RELAZIONE ESERCITAZIONE 17/05/21

Studente : Salvatore Maria Mobilia
Matricola : X81000882

Indice :

(1) Introduzione
(2) Classi e metodi
(3) Risultati
(4) Errori
(5) ToDoNext
(6) Allegati

(1) Introduzione:
In data odierna , è stata assegnata alla classe una esercitazione.
La suddetta era incentrata sull' utilizzo della OOP e delle Data Structures. Ci veniva chiesto di implementare una Struttura dati in grado di poter immagazzinare al suo interno dei valori inizialmente situati su un file testuale dal nome "Punteggi.txt".
La mia implementazione si è incentrata sulla costruzione di una Lista Doppiamente Linkata che si avvale dell' utilizzo di due Nodi (che in questo caso Prendono il nome di Player) fittizi (di nomi header e trailer, così come la letteratura a proposito prevede), al cui interno ho inserito i collegamenti : header→setNext(trailer) & trailer→setPrev(header).
In questo modo i facilita la stesura del codice per l' inserimento e la cancellazione di un nodo all' interno della lista.

(2) Classi e metodi:

I) player.h → al suo interno ho settato come variabili private tre std::string : ossia nome,cognome e circolo; una di tipo double : punteggio (ho preferito settarla a double, poiché nel file di input non comparivano le "f" finali al numero,che indicano un tipo float, e quindi sarebbe stata considerata double di default) e una variabile intera : anno.
Il file è un file di header, quindi al suo interno ho semplicemente dichiarato le variabili e i metodi(che descriverò più avanti) per poi sviluppare il codice nel file player.cpp.
II) player.cpp → al suo interno si trova l' implementazione reale della classe Player definita in player.h. I metodi presenti sono i metodi set (per settare i vari puntatori a valore necessari per la mia implementazione della lista con sentinelle), i metodi get (restituiscono i valori puntati dai puntatori), e l' overloading degli operatori :
Operator < : restituisce un valore booleano relativo al confronto fra il valore del punteggio acquisito da un player passato come riferimento e il valore del punteggio acquisito da this→player (semplicemente this).
Operator >= restituisce un valore booleano relativo al confronto fra il valore del punteggio acquisito da un player passato come riferimento e il valore del punteggio acquisito da this→player (semplicemente this).
Operator = : setta i valori di un player passato come parametro, uguali a quelli di this.
Operator << : restituisce un semplice listato dei valori contenuti all' interno

della classe con una semplice formattazione.
Operator >> : raccoglie da file di input i vari valori necessari per la costruzione dell' oggetto ,facendo attenzione a "tokenizare" la linea di input(Malgrado i miei sforzi non è funzionante, anche se la stampa dei valori risulta corretta : MustFix!).

Costruttori :
Default (necessario per header e trailer) : setta tutti i puntatori a valore a NULL , compresi il next e il prev del Player.
Con parametri: imposta tutti i puntatori a valori uguali a quelli forniti come parametri attuali, creando per ognuno un puntatore con l' utilizzo dell' operatore new(). In questo caso next e prev sono settati a NULL, per poter modificare i collegamenti solo in fase di costruzione della lista.

III) list.h → al suo interno si trovano le definizioni di variabili e metodi, implementati in list.cpp. Classe Template per poter immagazzinare ogetti custom. In essa ho definito un valore intero (n) che mi indica il numero totale di elementi nella lista,
tre puntatori a Nodo (Player *header,*trailer,*current) , un valore booleano (order) che indica alla lista il verso di ordinamento degli elementi.

IV) list.cpp → al suo interno si trova l' implementazione reale della classe List definita in list.h.
i)La classe List prevede in primis due metodi ausiliari di ricerca, inseriti nella sezione privata , che dipendono dal valore booleano (order).I metodi restituiscono il Player che possiede il valore del punteggio passato come parametro formale (NULL se non esiste all' interno della lista).
ii)La classe List prevede un Costruttore che prende come parametro un solo valore, order . Header verrà settato col costruttore di default di Player, stessa cosa vale per trailer (poi vengono inseriti all' interno del costruttore stesso i collegamenti this→header→setNext(this→trailer) e this→trailer→setPrev(this→header)), n è settato inizialmente a 0 e current a NULL.
iii)La classe List prevede un metodo per l' inserimento di un Nodo(Player) nella struttura. L' inserimento prevede come parametri: il nome , il cognome ,il circolo, il punteggio e l' anno del Player, creando nel corpo stesso del metodo , attraverso l' utilizzo dell' opeatore new(), il Nodo(Player) che ingloberà i valori di cui sopra, e dopo setterà i collegamenti relativi (in base a this→order), scorrendo la lista partendo da header→getNext(). Il tipo di ritorno è un riferimento alla lista, in modo tale da permettere inserimenti "a cascata".
iv) La classe List prevede un metodo per la cancellazione del player. La cancellazione prevede un solo parametro in input , il punteggio del Player, prevedendo a ottenere il nodo attraverso la funzione ausiliaria di ricerca(_searchT() se this→order == true, _searchF() se this→order == false).l tipo di ritorno è un riferimento alla lista, in modo tale da permettere

cancellazioni “a cascata”.
v) La lista prevede un metodo per la ricerca di un Player al suo interno (per l’utente), il cui valore di ritorno è un intero(booleano) che si ottiene dal confronto fra _searchT() oppure _searchF() e NULL.
vi) La lista prevede un set di tre metodi reset,hasNext e next che fanno da iteratore per la lista (usati per l’overloading dell’operatore <<) e quindi per scorrerla dal primo fino all’ultimo elemento (rispettivamente header→getNext() e trailer→getPrev()).
Vii) La lista prevede un metodo di controllo size “isEmpty()” che mi restituisce il valore del confronto fra this→n e 0.
viii) La lista prevede come plus (non è friend, ma esterno alla lista) un metodo per l’overloading dell’operatore << che sfrutta i tre metodi iteratori. Formatta l’output relativo alla lista sfruttando l’overloading dell’operatore << relativo alla classe Player.
ix)La lista prevede un metodo printCognome() che, preso in input un cognome, lo ricerca all’interno della lista e , se presente , ne stampa le informazioni.
x)La lista prevede un metodo cancAllPunteggio() , che preso in input un punteggio, cancella tutti i Players con quel valore dalla lista.

V) main.cpp → al suo interno sono definite e implementate , oltre che al main , due funzioni :
i) getNumLines() : restituisce il numero di linee presenti in un file passato come parametro.
ii) getData() : template → prende in input un file e una lista e crea la lista a partire dai dati contenuti all’interno del file.

(3) Risultati :

Le classi , a meno di qualche imprecisione (vedi (4) Errori) , stampano correttamente (seguendo i parametri bool relativi) i valori dei players presenti nella lista.

(4) Errori:
Nel file Player.cpp è presente l’implementazione relativa all’overloading dell’operatore >> . Se si prova ad utilizzare questo metodo , però , si incappa in un segmentation fault. I valori , debaggati stampando i vari tokens delle stringhe, risultano precisi (fase di fixing)

(5) ToDoNext
Bisogna fixare la questione relativa a (4), anche se il tutto funziona lo stesso.

(6) Allegati:
Come allegati, oltre ai file sorgente , vi sono sei immagini formato “.png” che attestano la corretta visualizzazione delle liste. Le prime tre sono relative alle liste ordinate in ordine crescente, le altre tre sono relative

alle liste ordinate in ordine decrescente.