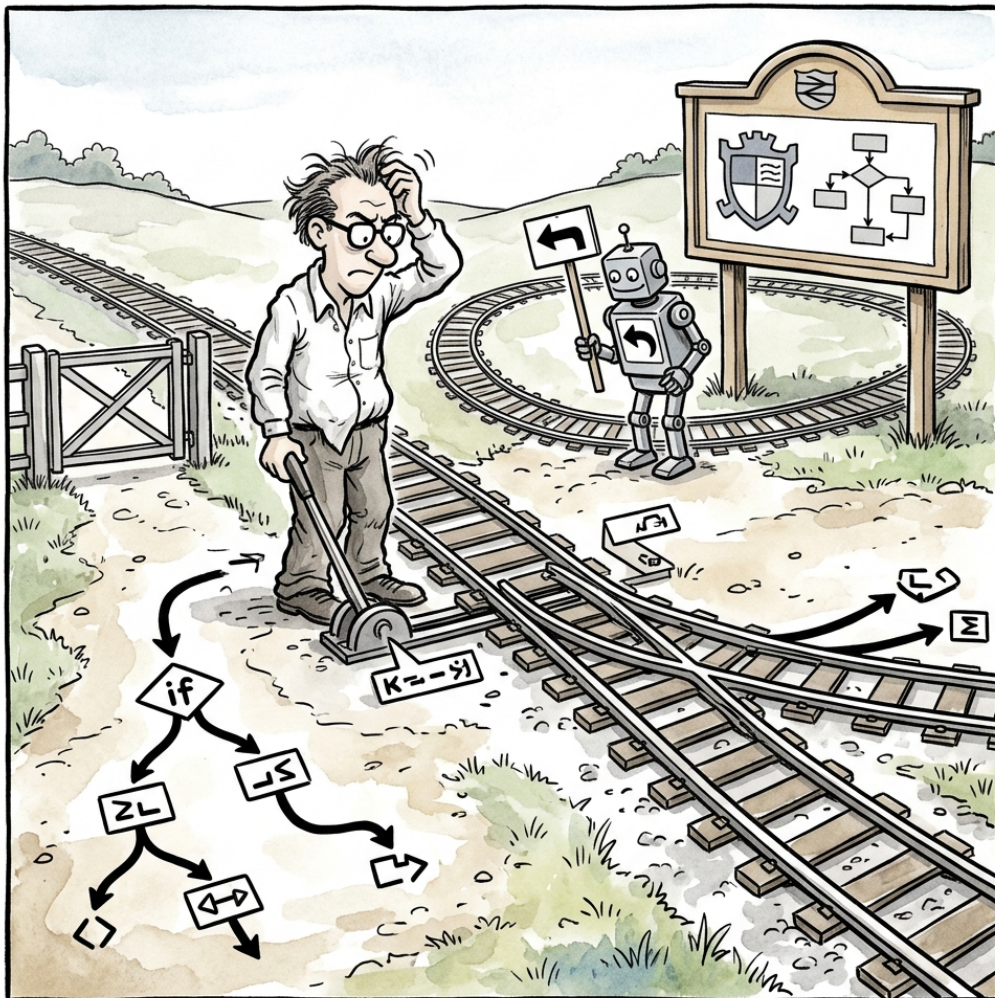


LEZIONE 4

Strutture Decisionali e l'Iterazione Fondamentale



Dispense di Programmazione 1 – Canale O-Z

Prof. Antonino Furnari

Corso di Laurea in Informatica

Università di Catania

Lezione 1

Strutture Decisionali e l'Iterazione Fondamentale

Nelle lezioni precedenti abbiamo completato la nostra cassetta degli attrezzi con i tipi di dato, la codifica dei reali e della tabella ASCII, e i dettagli sui modificatori di dimensione. Finora, però, i nostri programmi sono stati per lo più sequenze lineari di istruzioni. In questa lezione scopriremo come alterare questo flusso per prendere decisioni complesse e ripetere porzioni di codice. Partiremo analizzando come la CPU esegue i salti a basso livello (Assembly e `goto`), vedremo come il teorema di Böhm-Jacopini ci garantisca la possibilità di scrivere codice pulito senza usare salti incondizionati (spaghetti code), e impareremo a usare la selezione (`if/else`, `switch`, operatore ternario) e l'iterazione base (`while` con `break/continue`) in C. Infine, faremo la nostra prima conoscenza con il debugger **LLDB** per ispezionare il comportamento del codice in tempo reale.

1.1 Dalla RAM alla Programmazione Strutturata (Sotto il Cofano)

1.1.1 Il Flusso non Lineare e il Trucco del Salto (Goto)

Come fa il computer a saltare delle righe di codice o a ripetere un blocco di istruzioni? Fisicamente, la CPU possiede un registro speciale chiamato **Program Counter (PC)** che memorizza l'indirizzo della RAM dell'istruzione attualmente in esecuzione.

Normalmente il PC avanza in sequenza, istruzione dopo istruzione. Per deviare questo comportamento, la CPU non conosce blocchi o parentesi graffe: esegue semplicemente dei **salti (jump)** caricando un indirizzo diverso all'interno del Program Counter.

Consideriamo questa struttura condizionale in C e come viene convertita in linguaggio macchina dal compilatore:

```
1  /* Codice sorgente in C */
2  int x = 5;
3  if (x == 5) {
4      x = 10;
5  }
```

```
1 ; Equivalente Assembly concettuale (con il registro R1 che contiene il
   ↪ valore di x)
```

```
2  cmp R1, 5          ; Confronta il valore in R1 con il letterale 5
3  jne FINE_IF        ; Jump if Not Equal: se non sono uguali, salta a
   ↪ FINE_IF!
4  mov R1, 10         ; Se R1 == 5, esegui questo blocco (scrivi 10 in R1)
5  FINE_IF:
6  ; Il programma continua...
```

Nota

L'if tradotto al contrario Il compilatore traduce l'istruzione if al contrario: genera un'istruzione di salto condizionato (jne - salta se non sono uguali) che evita il blocco condizionale quando la condizione è **falsa**.

1.1.2 La Flessibilità del goto e lo Spaghetti Code

Anche in C è possibile scrivere salti incondizionati di basso livello abbinando l'istruzione goto a delle etichette:

```
1  int x = 10;
2  if (x > 5) goto SALTO;
3  printf("Questo messaggio viene ignorato.\n");
4
5  SALTO:
6  printf("Siamo saltati direttamente qui.\n");
```

Con il goto possiamo saltare in avanti (per saltare porzioni di codice) ma anche all'indietro (creando dei cicli). Negli anni '60, l'uso indiscriminato di questi salti liberi all'interno del codice portava a programmi disordinati definiti **spaghetti code**, complessi da comprendere e debuggare.

1.1.3 Il Teorema di Böhm-Jacopini

Nel 1966, gli informatici italiani **Corrado Böhm** e **Giuseppe Jacopini** pubblicarono una dimostrazione fondamentale che pose fine al caos dei salti non regolamentati:

Attenzione

Teorema di Böhm-Jacopini Qualsiasi algoritmo computabile può essere espresso utilizzando esclusivamente tre strutture di controllo fondamentali:

1. **Sequenza**: esecuzione ordinata di istruzioni una dopo l'altra.
2. **Selezione**: deviazione condizionale del flusso (es. bivio sì/no).
3. **Iterazione**: ripetizione di un blocco finché una condizione rimane vera.

Questo teorema assicura che possiamo bandire l'uso del goto senza perdere alcuna potenza computazionale, affidando la logica del nostro codice a strutture pulite e nidificate.

1.1.4 Concettualizzare con i Diagrammi di Flusso (Flowchart)

I diagrammi di flusso sono la visualizzazione grafica immediata delle tre strutture di controllo:

- **Rettangolo:** indica un'istruzione sequenziale.
- **Rombo:** indica una selezione condizionale che dirama il flusso.
- **Frecce all'indietro:** indicano un loop iterativo che ritorna in un punto precedente del programma.

```
1 graph TD
2   Start([Inizio]) --> RiempiCaldaia[Riempi la caldaia d'acqua]
3   RiempiCaldaia --> InserisciFiltro[Inserisci il filtro e il caffè?]
4   InserisciFiltro --> ChiudiMoka[Chiudi la moka e mettila sul fuoco]
5   ChiudiMoka --> CheckUscito{Il caffè ? uscito?}
6   CheckUscito -- No --> Attendi[Rimani in attesa]
7   Attendi --> CheckUscito
8   CheckUscito -- S? --> SpegniFuoco[Spegni il fuoco]
9   SpegniFuoco --> Versa[Versa il caffè? nella tazzina]
10  Versa --> End([Fine])
```

Il diagramma mostra chiaramente le tre strutture all'opera:

1. **Sequenza:** I passaggi iniziali si susseguono linearmente (Riempi caldaia → Inserisci filtro → Chiudi moka).
2. **Selezione:** Il rombo decisore (Il caffè è uscito?) separa il flusso in due percorsi alternativi basati su una risposta logica (Sì/No).
3. **Iterazione:** Il percorso che dal blocco di attesa ritorna al controllo del rombo forma un anello chiuso (ciclo) che ripete l'azione finché la condizione non cambia.

Suggerimento

Non esagerare con i disegni Non useremo software complessi per disegnare diagrammi di flusso: sono strumenti utili dal punto di vista concettuale ma non si usano nella progettazione di software complessi. Ci basta saperli tracciare rapidamente a mano libera.

1.2 Le Strutture Condizionali in C

1.2.1 La Selezione Completa: `if`, `else` ed `else if`

Completiamo la sintassi del blocco condizionale introducendo la via alternativa (`else`) e la cascata di controlli escludentisi (`else if`):

```
1  if (punti >= 90) {
2      printf("Eccellente\n");
3  } else if (punti >= 60) {
4      printf("Sufficiente\n");
5  } else {
6      printf("Non sufficiente\n");
7  }
```

Quando una delle condizioni a cascata risulta vera, la CPU esegue il relativo blocco e salta immediatamente oltre la fine della struttura dell'intera catena di `if`, ignorando i test successivi.

Nota

Graffe e istruzioni singole Se il blocco di codice contiene **una sola istruzione**, le parentesi graffe `{ } { }` sono opzionali:

```
1  if (temperatura > 30)
2      printf("Fa caldo\n");
3  else
4      printf("Si sta bene\n");
```

Tuttavia, è considerata una buona norma di programmazione inserire sempre le graffe per evitare errori logici di associazione quando si aggiungono istruzioni successivamente.

1.2.2 Selezione Multipla Discreta: `switch`

Quando dobbiamo confrontare una singola variabile discreta (es. intero o carattere) con molti valori fissi possibili, l'uso di molti `else if` risulta ridondante. Il C offre il costrutto `switch`:

```
1  #include <stdio.h>
2
3  int main(void) {
4      int scelta;
5      printf("---- MENU DISTRIBUTORE ---\n");
6      printf("1. Caff? (0.50?)\n");
7      printf("2. T? caldo (0.60?)\n");
8      printf("3. Cioccolata (0.70?)\n");
9      printf("Inserisci la tua scelta (1-3): ");
10     scanf("%d", &scelta);
11
12     switch (scelta) {
13         case 1:
14             printf("Hai selezionato: Caff?\n");
15             break;
16         case 2:
17             printf("Hai selezionato: T? caldo\n");
18             break;
19         case 3:
20             printf("Hai selezionato: Cioccolata\n");
21             break;
22         default:
```

```

23         printf("Selezione errata!\n");
24     }
25     return 0;
26 }

```

Attenzione

Il trabocchetto del Fall-Through Se omettete l'istruzione `break` alla fine di un `case`, la CPU non si ferma! Continuerà a eseguire a cascata le istruzioni dei `case` successivi anche se le loro condizioni non sono verificate. Questo comportamento può essere sfruttato intenzionalmente (ad esempio per far eseguire la stessa azione a 'a' e 'A'), ma nella maggior parte dei casi l'omissione del `break` è un bug.

Vincoli dello switch

- Può valutare **solo tipi discreti** (`int`, `char`, `short`, `long`). Non si possono usare `switch` con stringhe o con numeri a virgola mobile (`double`).
- Può verificare unicamente condizioni di **uguaglianza**, non relazioni d'ordine (non potete scrivere `case >= 18:`).

Nota

Lo `switch` non è indispensabile. Qualsiasi `switch` può essere riscritto in modo equivalente usando `if` ed `else if` annidati. È semplicemente uno strumento comodo per migliorare la leggibilità del codice.

1.2.3 L'Operatore Ternario (? :)

Consente di scrivere una selezione condizionale rapida per assegnare un valore a una variabile:

```

1  /* Con operatore ternario */
2  int max = (a > b) ? a : b;
3
4  /* Con if-else equivalente */
5  int max;
6  if (a > b) {
7      max = a;
8  } else {
9      max = b;
10 }

```

La sintassi è: `(condizione) ? valore_se_vero : valore_se_falso;`

1.3 L'Iterazione Fondamentale: Il Ciclo while

1.3.1 Il Costrutto `while`

Il ciclo `while` ripete l'esecuzione del blocco di codice racchiuso tra le graffe finché la sua condizione di controllo rimane vera (diversa da zero).

```
1 while (condizione) {  
2     // Codice da ripetere  
3 }
```

Attenzione

L'universalità del `while` Nelle prossime lezioni studieremo altri tipi di cicli come `for` o `do-while`. È fondamentale comprendere che **tutto ciò che si può implementare con gli altri cicli può essere espresso con un ciclo `while`**. Esso rappresenta l'iterazione base universale.

1.3.2 Evitare il Loop Infinito

Affinché il ciclo termini, all'interno del blocco delle graffe deve esistere un'istruzione che modifichi lo stato delle variabili coinvolte nella condizione di controllo, facendola diventare prima o poi falsa (0). Altrimenti il programma rimarrà bloccato in un loop infinito.

Vediamo un esempio di divisione intera eseguita per sottrazioni successive senza usare l'operatore `/`:

```
1 #include <stdio.h>  
2  
3 int main(void) {  
4     int dividendo = 17;  
5     int divisore = 5;  
6  
7     int quoziente = 0;  
8     int resto = dividendo;  
9  
10    // Sottraiamo finché il resto è maggiore o uguale al divisore  
11    while (resto >= divisore) {  
12        resto = resto - divisore; // Aggiorna lo stato!  
13        quoziente++;  
14    }  
15  
16    printf("Quoziente: %d, Resto: %d\n", quoziente, resto);  
17    return 0;  
18 }
```

1.3.3 Algoritmo del Massimo e Minimo (Sequenza Fissa)

Prima di addentrarci in cicli dal comportamento dinamico, studiamo un problema classico e fondamentale dell'informatica: **trovare il valore massimo e minimo all'interno di una sequenza di numeri** inseriti dall'utente.

Immaginiamo di dover leggere esattamente 10 numeri interi positivi. Per trovare gli estremi (massimo e minimo) senza dover memorizzare tutti i numeri contemporaneamente, applichiamo un algoritmo iterativo basato su **valori provvisori**:

1. Acquisiamo il primo numero e lo impostiamo provvisoriamente sia come massimo che come minimo.
2. Per ciascuno dei successivi 9 numeri: lo confrontiamo con il massimo provvisorio (se è maggiore, aggiorniamo il massimo) e con il minimo provvisorio (se è minore, aggiorniamo il minimo).

```
1  #include <stdio.h>
2
3  int main(void) {
4      int numero;
5      int max;
6      int min;
7
8      // Leggiamo il primo numero per impostare lo stato iniziale
9      printf("Inserisci il 1? numero: ");
10     if (scanf("%d", &numero) != 1) {
11         printf("Errore di lettura.\n");
12         return 1;
13     }
14     max = numero;
15     min = numero;
16
17     // Ciclo per i restanti 9 numeri
18     int i = 2; // Contatore: indica quale numero stiamo leggendo
19     while (i <= 10) {
20         printf("Inserisci il %d? numero: ", i);
21         if (scanf("%d", &numero) != 1) {
22             printf("Errore di lettura.\n");
23             return 1;
24         }
25
26         if (numero > max) {
27             max = numero; // Aggiorna il massimo provvisorio
28         }
29         if (numero < min) {
30             min = numero; // Aggiorna il minimo provvisorio
31         }
32         i++; // Passa al numero successivo
33     }
34
35     printf("\n--- Risultati ---\n");
36     printf("Il numero massimo inserito ?: %d\n", max);
37     printf("Il numero minimo inserito ?: %d\n", min);
38
39     return 0;
40 }
```

Suggerimento

Perché inizializzare con il primo elemento? Un errore comune nei principianti è inizializzare il massimo a 0 e il minimo a valori arbitrari (es. 99999). Questa scelta è rischiosa: se l'utente inserisse solo numeri negativi, il massimo rimarrebbe erroneamente 0 (che non fa parte della sequenza). Inizializzare `max` e `min` con il primo valore inserito garantisce la correttezza dell'algoritmo indipendentemente dal range dei dati.

1.3.4 Modificare il Flusso del Ciclo: `break` e `continue`

Possiamo alterare il comportamento standard di un ciclo a metà dell'esecuzione utilizzando due istruzioni speciali:

- **`break`**: esce immediatamente dal ciclo `while`, saltando alla prima istruzione utile dopo il blocco.
- **`continue`**: interrompe l'iterazione corrente a metà, saltando le righe sottostanti per tornare subito in cima a valutare nuovamente la condizione di controllo.

Per comprendere l'utilità di questi costrutti, analizziamo un problema reale: scrivere un programma che legga una sequenza di voti d'esame universitari inseriti da tastiera. Il programma deve:

1. Accettare solo voti validi compresi tra 18 e 30 (incluso 31 per indicare 30 e lode), saltando qualsiasi inserimento errato tramite l'istruzione **`continue`**.
2. Interrompere l'acquisizione non appena l'utente digita il valore di sentinella -1 tramite l'istruzione **`break`**.
3. Calcolare e visualizzare alla fine: il numero di esami validi inseriti, il voto massimo, il voto minimo e la media aritmetica dei voti validi.

```
1  #include <stdio.h>
2
3  int main(void) {
4      int voto = 0;
5      int somma = 0;
6      int conteggio = 0;
7
8      // Inizializziamo il massimo con il valore pi? basso possibile per un
      // esame
9      int max = 0;
10     // Inizializziamo il minimo con il valore pi? alto possibile per un
      // esame
11     int min = 100;
12
13     printf("Inserisci i voti degli esami (compresi tra 18 e 31; inserisci
      // -1 per terminare):\n");
14
15     while (1) { // Ciclo potenzialmente infinito, controllato all'interno
```

```

16     printf("Inserisci voto: ");
17     if (scanf("%d", &voto) != 1) {
18         printf("Inserimento non valido.\n");
19         break;
20     }
21
22     // Caso di uscita volontaria
23     if (voto == -1) {
24         break; // Esce immediatamente dal ciclo
25     }
26
27     // Filtro di validit?: i voti validi vanno da 18 a 30 (31 per
↪ lode)
28     if (voto < 18 || voto > 31) {
29         printf("Voto non valido! Deve essere compreso tra 18 e 31 (31
↪ = 30L).\n");
30         continue; // Salta il resto del corpo del ciclo e torna in
↪ cima
31     }
32
33     // Se siamo arrivati qui, il voto ? valido. Aggiorniamo le
↪ statistiche:
34     somma = somma + voto;
35     conteggio++;
36
37     if (voto > max) {
38         max = voto;
39     }
40     if (voto < min) {
41         min = voto;
42     }
43 }
44
45 // Calcolo e stampa dei risultati
46 if (conteggio > 0) {
47     double media = (double)somma / conteggio;
48     printf("\n--- Statistiche Esami ---\n");
49     printf("Esami superati: %d\n", conteggio);
50     printf("Voto massimo: %d\n", max);
51     printf("Voto minimo: %d\n", min);
52     printf("Media voti: %.2f\n", media);
53 } else {
54     printf("\nNessun voto valido inserito.\n");
55 }
56
57 return 0;
58 }

```

Nota

Perché qui `max = 0` e `min = 100` vanno bene? Poco sopra abbiamo sconsigliato di inizializzare massimo e minimo con valori arbitrari. Qui però l'intervallo dei dati è **noto e garantito dal filtro**: ogni voto che arriva all'aggiornamento di `max` e `min` è compreso

tra 18 e 31. Quindi il primo voto valido sarà sicuramente maggiore di 0 e minore di 100, e sostituirà entrambi i valori iniziali. Inoltre, se non viene inserito alcun voto valido, max e min non vengono mai stampati grazie al controllo `conteggio > 0`.

1.4 Quiz di Autovalutazione

Quiz 1

Domanda

1. Quale registro della CPU memorizza fisicamente l'indirizzo dell'istruzione in esecuzione?

- A) Il registro Accumulatore.
- B) Il Program Counter (PC).
- C) Il registro Stack Pointer.
- D) Il registro di Stato (Flags).

Soluzione
Risposta corretta: B
Spiegazione: Il Program Counter memorizza l'indirizzo di memoria dell'istruzione corrente e viene incrementato per puntare alla successiva o modificato direttamente durante le istruzioni di salto.

Quiz 2

Domanda

2. Cosa stabilisce il Teorema di Böhm-Jacopini?

- A) Che tutti i calcolatori devono seguire l'architettura di Von Neumann.
- B) Che qualsiasi algoritmo può essere espresso usando solo Sequenza, Selezione e Iterazione.
- C) Che il compilatore C converte i tipi reali sempre usando il bias 1023.
- D) Che l'uso del `goto` è obbligatorio per scrivere cicli infiniti.

Soluzione
Risposta corretta: B
Spiegazione: Dimostrato nel 1966, garantisce che i tre costrutti base siano sufficienti per esprimere qualsiasi logica computazionale, rendendo l'istruzione di salto incondizionato `goto` non necessaria.

Quiz 3

Domanda

3. Cosa succede se si omette il comando `break` in uno dei blocchi `case` di uno `switch`?

- A) Il compilatore segnala un errore di sintassi e arresta la compilazione.
- B) Il programma crasha con un errore di segmentation fault.
- C) La CPU prosegue eseguendo le istruzioni dei `case` successivi a cascata (fall-through).
- D) Lo `switch` viene convertito automaticamente in una divisione intera.

Soluzione
Risposta corretta: C
Spiegazione: Il `break` serve a uscire dallo `switch`. Se manca, l'esecuzione prosegue linearmente sulle righe dei casi successivi, indipendentemente dal valore dei loro controlli.

Quiz 4

Domanda

4. Quale dei seguenti tipi non può essere valutato all'interno di una selezione `switch`?

- A) `int`
- B) `char`
- C) `double`
- D) `short`

Soluzione
Risposta corretta: C
Spiegazione: Il costrutto `switch` in C opera esclusivamente su valori discreti convertibili in interi. I numeri in virgola mobile (`double` e `float`) e le stringhe sono esclusi.

Quiz 5

Domanda

5. Quale espressione con l'operatore ternario equivale a questo blocco?

```
1 int ris;
2 if (valore > 0) ris = 1;
3 else ris = 0;
```

- A) `int ris = (valore > 0) ? 0 : 1;`

- B) `int ris = (valore > 0) ? 1 : 0;`
- C) `int ris = (valore > 0) : 1 ? 0;`
- D) `int ris = valore > 0;`

Soluzione
Risposta corretta: B (anche la D è corretta in C, ma la B usa esplicitamente l'operatore ternario)
Spiegazione: La sintassi dell'operatore ternario assegna il primo valore (1) se la condizione è vera, altrimenti il secondo (0).

Quiz 6

Domanda

6. Che cos'è l'universalità del ciclo `while`?

- A) Significa che funziona su qualsiasi sistema operativo.
- B) Significa che qualsiasi ciclo scritto con `for` o `do-while` può essere espresso tramite il `while`.
- C) Significa che il `while` non può mai produrre un ciclo infinito.
- D) Significa che può valutare solo condizioni logiche ma non confronti numerici.

Soluzione
Risposta corretta: B
Spiegazione: Il `while` rappresenta la forma più pura ed elementare di iterazione. Gli altri cicli sono zuccheri sintattici che facilitano la scrittura, ma non aggiungono potenza di calcolo.

Quiz 7

Domanda

7. Cosa fa l'istruzione `continue` all'interno di un ciclo `while`?

- A) Esce immediatamente dal ciclo saltando alla fine del programma.
- B) Salta la parte rimanente del blocco del ciclo solo per l'iterazione corrente, ritornando in cima a valutare la condizione.
- C) Esegue di nuovo l'istruzione precedente per correggere eventuali errori.
- D) Azzera il contatore del ciclo portando la condizione a vero.

Soluzione
Risposta corretta: B
Spiegazione: `continue` interrompe solo il passaggio corrente. Riporta il flusso in cima al controllo del ciclo per l'iterazione successiva.

Quiz 8

Domanda

8. Nel programma di esempio sui voti universitari, cosa succede se l'utente inserisce la sequenza di numeri: 25, 12, 30, -1?

- A) Il programma stampa una media pari a 22.33 (escludendo solo -1).
- B) Il programma termina restituendo un errore a causa del valore 12.
- C) Il programma ignora il voto 12 (tramite `continue`) e calcola le statistiche (media 27.50) su 25 e 30 dopo l'uscita tramite `break` con -1.
- D) Il programma va in loop infinito.

Soluzione
Risposta corretta: C
Spiegazione: Il valore 12 è fuori dall'intervallo [18, 31], quindi attiva la condizione d'errore e l'istruzione `continue` che salta le istruzioni di accumulo e conteggio. Il valore -1 provoca l'attivazione del `break` che interrompe il ciclo e avvia la stampa finale delle medie basate solo sui voti validi (25 e 30, la cui media è 27.50).

1.5 Esercizi Pratici

Esercizio 1: Il Calcolo del Massimo tra tre numeri

Domanda

Scrivete un programma che legga tre numeri interi da tastiera e stampi il valore massimo tra i tre, usando una cascata di `if/else` ed evitando l'uso di operatori logici complessi.

Spiegazione dell'algoritmo: Assumiamo inizialmente che a sia il massimo. Successivamente confrontiamo questo valore provvisorio con b ed eventualmente lo aggiorniamo, ripetendo lo stesso passaggio con c.

```

1  #include <stdio.h>
2
3  int main(void) {
4      int a, b, c;
5      printf("Inserisci tre numeri interi: ");
6      scanf("%d %d %d", &a, &b, &c);
7
8      int max = a;
9
10     if (b > max) {
11         max = b;
12     }
13     if (c > max) {
14         max = c;
15     }
16     printf("Il valore massimo inserito è: %d\n", max);
17     return 0;
18 }
19

```

Soluzione

Esercizio 2: Calcolatore di Tariffe Distributore

Domanda

Scrivete un programma che simuli la selezione di una scelta menu (1=Caffè, 2=Cioccolata, 3=Tè) e stampi il costo, usando lo switch con gestione del caso di errore tramite default.

Spiegazione dell'algoritmo: Il costrutto switch seleziona il blocco corretto controllando l'intero scelta. L'istruzione break garantisce l'uscita dallo switch evitando l'esecuzione del codice sottostante.

```

1  #include <stdio.h>
2
3  int main(void) {
4      int scelta;
5      printf("Menu Bevande:\n1. Caffè\n2. Cioccolata\n3. Tè\nScelta: ");
6      scanf("%d", &scelta);
7
8      switch (scelta) {
9          case 1:
10             printf("Costo Caffè: 0.50?\n");
11             break;
12          case 2:
13             printf("Costo Cioccolata: 0.70?\n");
14             break;
15          case 3:
16             printf("Costo Tè: 0.60?\n");
17             break;
18          default:
19             printf("Errore: Selezione non valida!\n");
20     }
21     return 0;
22 }

```

Soluzione

Esercizio 3: Divisione intera senza l'operatore di

Domanda

divisione

Scrivete un programma che calcoli la divisione intera e il resto tra due numeri interi positivi usando solo sottrazioni ripetute all'interno di un ciclo while.

Spiegazione dell'algoritmo: Sottraiamo il divisore dal dividendo cumulato in resto fino a quando non è più possibile sottrarre senza andare in negativo. Il numero di sottrazioni effettuate corrisponde al quoziente.

```

1  #include <stdio.h>
2
3  int main(void) {
4      int dividendo, divisore;
5      printf("Inserisci dividendo e divisore positivi: ");
6      scanf("%d %d", &dividendo, &divisore);
7
8      if (divisore <= 0 || dividendo < 0) {
9          printf("Errore: inserisci valori validi!\n");
10         return 1;
11     }
12
13     int quoziente = 0;
14     int resto = dividendo;
15
16     while (resto >= divisore) {
17         resto = resto - divisore;
18         quoziente++;
19     }
20
21     printf("Risultato: %d con resto %d\n", quoziente, resto);
22     return 0;
23 }
```

Soluzione

Esercizio 4: Massimo Comun Divisore (MCD)

Domanda

Implementate l'algoritmo di Euclide a sottrazioni successive per calcolare il MCD tra due numeri interi positivi letti da tastiera.

Spiegazione dell'algoritmo: Ripetiamo la sottrazione del minore dal maggiore finché le due variabili temp_a e temp_b non contengono lo stesso valore, che per definizione di Euclide corrisponde al MCD.

```

1  #include <stdio.h>
2
3  int main(void) {
4      int a, b;
5      printf("Inserisci due numeri interi positivi: ");
6      scanf("%d", &a, &b);
7
8      if (a <= 0 || b <= 0) {
9          printf("I numeri devono essere positivi!\n");
10         return 1;
11     }
12     int temp_a = a;
13     int temp_b = b;
14
15     while (temp_a != temp_b) {
16         if (temp_a > temp_b) {
17             temp_a = temp_a - temp_b;
18         } else {
19             temp_b = temp_b - temp_a;
20         }
21     }
22
23     printf("Il MCD tra %d e %d è %d\n", a, b, temp_a);
24     return 0;
25 }

```

Soluzione

Esercizio 5: Stampa dei soli numeri dispari con continue

Domanda

Scrivete un programma che stampi tutti i numeri da 1 a 15, escludendo i numeri pari tramite l'uso dell'istruzione `continue` all'interno di un ciclo `while`.

Spiegazione dell'algoritmo: A ogni iterazione incrementiamo `i`. Se `i` è pari (`i % 2 == 0`), l'istruzione `continue` fa ripartire immediatamente il ciclo dal controllo, ignorando l'istruzione `printf` sottostante.

```

1  #include <stdio.h>
2
3  int main(void) {
4      int i = 0;
5      while (i < 15) {
6          i++;
7          if (i % 2 == 0) {
8              continue; // Salta il resto del ciclo e torna in cima
9          }
10         printf("%d ", i);
11     }
12     printf("\n");
13     return 0;
14 }

```

Soluzione

Esercizio 6: Somma di numeri fino a inserimento dello zero

Domanda

Scrivete un programma che continui a richiedere numeri interi all'utente sommandoli tra loro. Il ciclo deve interrompersi non appena l'utente digita il valore 0. Stampate infine la somma totale.

Spiegazione dell'algoritmo: Fin tanto che l'utente non inserisce 0, la condizione `numero != 0` rimane vera e il ciclo accumula il valore letto nella variabile `somma`.

```

1  #include <stdio.h>
2
3  int main(void) {
4      int numero = -1; // Inizializzato a un valore diverso da zero
5      int somma = 0;
6
7      while (numero != 0) {
8          printf("Inserisci un numero (0 per terminare): ");
9          scanf("%d", &numero);
10         somma = somma + numero;
11     }
12     printf("La somma totale dei numeri inseriti è: %d\n", somma);
13     return 0;
14 }
15

```

Soluzione

Esercizio 7: Conversione carattere maiuscolo/minuscolo**Domanda**

Scrivete un programma che acquisisca un singolo carattere da tastiera. Se è una lettera maiuscola, stampate la rispettiva lettera minuscola sfruttando la tabella ASCII (senza usare funzioni di libreria), altrimenti stampate un errore.

Spiegazione dell'algoritmo: Sfruttiamo la codifica ordinata ASCII. Le lettere maiuscole e minuscole distano sempre 32 posizioni tra loro. Sommando questa differenza passiamo al carattere minuscolo equivalente.

```

1  #include <stdio.h>
2
3  int main(void) {
4      char c;
5      printf("Inserisci un carattere maiuscolo: ");
6      scanf("%c", &c); // Lo spazio prima di %c consuma eventuali
                          // whitespace residui
7
8      if (c >= 'A' && c <= 'Z') {
9          // La distanza tra maiuscole e minuscole in ASCII è
          // costante ('a' - 'A' = 32)
10         char minuscolo = c + ('a' - 'A');
11         printf("Lettera minuscola: %c\n", minuscolo);
12     } else {
13         printf("Errore: il carattere inserito non è una lettera
14             ↪ minuscola!\n");
15     }
16     return 0;
17 }
```

Soluzione**Esercizio 8: Menu Distributore Ciclico con Uscita****Domanda**

volontaria

Scrivete un programma che simuli il distributore di bevande dell'Esercizio 2 all'interno di un ciclo `while`. Il menu deve continuare a ripresentarsi all'utente dopo ogni acquisto finché non viene selezionata l'opzione di uscita (tasto 0).

Spiegazione dell'algoritmo: Il ciclo `while` continua a riproporre il menu fino a quando `scelta != 0` diventa falsa. Lo `switch` all'interno gestisce le opzioni di acquisto e l'opzione di uscita.

```

1  #include <stdio.h>
2
3  int main(void) {
4      int scelta = -1;
5
6      while (scelta != 0) {
7          printf("\n--- DISTRIBUTORE AUTOMATICO ---\n");
8          printf("1. Caffè (0.50?)\n");
9          printf("2. Cioccolata (0.70?)\n");
10         printf("3. Tè (0.60?)\n");
11         printf("0. Esci\n");
12         printf("Scelta: ");
13         scanf("%d", &scelta);
14
15         switch (scelta) {
16             case 0:
17                 printf("Arrivederci!\n");
18                 break;
19             case 1:
20                 printf("Prezzo: 0.50?\n");
21                 break;
22             case 2:
23                 printf("Prezzo: 0.70?\n");
24                 break;
25             case 3:
26                 printf("Prezzo: 0.60?\n");
27                 break;
28             default:
29                 printf("Scelta errata!\n");
30         }
31     }
32     return 0;
33 }

```

Soluzione

Esercizio 9: Calcolo potenza senza pow

Domanda

Scrivete un programma che calcoli la potenza base elevato esponente (b^e) per esponenti interi non negativi, usando un ciclo `while` e moltiplicazioni ripetute.

Soluzione

```

1  #include <stdio.h>
2
3  int main(void) {
4      double base;
5      int esponente;
6      printf("Inserisci la base (reale) e l'esponente (intero)
    ↳ positivo: ");
7      scanf("%lf %d", &base, &esponente);
8
9      if (esponente < 0) {
10         printf("L'esponente deve essere non negativo!\n");
11         return 1;
12     }
13
14     double risultato = 1.0;
15     int i = 0;
16
17     while (i < esponente) {
18         risultato = risultato * base;
19         i++;
20     }
21
22     printf("Risultato: %.6f\n", risultato);
23     return 0;
24 }

```

Spiegazione dell'algoritmo: Inizializziamo il risultato a 1.0. Moltiplichiamo il risultato per la base tante volte quante indichiate dall'esponente, usando la variabile contatore `i` come controllo.

Esercizio 10: Fibonacci Limitato**Domanda**

Scrivete un programma che stampi tutti i numeri della celebre successione di Fibonacci (1, 1, 2, 3, 5, 8, 13...) che siano inferiori o uguali a un valore massimo inserito dall'utente, usando un ciclo `while`.

Soluzione

```
1 #include <stdio.h>
2
3 int main(void) {
4     int limite;
5     printf("Inserisci il valore massimo: ");
6     scanf("%d", &limite);
7
8     int a = 1;
9     int b = 1;
10
11     printf("Successione di Fibonacci: ");
12     while (a <= limite) {
13         printf("%d ", a);
14         int successivo = a + b;
15         a = b;
16         b = successivo;
17     }
18     printf("\n");
19     return 0;
20 }
```

Spiegazione dell'algoritmo: La successione di Fibonacci si ottiene sommando i due termini precedenti. A ogni passaggio calcoliamo il successivo, facciamo slittare in avanti i valori delle variabili a e b e ci fermiamo non appena superiamo la soglia impostata.