Fondamenti di Informatica T-1 modulo 2

Laboratorio 10:

preparazione alla prova d'esame

Esercizio 1 - Gestione degli impegni

- Gli ***impegni giornalieri*** dei dipendenti di un'azienda devono essere aggiornati con una serie di nuove richieste
- Gli ***impegni già fissati sono memorizzati in un file di testo***, all'interno del quale ogni riga rappresenta gli impegni di un dipendente, secondo il seguente formato:
  - Codice numerico del dipendente (intero)
  - Nome del dipendente (al più di 20 caratteri), seguito da un ';'
  - Sequenza di impegni del dipendente, costituita da
    - Ora di inizio dell'impegno, con formato HH:MM (HH e MM interi), seguita da uno spazio
    - Ora di fine dell'impegno, con formato HH:MM, seguita da un ';' (a meno che non si tratti dell'ultimo impegno, in questo caso c'è un fine linea)

Gestione degli impegni
Requisiti generali

- Si fissi come ***ipotesi*** che la ***sequenza di impegni sia ordinata rispetto al tempo***, e che ogni lavoratore abbia un massimo di 20 impegni. Nota: ogni impegno ha un orario di inizio superiore o uguale alle 09:00 e un orario di fine inferiore o uguale alle 18:00
- Un secondo ***file binario*** contiene le informazioni relative ad ***ulteriori richieste di impegni da eseguire (possibilmente) in giornata***. Il primo campo del file riporta il numero di richieste contenute, seguito dalle varie richieste. Ogni richiesta è costituita dal ***codice del dipendente*** a cui si riferisce e da un intero rappresentate i ***minuti da dedicare*** per l'impegno richiesto

Parte 1
Strutture dati e relative operazioni

- Definire delle strutture atte a contenere i dati relativi a dipendenti, impegni e riunioni
- Il tipo **event** deve contenere i dati di un impegno (orario di inizio e di fine) sfruttando una seconda struttura dati chiamata **time** (contenente ore e minuti)
- Il tipo **worker** contiene i dati relativi ad un dipendente: codice numerico, nome e sequenza di impegni da modellare come un vettore di **event *allocato staticamente***
  - *Nota: ricordarsi che servirà memorizzare anche la dimensione logica di tale vettore*
- Infine, il tipo **request** memorizza i dati di una richiesta di impegno, ovvero codice del dipendente e numero di minuti

Parte 1
Strutture dati e relative operazioni

Dichiarare e definire le seguenti funzioni:

- **timeDifference** deve prendere in ingresso due orari e restituirne la differenza in minuti (valore assoluto, ricordandosi di effettuare le opportune conversioni minuti/ore… la differenza tra 10:20 e 8:50 è pari a 90 minuti)
- **getTranslatedTime** deve prendere in ingresso un orario e un valore in minuti e restituire l'orario che si ottiene aggiungendo i minuti dati all'orario (ricordarsi di effettuare le opportune conversioni minuti/ore)

Parte 2
Lettura dai file

- Realizzare una funzione **readWorkers** che prenda in input il nome del file di testo contenente i dati dei dipendenti, lo apra in lettura e ***legga i dati di tutti i dipendenti memorizzandoli in una lista di worker***
- Realizzare una funzione **readRequests** che prenda in input il nome del file binario contenente i dati delle richieste, lo apra in lettura e legga i dati di tutte le richieste memorizzandoli in un ***vettore di meeting allocato dinamicamente della dimensione strettamente necessaria***. Si ricorda che il numero di richieste è memorizzato come primo campo del file
- Realizzare due funzioni di stampa verificando la corretta lettura dei dati

Parte 3
inserimento richiesta

Realizzare una funzione **insertNewEvent** che prenda in input una richiesta di impegno e la lista di dipendenti, ottenendo il dipendente coinvolto e cercando di inserire la richiesta nella sua agenda. Si adotti la seguente politica per l'inserimento:

- la richiesta deve essere inserita ***prima possibile tra i vari impegni già presi dal dipendente*** (ovvero appena il dipendente ha un tempo libero capace di contenere il nuovo impegno), senza ovviamente travalicare i limiti della giornata lavorativa (09:00 – 18:00); al fine di individuare l'ammontare dei tempi liberi, si utilizzi la funzione **timeDifference** sviluppata al punto 1.
- se è possibile inserire la richiesta in agenda, la funzione si deve incaricare di creare effettivamente il nuovo impegno (l'impegno partirà o alle 09:00 o al tempo di fine dell'impegno immediatamente precedente) e di inserirlo nel vettore degli impegni del dipendente sfruttando **l'inserimento ordinato**.
- La funzione deve restituire un booleano che attesti se l'inserimento è stato effettuato o meno

Parte 4
Gestione delle richieste

- Realizzare una funzione **handleRequests** che prenda in input una lista di dipendenti e un vettore di richieste, cercando di inserirle nelle varie agende. A tal fine, si scandisca il vettore di richieste, utilizzando la funzione **insertNewEvent** per l'inserimento.
  - Se la richiesta non è stata inserita, si stampi a video un messaggio di errore del tipo "il dipendente XXX non ha tempo sufficiente per gestire un impegno di YYY minuti"
- Realizzare una funzione **printWorkers** che stampi su un file di testo una lista di dipendenti e relativi impegni, con lo stesso formato utilizzato in lettura.
- Nel main, si utilizzino le funzioni sviluppate per acquisire i dati in input, gestire le richieste e stampare su file la lista dei dipendenti dopo l'aggiornamento.

Esercizio 2: Prenotazione di pizze

- Una catena americana di pizzerie permette la prenotazione di pizze tramite il proprio sito Web
- Si deve realizzare un programma che, forniti un insieme di ordini di pizze e il listino dei prezzi dei singoli ingredienti, calcoli la spesa da fatturare ad ogni cliente. Le informazioni sono contenute in due differenti file, **ingredienti.txt** e **ordini.bin**.
- Il file di testo **ingredienti.txt** memorizza, in ogni riga,
  - nome dell'ingrediente (al più 24 caratteri, senza spazi)
  - prezzo di tale ingrediente (un float)

Prenotazione di pizze Requisiti generali

- Il file binario **ordini.bin** memorizza invece gli ordini:
  - all'inizio è salvato il numero di ordini presenti in tutto il file (int)
  - Di seguito, sono registrate ***tutte le pizze ordinate***: esattamente in questo ordine, per ogni pizza, sono salvati
    - nome del cliente che ha ordinato la pizza (al più 64 caratteri)
    - numero di ingredienti presenti su quella pizza (al più 10)
    - array di 10 elementi (non tutti usati) di ingredienti (composti dalla coppia nomeIngrediente-prezzo)
- Ovviamente il file ***può contenere più pizze ordinate dalla stessa persona*** e le liste degli ingredienti ivi registrati, anche se composte dalla coppia nomeIngrediente-prezzo, in realtà riportano in modo uniforme solo il nome dell'ingrediente: infatti il prezzo è soggetto a frequenti modifiche (fa testo in tal senso il solo listino prezzi ufficiale dato nel file precedente)

Parte 1
Lettura e scrittura degli ingredienti

- Il candidato realizzi un modulo per la ***gestione del listino degli ingredienti***
- In particolare, definisca una opportuna struttura dati **ingrediente**, per tenere traccia del nome di un ingrediente (al più 24 caratteri senza spazi) e del suo prezzo (double)
- Il candidato poi definisca:
  - insieme di primitive opportune per poter usare/gestire liste di strutture dati **ingrediente** (si faccia riferimento, a tal scopo, alle primitive sulle liste viste a lezione)
  - funzione **leggiIngredienti(…)** che, ricevuto in ingresso il nome di un file, apra il file e restituisca una lista di strutture dati **ingrediente** lette da tale file
  - funzione **scriviIngredienti(…)** che, ricevuto come parametri d'ingresso il nome di un file di testo e una lista di strutture dati **ingrediente**, scriva le informazioni relative agli ingredienti sul file indicato. In particolare, su ogni riga si scriva il nome dell'ingrediente e, separato da uno spazio, il suo prezzo
- Nel main(), testare il funzionamento del modulo utilizzando il file di testo dato

Parte 2
Modifica dei prezzi

- Il candidato estenda il modulo definito al punto precedente realizzando:
  - funzione **trovaPrezzo(…)** che, ricevuti in ingresso il nome di un ingrediente e una lista di strutture dati **ingrediente**, restituisca il prezzo di tale ingrediente
  - funzione **aggiornaPrezzo(…)** che, ricevuti in ingresso la lista di strutture dati **ingrediente**, il nome di un ingrediente e un nuovo prezzo, restituisca la lista di ingredienti dove il prezzo dell'ingrediente specificato è stato aggiornato al nuovo valore. A tal scopo, il candidato consideri di accedere alla lista usando la notazione a puntatori
- Nel main() il candidato modifichi la lista di ingredienti letti nell'esercizio precedente e controlli, salvando la lista aggiornata su un file, che tale modifica sia avvenuta con successo

Parte 3
Gestione delle pizze

- Il candidato realizzi un modulo per la ***gestione di una singola pizza***
- In particolare, si definisca una struttura dati **Pizza** contenente le seguenti informazioni (esattamente in questo ordine):
  - nome del cliente che ha ordinato la pizza (al più 64 caratteri)
  - numero di ingredienti (int)
  - array di strutture dati **ingrediente**, di dimensione statica di 10 elementi
- Tipicamente una pizza conterrà meno di 10 ingredienti: il numero di tali ingredienti è indicato nell'apposito campo
- Il candidato poi implementi una funzione **calcolaPrezzo(…)** che, ricevuti in ingresso una struttura dati di tipo **Pizza** e una lista di strutture dati **ingrediente**, calcoli il costo di tale pizza e lo restituisca come risultato della funzione

Parte 4
Gestione degli ordini

- Il candidato realizzi un modulo di gestione degli ordini, che sono registrati su un file binario (a titolo di esempio, si consideri il file **ordini.bin** dato)
- Su tale file viene salvato innanzitutto il numero di pizze ordinate (int) e a seguire delle strutture dati di tipo **Pizza**
- Uno stesso cliente può ordinare più pizze, che possono essere registrate nel file anche in ordine non consecutivo; il file può contenere gli ordini relativi a diversi clienti. Il candidato implementi:
  - funzione **leggiOrdini(…)** che, ricevuto in ingresso il nome di un file di ordini, apra tale file, legga il numero di pizze ivi registrate, allochi dinamicamente memoria sufficiente, e legga le strutture dati di tipo **Pizza** presenti in tale file. La funzione dovrà restituire un ***puntatore all'area di memoria allocata*** e, tramite un parametro passato per riferimento, il numero di pizze lette
  - funzione **sort(…)** che, utilizzando un algoritmo di ordinamento a scelta del candidato (tra quelli visti a lezione), ordini un array di strutture dati di tipo **Pizza** in base al nome del cliente che ha ordinato tale pizza

Parte 5
Calcolo della spesa dei clienti

- Il candidato, utilizzando le funzioni definite negli esercizi precedenti, provveda a leggere il listino degli ingredienti dal file **ingredienti.txt**, aggiorni il prezzo dell'ingrediente "salame_piccante" a 2.00 euro e salvi (sovrascrivendo il file originale) la nuova lista di prezzi nello stesso file
- Quindi il candidato legga dal file **ordini.bin** un array di pizze richieste da diversi clienti, ordini tale array in base al nome del cliente e calcoli per ogni cliente la spesa totale (data dalla somma del costo di ogni pizza richiesta dallo stesso cliente)

  Suggerimento: si sfrutti il fatto che l'array è ordinato proprio in base al nome del cliente e che, quindi, pizze richieste dalla stessa persona risulteranno essere adiacenti