

Eco-design Digitale di Base per i servizi ICT

Programmazione in C e Python

Obiettivi della lezione

- Comprendere l'uso dei puntatori in C e il loro ruolo nella gestione della memoria
- Conoscere le strutture (struct) e il loro utilizzo per aggregare dati
- Effettuare un primo accesso a file di testo con funzioni base

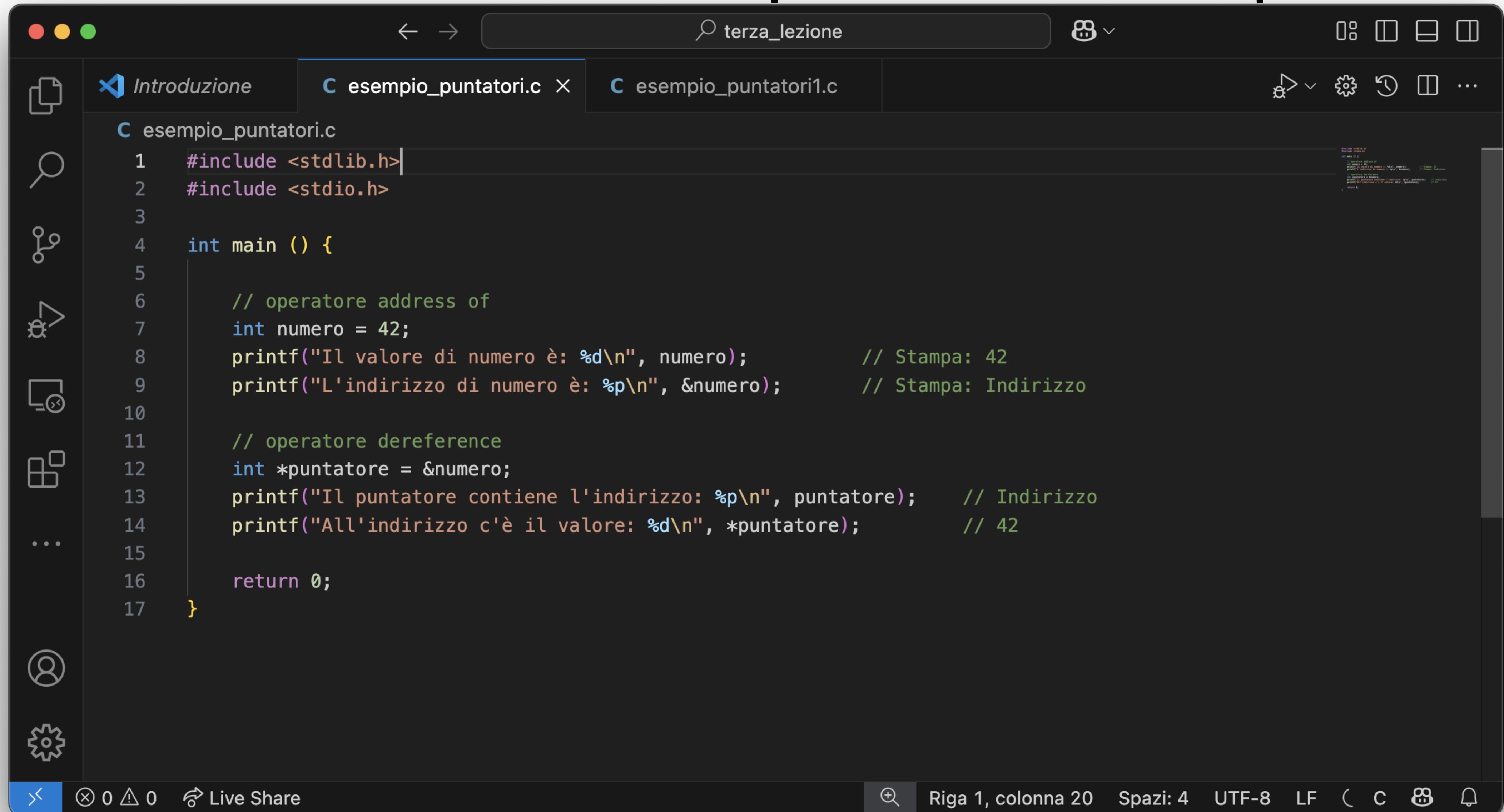
Concetto base dei puntatori

- Cos'è un puntatore?

Variabile speciale che **memorizza l'indirizzo di memoria di un'altra variabile.**

- Operatori fondamentali:
 - «&» (Address of): operatore che ritorna l'indirizzo della variabile
 - «*» (Dereference): operatore che accede al valore contenuto nell'indirizzo puntato

Concetto base dei puntatori: esempi

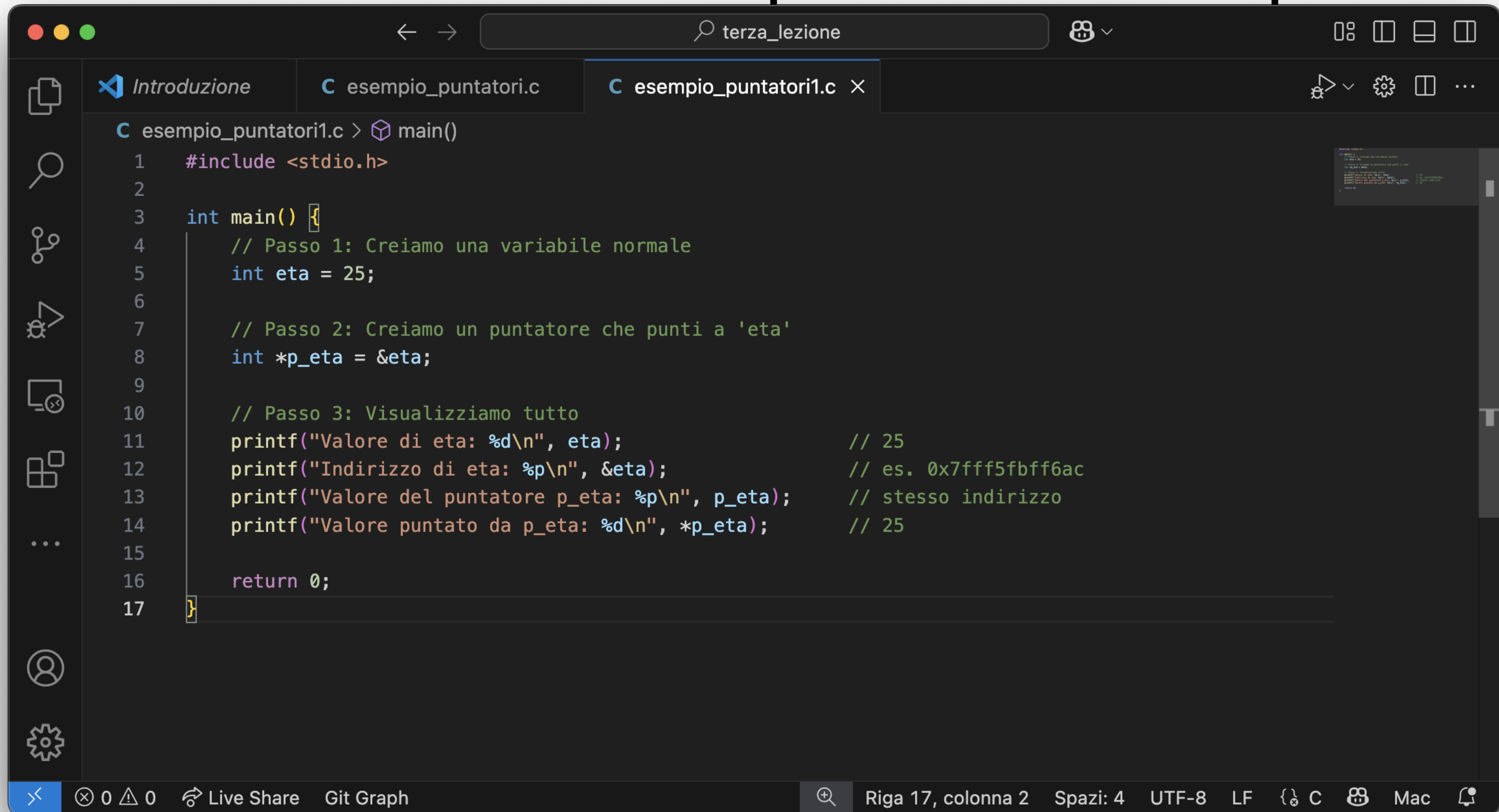


The screenshot shows a code editor with three tabs: 'Introduzione', 'esempio_puntatori.c', and 'esempio_puntatori1.c'. The active tab is 'esempio_puntatori.c'. The code is written in C and demonstrates basic pointer operations. It includes headers for `stdlib.h` and `stdio.h`. The `main` function declares an integer `numero` with the value 42. It then prints the value of `numero` and its memory address using `printf`. Next, it declares a pointer `puntatore` and assigns it the address of `numero`. It then prints the address stored in `puntatore` and the value at that address (which is 42). The program ends with `return 0;`.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main () {
5
6      // operatore address of
7      int numero = 42;
8      printf("Il valore di numero è: %d\n", numero);           // Stampa: 42
9      printf("L'indirizzo di numero è: %p\n", &numero);        // Stampa: Indirizzo
10
11     // operatore dereference
12     int *puntatore = &numero;
13     printf("Il puntatore contiene l'indirizzo: %p\n", puntatore); // Indirizzo
14     printf("All'indirizzo c'è il valore: %d\n", *puntatore);    // 42
15
16     return 0;
17 }
```

The editor interface includes a sidebar with icons for file explorer, search, source control, and other tools. The bottom status bar shows 'Riga 1, colonna 20', 'Spazi: 4', 'UTF-8', 'LF', and other editor settings.

Concetto base dei puntatori: esempi



The screenshot shows a Visual Studio Code editor window with the title bar 'terza_lezione'. The editor has three tabs: 'Introduzione', 'esempio_puntatori.c', and 'esempio_puntatori1.c'. The active tab is 'esempio_puntatori1.c', which contains the following C code:

```
C esempio_puntatori1.c > main()
1  #include <stdio.h>
2
3  int main() {
4      // Passo 1: Creiamo una variabile normale
5      int eta = 25;
6
7      // Passo 2: Creiamo un puntatore che punti a 'eta'
8      int *p_eta = &eta;
9
10     // Passo 3: Visualizziamo tutto
11     printf("Valore di eta: %d\n", eta);           // 25
12     printf("Indirizzo di eta: %p\n", &eta);       // es. 0x7fff5fbff6ac
13     printf("Valore del puntatore p_eta: %p\n", p_eta); // stesso indirizzo
14     printf("Valore puntato da p_eta: %d\n", *p_eta); // 25
15
16     return 0;
17 }
```

The status bar at the bottom shows 'Riga 17, colonna 2', 'Spazi: 4', 'UTF-8', 'LF', 'C', 'Mac', and a notification icon.

Buone pratiche per l'inizializzazione

```
int x = 100;
```

- Inizializzazione diretta:

```
int* ptr1 = &x;
```

- Dichiarazione e assegnazione separata:

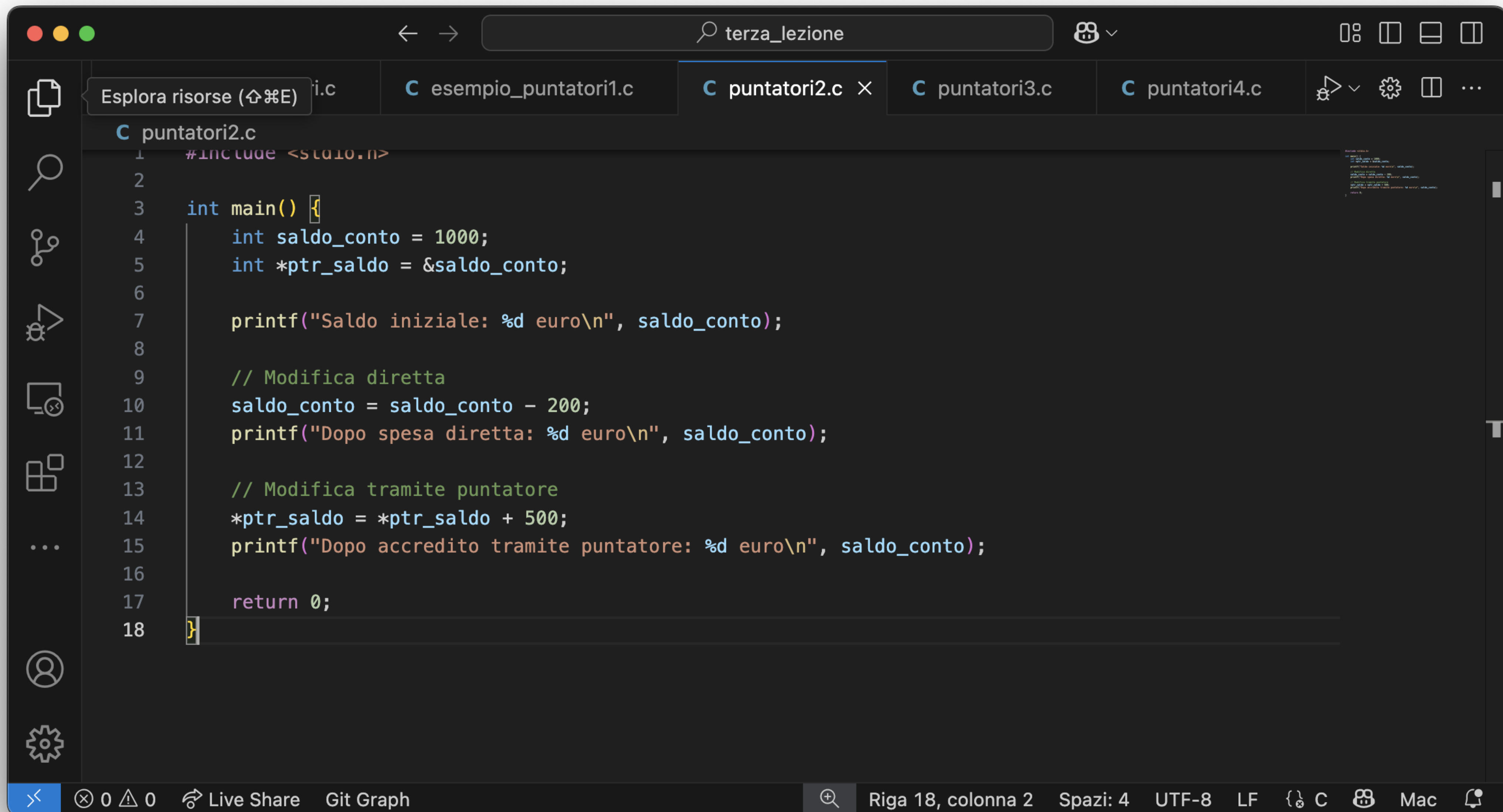
```
int *ptr2;
```

```
ptr2 = &x
```

- Inizializzazione a NULL per sicurezza

```
int *ptr3 = NULL;
```


Buone pratiche per l'inizializzazione

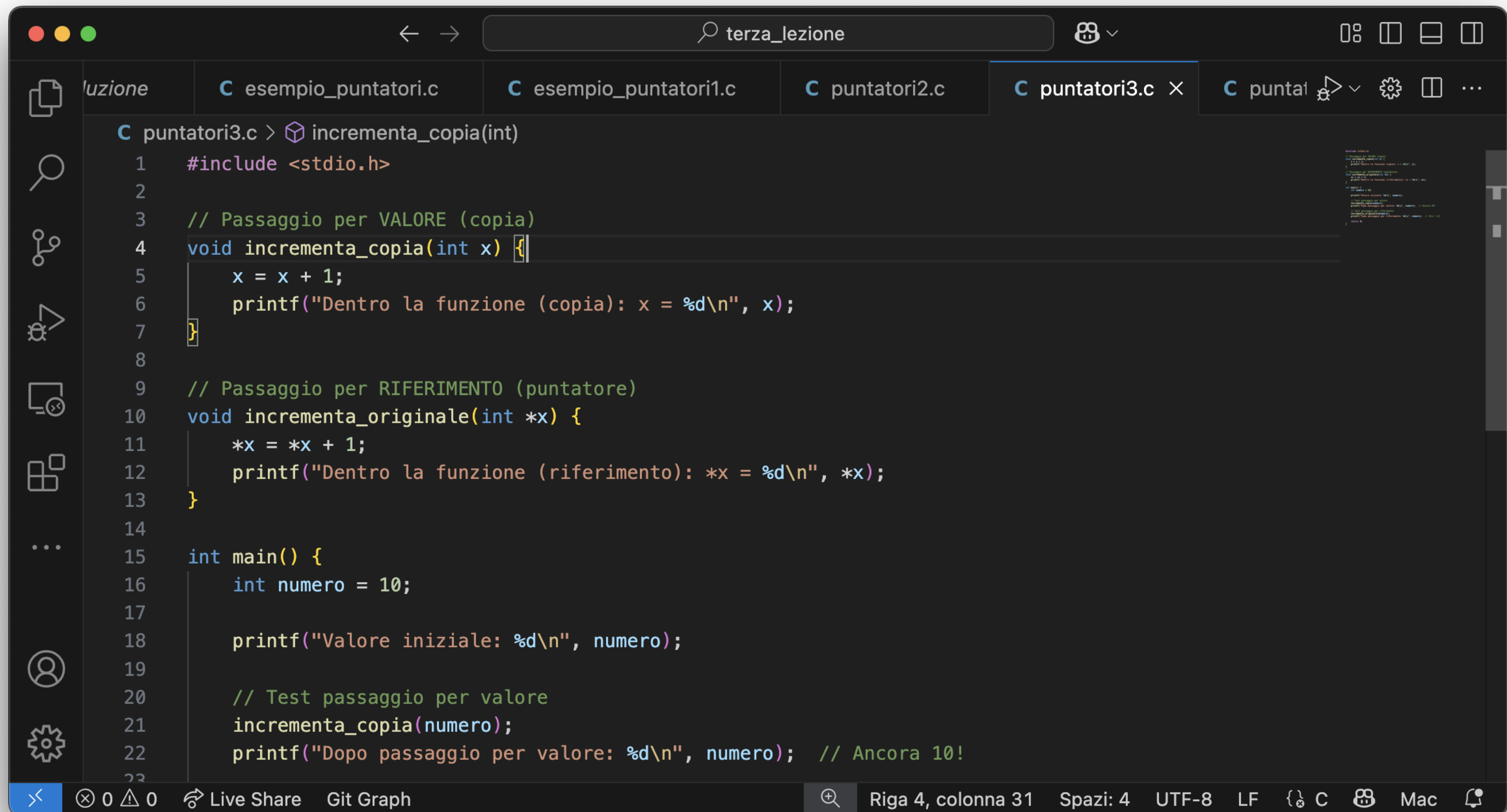


The screenshot shows a code editor with a dark theme. The top bar includes a search icon, the text 'terza_lezione', and window management icons. The editor has several tabs: 'Esplora risorse (⌘E)', 'i.c', 'esempio_puntatori1.c', 'puntatori2.c' (active), 'puntatori3.c', and 'puntatori4.c'. The active tab 'puntatori2.c' displays the following C code:

```
1 #include <stdio.h>
2
3 int main() {
4     int saldo_conto = 1000;
5     int *ptr_saldo = &saldo_conto;
6
7     printf("Saldo iniziale: %d euro\n", saldo_conto);
8
9     // Modifica diretta
10    saldo_conto = saldo_conto - 200;
11    printf("Dopo spesa diretta: %d euro\n", saldo_conto);
12
13    // Modifica tramite puntatore
14    *ptr_saldo = *ptr_saldo + 500;
15    printf("Dopo accredito tramite puntatore: %d euro\n", saldo_conto);
16
17    return 0;
18 }
```

The bottom status bar shows 'Riga 18, colonna 2', 'Spazi: 4', 'UTF-8', 'LF', 'C', 'Mac', and a bell icon.

Buone pratiche per l'inizializzazione



```
terza_lezione
C esempio_puntatori.c
C esempio_puntatori1.c
C puntatori2.c
C puntatori3.c
C puntal

C puntatori3.c > incrementa_copia(int)
1  #include <stdio.h>
2
3  // Passaggio per VALORE (copia)
4  void incrementa_copia(int x) {
5      x = x + 1;
6      printf("Dentro la funzione (copia): x = %d\n", x);
7  }
8
9  // Passaggio per RIFERIMENTO (puntatore)
10 void incrementa_originale(int *x) {
11     *x = *x + 1;
12     printf("Dentro la funzione (riferimento): *x = %d\n", *x);
13 }
14
15 int main() {
16     int numero = 10;
17
18     printf("Valore iniziale: %d\n", numero);
19
20     // Test passaggio per valore
21     incrementa_copia(numero);
22     printf("Dopo passaggio per valore: %d\n", numero); // Ancora 10!
23 }
```

0 0 Live Share Git Graph Riga 4, colonna 31 Spazi: 4 UTF-8 LF { } C Mac



Altri esercizi

Scrivere un programma che utilizza una funzione che prende come parametri due interi e scambia i loro valori.

Puntatori e Array

- **Concetto chiave:** Il nome di un array è un puntatore costante al primo elemento.

Differenze fondamentali

Caratteristica	Array	Puntatore
Memoria	Blocco contiguo fisso	Variabile che contiene indirizzo
Riassegnazione	 Impossibile	 Possibile
Sizeof	Dimensione totale	Dimensione del puntatore
Modifica	Solo contenuto	Può cambiare dove punta

Gestione memoria: Stack vs Heap

Stack

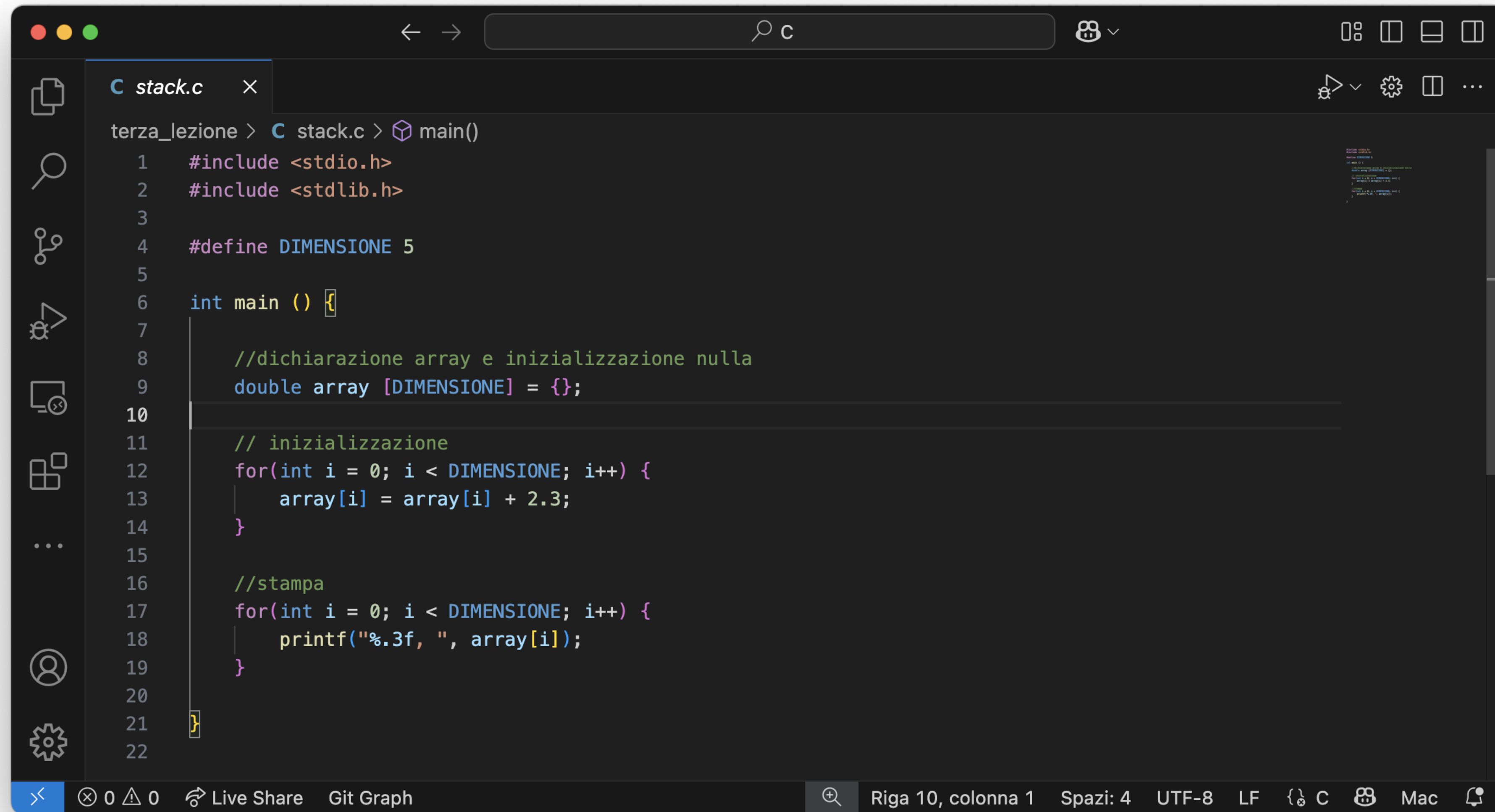
- Memoria gestita **automaticamente** dal compilatore.
- Usato per variabili locali, parametri di funzione, array "automatici".
- **Veloce**, ma con spazio limitato.
- Memoria **liberata automaticamente** alla fine del blocco/funzione.

Gestione memoria: Stack vs Heap

Heap

- Memoria gestita **manualmente** dal programmatore.
- Usato per oggetti grandi o con durata variabile.
- Più **flessibile**, ma più lenta e soggetta a errori (memory leak).
- Necessita di malloc()/calloc() per allocare e free() per liberare.

Gestione Stack

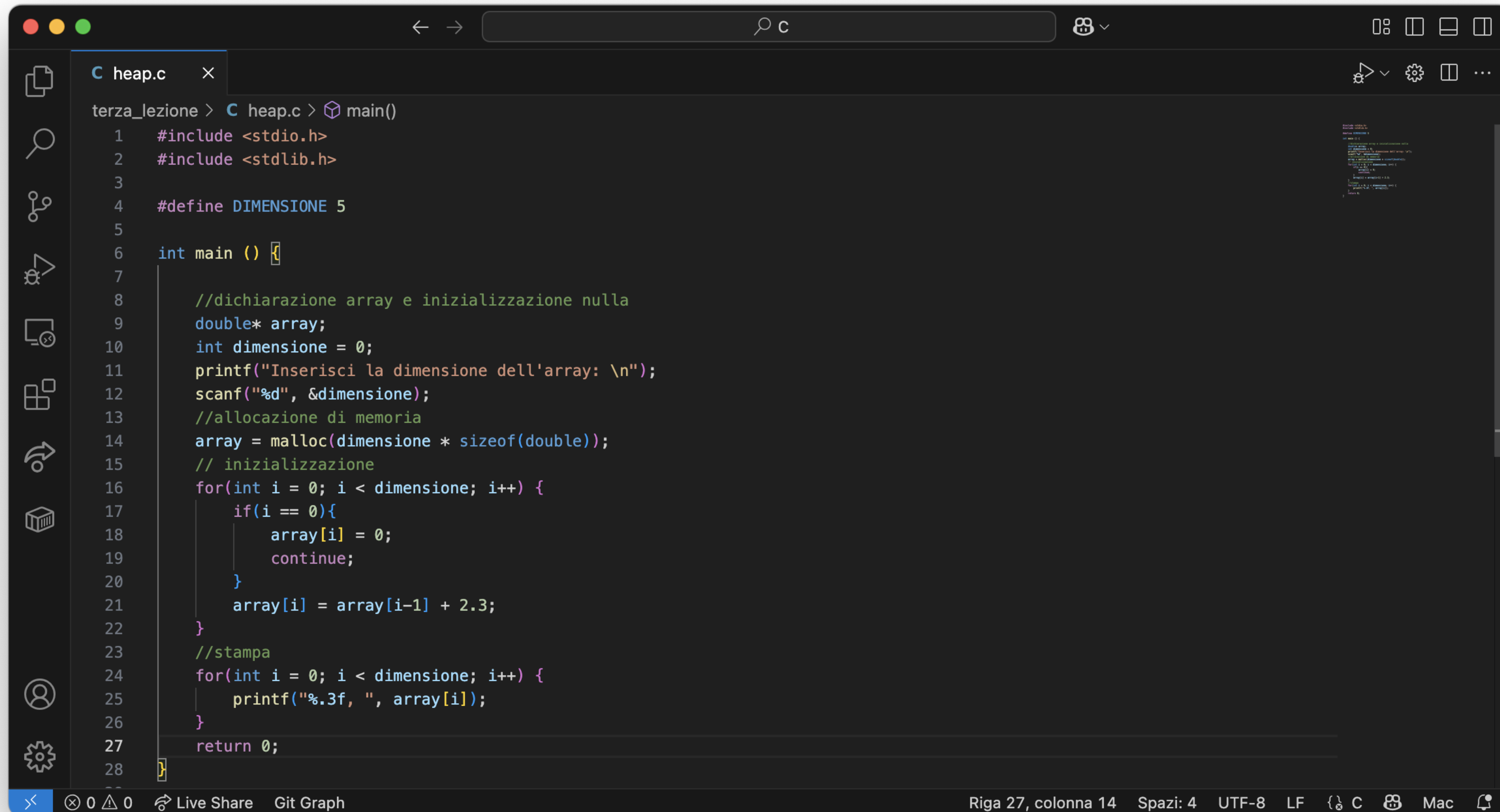


The image shows a code editor window with a dark theme. The title bar at the top has standard macOS window controls (red, yellow, green buttons) and navigation icons (back, forward, search, and window management). The editor has a sidebar on the left with icons for file explorer, search, source control, and other tools. The main editor area displays a C program named `stack.c`. The code is as follows:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define DIMENSIONE 5
5
6  int main () {
7
8      //dichiarazione array e inizializzazione nulla
9      double array [DIMENSIONE] = {};
10
11      // inizializzazione
12      for(int i = 0; i < DIMENSIONE; i++) {
13          array[i] = array[i] + 2.3;
14      }
15
16      //stampa
17      for(int i = 0; i < DIMENSIONE; i++) {
18          printf("%.3f, ", array[i]);
19      }
20
21  }
```

The status bar at the bottom shows various settings: a zoom level of 0, a warning icon, a "Live Share" button, a "Git Graph" button, a magnifying glass icon, the current position "Riga 10, colonna 1", the number of spaces "Spazi: 4", the encoding "UTF-8", the line ending "LF", the language "C", and the operating system "Mac".

Gestione Heap



The image shows a code editor window with a dark theme. The title bar at the top has standard macOS window controls (red, yellow, green buttons) and a search bar containing the letter 'c'. The editor has a sidebar on the left with icons for file explorer, search, source control, and other tools. The main area displays a C program named 'heap.c' with the following code:

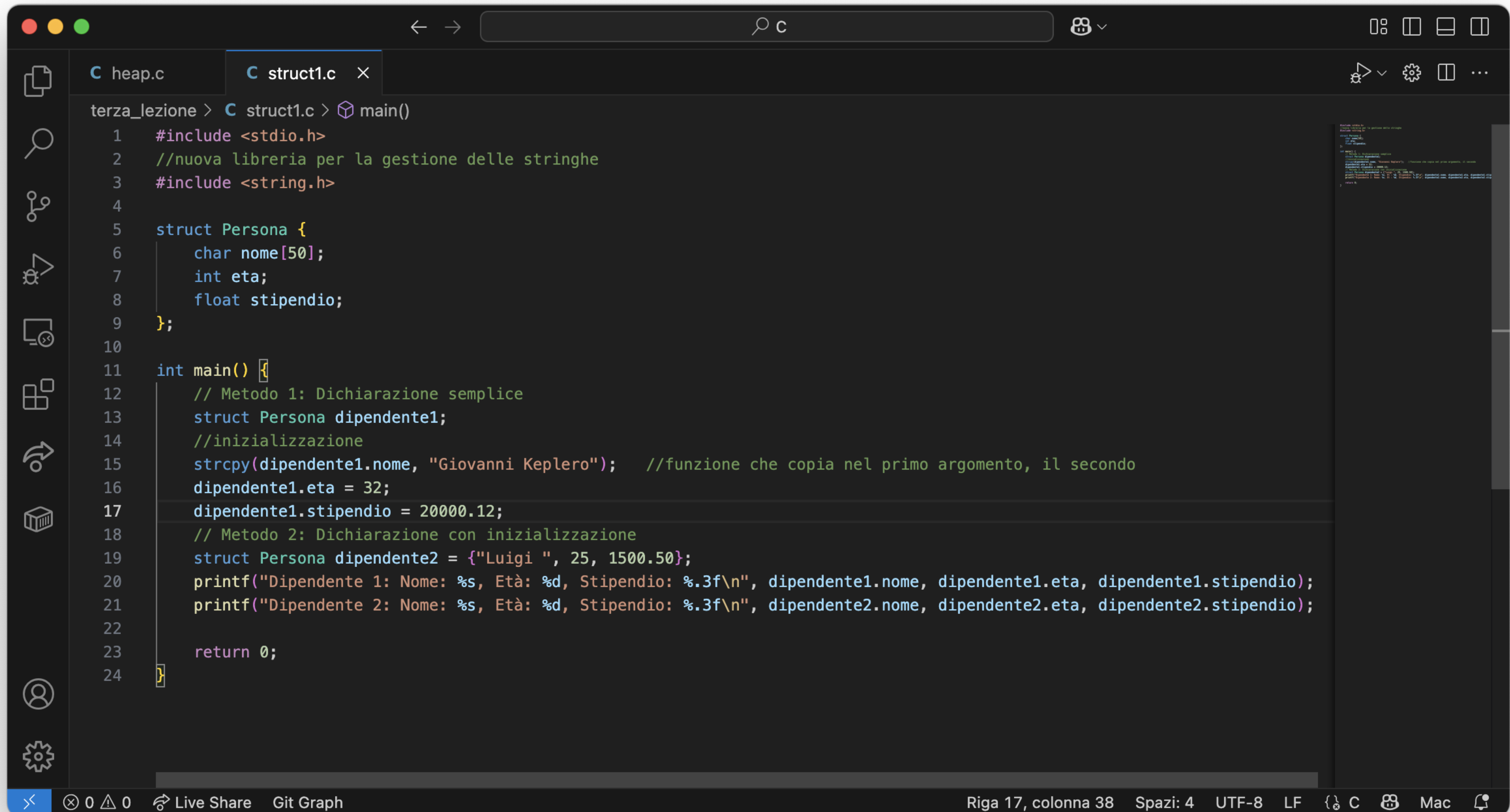
```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define DIMENSIONE 5
5
6  int main () {
7
8      //dichiarazione array e inizializzazione nulla
9      double* array;
10     int dimensione = 0;
11     printf("Inserisci la dimensione dell'array: \n");
12     scanf("%d", &dimensione);
13     //allocazione di memoria
14     array = malloc(dimensione * sizeof(double));
15     // inizializzazione
16     for(int i = 0; i < dimensione; i++) {
17         if(i == 0){
18             array[i] = 0;
19             continue;
20         }
21         array[i] = array[i-1] + 2.3;
22     }
23     //stampa
24     for(int i = 0; i < dimensione; i++) {
25         printf("%.3f, ", array[i]);
26     }
27     return 0;
28 }
```

The status bar at the bottom shows the current cursor position as 'Riga 27, colonna 14', along with other settings like 'Spazi: 4', 'UTF-8', 'LF', and the operating system 'Mac'.

Strutture di dati in C: **struct**

- Concetto fondamentale: una **struct** (struttura) è un tipo di dato definito dall'utente che permette di **aggregare variabili di tipi diversi** sotto un unico nome.
- Vantaggi:
 1. **Organizzazione**: Raggruppano dati correlati
 2. **Leggibilità**: Il codice è più chiaro e comprensibile
 3. **Manutenibilità**: Più facile modificare e aggiornare
 4. **Riusabilità**: Si possono creare tipi personalizzati riutilizzabili
 5. **Modularità**: Facilitano la programmazione a oggetti

Sintassi struct



```
1  #include <stdio.h>
2  //nuova libreria per la gestione delle stringhe
3  #include <string.h>
4
5  struct Persona {
6      char nome[50];
7      int eta;
8      float stipendio;
9  };
10
11 int main() {
12     // Metodo 1: Dichiarazione semplice
13     struct Persona dipendente1;
14     //inizializzazione
15     strcpy(dipendente1.nome, "Giovanni Keplero"); //funzione che copia nel primo argomento, il secondo
16     dipendente1.eta = 32;
17     dipendente1.stipendio = 20000.12;
18     // Metodo 2: Dichiarazione con inizializzazione
19     struct Persona dipendente2 = {"Luigi ", 25, 1500.50};
20     printf("Dipendente 1: Nome: %s, Età: %d, Stipendio: %.3f\n", dipendente1.nome, dipendente1.eta, dipendente1.stipendio);
21     printf("Dipendente 2: Nome: %s, Età: %d, Stipendio: %.3f\n", dipendente2.nome, dipendente2.eta, dipendente2.stipendio);
22
23     return 0;
24 }
```

Riga 17, colonna 38 Spazi: 4 UTF-8 LF C Mac

Operare con le struct

- **Operatore punto (.)**

L'operatore punto viene utilizzato quando abbiamo una **variabile struct diretta**

- **Operatore freccia (->)**

L'operatore freccia viene utilizzato quando abbiamo un **puntatore a struct**

Struct e funzioni

- **Passaggio per valore:**
Modifico solamente una copia della struct
- **Passaggio per riferimento:**
Modifico la struct che sto passando come parametro.

Accesso ai file in C

- Operazione tra le più importanti in C, poiché si riesce ad interagire con il sistema di archiviazione permanente.
- Libreria: `#include <stdio.h>`
- Funzioni principali:
 `FILE *fp`: struct che rappresenta il file;

Aprire un file: `fopen()`

- Punto di partenza per qualsiasi operazione sui file
- 2 parametri: nome del file (con eventuale path), modalità di apertura
- `"r"` (read): apre per sola lettura. Il file deve esistere
- `"w"` (write): apre per scrittura, cancellando il contenuto esistente
- `"a"` (append): apre per aggiungere dati alla fine del file
- `"r+"`: apre per lettura e scrittura (il file deve esistere)
- `"w+"`: apre per lettura e scrittura (crea o sovrascrive il file)

Aprire un file: `fopen()`

- Verificare sempre se `fp != NULL` dopo `fopen()`
- Un tentativo di usare un puntatore `NULL` causerebbe un crash del programma. Gli errori più comuni includono:

File inesistente (modalità "r")

Permessi insufficienti

Percorso non valido

- Un programma robusto deve sempre gestire questi casi e informare l'utente del problema in modo comprensibile

Stampare su un file: `fprintf()`

- La funzione `fprintf()` è l'equivalente di `printf()` per i file.

Parametri: puntatore al file, seguito dalla stringa di formato e dalle variabili.

- È importante ricordare che i dati potrebbero essere bufferizzati, quindi non vengono scritti immediatamente sul disco.

Leggere da un file: `fscanf()`

- La funzione `fscanf()` legge dati formattati da un file. Utile quando i dati nel file hanno una struttura prevedibile.
- Il valore di ritorno di `fscanf()` indica quanti elementi sono stati letti con successo, permettendo di verificare se la lettura è andata a buon fine.

Chiusura un file: `fclose()`

- Ogni file aperto deve essere chiuso con `fclose()`. Questa operazione:
 - Libera le risorse di sistema
 - Garantisce che tutti i dati bufferizzati vengano scritti
 - Evita perdite di memoria
 - Rispetta i limiti del sistema sui file aperti
- Dimenticare di chiudere i file può causare problemi di performance e, nei casi peggiori, perdita di dati.

Esercizi

- **Esercizio: rubrica con struct**

Definire una struct Persona

Chiedere nome ed età a N persone

Salvare su file rubrica.txt usando fprintf

- **Esercizio: Archivio studenti**

Creare un programma che gestisce un archivio studenti con struct, puntatori e salvataggio su file