INGEGNERIA DEL SOFTWARE
PROGRAMMAZIONE AVANZATA
ED ELEMENTI DI INGEGNERIA DEL SOFTWARE
21 Gennaio 2020

Name: _____ Surname: _____

Student ID: _____ MAIL: _____

ESERCIZIO 1

Considerate un programma che rappresenti varie caratteristiche e comportamenti di un'automobile.

Un'automobile può montare pneumatici di tipo `A` o `B`.

Il tipo `A` offre un'aderenza del 95% fino a una velocità di 70 Km/h, e dell'85% per velocità superiori.

Il tipo `B` offre un'aderenza del 98% fino a una velocità di 65 Km/h, e dell'87% per velocità superiori.

La funzione **`tire_adherence()`** deve restituire l'aderenza (valore numerico tra 0 e 1, per esempio 0.98 per indicare il 98%) per una data auto, a seconda del pneumatico montato.

Realizzate tutte le funzionalità necessarie a risolvere il problema, e scrivete un piccolo main che illustri le funzionalità.

Tenete presenti i seguenti punti
- Le funzionalità dovrebbero essere separate dal main(), in modo che differenti main() possano utilizzare le funzionalità a disposizione.
- Il programma dovrebbe essere modulare.
-Potrei avere più di una automobile nello stesso programma. Devo quindi poter distinguere una automobile dall'altra.
- Scegliete un design pattern specifico che agevoli la **progettazione in vista del cambiamento**. Per esempio potrei avere necessità di ampliare il numero di tipi di pneumatici a disposizione.
- Tenete presente che la funzione `tire_adherence()` (in altre versioni del programma) potrebbe calcolare l'aderenza utilizzando modelli più complicati che tengono conto anche del carico dell'auto, del tipo di strada, ecc.

Traccia di soluzione

Si applichi strategy pattern.
Si definisce una struttura Car che contenga, fra i campi, un puntatore a funzione che punta ad una delle possibili strategie concrete (`tire_adherence_A()`, `tire_adherence_B()`, ecc).
La funzione può accettare la velocità come parametro d'ingresso, oppure la velocità può essere memorizzata in un campo della struttura.

NB non utilizzate '**auto'** come nome della struttura o delle variabili, perché '**auto'** è una parola riservata c.

Esercizio 2

Mario è un pianificatore compulsivo e vuole pianificare le sue prossime letture. Gli amici gli suggeriscono di definire una lista di libri e di ordinarli secondo un criterio di interesse. **Le informazioni relative a ciascun libro saranno semplicemente il titolo ed un numero che rappresenta l'interesse provato da Mario**.

A Mario questa soluzione non piace perché il suo interesse per i libri nella lista cambierà man mano che va avanti nella lettura. Perché ordinare l'intero elenco, se poi dovrà cambiare l'ordinamento dopo la prima lettura?
[nb. qui 'lista' non è un termine tecnico che indica un tipo di dato, ma solo un termine usato per descrivere il problema dal punto di vista dell'utente]

Realizzate un programma che aiuti Mario in questo compito.

1) Il programma deve contenere una funzione **prossimoLibro()** che comunica a Mario quale sarà il prossimo libro, e lo elimini dalla struttura dati.

Esempio. Se i libri (e relativi punteggi) sono I Demoni (10), La valle dell'eden (25), Cecità (32), Saggio sulla lucidità (30), la funzione **prossimoLibro()** renderà il libro Cecità, che verrà eliminato dalla struttura dati. Una successiva chiamata alla funzione **prossimoLibro()** renderà Saggio sulla lucidità.

2) Il programma deve contenere anche una funzione **incrementaPunteggio**(i). Se Mario dovesse cambiare idea circa il libo i-esimo chiamerà la funzione **incrementaPunteggio**(i) che aumenta di 10 la preferenza di Mario per il libro *i*.

Esempio. Se La valle dell'Eden è identificato dall'indice 1, **incrementaPunteggio**(1) porterà a 35 il suo punteggio. Una successiva chiamata della funzione **prossimoLibro ()** renderà La valle dell'Eden.

Entrambe le funzioni devono riorganizzare la struttura dati in modo che la struttura dati sia pronta ad una nuova chiamata della funzione prossimoLibro ().

Si suppone che le strutture dati e le funzioni elencate qui sotto siano già a disposizione (le funzioni sono a disposizione su Moodle).

```
typedef struct {
    char * nome;
    int interesse;
} T_libro

parent();
left();
right();
build_max_heap();
max_heapify();
```

Occorre

- implementare le funzioni **prossimoLibro** () e **incrementaPunteggio** (i), e mostrare un main minimale che le utilizzi.
- Esaminare la complessità computazionale (anche solo in linea di massima) in modo da giustificare le scelte svolte

Traccia di soluzione

I dati vanno gestiti con un max-heap.

prossimoLibro () è una funzione standard di heap. Dopo aver estratto il primo elemento dell'heap, copia l'ultimo elemento in prima posizione, decrementa heap_size e ripristina nel modo ottimo la proprietà max-heap. Vedi slides per dettagli.

incrementaPunteggio (i) . Dopo aver incrementato il punteggio la proprietà max-heap potrebbe essere violata. Occorre ripristinarla considerando che per farlo è sufficiente confrontare il nodo `i` con il nodo `parent(i)`, se necessario scambiarli, e proseguire andando verso l'alto.
Applicare `max_heapify()` è errato e applicare `build_max_heap ()` è inutilmente costoso.