	Politecnico di Milano Scuola di Ingegneria Industriale e dell'Informazione **INFORMATICA B** Appello del 26/06/2024		COGNOME E NOME
	Fila	Colonna	MATRICOLA

- Il presente plico contiene 3 esercizi e 2 domande e **deve essere debitamente compilato con cognome e nome, e numero di matricola**.
- Il tempo a disposizione è di 1 ora e 30 minuti.
- Non separate questi fogli. Scrivete la soluzione solo sui fogli distribuiti, utilizzando il retro delle pagine in caso di necessità. Cancellate le parti di brutta (o ripudiate) con un tratto di penna.
- Ogni parte non cancellata a penna sarà considerata parte integrante della soluzione.
- È possibile scrivere a matita (e non occorre ricalcare al momento della consegna!).
- **Qualsiasi tentativo di comunicare con altri studenti comporta l'espulsione dall'aula**.
- È possibile ritirarsi senza penalità.
- **Non è possibile lasciare l'aula conservando il tema della prova in corso**.
- **Per il superamento dell'esame è necessario dimostrare sufficienti competenze sia in C sia in Matlab, e quindi saper impostare correttamente esercizi in entrambi i linguaggi**.
- **MOLTO IMPORTANTE**: **risposte poco leggibili** (scritte molto piccolo, con calligrafia poco comprensibile, o molto disordinate) **non saranno considerate nella valutazione**.

Esercizio 1 (10 punti)

Un cinema multisala, gestito da ex-studenti del corso di Informatica B, vuole gestire la prenotazione dei posti a sedere per le sue sale cinematografiche attraverso un software in linguaggio C. Ogni sala è caratterizzata da:

- Numero della sala (intero);
- Numero di file di poltrone (intero);
- Numero di poltrone (per semplicità, nel seguito, verranno chiamati posti) per fila (intero);
- Matrice di poltrone, dove ogni elemento è 0 (libero) o 1 (occupato).

Domanda 1.1 (punti 2.5) Definire una struttura dati `Sala` per rappresentare una sala cinematografica, avente **al massimo** 100 file e 100 posti per fila, e un array `multisala` per memorizzare fino a 10 sale

Si risponda alle seguenti ulteriori domande assumendo che la variabile `multisala` sia stata correttamente riempita con le informazioni relative a 10 sale.

Domanda 1.2 (punti 2.5) Facendo riferimento ai dati nella variabile `multisala`, scrivere una porzione di codice che, richiesti all'utente il numero della sala, la fila e il posto desiderati, controlli se la sala esiste e se la fila e il posto indicati sono effettivamente presenti nella sala e, se il posto è libero, lo occupi. La porzione di codice stamperà il valore 1 se la prenotazione è andata a buon fine, 0 altrimenti.

Domanda 1.3 (punti 2.5) Facendo riferimento ai dati nella variabile `multisala`, scrivere una porzione di codice che, richiesto all'utente il numero della sala, stampi a schermo il numero totale di posti liberi nella sala.

Domanda 1.4 (punti 2.5) Facendo riferimento ai dati nella variabile `multisala`, scrivere una porzione di codice che, richiesto all'utente il numero della sala, stampi a video una rappresentazione della sala con i posti liberi indicati da "O" e i posti occupati da "X".

Esempio di stampa per una sala con 3 file, 5 posti per ciascuna e 4 posti occupati in totale:

```
  1 2 3 4 5

1 O O X O O

2 X O O O X

3 O X O O O
```

Soluzione

Domanda 1.1

```c
#include <stdio.h>

#define MAX_FILE 100
#define MAX_POSTI 100
#define NUM_SALE 10

typedef struct {
    int numero_sala;
    int num_file;
    int num_posti_per_fila;
    int poltrone[MAX_FILE][MAX_POSTI];
} Sala;

Sala multisala[NUM_SALE];
```

Domanda 1.2

```c
#include <stdio.h>

int main() {
    // Supponiamo che multisala sia già inizializzato con dati validi

    int numero_sala, fila, posto;
    printf("Inserisci il numero della sala, la fila e il posto desiderati: ");
    scanf("%d %d %d", &numero_sala, &fila, &posto);

    int prenotazione_riuscita = 0;
    for (int i = 0; i < NUM_SALE; i++) {
        if (multisala[i].numero_sala == numero_sala) {
            if (fila >= 1 && fila <= multisala[i].num_file && posto >= 1 && posto <=
multisala[i].num_posti_per_fila) {
                if (multisala[i].poltrone[fila - 1][posto - 1] == 0) {
                    multisala[i].poltrone[fila - 1][posto - 1] = 1;
                    prenotazione_riuscita = 1;
                }
            }
        }
    }

    printf("%d\n", prenotazione_riuscita);
    return 0;
}
```

Domanda 1.3

```c
#include <stdio.h>

int main() {
    // Supponiamo che multisala sia già inizializzato con dati validi
```

```
    int numero_sala;
    printf("Inserisci il numero della sala: ");
    scanf("%d", &numero_sala);

    int posti_liberi = 0;
    for (int i = 0; i < NUM_SALE; i++) {
        if (multisala[i].numero_sala == numero_sala) {
            for (int j = 0; j < multisala[i].num_file; j++) {
                for (int k = 0; k < multisala[i].num_posti_per_fila; k++) {
                    if (multisala[i].poltrone[j][k] == 0) {
                        posti_liberi++;
                    }
                }
            }
        }
    }

    printf("Posti liberi nella sala %d: %d\n", numero_sala, posti_liberi);
    return 0;
}
```

Domanda 1.4

```
#include <stdio.h>

int main() {
    // Supponiamo che multisala sia già inizializzato con dati validi

    int numero_sala;
    printf("Inserisci il numero della sala: ");
    scanf("%d", &numero_sala);

    for (int i = 0; i < NUM_SALE; i++) {
        if (multisala[i].numero_sala == numero_sala) {
            // Stampa l'intestazione dei numeri dei posti
            printf("   ");
            for (int k = 1; k <= multisala[i].num_posti_per_fila; k++) {
                printf("%d ", k);
            }
            printf("\n");
            // Stampa le file con i posti
            for (int j = 0; j < multisala[i].num_file; j++) {
                printf("%d ", j + 1);
                for (int k = 0; k < multisala[i].num_posti_per_fila; k++) {
                    if (multisala[i].poltrone[j][k] == 0) {
                        printf("O ");
                    } else {
                        printf("X ");
                    }
                }
```

```
            printf("\n");
        }
    }
}

    return 0;
}
```

Esercizio 2 (10 punti)

Un'azienda automobilistica ha condotto una serie di test su diversi modelli di auto per valutare le loro prestazioni. I dati raccolti sono memorizzati in un file binario MATLAB chiamato `dati_auto.mat`, che contiene le seguenti variabili:

- `modello`: un array di stringhe contenente i nomi dei modelli di auto testati;
- `accelerazione`: un array contenente i valori di accelerazione da 0 a 100 km/h (in secondi) per ciascun modello;
- `consumo`: un array contenente i valori di consumo di carburante (in litri per 100 km) per ciascun modello;
- `potenza`: un array contenente i valori di potenza del motore (in cavalli vapore) per ciascun modello.
- `Prezzo`: un array contenente il prezzo in € per ciascun modello.

Si scriva uno script MATLAB che esegua le seguenti operazioni.

Domanda 2.1 (2 punti) Dopo aver caricato il file `dati_auto.mat`, calcolare e visualizzare un grafico che mostri la relazione tra accelerazione e consumo di carburante. Ogni punto nel grafico rappresenterà un modello di auto, con l'accelerazione sull'asse x e il consumo sull'asse y.

Domanda 2.2 (3 punti) Identificare il modello di auto con la migliore accelerazione (tempo più basso) e il modello con il miglior consumo di carburante (valore più basso).

- Stampare i nomi dei modelli e i relativi valori di accelerazione e consumo.
- Si evidenzi sul grafico precedente il modello di auto con la migliore accelerazione con una stella (*) e il modello con il miglior consumo di carburante con un triangolo (^).

Domanda 2.3 (3 punti) Scrivere una funzione MATLAB che prenda in input gli array potenza e consumo e restituisca:

- Il valore medio del rapporto potenza/consumo per tutti i modelli.
- I nomi dei modelli che hanno un rapporto potenza/consumo superiore alla media.

Scrivere poi uno script in cui si chiama la funzione e si stampano a schermo i risultati ottenuti.

Domanda 2.4 (2 punti) Definito come "efficiente" un modello che soddisfa tutte le seguenti condizioni:

- Consumo di carburante inferiore a 6 l/100km.
- Potenza del motore superiore a 100 CV.
- Prezzo inferiore a 30000 €.
- Un rapporto potenza/consumo superiore alla media

Stampare il nome del modello, la marca, il consumo, la potenza e il prezzo delle auto efficienti.

Soluzione

Domanda 2.1

```
% Carica i dati dal file dati_auto.mat
load('dati_auto.mat');

figure;
plot(accelerazione, consumo, 'o');
xlabel('Accelerazione (0-100 km/h) [s]');
ylabel('Consumo di carburante (l/100 km)');
title('Relazione tra Accelerazione e Consumo di carburante');
grid on;
```

Domanda 2.2

```
% Migliore accelerazione
[~, idx_migliore_accelerazione] = min(accelerazione);
miglior_modello_accelerazione = modello(idx_migliore_accelerazione, :);
valore_migliore_accelerazione = accelerazione(idx_migliore_accelerazione);

% Miglior consumo
[~, idx_miglior_consumo] = min(consumo);
miglior_modello_consumo = modello(idx_miglior_consumo, :);
valore_miglior_consumo = consumo(idx_miglior_consumo);

% Stampa dei risultati
fprintf('Migliore accelerazione: %s con %.2f secondi\n', miglior_modello_accelerazione,
valore_migliore_accelerazione);
fprintf('Miglior consumo: %s con %.2f litri/100 km\n', miglior_modello_consumo, valore_miglior_consumo);

% Evidenziare i modelli sul grafico
hold on;
plot(accelerazione(idx_migliore_accelerazione), consumo(idx_migliore_accelerazione), 'rp', 'MarkerSize',
10, 'MarkerFaceColor', 'r'); % stella rossa
plot(accelerazione(idx_miglior_consumo), consumo(idx_miglior_consumo), 'g^', 'MarkerSize', 10,
'MarkerFaceColor', 'g'); % triangolo verde
legend('Modelli', 'Migliore Accelerazione', 'Miglior Consumo', 'Location', 'Best');
hold off;
```

Domanda 2.3

```
function [media_rapporto, modelli_superiori_alla_media] = calcola_rapporto(potenza, consumo, modelli)
    rapporto = potenza ./ consumo;
    media_rapporto = mean(rapporto);
    modelli_superiori_alla_media = modelli(rapporto > media_rapporto, :);
end

% Chiamare la funzione e visualizzare i risultati
[media_rapporto, modelli_superiori_alla_media] = calcola_rapporto(potenza, consumo, modello);
fprintf('Valore medio del rapporto potenza/consumo: %.2f\n', media_rapporto);
fprintf('Modelli con rapporto potenza/consumo superiore alla media:\n');
```

```
disp(modelli_superiori_alla_media);
```

Domanda 2.4

```
efficienza_criteria = consumo < 6 & potenza > 100 & prezzo < 30000;

% Calcolare il rapporto potenza/consumo
rapporto_potenza_consumo = potenza ./ consumo;

% Media del rapporto potenza/consumo
media_rapporto = mean(rapporto_potenza_consumo);

% Ulteriore criterio di efficienza
efficienza_criteria = efficienza_criteria & (rapporto_potenza_consumo > media_rapporto);

% Stampare i risultati
fprintf('Modelli di auto efficienti:\n');
fprintf('Modello\t\tConsumo\t\tPotenza\t\tPrezzo\n');
for i = 1:length(modello)
    if efficienza_criteria(i)
        fprintf('%s\t\t%.2f\t\t%.2f\t\t%.2f\n', modello(i, :), consumo(i), potenza(i), prezzo(i));
    end
end
```

Esercizio 3 (6 punti)

Si consideri il seguente codice C:

```
#include <stdio.h>

int main() {
    char buffer[16];
    int i;
    short int j;

    printf("Inserisci stringa: \n");
    scanf("%s", buffer);

    printf("Inserisci un numero intero: ");
    scanf("%d", &i);

    if (i != 0) {
        printf("Welcome to level 1\n");

        j = i;

        if (j == 0) {
            printf("Welcome to level 2\n");
        } else {
            printf("FAILED\n");
        }
    } else {
        printf("FAILED\n");
    }
    return 0;
}
```

Si risponda alle seguenti domande:

Domanda 3.1 (2 punti) Nel caso in cui gli input forniti al programma sono: “centocinquantacinque” per `buffer` e 10 per `i`, il programma stamperà `"Welcome to level 1”,` in seguito `"FAILED",` e terminerà mostrando un messaggio di errore. Si fornisca una motivazione del comportamento descritto, tenendo conto delle dimensioni e del tipo delle variabili utilizzate.

Domanda 3.2 (punti 2) Nel caso in cui gli input forniti al programma sono: “StringaGenerica” per `buffer` e 10 per `i`, il programma stamperà `"Welcome to level 1"` e in seguito `"FAILED"`, terminando correttamente l’esecuzione. Si fornisca una motivazione del comportamento descritto, tenendo conto delle dimensioni e del tipo delle variabili utilizzate.

Domanda 3.3 (punti 2) Nel caso in cui gli input forniti al programma sono: “StringaGenerica” per `buffer` e 65536 per `i`, il programma stamperà `"Welcome to level 1",` in seguito `"Welcome to level 2", "`, terminando correttamente l’esecuzione. Si fornisca una motivazione del comportamento descritto, tenendo conto delle dimensioni e del tipo delle variabili utilizzate.

Domanda 1 (3 punti)

Si consideri un sistema operativo in esecuzione su una CPU single-core che utilizza una politica di scheduling round-robin con un quanto di tempo di 10 millisecondi. Si supponga inoltre che il tempo per i cambi di contesto sia così basso da essere trascurabile. Supponiamo che al tempo t=0 siano presenti nella coda di scheduling tre processi, nell'ordine P1, P2 e P3, tutti nello stato di pronto, con tempi di esecuzione rispettivamente di 30 ms, 15 ms e 45 ms. Assumendo che non ci siano interruzioni interne o esterne e che i tre processi non si blocchino mai per eseguire operazioni di I/O, descrivere come il sistema operativo gestirà l'esecuzione di questi processi, indicando l'ordine in cui verranno eseguiti e i relativi tempi di attesa (Tempo di attesa = Tempo di completamento - Tempo di esecuzione).

Soluzione

Passo 1: Ciclo iniziale di 10 ms

- **Tempo 0-10 ms:**
  - Esegue **P1** (rimangono 20 ms di P1)
- **Tempo 10-20 ms:**
  - Esegue **P2** (rimangono 5 ms di P2)
- **Tempo 20-30 ms:**
  - Esegue **P3** (rimangono 35 ms di P3)

Passo 2: Secondo ciclo di 10 ms

- **Tempo 30-40 ms:**
  - Esegue **P1** (rimangono 10 ms di P1)
- **Tempo 40-45 ms:**
  - Esegue **P2** (termina P2)
- **Tempo 45-55 ms:**
  - Esegue **P3** (rimangono 25 ms di P3)

Passo 3: Terzo ciclo di 10 ms

- **Tempo 55-65 ms:**
  - Esegue **P1** (termina P1)
- **Tempo 65-75 ms:**
  - Esegue **P3** (rimangono 15 ms di P3)

Passo 4: Quarto ciclo di 10 ms

- **Tempo 75-85 ms:**
  - Esegue **P3** (rimangono 5 ms di P3)

Passo 5: Quinto ciclo di 10 ms

- **Tempo 85-90 ms:**
  - Esegue **P3** (termina P3)

Tempi di attesa

- **P1:**
  - Inizia a tempo 0, finisce a tempo 65ms.
  - Tempo di attesa = Tempo di fine - Tempo di esecuzione totale = 65 - 30 = 35 ms.
- **P2:**
  - Inizia a tempo 10, finisce a tempo 45 ms.
  - Tempo di attesa = Tempo di fine - Tempo di esecuzione totale = 45 - 15 = 30 ms.
- **P3:**
  - Inizia a tempo 20, finisce a tempo 90 ms.
  - Tempo di attesa = Tempo di fine - Tempo di esecuzione totale = 90 - 45 = 45 ms.

Ordine di esecuzione

1. P1 (0-10 ms)
2. P2 (10-20 ms)
3. P3 (20-30 ms)
4. P1 (30-40 ms)
5. P2 (40-45 ms)
6. P3 (45-55 ms)
7. P1 (55-65 ms)
8. P3 (65-75 ms)
9. P3 (75-85 ms)
10. P3 (85-90 ms)

Tempi di attesa

- P1: 35 ms
- P2: 30 ms
- P3: 45 ms

Domanda 2 (3 punti)

Discutere le principali differenze tra linguaggi di programmazione compilati e linguaggi di programmazione interpretati, elencando i loro vantaggi e svantaggi. Inoltre, descrivere un esempio di applicazione concreta in cui utilizzare un linguaggio di programmazione compilato risulta essere una scelta migliore rispetto ad usare un linguaggio di programmazione interpretato, ed un esempio in cui invece utilizzare un linguaggio di programmazione interpretato risulta essere una scelta migliore rispetto ad usare un linguaggio di programmazione compilato.