

Modellare la Realtà (Struct, Padding e Persistenza a Blocchi)



Dispense di Programmazione 1 – Canale O-Z

Prof. Antonino Furnari

Corso di Laurea in Informatica

Università di Catania

Lezione 1

Modellare la Realtà (Struct, Padding e Persistenza a Blocchi)

Nelle lezioni precedenti abbiamo visto come gestire insiemi di dati omogenei tramite gli array e come persistere le informazioni su file di testo e binari. Tuttavia, la realtà che ci circonda è complessa ed eterogenea: un'entità del mondo reale come uno "Studente" non è definibile tramite un singolo numero o una singola stringa, ma possiede un insieme di caratteristiche di tipo diverso (un nome, un numero di matricola intero, una media pesata float).

In questa lezione supereremo la fragilità degli array paralleli introducendo il concetto di **Struttura (struct)**, che permette di aggregare tipi di dato eterogenei in un unico record logico. Esploreremo la fisica della memoria analizzando come vengono passate le struct alle funzioni (per valore vs riferimento con l'operatore freccia ->) e svelando le bolle di memoria invisibili create dal compilatore per l'efficienza dei bus hardware (**Padding e Allineamento**). Infine, vedremo come le struct rivoluzionano la persistenza binaria consentendo la scrittura e la lettura massiva a blocchi e l'accesso diretto istantaneo tramite la navigazione con `fseek`.

1.1 La Fine degli Array Paralleli e l'Incapsulamento

1.1.1 Il Problema: I limiti dei tipi predefiniti

Fino ad ora, per modellare una tabella di elementi eterogenei, siamo stati costretti ad adottare la tecnica degli **array paralleli**. Se ad esempio desideriamo gestire un elenco di esami universitari composto da materia, CFU e voto, dichiariamo tre array distinti che condividono implicitamente lo stesso indice logico:

```
1 char materie[100][50]; // Nomi degli esami
2 int cfu[100];           // Crediti di ciascun esame
3 int voti[100];          // Voto conseguito
```

Sebbene questa soluzione funzioni, essa presenta gravissimi problemi di manutenibilità e sicurezza dei dati. Consideriamo il caso in cui si voglia ordinare gli esami in base al voto usando l'algoritmo Bubble Sort. Per non mescolare i dati, siamo costretti a effettuare lo scambio (swap) per ciascuno dei tre array paralleli coordinando manualmente gli indici:

```
1 // Scambio del voto
2 int temp_voto = voti[i];
```

```
3 voti[i] = voti[j];
4 voti[j] = temp_voto;
5
6 // Scambio dei CFU
7 int temp_cfu = cfu[i];
8 cfu[i] = cfu[j];
9 cfu[j] = temp_cfu;
10
11 // Scambio della stringa materia
12 char temp_mat[50];
13 strcpy(temp_mat, materie[i]);
14 strcpy(materie[i], materie[j]);
15 strcpy(materie[j], temp_mat);
```

Se il programmatore dimentica di scambiare anche solo una delle variabili (ad esempio omettendo lo swap del nome della materia), l'associazione logica viene irrimediabilmente distrutta, attribuendo ad esempio i CFU di Matematica all'esame di Storia. Il C mette a disposizione tipi primitivi (int, float, char), ma non possiede un tipo nativo denominato Esame. Diventa quindi necessario creare un nostro tipo di dato personalizzato.

1.1.2 La Creazione di un Nuovo Tipo: La struct

In C, la parola chiave struct (Struttura) permette di raggruppare variabili di tipo diverso all'interno di un unico contenitore logico e fisico. Ciascuna variabile interna alla struttura viene chiamata **campo** o **membro**.

Definizione di una Struct

La definizione descrive il modello ("progetto") della struttura senza allocare spazio in memoria RAM:

```
1 struct materia_s {
2     char nome[50];
3     char codice[10];
4     int cfu;
5     int voto;
6 }; // ATTENZIONE: Il punto e virgola finale ? obbligatorio!
```

Per istanziare una variabile di questo tipo e accedere ai suoi membri, utilizziamo l'**operatore punto (.)**:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5     // Dobbiamo ripetere la parola chiave 'struct'
6     struct materia_s analisi;
7
8     // Accesso e scrittura nei campi (i campi array si riempiono con
9     ↪ strcpy)
10    strcpy(analisi.nome, "Analisi");
11    strcpy(analisi.codice, "AN29345");
```

```
11     analisi.cfu = 9;
12     analisi.voto = 28;
13
14     printf("Voto in analisi: %d\n", analisi.voto);
15     return 0;
16 }
```

Definizione Elegante con typedef

Dover anteporre costantemente `struct` davanti al nome del tipo risulta ridondante. Possiamo utilizzare la keyword `typedef` per creare un alias (sinonimo) semplificato:

```
1  typedef struct {
2      char nome[50];
3      char codice[10];
4      int cfu;
5      int voto;
6  } Esame; // "Esame" diventa a tutti gli effetti un nuovo tipo di dato!
```

È anche possibile scorporare la definizione della struct e il typedef:

```
1  struct esame {
2      char nome[50];
3      char codice[10];
4      int cfu;
5      int voto;
6  };
7
8  typedef struct esame Esame; // "Esame" diventa a tutti gli effetti un
    ↪ nuovo tipo di dato!
```

Nota

Nota Per convenzione stilistica diffusa, i tipi personalizzati definiti dal programmatore si scrivono adottando la nomenclatura **PascalCase** (es. `Esame` o `StudenteIngegneria`), distinguendoli a colpo d'occhio dai tipi primitivi nativi (`int`, `float`, `char`) scritti interamente in minuscolo.

1.1.3 Inizializzazione e Assegnazione Magica

Inizializzazione a Graffe

Una struct può essere inizializzata sinteticamente all'atto della dichiarazione usando una lista di valori racchiusi tra parentesi graffe `{...}`, rispettando rigorosamente l'ordine di dichiarazione dei campi:

```
1  // Corretto: {nome, codice, cfu, voto}
2  Esame prog = {"Programmazione 1", "INF101", 12, 30};
3
```

```
4 // Errato o disallineato:  
5 Esame errore = {"Fisica", "FIS01", 18, 6}; // Assegna 18 a cfu e 6 a voto  
    ↪ !
```

Copia per Assegnazione

A differenza degli array tradizionali (che non possono essere duplicati tramite l'operatore =), le struct godono del privilegio della **copia diretta per assegnamento**. Il compilatore effettua autonomamente un dump perfetto bit-a-bit dell'intera area di memoria occupata dalla struct:

```
1 Esame backup = prog; // Copia immediata di tutti i campi in RAM
```

1.2 Puntatori a Struct e Fisica della Memoria

1.2.1 Passaggio per Valore vs Riferimento: L'Operatore Freccia (->)

Sommando lo spazio occupato dai campi di Esame ricaviamo la sua dimensione: 50 (nome) + 10 (codice) + 4 (cfu) + 4 (voto) = 68 byte.

Passaggio per Valore (Inefficiente)

Se passiamo una struct a una funzione per valore, l'intera struttura da 68 byte viene interamente duplicata sullo Stack all'interno del record di attivazione della funzione chiamata:

```
1 void stampa_copia(Esame e) {  
2     printf("%s: %d\n", e.nome, e.voto);  
3 }
```

Qualora la struct contenesse array di dimensioni maggiori, questa operazione comporterebbe un pesante spreco di memoria Stack e cicli di clock per effettuare la fotocopia dei byte.

Passaggio per Riferimento (Efficiente)

Per ottimizzare le prestazioni, passiamo unicamente l'indirizzo di memoria della struct (8 byte su sistemi a 64-bit). Per accedere ai campi della struct puntata abbiamo due opzioni sintattiche:

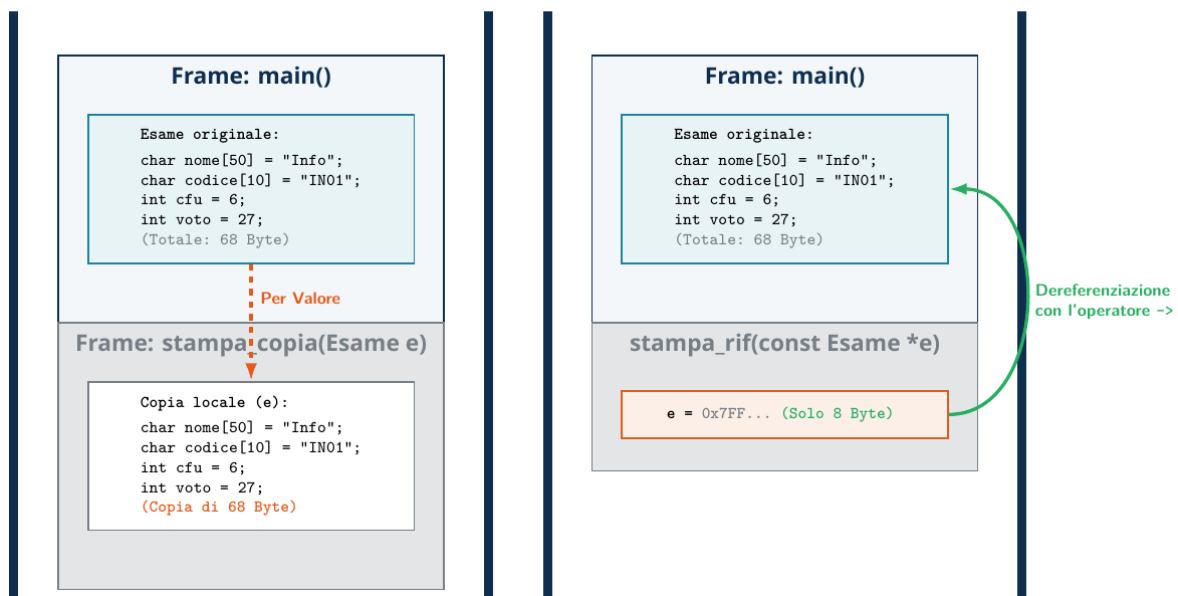
1. **Sintassi Esplicita**: Dereferenziare il puntatore e usare l'operatore punto. Richiede le parentesi a causa della precedenza degli operatori: `(*e).nome`.
2. **Sintassi Semplificata (Operatore Freccia ->)**: Zucchero sintattico che sostituisce la combinazione di dereferenziazione e punto: `e->nome`.

```

1 #include <stdio.h>
2
3 void stampa_rif(const Esame *e) {
4     // Accesso tramite operatore freccia ->
5     printf("%s (%s) - CFU: %d, Voto: %d\n", e->nome, e->codice, e->cfu, e
6     ↪ ->voto);
7 }
8
9 int main(void) {
10     Esame info = {"Informatica", "INF01", 6, 27};
11     stampa_rif(&info); // Passiamo l'indirizzo
12     return 0;

```

Scenario A: Passaggio per Valore Scenario B: Passaggio per Riferimento



Suggerimento

Regola Pratica Se possiedi direttamente la variabile struct (l'oggetto in sé) utilizza l'operatore punto `..`. Se possiedi un puntatore alla struct, utilizza l'operatore freccia `->`.

1.2.2 Sotto il Cofano: Padding e Allineamento (Data Alignment)

Consideriamo l'esperimento di verificare lo spazio occupato in RAM da due struct apparentemente identiche nei campi, ma disposte in ordine differente:

```

1 typedef struct {
2     char c;      // 1 byte
3     double d;    // 8 byte
4     int i;       // 4 byte
5 } ProvaA;
6

```

```
7 typedef struct {  
8     double d;    // 8 byte  
9     int i;       // 4 byte  
10    char c;      // 1 byte  
11 } ProvaB;
```

A livello algebrico, lo spazio occupato dai campi dovrebbe essere pari a $1 + 8 + 4 = 13$ byte. Tuttavia, compilando ed eseguendo `sizeof` su una macchina a 64-bit ricaviamo risultati sorprendenti:

- `sizeof(ProvaA)` restituisce **24 byte**.
- `sizeof(ProvaB)` restituisce **16 byte**.

Il Padding

Questo fenomeno è causato dal **Padding** (imbottitura) introdotto dal compilatore. I moderni processori leggono la memoria RAM a blocchi allineati (solitamente di 8 byte su sistemi a 64-bit). Se una variabile multi-byte (come un `double` da 8 byte) venisse memorizzata a cavallo tra due blocchi fisici di memoria, la CPU sarebbe costretta a eseguire due letture RAM distinte per ricomporre il dato, rallentando l'esecuzione.

Per ottimizzare gli accessi hardware, il compilatore applica le seguenti regole di allineamento:

1. Ogni tipo di dato nativo deve iniziare a un indirizzo di memoria che sia multiplo della sua stessa dimensione (es. un `double` allineato a indirizzi multipli di 8, un `int` a multipli di 4).
2. Il compilatore inietta byte vuoti (padding) invisibili per ripristinare l'allineamento ottimale.

Analizziamo il posizionamento dei byte per le due strutture:

- **ProvaA (char c, double d, int i):**
 - Il carattere `c` occupa la posizione 0.
 - Il campo successivo `d` (`double`) richiede allineamento a 8 byte. Il compilatore inserisce **7 byte di padding** vuoti.
 - Il `double` viene allocato dall'offset 8 all'offset 15.
 - L'intero `i` viene allocato dall'offset 16 al 19 (allineamento a 4 rispettato).
 - La dimensione totale della struct deve essere multipla del campo più grande (`double = 8`). Vengono iniettati altri **4 byte di padding** finali, portando il totale a **24 byte** (offset 0-23).
- **ProvaB (double d, int i, char c):**
 - Il `double` viene allocato al blocco iniziale dall'offset 0 al 7.
 - L'intero `i` occupa la posizione dall'offset 8 all'11 (allineamento a 4 rispettato).
 - Il carattere `c` occupa l'offset 12.
 - Vengono iniettati **3 byte di padding** finali per allineare l'intera struttura al multiplo di 8, portando il totale a **16 byte** (offset 0-15).

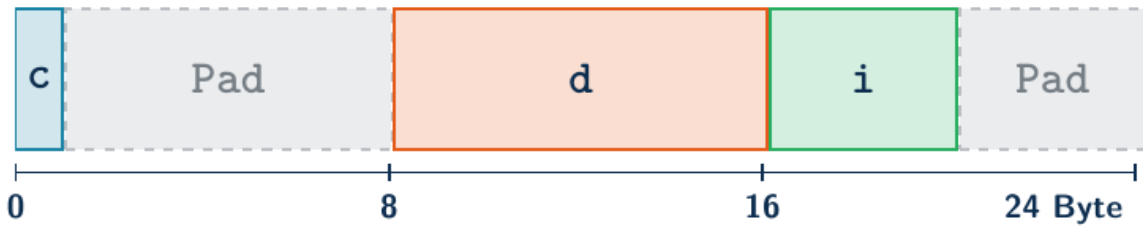
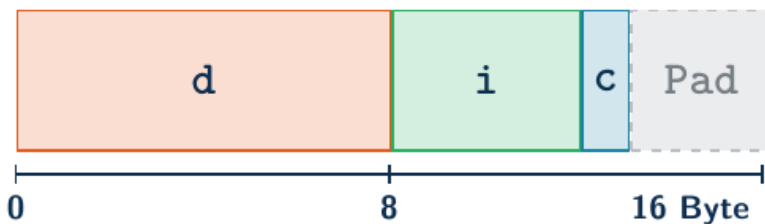
struct ProvaA (Dimensione Totale = 24 Byte)**struct ProvaB (Dimensione Totale = 16 Byte)**

Figura 1.1: Padding e Allineamento

Attenzione

Attenzione Nei normali esercizi di laboratorio non è richiesto riordinare ossessivamente le struct per risparmiare memoria. È tuttavia fondamentale, in contesti a basso livello (sistemi operativi, sistemi embedded e programmazione di rete), conoscere l'esistenza del padding per prevenire anomalie e sprechi di RAM.

1.2.3 Enumerazioni (enum): Un tocco di leggibilità

Spesso all'interno delle struct è necessario salvare lo stato di un oggetto (es. un esame può essere "In Corso", "Superato" o "Bocciato"). Utilizzare numeri interi generici (0, 1, 2) rende il codice cryptico ed esposto a errori.

Il C permette di creare delle **Enumerazioni (enum)**, che associano identificatori mnemonici a costanti intere incrementali (di default partendo da 0):

```

1  #include <stdio.h>
2
3  // Definizione dell'enumerazione
4  typedef enum {
5      IN_CORSO,    // 0
6      SUPERATO,    // 1
7      BOCCIATO     // 2
8  } StatoEsame;
9
10 // Utilizzo all'interno di una Struct
11 typedef struct {
12     char materia[50];
13     StatoEsame stato;

```

```
14 } EsameCarriera;
15
16 int main(void) {
17     EsameCarriera prog = {"Programmazione 1", IN_CORSO};
18
19     if (prog.stato == IN_CORSO) {
20         printf("Stai ancora studiando %s. Coraggio!\n", prog.materia);
21     }
22
23     prog.stato = SUPERATO; // Aggiornamento dello stato
24     return 0;
25 }
```

1.3 Il Trionfo di fread, fwrite ed fseek

1.3.1 Unire la Teoria: Struct e File Binari

Nella Lezione 12 abbiamo notato la difficoltà di gestire file binari composti da stringhe e dati a lunghezza variabile: senza una dimensione nota e costante, risulta impossibile calcolare l'offset fisico per saltare a un record specifico.

La struct rappresenta il mattoncino fondamentale per risolvere questo problema: poiché la dimensione di una struct (ad esempio `sizeof(Esime)`) è fissa e nota a tempo di compilazione, un file binario composto da una successione di struct si comporterà fisicamente come un array sequenziale su disco.

1.3.2 Il Salvataggio Istantaneo dell'intero Database

Grazie alla contiguità fisica della memoria RAM, possiamo salvare un intero array di struct su disco con una singola invocazione di `fwrite`, riversando i blocchi in un unico colpo:

```
1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int cfu;
6      int voto;
7  } Esame;
8
9  int main(void) {
10     // Array di struct in RAM
11     Esame libretto[3] = {
12         {"Analisi 1", 9, 28},
13         {"Programmazione 1", 12, 30},
14         {"Fisica 1", 6, 25}
15     };
16
17     FILE *fp = fopen("database.bin", "wb");
18     if (fp != NULL) {
19         // Scrittura in blocco: (sizeof(Esime) * 3) byte trasferiti sul
20         ↪ disco
21         fwrite(libretto, sizeof(Esime), 3, fp);
22     }
```

```

21         fclose(fp);
22         printf("Database salvato con successo!\n");
23     }
24     return 0;
25 }

```

1.3.3 Accesso Diretto

Grazie alla struttura a blocchi costanti, la testina di lettura può saltare istantaneamente all'inizio del record N -esimo senza dover scansionare i record precedenti. L'offset esatto in byte a partire dall'inizio del file (`SEEK\{\}_SET`) è calcolato con la formula:

$$\text{Offset} = \text{indice} \times \text{sizeof}(\text{Struct})$$

```

1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int cfu;
6      int voto;
7  } Esame;
8
9  int main(void) {
10     FILE *fp = fopen("database.bin", "rb");
11     if (fp == NULL) {
12         return 1;
13     }
14
15     Esame cercato;
16     int indice_desiderato = 1; // Vogliamo leggere il secondo esame (
    ↪ indice 1)
17
18     // SALTO MATEMATICO
19     fseek(fp, indice_desiderato * sizeof(Esame), SEEK_SET);
20
21     // Lettura isolata dell'unico record posizionato sotto la testina
22     fread(&cercato, sizeof(Esame), 1, fp);
23     fclose(fp);
24
25     printf("Esame estratto al volo: %s, Voto: %d\n", cercato.nome,
    ↪ cercato.voto);
26     return 0;
27 }

```

Nota: il diagramma riporta blocchi da 68 byte (dimensione della `Esame` con il campo codice); con la `Esame` di questa sezione, priva di codice, `sizeof(Esame)` vale 60 byte su x86-64 (50 + 2 di padding + 4 + 4), quindi il record di indice 1 inizia all'offset 60.

File Binario database.bin (Blocchi da 68 Byte)

```
fseek(fp, 1 * sizeof(Esame), SEEK_SET);
```

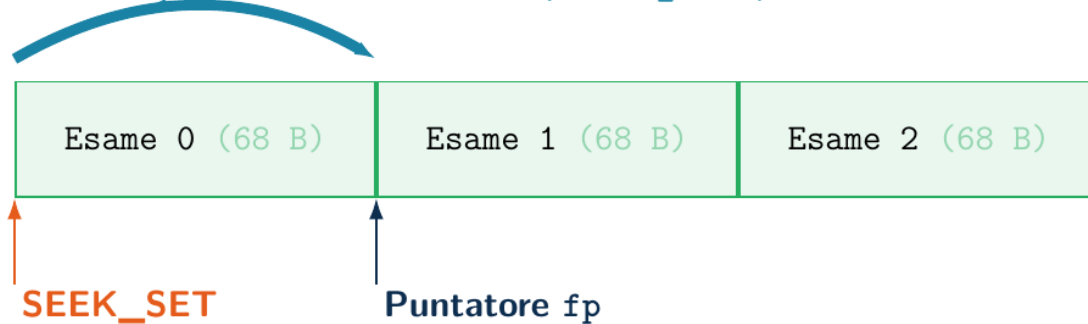


Figura 1.2: I/O Diretto con Array di Struct su File Binario

1.4 Quiz di Auto-valutazione

Quiz 1

Domanda

1. Quale delle seguenti affermazioni sulla definizione di una struct in C è corretta?

- A) La definizione alloca immediatamente spazio nella memoria RAM.
- B) È obbligatorio includere il punto e virgola ; dopo la chiusura della parentesi graffa.
- C) Permette di raggruppare unicamente elementi dello stesso tipo.

Risposta corretta: B
Spiegazione: La definizione di una struct funge solo da modello ("stampo") per il compilatore e non alloca memoria (che viene allocata solo all'istanza delle variabili). La sintassi del linguaggio C impone tassativamente il punto e virgola ; al termine della definizione della struttura.

Soluzione

Quiz 2

Domanda

2. Cosa accade quando si esegue l'operazione di assegnamento structA = structB in C?

- A) Viene copiato solo l'indirizzo di memoria del primo elemento (come per gli array).
- B) Il compilatore restituisce un errore, poiché le struct non si possono copiare con =.

- C) Viene effettuata una copia fisica bit-a-bit di tutti i campi in una nuova porzione di RAM.

Soluzione
Risposta corretta: C
Spiegazione: A differenza degli array, le struct in C sono trattate come valori atomici per l'assegnamento. Il compilatore genera codice per copiare l'intera porzione di memoria occupata dalla struct sorgente in quella di destinazione.

Quiz 3

Domanda

3. Data la struct `typedef struct \{\} { int x; char y; \{\}} Punto; Punto p; Punto *ptr = \{\}&p;`, qual è la sintassi corretta per accedere al campo y tramite il puntatore?

- A) `ptr->y`
- B) `*ptr.y`
- C) `ptr.y`

Soluzione
Risposta corretta: A
Spiegazione: L'operatore freccia `->` è lo zucchero sintattico utilizzato per accedere ai membri di una struttura puntata da un puntatore. Scrivere `ptr->y` equivale a scrivere `(*ptr).y`. La scrittura `*ptr.y` senza parentesi è errata poiché l'operatore punto `.` ha una precedenza maggiore rispetto all'operatore di dereferenziazione `*`.

Quiz 4

Domanda

4. Perché `sizeof(struct \{\} { char c; double d; \{\}})` restituisce solitamente 16 anziché 9?

- A) A causa dell'allineamento dei dati (padding) effettuato dal compilatore per l'efficienza della CPU.
- B) Perché il compilatore C riserva byte aggiuntivi per i nomi delle variabili.
- C) Perché il tipo `char` viene convertito automaticamente in `double`.

Soluzione
Risposta corretta: A
Spiegazione: Le CPU a 64-bit accedono in modo efficiente a dati situati ad indirizzi multipli di 8. Per evitare accessi disallineati, il compilatore inserisce 7 byte di padding vuoti subito dopo il char da 1 byte, allineando il double (8 byte) ai blocchi hardware della CPU.

Quiz 5

Domanda

5. Cos'è una enum in C?

- A) Un tipo di dato che alloca dinamicamente spazio nello Stack.
- B) Un insieme di costanti intere con nomi mnemonici per migliorare la leggibilità del codice.
- C) Un file binario in cui memorizzare le struct.

Soluzione
Risposta corretta: B
Spiegazione: Le enumerazioni consentono di mappare nomi mnemonici a valori interi sequenziali (es. 0, 1, 2). Sotto il cofano, le variabili di tipo enum sono gestite dal compilatore come normali numeri interi, ma prevenengono l'uso di "numeri magici" sparsi nel codice.

Quiz 6

Domanda

6. Se definiamo `typedef struct \{\}\ char nome[20]; \{\}\ Persona;`, quanti byte occuperà un puntatore `Persona *p` in un sistema a 64-bit?

- A) 20 byte.
- B) 8 byte.
- C) 4 byte.

Soluzione
Risposta corretta: B
Spiegazione: La dimensione di un puntatore dipende esclusivamente dall'architettura del processore (indirizzamento a 64-bit = 8 byte, 32-bit = 4 byte), indipendentemente dal tipo di dato o dalla dimensione della struttura a cui punta.

Quiz 7

Domanda

7. Qual è la formula matematica per calcolare l'offset in byte per raggiungere il record N -esimo (con indice 0-based) all'interno di un file binario strutturato con struct di tipo Record?

- A) $N * \text{sizeof}(\text{Record})$
- B) $(N - 1) * \text{sizeof}(\text{Record})$
- C) $N * \text{sizeof}(\text{Record}) + 1$

Soluzione
Risposta corretta: A
Spiegazione: Avendo i record una dimensione fissa costante, il record all'indice N si trova esattamente dopo che si sono saltati N record interi di dimensione $\text{sizeof}(\text{Record})$. Per l'indice 0, lo spostamento sarà di 0 byte (l'inizio del file).

Quiz 8

Domanda

8. Per quale motivo gli array paralleli sono considerati una pratica fragile in programmazione C?

- A) Perché il compilatore impedisce la compilazione di più di due array contemporaneamente.
- B) Perché richiedono la gestione sincrona manuale degli indici; se si effettua uno swap su un solo array, i dati degli altri si disallineano corrompendo l'informazione.
- C) Perché consumano il doppio dello spazio in memoria rispetto a una struct equivalente.

Soluzione
Risposta corretta: B
Spiegazione: Gli array paralleli non garantiscono l'integrità del dato a livello di compilatore. Qualsiasi operazione di ordinamento o inserimento costringe a manipolare simultaneamente tutti gli array; un singolo errore disallinea irrimediabilmente i record.

Quiz 9

Domanda

9. Con quale costante in `fseek` si indica che lo spostamento deve avvenire a partire dalla posizione corrente del cursore?

- A) `SEEK\{\}_SET`

- B) SEEK\{}_END
- C) SEEK\{}_CUR

Soluzione
Risposta corretta: C
Spiegazione: In fseek, SEEK\{}_SET indica l'inizio del file, SEEK\{}_END indica la fine del file, e SEEK\{}_CUR indica la posizione corrente in cui si trova la testina di lettura/scrittura.

Quiz 10

Domanda

10. Qual è l'effetto di omettere il punto e virgola ; alla fine della definizione di una struct?

- A) Nessuno, il compilatore lo aggiunge automaticamente.
- B) Un errore di compilazione ("syntax error").
- C) La struct viene allocata nello Heap invece che nello Stack.

Soluzione
Risposta corretta: B
Spiegazione: La sintassi formale del linguaggio C richiede obbligatoriamente il punto e virgola al termine della definizione di una struttura. In caso contrario, il compilatore genera un errore bloccante.

1.5 Esercizi Pratici

Esercizio 1: Definizione e Stampa

Domanda

(stampa\{}_punto.c)

Si definisca una struct Punto3D avente tre coordinate reali (x, y, z). Si scriva una funzione stampa\{}_punto che accetti la struct per valore e ne stampi le coordinate nel formato (x, y, z).

Spiegazione dell'algoritmo: Definiamo il tipo Punto3D contenente tre variabili double. La funzione stampa_punto\{\}_punto riceve una copia locale della struttura e stampa i singoli campi accedendovi con l'operatore punto (.).

```

1  #include <stdio.h>
2
3  // Definizione della struct
4  typedef struct {
5      double x;
6      double y;
7      double z;
8  } Punto3D;
9
10 // Funzione che accetta la struct per valore
11 void stampa_punto(Punto3D p) {
12     printf("%.2f, %.2f, %.2f\n", p.x, p.y, p.z);
13 }
14
15 int main(void) {
16     Punto3D origine = {0.0, 0.0, 0.0};
17     Punto3D p1 = {3.5, -2.0, 7.8};
18     printf("Origine: ");
19     stampa_punto(origine);
20
21     printf("Punto 1: ");
22     stampa_punto(p1);
23
24     return 0;
25 }
26

```

Soluzione

Esercizio 2: Modifica per Riferimento

Domanda

(trasla\{\}_punto.c)

Si scriva una funzione trasla\{\}_punto che accetti un puntatore a Punto3D e tre valori reali dx, dy, dz per traslare il punto sommandoli alle coordinate originali.

Spiegazione dell'algoritmo: Passando il puntatore Punto3D *p, la funzione può accedere e modificare le coordinate reali del punto originale allocato nel frame del main tramite l'operatore freccia ->, evitando di creare copie ridondanti in memoria Stack.

```

1  #include <stdio.h>
2
3  typedef struct {
4      double x;
5      double y;
6      double z;
7  } Punto3D;
8
9  // Modifica tramite puntatore e operatore freccia ->
10 void trasla_punto(Punto3D *p, double dx, double dy, double dz) {
11     p->x += dx;
12     p->y += dy;
13     p->z += dz;
14 }
15
16 void stampa_punto(Punto3D p) {
17     printf("%.2f, %.2f, %.2f\n", p.x, p.y, p.z);
18 }
19
20 int main(void) {
21     Punto3D p = {1.0, 2.0, 3.0};
22
23     printf("Punto originale: ");
24     stampa_punto(p);
25
26     // Passiamo l'indirizzo di memoria di p
27     trasla_punto(&p, 1.5, -0.5, 2.0);
28
29     printf("Punto traslato: ");
30     stampa_punto(p);
31
32     return 0;
33 }

```

Soluzione

Esercizio 3: Calcolo Statistiche su Array di Struct

Domanda

(statistiche\{}_studenti.c)

Si definisca una struct `Studente` contenente il nome (stringa max 50 caratteri) e la media dei voti (valore float). Si scriva un programma che istanzi un array di 3 studenti, legga i dati da tastiera e visualizzi la media complessiva della classe.

Spiegazione dell'algoritmo: Creiamo un array in cui ogni cella contiene una struct studente. Nel ciclo for accediamo ai campi di ciascun elemento combinando l'indice dell'array e l'operatore punto: `classe[i].nome` e `classe[i].media` (notare l'operatore indirizzato `&` posizionato davanti al membro per leggerlo con `scanf`).

```

1  #include <stdio.h>
2  #include <string.h>
3
4  typedef struct {
5      char nome[50];
6      float media;
7  } Studente;
8
9  int main(void) {
10     Studente classe[3];
11     float somma = 0.0;
12
13     for (int i = 0; i < 3; i++) {
14         printf("Inserisci il nome dello studente %d: ", i + 1);
15         // Lettura stringa con gestione del fine riga
16         scanf("%49s", classe[i].nome);
17
18         printf("Inserisci la media dello studente %d: ", i + 1);
19         scanf("%f", &classe[i].media);
20
21         somma += classe[i].media;
22     }
23
24     float media_generale = somma / 3.0;
25     printf("\nMedia generale della classe: %.2f\n", media_generale);
26
27     return 0;
28 }

```

Soluzione

Esercizio 4: Ricerca per Chiave in Archivio

Domanda

(cerca\{}_studente.c)

Aggiungere all'esercizio precedente un campo intero matricola. Si realizzi una funzione che, dato un array di studenti e una matricola da cercare, restituisca l'indice dello studente trovato, oppure -1 se non presente.

Spiegazione dell'algoritmo: Scansiamo sequenzialmente l'array di struct. Ad ogni iterazione, confrontiamo il valore della variabile cercata con il campo elenco[i].matricola. Non appena riscontriamo uguaglianza, interrompiamo restituendo la posizione corrente dell'array.

```

1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int matricola;
6      float media;
7  } Studente;
8
9  // Funzione di ricerca lineare
10 int cerca_studente(const Studente elenco[], int n, int
    ↪ matricola_cercata) {
11     for (int i = 0; i < n; i++) {
12         if (elenco[i].matricola == matricola_cercata) {
13             return i; // Restituisce l'indice della corrispondenza
14         }
15     }
16     return -1; // Non trovato
17 }
18
19 int main(void) {
20     Studente archivio[3] = {
21         {"Alice", 104523, 28.5},
22         {"Bob", 104589, 24.2},
23         {"Charlie", 104612, 27.8}
24     };
25
26     int matr = 104589;
27     int idx = cerca_studente(archivio, 3, matr);
28
29     if (idx != -1) {
30         printf("Studente trovato: %s, Media: %.2f\n", archivio[idx
    ↪ ].nome, archivio[idx].media);
31     } else {
32         printf("Nessuno studente con matricola %d in archivio.\n",
    ↪ matr);
33     }
34
35     return 0;
36 }

```

Soluzione

Esercizio 5: Scambio e Ordinamento di Struct

Domanda

(ordina\{}_studenti.c)

Si scriva una funzione per scambiare due struct `Studiante` e la si utilizzi per ordinare in senso decrescente (in base alla media) l'array di studenti precedentemente creato.

Spiegazione dell'algoritmo: La funzione `scambia` sfrutta la caratteristica nativa del C di copiare per intero le struct tramite assegnamento (`temp = *a;`). Nel ciclo del Bubble Sort, verifichiamo la condizione sulla media e scambiamo le intere strutture dati con un'unica chiamata, mantenendo l'allineamento dei record.

```

1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int matricola;
6      float media;
7  } Studiante;
8
9  // Funzione di swap per intere struct
10 void scambia_studenti(Studiante *a, Studiante *b) {
11     // Sfruttiamo la copia diretta del C
12     Studiante temp = *a;
13     *a = *b;
14     *b = temp;
15 }
16
17 int main(void) {
18     Studiante archivio[3] = {
19         {"Bob", 104589, 24.2},
20         {"Alice", 104523, 28.5},
21         {"Charlie", 104612, 27.8}
22     };
23
24     // Ordinamento Bubble Sort
25     for (int i = 0; i < 3 - 1; i++) {
26         for (int j = 0; j < 3 - i - 1; j++) {
27             if (archivio[j].media < archivio[j+1].media) {
28                 // Passiamo gli indirizzi per lo swap
29                 scambia_studenti(&archivio[j], &archivio[j+1]);
30             }
31         }
32     }
33
34     printf("Archivio ordinato per media decrescente:\n");
35     for (int i = 0; i < 3; i++) {
36         printf("%s (Matr: %d) - Media: %.2f\n", archivio[i].nome,
37             ↪ archivio[i].matricola, archivio[i].media);
38     }
39     return 0;
40 }

```

Soluzione

Esercizio 6: Gestione di Stati tramite**Domanda**

`enum(carriera\{}_esami.c)`

Si definisca una struct `Corso` contenente il nome della materia e un campo `stato` di tipo `enum` (`DA\{}_SOSTENERE`, `SUPERATO`, `FALLITO`). Si scriva una funzione che stampi a schermo l'elenco dei corsi colorando simbolicamente l'output a seconda dello stato.

Soluzione

```

1  #include <stdio.h>
2
3  typedef enum {
4      DA_SOSTENERE,
5      SUPERATO,
6      FALLITO
7  } StatoCorso;
8
9  typedef struct {
10     char materia[50];
11     StatoCorso stato;
12 } Corso;
13
14 void stampa_libretto(const Corso libretto[], int n) {
15     for (int i = 0; i < n; i++) {
16         printf("Materia: %-20s -> Stato: ", libretto[i].materia);
17         switch (libretto[i].stato) {
18             case DA_SOSTENERE:
19                 printf("[ ] Da Sostenere\n");
20                 break;
21             case SUPERATO:
22                 printf("[X] Superato\n");
23                 break;
24             case FALLITO:
25                 printf("[!] Fallito\n");
26                 break;
27         }
28     }
29 }
30
31 int main(void) {
32     Corso esami[3] = {
33         {"Programmazione 1", SUPERATO},
34         {"Architettura", DA_SOSTENERE},
35         {"Fisica 1", FALLITO}
36     };
37     stampa_libretto(esami, 3);
38     return 0;
39 }
40

```

Spiegazione dell'algoritmo: Il tipo StatoCorso mappa i tre possibili valori simbolici. Nella funzione stampa_{}_libretto utilizziamo un costrutto switch sul campo enum libretto[i].stato per determinare in modo leggibile quale descrizione testuale stampare.

Esercizio 7: Persistenza a Blocchi su Disco

Domanda

(salva\{}_archivio.c)

Si scriva un programma C che legga un array statico di 3 struct `Studiante` e lo scriva integralmente in formato binario all'interno del file `studenti.bin` in un solo colpo tramite la funzione `fwrite`.

Spiegazione dell'algoritmo: Apriamo il file con modalità "wb" (write binary). La funzione `fwrite` riceve come primo parametro il puntatore alla base dell'array `elenco`, e copia sul file $3 \times \text{sizeof}(\text{Studiante})$ byte consecutivi, effettuando il dump della RAM su disco.

```

1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int matricola;
6      float media;
7  } Studiante;
8
9  int main(void) {
10     Studiante elenco[3] = {
11         {"Alice", 104523, 28.5},
12         {"Bob", 104589, 24.2},
13         {"Charlie", 104612, 27.8}
14     };
15     FILE *fp = fopen("studenti.bin", "wb");
16     if (fp == NULL) {
17         fprintf(stderr, "Errore di apertura del file in scrittura.\n");
18         return 1;
19     }
20
21     // Salvataggio massivo dell'intero array in RAM sul file stream
22     size_t scritti = fwrite(elenco, sizeof(Studiante), 3, fp);
23     fclose(fp);
24     printf("Salvati correttamente %zu record di dimensione %zu byte\n", scritti, sizeof(Studiante));
25     return 0;
26 }

```

Soluzione

Esercizio 8: Recupero Blocco Dati da File Binario

Domanda

(carica\{}_archivio.c)

Si realizzi il programma speculare al precedente: il programma apre `studenti.bin` in

lettura binaria, carica l'intero blocco di dati all'interno di un array locale in RAM tramite `fread` e ne stampa a schermo le informazioni.

Spiegazione dell'algoritmo: Utilizziamo `fread` fornendo l'indirizzo dell'array `archivio_letti` in RAM la struttura originale delle struct.

```

1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int matricola;
6      float media;
7  } Studente;
8
9  int main(void) {
10     Studente archivio_letti[3];
11
12     FILE *fp = fopen("studenti.bin", "rb");
13     if (fp == NULL) {
14         fprintf(stderr, "Errore di apertura del file in lettura.
15         ↪ Eseguire prima il salvataggio.\n");
16     }
17     // Caricamento massivo del record dal file binario
18     size_t letti = fread(archivio_letti, sizeof(Studente), 3, fp);
19     fclose(fp);
20
21     printf("Caricati con successo %zu record.\n", letti);
22     for (int i = 0; i < 3; i++) {
23         printf("Studente %d: %s (Matr: %d) -> Media: %.2f\n",
24             i + 1, archivio_letti[i].nome, archivio_letti[i].
25             ↪ matricola, archivio_letti[i].media);
26     }
27     return 0;
28 }
29

```

Soluzione

Esercizio 9: Accesso Casuale a Record di Struct

Domanda

(`leggi\{}_record.c`)

Si scriva un programma C che consenta all'utente di inserire da tastiera un indice N (0, 1, 2) e, tramite `fseek`, legga direttamente ed esclusivamente il record corrispondente del file `studenti.bin`, senza caricare in memoria gli altri record.

Soluzione

Spiegazione dell'algoritmo: Sfruttando la dimensione fissa di `Studente` nota a tempo di compilazione, posizioniamo la testina di lettura all'offset in byte pari a `indice * sizeof(Studente)`. In questo modo la chiamata a `fread` estrae esclusivamente i byte associati allo studente richiesto.

```

1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int matricola;
6      float media;
7  } Studente;
8
9  int main(void) {
10     FILE *fp = fopen("studenti.bin", "rb");
11     if (fp == NULL) {
12         fprintf(stderr, "Errore: impossibile leggere 'studenti.bin'
        ↵ ".\n");
13     }
14     return 1;
15 }
16
17 int indice = 0;
18 printf("Inserisci l'indice del record da caricare (0, 1, o 2):
        ↵ ";
19 if (scanf("%d", &indice) != 1 || indice < 0 || indice > 2) {
20     printf("Indice non valido!\n");
21     fclose(fp);
22     return 1;
23 }
24 // Salto mirato: calcoliamo l'offset matematico
25 long offset = (long)(indice * sizeof(Studente));
26 fseek(fp, offset, SEEK_SET);
27 // Lettura del singolo record isolato
28 Studente cercato;
29 size_t letti = fread(&cercato, sizeof(Studente), 1, fp);
30 fclose(fp);
31 if (letti == 1) {
32     printf("\n[RECORD %d ESTRATTO]\n", indice);
33     printf("%s\n", cercato.nome);
34     printf("Matricola: %d\n", cercato.matricola);
35     printf("Media: %.2f\n", cercato.media);
36 } else {
37     printf("Errore durante la lettura del record.\n");
38 }
39
40 }
41
42 return 0;
43 }

```

Esercizio 10: Aggiornamento Diretto su File Binario

Domanda

(aggiorna\{}_record.c)

Si realizzi una semplice utility C che consenta di modificare in loco la media di un determinato studente salvato su `studenti.bin`. Il programma deve:

1. Chiedere l'indice da modificare e il nuovo valore della media.
2. Posizionarsi sul record desiderato.
3. Sovrascrivere esclusivamente quel record sul file aperto in modalità di aggiornamento "`r+b`".

--

Spiegazione dell'algoritmo: L'apertura "r+b" consente di modificare porzioni del file preservando il resto del contenuto. Effettuiamo una prima fseek per leggere lo studente, aggiorniamo il campo media nella struct in RAM, dopodiché eseguiamo una seconda fseek per riposizionare indietro il cursore del file (poiché la precedente fread lo aveva fatto avanzare di sizeof(Studente) byte) ed effettuiamo la sovrascrittura tramite fwrite.

```

1  #include <stdio.h>
2
3  typedef struct {
4      char nome[50];
5      int matricola;
6      float media;
7  } Studente;
8
9  int main(void) {
10     // Apriamo il file in lettura/scrittura senza distruggerlo (r+b)
11     FILE *fp = fopen("studenti.bin", "r+b");
12     if (fp == NULL) {
13         fprintf(stderr, "Errore: impossibile aprire il file. Eseguì
14         prima il salvataggio.\n");
15         return 1;
16     }
17     int indice = 0;
18     float nuova_media = 0.0;
19
20     printf("Inserisci l'indice dello studente da aggiornare (0, 1 o
21     2): ");
22     scanf("%d", &indice);
23     printf("Inserisci la nuova media: ");
24     scanf("%f", &nuova_media);
25
26     // 1. Spostiamo la testina sul record da aggiornare per
27     leggere prima
28     fseek(fp, indice * sizeof(Studente), SEEK_SET);
29     Studente s;
30     fread(&s, sizeof(Studente), 1, fp);
31
32     // 2. Modifichiamo il dato in memoria RAM
33     s.media = nuova_media;
34
35     // 3. Riposizioniamo la testina sullo stesso record prima di
36     sovrascriverlo
37     // (Nota: la fread ha fatto avanzare il cursore interno del
38     file)
39     fseek(fp, indice * sizeof(Studente), SEEK_SET);
40
41     // 4. Sovrascrittura mirata del blocco modificato
42     fwrite(&s, sizeof(Studente), 1, fp);
43     fclose(fp);
44     printf("Media aggiornata con successo su disco!\n");
45     return 0;
46 }

```