Libro.h

mercoledì 28 giugno 2023 21:53

```
#include <iostream>
using namespace std;
#ifndef LIBRO_H
#define LIBRO_H
class Libro{
    protected:
        string titolo;
        string autore;
        int anno;
        virtual void Stampa(ostream &os) const = 0; //la utilizzo solo per
l'overloading dell'operatore <<
    public:
        Libro(string _titolo, string _autore, int _anno) :
            titolo(_titolo), autore(_autore), anno(_anno){};

        virtual ~Libro(){};
        string getTitolo() const{ return this->titolo;}
        string getAutore() const{ return this->autore;}
        int getAnno() const{ return this->anno;}

        virtual string genere() const = 0;
        virtual void stampa() const = 0;
        friend ostream &operator <<(ostream &os, const Libro &lb);

};
bool operator <(const Libro& l1, const Libro& l2){
    return l1.getTitolo() < l2.getTitolo();
}
/*
bool operator <(const string &s, const Libro& l2){
    return s < l2.getTitolo();
}*/
bool operator==(const Libro& l1, const Libro& l2){
    return l1.getTitolo() == l2.getTitolo();
}
/*bool operator==(const string &s, const Libro& l2){
    return s == l2.getTitolo();
}*/

ostream &operator <<(ostream &os, const Libro &lb){
        lb.Stampa(os);
        return os;
}
class Romanzo : public Libro{
    private:
        void Stampa(ostream &os) const;
    public:
        Romanzo(string _titolo, string _autore, int _anno)
            : Libro(_titolo, _autore, _anno){};
        virtual ~Romanzo(){};
        string genere() const{ return "romanzo";}
        void stampa() const;
};
void Romanzo::stampa() const{
    cout << "Titolo: " << getTitolo() <<
```

```
    "\tAutore: " << getAutore() << "\tAnno pubblicazione: " << getAnno()
    << "\tGenere: " << genere() << endl;
}
void Romanzo::Stampa(ostream &os) const{
    os << "Titolo: " << getTitolo() <<
    "\tAutore: " << getAutore() << "\tAnno pubblicazione: " << getAnno()
    << "\tGenere: " << genere() << endl;
}


class Saggio : public Libro{
    private:
        void Stampa(ostream &os) const;
    public:
        Saggio(string _titolo, string _autore, int _anno)
            : Libro(_titolo, _autore, _anno){};
        virtual ~Saggio(){};
        string genere() const{ return "saggio";}
        virtual void stampa() const;
};
void Saggio::stampa() const{
    cout << "Titolo: " << getTitolo() <<
    "\tAutore: " << getAutore() << "\tAnno pubblicazione: " << getAnno()
    << "\tGenere: " << genere() << endl;
}
void Saggio::Stampa(ostream &os) const{
    os << "Titolo: " << getTitolo() <<
    "\tAutore: " << getAutore() << "\tAnno pubblicazione: " << getAnno()
    << "\tGenere: " << genere() << endl;
}

#endif
```

BST template

mercoledì 28 giugno 2023 21:54

```
#include <iostream>
#include "libro.h"
using namespace std;
#ifndef BST_H
#define BST_H

template <typename T = Libro*>
class Nodo{
    public:
        T dato;
        Nodo<T>* left;
        Nodo<T>* right;
        Nodo<T>* padre;
        Nodo(T el) : dato(el), left(nullptr), right(nullptr), padre(nullptr)
{};
};

template <typename T = Libro*>
class BST{
    private:
        Nodo<T>* radice;
        void inOrder(Nodo<T>* ptr) const;
        Nodo<T>* ricerca(Nodo<T>* x, T el) const;
        void trapianta(Nodo<T>*u, Nodo<T>* v); //-> per cancellazione
        Nodo<T>* minimo(Nodo<T>* x) const; //->per cancellazione
        void cancella(Nodo<T>* z);


    public:
        BST() : radice(nullptr){};
        ~BST(){}; //IMPLEMENTA

        void inOrder() const; //funzione di interfaccia
        void inserisci(T el);
        Nodo<T>* ricerca(T el) const; //func di interfaccia
        //wrapper cancellazione
        bool cancella(T el);

};
template <typename T>
void BST<T>::inOrder(Nodo<T>* ptr) const{

    if(ptr != nullptr){
        inOrder(ptr->left);
        cout << *ptr->dato;
        inOrder(ptr->right);
    }
}
template <typename T>
void BST<T>::inOrder() const{
    inOrder(this->radice);
}
template <typename T>
void BST<T>::inserisci(T el){
    Nodo<T>* nuovo = new Nodo<T>(el);
```

```
    Nodo<T>* y = nullptr;
    Nodo<T>* x = this->radice;

    while( x!= nullptr){
        y = x;
        if(*el < *x->dato){
            x = x->left;
        }
        else{
            x = x->right;
        }
    }

    nuovo->padre = y;
    if(y == nullptr){
        radice = nuovo;
    }
    else if(*el < *y->dato){
        y->left = nuovo;
    }
    else{
        y->right = nuovo;
    }
}

template <typename T>
Nodo<T>* BST<T>::ricerca(Nodo<T>* x, T el) const{

    if(x == nullptr || *x->dato == *el){
        return x;
    }
    if(*el < *x->dato){
        return ricerca(x->left, el);
    }
    else{
        return ricerca(x->right, el);
    }
}
template <typename T>
Nodo<T>* BST<T>::ricerca(T el) const{
    return ricerca(this->radice, el);
}
template <typename T>
void BST<T>::trapianta(Nodo<T>*u, Nodo<T>* v){
    if(u->padre == nullptr){
        radice = v;
    }
    else if(u->padre->left == u){
        u->padre->left = v;
    }
    else{
        u->padre->right = v;
    }
    if(v!= nullptr){
        v->padre = u->padre;
    }

}
template <typename T>
Nodo<T>* BST<T>::minimo(Nodo<T>* x) const{

    if(x == nullptr){
```

```
        return nullptr;
    }

    while( x->left!= nullptr){
        x = x->left;
    }
    return x;

}

template <typename T>
void BST<T>::cancella(Nodo<T>* z){
    Nodo<T>* y;
    if(z->right == nullptr){
        trapianta(z, z->left);
    }
    else if(z->left == nullptr){
        trapianta(z, z->right);
    }
    else{
        y = minimo(z->right);
        if(y->padre != z){ //se y non è figlio diretto di z devo prima
sistemare la gerarchia del sottoalbero che ha come radice y
            trapianta(y, y->right);
            y->right = z->right;
            y->right->padre = y;
        }
        trapianta(z, y);
        y->left = z->left;
        y->left->padre = y;
    }
    delete z;
}
template <typename T>
bool BST<T>::cancella(T el){
    Nodo<T>* temp = ricerca(el);
    if(temp == nullptr){
        return false;
    }
    else{
        cancella(temp);
        return true;
    }
}

#endif
```

Testo

mercoledì 28 giugno 2023 21:51

PROVA D'ESAME 1

Si consideri una classe virtuale Libro che rappresenta un libro.

La classe ha i seguenti attributi privati:

- *titolo, autore, anno*

La classe ha i seguenti metodi pubblici:

- *costruttore, get degli attributi, stampa*
- *un metodo virtuale puro chiamato genere che restituisce il genere del libro*

Si considerino due classi derivate da Libro: Romanzo e Saggio.

Ognuna di queste classi implementa il metodo genere in modo appropriato per il tipo di libro.

Si consideri una classe template AlberoBinario<T> che rappresenta una struttura dati ad albero binario di ricerca di elementi di tipo T.

La classe ha i seguenti attributi privati:

- *radice: un puntatore al nodo radice dell'albero*
- *dimensione: il numero di nodi nell'albero*

La classe ha i seguenti metodi pubblici:

- *costruttore, distruttore*
- *inserisci: inserisce un nuovo elemento nell'albero in base al suo valore*
- *rimuovi: rimuove un elemento dall'albero dato il suo valore e restituisce true se l'elemento è stato trovato e rimosso, altrimenti restituisce false*
- *cerca: cerca un elemento nell'albero dato il suo valore e restituisce un puntatore all'elemento*
- *se trovato, altrimenti restituisce nullptr*
- *stampa: stampa tutti gli elementi nell'albero in ordine crescente dei loro valori*

Si scriva un programma che crea un albero binario di puntatori a libri e inserisce

nell'albero i seguenti libri:

- *Il nome della rosa, autore Umberto Eco, anno 1980, tipo Romanzo*
- *Il mondo nuovo, autore Aldous Huxley, anno 1932, tipo Romanzo*
- *Breve storia del tempo, autore Stephen Hawking, anno 1988, tipo Saggio*
- *Orgoglio e pregiudizio, autore Jane Austen, anno 1813, tipo Romanzo*
- *Il gene egoista, autore Richard Dawkins, anno 1976, tipo Saggio*
- *Il signore degli anelli, autore J.R.R. Tolkien, anno 1954, tipo Romanzo*

Il programma poi chiede all'utente di inserire il titolo di un libro da rimuovere dall'albero e stampa un messaggio di conferma se il libro è stato rimosso con successo, altrimenti stampa un messaggio di errore. Il programma poi stampa tutti i libri rimanenti nell'albero usando il metodo stampa della classe AlberoBinario.

Output atteso inserendo il titolo Il mondo nuovo:

Inserisci il titolo di un libro da rimuovere: Il mondo nuovo

Il libro Il mondo nuovo è stato rimosso con successo dall'albero.

Libri rimanenti nell'albero:

Breve storia del tempo - Stephen Hawking - 1988 - Saggio

Il gene egoista - Richard Dawkins - 1976 - Saggio

Il nome della rosa - Umberto Eco - 1980 - Romanzo

Il signore degli anelli - J.R.R. Tolkien - 1954 - Romanzo

Orgoglio e pregiudizio - Jane Austen - 1813 – Romanzo

Output atteso inserendo il titolo Harry Potter:

Inserisci il titolo di un libro da rimuovere: Harry Potter

Nessun libro con questo titolo trovato nell'albero.

main.cpp

mercoledì 28 giugno 2023 21:54

```cpp
#include <iostream>
#include <string>
#include "libro.h"
#include "BST.h"
using namespace std;
//Nota: nella mia implementazione, inserisco i Libri nel BST template in
funzione
//del loro titolo.
int main(){
    Libro* r1 = new Romanzo("Il nome della rosa", "Umberto Eco", 1980);
    Libro* r2 = new Romanzo("Il mondo nuovo","Aldous Huxley",1932);
    Libro* s1 = new Saggio("Breve storia del tempo", "Stephen Hawking", 1988);
    Libro* r3 = new Romanzo("Orgoglio e pregiudizio", "Jane Austen", 1813);
    Libro* s2 = new Saggio("Il gene egoista", "Richard Dawkins", 1976);
    Libro* r4 = new Romanzo("Il signore degli anelli", "J.R.R. Tolkien",
1954);
    // r1->stampa();
    // s2->stampa();
    // cout << *r1;
    // cout << *s2;
    BST alberello; // T = Libro* in automatico
    alberello.inserisci(r1);
    alberello.inserisci(r2);
    alberello.inserisci(s1);
    alberello.inserisci(r3);
    alberello.inserisci(s2);
    alberello.inserisci(r4);
    cout << "STAMPA RACCOLTA LIBRI ORDINATI SECONDO L'ANNO: " << endl;
    alberello.inOrder();
    int risp;
    cout << "INSERISCI L'ANNO DEL LIBRO CHE VUOI RIMUOVERE: ";
    cin >> risp;
    //so che è un brutto modo ma non saprei come altro fare perchè STO
IMPAZZENDO VISTO CHE STO CAZZO DI ALBERO E' TEMPLATE NON POSSO FARE NU CAZZ
    //Cerco l' anno tra i Libri che ho creato e poi se lo trovo inserisco il
risultato in tmp
    Nodo<Libro*>* tmp;
    if(risp == r1->getAnno()){
        tmp = alberello.ricerca(r1);
    }
    else if(risp == r2->getAnno()){
        tmp = alberello.ricerca(r2);
    }
    else if(risp == s1->getAnno()){
        tmp = alberello.ricerca(s1);
    }
    else if(risp == r3->getAnno()){
        tmp = alberello.ricerca(r3);
    }
    else if(risp == s2->getAnno()){
        tmp = alberello.ricerca(s2);
    }
    else if(risp == r4->getAnno()){
        tmp = alberello.ricerca(r4);
    }
```

```
    else{
        cout << "Libro non esistente, spiaze" << endl;
    }

    //controllo se il nodo e' effettivamente contenuto nell'alberello. Se c'è,
lo elimino
    if(tmp != nullptr){
        alberello.cancella(tmp->dato);
        cout << "Il libro " << tmp->dato->getTitolo() << " e' stato rimosso
con successo!" << endl;
    }
    else{
        cout << "Libro non trovato nell'alberello." << endl;
    }
    cout << "\n\nLIBRI RIMANENTI NELL'ALBERO: \n";
    alberello.inOrder();


    return 0;
}
```