.

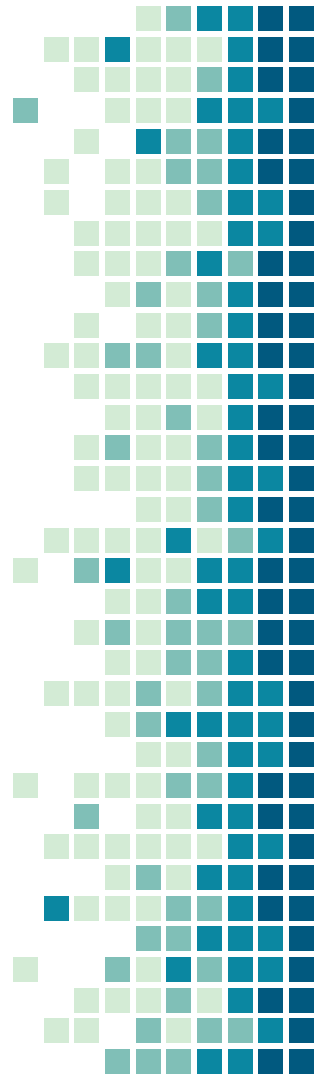


Esercizio 3

Versione 1: scrivere un programma in cui si dichiarano tre variabili intere A, B, e C e si assegnino a esse tre valori a piacere dello studente. Il programma deve calcolare la media dei tre valori, memorizzarla in una variabile di tipo float e visualizzarla con 2 decimali.

Versione 2: ripetere l'esercizio della versione 1 dichiarando tre variabili di tipo float.

Non devono essere presenti magic numbers e si deve commentare opportunamente tutto il codice.

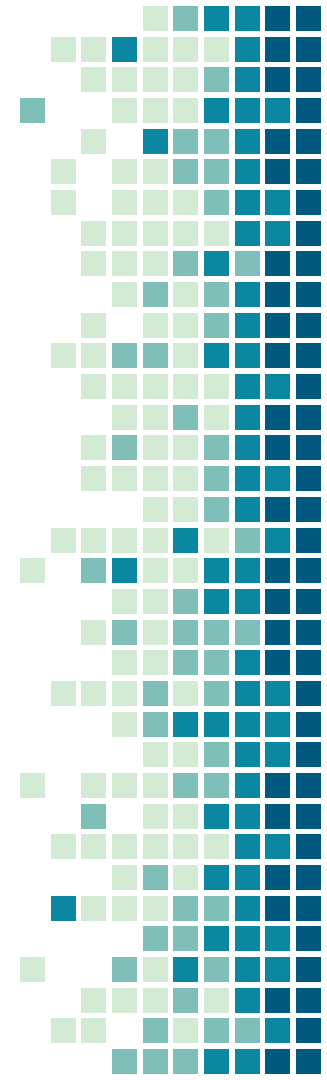


Esercizio 4

Scrivere un programma che, dato un numero complessivo di gatti e il numero di questi per fila, fornisca in output:

- il numero di file risultanti
- il numero di gatti rimanenti nel caso in cui l'ultima fila non sia completa.

Non devono essere presenti magic numbers e si deve commentare opportunamente tutto il codice.



Esercizio 5

Scrivere un programma che, ricevuto in input un numero di gradi Celsius, lo trasformi in Fahrenheit secondo la formula $F = (C * 1.8) + 32$. Effettuare poi la conversione inversa secondo la formula $C = (F - 32) / 1.8$ e controllare mediante stampa a video la correttezza del risultato.

Non devono essere presenti magic numbers e si deve commentare opportunamente tutto il codice.



Esercizio 6

Scrivere un programma che permetta il calcolo del polinomio $5x^4 - 8x^3 + 4x^2 + 3x - 4$. Assegnare alla variabile x un valore a piacere e stampare il risultato a video.



Esercizio 7

Scrivere un programma che permetta di calcolare il costo finale di un certo prodotto. Assegnare a piacere il prezzo del prodotto in una variabile float **prezzoProdotto** e associare a una macro **IVA** una percentuale a piacere. Stampare poi in output il seguente messaggio:

Importo iniziale: __. __ EUR

IVA applicata (__%): __. __ EUR

Importo finale: __. __ EUR

Tutti gli importi e la percentuale dovranno essere sostituiti dal valore corrispondente. Per gli importi dovranno essere stampate solo due cifre decimali.

Non devono essere presenti magic numbers e si deve commentare opportunamente tutto il codice.

