IP a.a. 2020/19 - Esercitazione per l'esame

Prima di cominciare lo svolgimento leggete attentamente tutto il testo.

L'esame è un mini-progetto in cui sono dati alcuni elementi e voi dovete progettare ex novo tutto quello che manca per arrivare a soddisfare le richieste del testo.

Dovete realizzare le funzioni richieste, esattamente con la *segnatura* con cui sono indicate: nome, tipo restituito, tipo degli argomenti, nell'ordine. Potete inoltre realizzare altre funzioni in tutti i casi in cui lo ritenete appropriato.

Nel file zip trovate

- un file `contact.h` contenente le definizioni di tipo date e le intestazioni delle funzioni, sia quelle date, sia quelle che dovete implementare
- un file `main.cpp` contenente un main da usare per fare testing delle vostre implementazioni e la realizzazione (corpo) delle funzioni fornite da noi.
- un file `def` contenente una definizione di tipo di dato incompleta, che dovete realizzare voi e integrare con le eventuali funzioni di manipolazione
- un file `gpsdata.txt` contenente dati di prova
- un file `Cognome.cpp` contenente lo scheletro delle funzioni che dovete realizzare voi

Non dovete modificare i file `contact.h` e `main.cpp`.
Dovete modificare il file `def`.
Dovete modificare il file `Cognome.cpp` (al quale dovrete anche dare il vostro cognome). Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo a `contact.h`. Non siete però autorizzati a usare `algorithms`; negli esami del primo anno dovete dimostrare di essere in grado di programmare voi gli algoritmi.

Ricordatevi di procedere ordinatamente:

- create il file `<VostroCognome>.cpp`;
- definite in questo file e in `dbdef` **tutte** le funzioni richieste con un corpo minimo (ad esempio vuoto se il tipo di ritorno è `void`, o `return nullptr;` se la funzione restituisce un tipo puntatore).
- implementate una funzione alla volta, testatela e salvate il lavoro fatto, per esempio rinominando il file con un numero di versione progressivo e proseguendo poi sul file originale. In questo modo potete fare il "backtracking", tornare indietro a una versione funzionante se vi trovate in difficoltà con una modifica fatta.

Obiettivo

Vogliamo utilizzare i dati di una app di tracciamento dei contagi per identificare la posizione geografica di un focolaio di malattia infettiva (esempio: COVID-19).

Per fare questo vogliamo leggere un flusso di informazioni da un dispositivo GPS, in formato standard: il formato si chiama NMEA e il tipo di record che ci interessa si chiama GPRMC. Ogni record di informazioni (chiamato "sentence" nel gergo dello standard NMEA) corrisponde all'evento in cui qualcuno è venuto in contatto con un individuo infetto. Di tale evento sono riportate data, ora e posizione.

Il file "gpsdata.txt" contiene 15 record, uno per linea.

Tali record andranno letti da file; memorizzati in una struttura dati apposita; ordinati; stampati in ordine; infine andrà calcolata la posizione centrale, ovvero il baricentro di tutte le coordinate. Le coordinate del baricentro non sono altro che la media delle coordinate di tutti i record.

Il baricentro indica la posizione più probabile per il focolaio.

NOTA: L'applicazione di analisi della posizione non è realistica. Il formato dei dati GPS è realistico.

1 Struttura `Gprmc` (FORNITA)

La struct `Gprmc` contiene i dati letti da un record, ed è definita come segue:

```
struct Gprmc {
    // dati grezzi letti da file
    float lat;  // latitudine
    float lon;  // longitudine

    int time;   // orario
    int date;   // data

    // dati formattati
    int h;      // ore
    int m;      // minuti
    int s;      // secondi

    int gg;     // giorno
    int mm;     // mese
    int aaaa;   // anno
};
```

I membri `lat` e `lon` sono latitudine e longitudine in "gradi decimali" (così evitiamo i numeri in base 60). I membri `time` e `date` sono rispettivamente l'ora e il giorno dell'evento, codificati nelle cifre di un numero intero nel seguente modo:

Orario $\longrightarrow$ `HHMMSS`
Data $\longrightarrow$ `GGMMAA`

ovvero

- **per l'orario** le due cifre più a sinistra sono le ore; le due centrali sono i minuti; le due più a destra sono i secondi
- **per la data** le due cifre più a sinistra sono il giorno; le due centrali sono il mese; le due più a destra sono l'anno espresso a due cifre.

2 Funzione `readGpsRecord` (FORNITA)

La funzione `readGpsRecord` legge una linea e restituisce il suo contenuto in una struct `Gprmc`.

La funzione legge i primi quattro membri. Successivamente, data e ora andranno interpretate nelle loro componenti individuali, come indicato sopra, e usate per assegnare un valore ai restanti 6 membri (vedi specifiche più oltre).

3 Funzione `writeGpsRecord` (FORNITA)

```
void writeGpsRecord(Gprmc);
```

Riceve in ingresso una struct `Gprmc` e ne stampa il contenuto su `std::cout`.

La funzione stampa data e ora leggendo le loro componenti individuali (membri `gg mm aaaa h m s`), che quindi dovranno essere già state convertite.

4 Funzione `convDateTime` (DA FARE)

```
void convDateTime(Gprmc &);
```

Riceve in ingresso un riferimento a struct `Gprmc` e calcola e assegna i valori dei membri `gg mm aaaa` a partire dal membro `date`, e i valori dei membri `h m s` a partire dal membro `time`, secondo il criterio indicato sopra.

NOTA: L'anno deve essere espresso in 4 cifre sommando 2000.

5 Tipo di dato `ContactList` (DA FARE)

I record letti da file devono essere memorizzati in una struttura dati apposita. Definire il tipo di dato `ContactList` come lista collegata semplice, i cui elementi hanno come contenuto un record di tipo `Gprmc`, e tutte le funzioni necessarie a operare su di essa.

Il tipo, e i prototipi delle relative funzioni, vanno scritti nel file "def" che viene automaticamente incluso nello header "contact.h". In questo modo è come se venissero scritte direttamente nello header.

6 Funzione `leggiGps` (DA FARE)

```
ContactList leggiGps(std::string)
```

Apre il file di dati GPS il cui nome è passato come argomento; lo legge utilizzando `gpsReadRecord`, costruisce una `ContactList` con i dati letti, e la restituisce in uscita (dopo avere chiuso il file).

Sollevare una eccezione di tipo `std::string` se non è possibile aprire il file. Non è necessario catturare (`catch`) e trattare l'eccezione; il trattamento è già previsto nel programma principale (fornito).

7 Funzione `scriviGps` (DA FARE)

```
void scriviGps(ContactList)
```

riceve in ingresso una `ContactList`, e utilizzando `gpsWriteRecord` scrive su cout tutti i suoi record.

8 Funzione `dist` (DA FARE)

```
float dist(Gprmc r1, Gprmc r2);
```

Restituisce la distanza euclidea tra le coordinate di `r1` e le coordinate di `r2`.

La distanza euclidea tra i punti (x_1, y_1) e (x_2, y_2) è quella definita dal teorema di Pitagora:

$$d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

(non serve fare la radice quadrata).

9 Funzione `sort` (DA FARE)

```
void sort(ContactList &);
```

Ordina una `ContactList`, passata come argomento reference, in ordine di distanza. Ogni elemento della lista deve essere quello a distanza minima da quello precedente. Si parte dal primo elemento della lista, che resta quello che è all'inizio, e si ordinano i successivi.

Per l'ordinamento, confrontare gli elementi usando la funzione `dist`

10 Funzione `baricentro` (DA FARE)

```
void baricentro(ContactList c, int &latB, int & lonB);
```

Calcola la latitudine media e la longitudine media nella `ContactList c`, assegna i valori rispettivamente a `latB` e `lonB`.