Dipendente.h

mercoledì 28 giugno 2023 21:37

```
#include <iostream>
#include <string>
using namespace std;
class Dipendente{
protected:
    string cod_fiscale;
    int giorno;
    int mese;
    int anno;
    int paga_base;

    virtual void stampa(ostream &os) const{
        os << "CODICE FISCALE: " << cod_fiscale
        << "\tDATA DI NASCITA: " << giorno << "/"
        << mese << "/" << anno << "\tPAGA BASE: " << paga_base << endl;
    }
public:
    Dipendente(string _cod_fiscale, int _giorno, int _mese, int _anno, int
_paga_base) :
    cod_fiscale(_cod_fiscale), giorno(_giorno), mese(_mese), anno(_anno),
paga_base(_paga_base){};
    virtual ~Dipendente(){};  // virtual destructor
    int getPagaBase() const{return paga_base;}
    friend ostream &operator <<(ostream &os, const Dipendente& dip){
        dip.stampa(os);
        return os;
    }
};
class DipendenteOccasionale : public Dipendente{
protected:
    string mese1;
    string mese2;
    string mese3;
    int paga_extra;
    void stampa(ostream& os) const override{
        Dipendente::stampa(os);
        os << "TRIMESTRE LAVORATIVO: "<< mese1 << "/" << mese2 << "/" << mese3
        << "\tPAGA EXTRA: " << paga_extra <<" euro.\n" <<endl;
    }
public:
    DipendenteOccasionale(string _cod_fiscale, int _giorno, int _mese, int
_anno, int _paga_base,
    string _mese1, string _mese2, string _mese3, int _paga_extra)
    : Dipendente(_cod_fiscale,_giorno, _mese, _anno, _paga_base),
    mese1(_mese1), mese2(_mese2), mese3(_mese3), paga_extra(_paga_extra){};
    ~DipendenteOccasionale() override{};
};
class Azienda{
protected:
    string nome;
    string partita_iva;
    Dipendente** ptr_dip;
    int dim_azienda = 10;
public:
    Azienda(string _nome,string _partita_iva, Dipendente** _ptr_dip) :
```

```
    nome(_nome),partita_iva(_partita_iva), ptr_dip(_ptr_dip){};
    int getDimAzienda() const{ return this->dim_azienda;}
    friend ostream &operator <<(ostream &os, const Azienda &az){
        os << "NOME AZIENDA: " << az.nome << "\tPARTITA IVA: "
<< az.partita_iva
        << "\nDIPENDENTI AZIENDA: " << endl;

        for(int i = 0; i < az.dim_azienda && az.ptr_dip[i] != nullptr; i++){
            os << i << ")" << *az.ptr_dip[i] << endl;
        }
        return os;
    }
    void aggiungiDipendenti(Dipendente* dip, int& index){
        if (index < dim_azienda) {
            ptr_dip[index] = dip;
            index++;
        }
    }
    void calcolaStipendi(int &index){
        long stipendio = 0;
        for(int i = 0; i < index ; i++){
            stipendio += ptr_dip[i]->getPagaBase();
        }
        cout << "stipendio: " <<  stipendio << endl;
    }
};
```

main.cpp

mercoledì 28 giugno 2023 21:44

```
#include <iostream>
#include "Dipendente.h"
using namespace std;
int main(){
    Dipendente *dip1 = new Dipendente("AAABBB80T21C351Y", 02, 02, 1980, 1000);
    Dipendente *dip2 = new Dipendente("CCCDDD82C21C351Y", 21, 4, 1982, 1200);
    Dipendente *dip3 = new Dipendente("EEEFFF87C28C351Y", 29, 8, 1987, 1200);
    Dipendente *occ1 = new DipendenteOccasionale("HHH81R23C351Y", 29, 8, 1981,
800, "Gennaio", "Febbraio", "Marzo", 300);
    int dim = 10;
    int index = 0;
    Dipendente* arr[dim] = {nullptr};
    Azienda az("AZIENDA A", "1234454545454", arr);
    az.aggiungiDipendenti(dip1, index);
    az.aggiungiDipendenti(dip2, index);
    az.aggiungiDipendenti(dip3, index);
    az.aggiungiDipendenti(occ1, index);
    cout << az;
    az.calcolaStipendi(index);
    // Delete the dynamically allocated objects
    for (int i = 0; i < index; ++i) {
        delete arr[i];
    }
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 21:44

ESAME PROG2 O-Z

Si scriva una classe **Dipendente** che permetta di rappresentare un generico dipendente di un'azienda. Tale classe deve contenere una stringa rappresentante il codice fiscale (univoco per ogni dipendente), ed una data di nascita (implementazione arbitraria a cura dello studente).

Dotare, quindi, la classe dei seguenti metodi:

- Un **costruttore con parametri**;
- Metodi **get e set** per gli attributi;
- **L'overloading dell'operatore operator<<,** che permetta di stampare Su standard output le informazioni sul Dipendente.

Estendere la classe **Dipendente** definendo una sottoclasse:

- La classe **Dipendente Occasionale** che ha assegnato un trimestre lavorativo come attributo (implementazione arbitraria a cura dello studente) ed un intero che rappresenta la paga extra.

Definire, quindi, una classe **Azienda**, che abbia come dati:

- Una **stringa** rappresentante la partita iva dell'azienda;
- Una **stringa** rappresentante il nome;
- Un **array di puntatori** a dipendenti creati dinamicamente;

Dotare la classe dei seguenti metodi:

- Un **costruttore con parametri**.
- **L'overloading dell'operatore<<,** che permetta di stampare su standard input le info sull'azienda nome, partita iva e lista dei dipendenti (con specifiche sui Dipendenti Occasionali).
- Un metodo **aggiungiDipendente()** che aggiunge un dipendente nell'array
- Un metodo **calcolaStipendi()** che prende in input il mese e restituisce la somma degli stipendi da pagare
- Un metodo **calcolaPremio()** che prende in input un intero età minima ed un intero anno e restituisce il valore del premio da assegnare ai dipendenti. Il metodo calcola la somma della paga base dei suoi dipendenti (non occasionali) con età maggiore di età minima, e ne restituisce il 5% come valore premio.

Si noti che per poter aggiungere un dipendente è necessario che ci sia spazio sufficiente nell' array. Si noti inoltre, che la classe Azienda fa uso della composizione: al suo interno, infatti, utilizza la classe Dipendente.

Per cui, dove necessario, è consigliato fare uso dei metodi della classe Dipendente e dei metodi della classe Azienda.

Ad esempio, nel metodo per la stampa della classe Azienda è bene utilizzare il metodo per la stampa della classe Dipendente per stampare su standard output i singoli dipendenti.

Testare il tutto con un main() che inserisca un Azienda e calcoli gli stipendi di marzo, giugno, settembre e dicembre (calcolo per mese, non per trimestre), ed il premio per i dipendenti con almeno 40 anni nel 2023:

AziendaA, partita iva 12345678901, e la seguente lista dei dipendenti:

- Dip. AAABBB80T21C351Y, 02/02/1980 paga base 1000

- Dip. CCCDDD82C21C351Y, 21/04/1982 paga base 1200

- Dip EEEFFF87C28C351Y, 29/08/1987 poga base 1200

- Occ, HHH81R23C351Y, 29/08/1981 paga base 800, paga extra 300, trimestre 1.

PRODOTTO VENDITORI TUTTO

mercoledì 28 giugno 2023 21:20

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <algorithm>
using namespace std;
class Prodotto {
    int codice;
    double prezzo;
public:
    Prodotto(): codice(0), prezzo(0.0) {}
    Prodotto(int cod, double pre): codice(cod), prezzo(pre) {}
    int getCodice() const {
        return codice;
    }
    void setCodice(int cod) {
        codice = cod;
    }
    double getPrezzo() const {
        return prezzo;
    }
    void setPrezzo(double pre) {
        prezzo = pre;
    }
    void leggi() {
        cout << "Inserisci codice prodotto: ";
        cin >> codice;
        cout << "Inserisci prezzo prodotto: ";
        cin >> prezzo;
    }
    friend ostream& operator<<(ostream& os, const Prodotto& prod);
};
ostream& operator<<(ostream& os, const Prodotto& prod) {
    os << "Codice Prodotto: " << prod.codice << ", Prezzo: " << prod.prezzo;
    return os;
}
class Venditore {
    int codice;
    vector<Prodotto> prodotti;
public:
    Venditore(): codice(0) {}
    Venditore(int cod): codice(cod) {}
    Venditore(const Venditore& v): codice(v.codice), prodotti(v.prodotti) {}
    int getCodice() const {
        return codice;
    }
    void setCodice(int cod) {
        codice = cod;
    }
    void leggi() {
        cout << "Inserisci codice venditore: ";
        cin >> codice;

        int numProdotti;
        cout << "Inserisci numero di prodotti: ";
```

```
        cin >> numProdotti;

        for (int i = 0; i < numProdotti; ++i) {
            Prodotto prod;
            prod.leggi();
            prodotti.push_back(prod);
        }
    }
    friend ostream& operator<<(ostream& os, const Venditore& vend);
    double calcolaPrezzoTotale() const {
        double totale = 0;
        for (Prodotto prod : prodotti) {
            totale += prod.getPrezzo();
        }
        return totale;
    }
    void aggiungiProdotto(Prodotto prod) {
        prodotti.push_back(prod);
    }
};
ostream& operator<<(ostream& os, const Venditore& vend) {
    os << "Codice Venditore: " << vend.codice << "\n";
    os << "Prodotti:\n";
    for (Prodotto prod : vend.prodotti) {
        os << prod << "\n";
    }
    return os;
}
struct VenditoreInfo {
    int codice;
    double prezzoTotale;
    VenditoreInfo(int cod, double prezzo): codice(cod), prezzoTotale(prezzo)
{}
    bool operator<(const VenditoreInfo& other) const {
        return prezzoTotale < other.prezzoTotale;
    }
    bool operator>(const VenditoreInfo& other) const {
        return prezzoTotale > other.prezzoTotale;
    }
};
int main() {
    ifstream input("prodotti.txt");
    vector<Venditore> venditori;
    if (input.is_open()) {
        string line;
        while (getline(input, line)) {
            Venditore vend;
            vend.setCodice(stoi(line));
            getline(input, line);
            int numProdotti = stoi(line);
            for (int i = 0; i < numProdotti; ++i) {
                getline(input, line);
                stringstream ss(line);
                int cod;
                double prezzo;
                char comma;
                ss >> cod >> comma >> prezzo;
                Prodotto prod(cod, prezzo);
                vend.aggiungiProdotto(prod);
            }
            venditori.push_back(vend);
        }
```

```
    } else {
        cout << "Impossibile aprire il file\n";
    }
    vector<VenditoreInfo> infoVenditori;
    for (Venditore vend : venditori) {
        infoVenditori.push_back(VenditoreInfo(vend.getCodice(),
vend.calcolaPrezzoTotale()));
    }
    cout << "Ordinare i venditori in ordine crescente o decrescente? (c/d): ";
    char scelta;
    cin >> scelta;
    if (scelta == 'c') {
        sort(infoVenditori.begin(), infoVenditori.end());
    } else if (scelta == 'd') {
        sort(infoVenditori.begin(), infoVenditori.end(),
greater<VenditoreInfo>());
    }
    for (VenditoreInfo info : infoVenditori) {
        cout << "Codice Venditore: " << info.codice << ", Prezzo Totale: "
<< info.prezzoTotale << "\n";
    }
    return 0;
}
```

file input prodotti.txt

mercoledì 28 giugno 2023 21:24

```
12345
3
101,12.99
102,19.99
103,14.99
67890
2
104,9.99
105,15.99
54321
1
106,24.99
```

Testo

mercoledì 28 giugno 2023 21:26

Prodotti e venditori (esercitazione)

Definire una classe ***Prodotto*** che permetta di rappresentare un generico prodotto da vendere. Tale classe deve contenere almeno: un intero rappresentante il codice del prodotto, ed un double rappresentante il prezzo (in euro).

Dotare, quindi, la classe dei seguenti metodi:

- Un costruttore senza parametri (per default, impostare tutti i dati a zero);
- Un costruttore con due parametri (il codice ed il prezzo);
- Metodi get e set per il codice ed il prezzo;
- L'overloading dell'operatore operator<<, che permetta di stampare su standard output le informazioni su un prodotto.

Può essere utile definire un metodo void leggi() che permetta di leggere da standard input le informazioni riguardanti un prodotto, in modo tale da creare a runtime delle istanze di Prodotto.

Estendere la classe prodotto definendo due sottoclassi:

- La classe ProdottoAlimentare aggiunge una data di scadenza ed un metodo *verifica()* che restituisce true solo se il prodotto è ancora buono, altrimenti genera false se è scaduto.
- La classe ProdottoPerBambini aggiunge la fascia d'età come attributo. Le fasce disponibili sono 0-3, 3-6, 6-9, 9-12.

Definire, quindi, una classe ***Venditore***, che abbia come dati:

- Un intero, rappresentante il codice del venditore;
- Un array di lunghezza variabile contenente i prodotti, rappresentante l'elenco dei prodotti venduti;
- Un intero, rappresentante il numero di prodotti nell'array;

Dotare la classe dei seguenti metodi:

- Un costruttore senza parametri (per default, impostare tutti i dati a zero);
- Un costruttore con un parametro (il codice);
- Il costruttore di copia (facoltativo);
- L'overloading dell'operatore operator<<, che permetta di stampare su standard input le informazioni sul venditore: il codice e l'elenco dei prodotti venduti;
- Un metodo **double calcolaPrezzoTotale()** che restituisca la somma dei prezzi di tutti i prodotti del venditore;
- Un metodo **double calcolaValoreMagazzino()** che restituisca la somma dei prezzi dei prodotti del venditore escludendo quelli scaduti;
- Un metodo **void aggiungiProdotto()** che permetta di aggiungere un prodotto nell'elenco dei prodotti.

Anche in questo caso può essere utile un metodo leggi() per agevolare l'inserimento di un nuovo venditore.

Si noti che per poter aggiungere un prodotto è necessario che ci sia spazio sufficiente nell'array. Si noti, inoltre, che la classe Venditore si fa uso della composizione: al suo interno, infatti, utilizza la classe Prodotto. Per cui, dove necessario, è consigliato fare uso dei metodi della classe Prodotto nei metodi della classe Venditore. Ad esempio, nel metodo per la stampa della classe Venditore è bene utilizzare il metodo per la stampa della classe Prodotto per stampare su standard output i singoli prodotti.

Infine, realizzare un main() che crea un certo numero di venditori (almeno 3) ciascuno con almeno 5 prodotti, e permetta mediante un menu di stampare i venditori ed i loro prezzi totali in ordine crescente o decrescente a scelta dell'utente.

Alcuni suggerimenti:

- gestire la scadenza del prodotto in maniera semplice;
- organizzare i valori dei venditori e dei prodotti in dei file di testo, da leggere via codice mediante il metodo leggi();
- per assicurarsi che i codici di venditori e prodotti siano univoci sfruttare le variabili statiche.

Animali.h

mercoledì 28 giugno 2023 21:30

```
#include <iostream>
#include <string>
using namespace std;
#ifndef ANIMALI_H
#define ANIMALI_H

class Animale{
    protected:
        string nome;
        string razza;
        int eta;
        virtual void Stampa(ostream &os) const = 0;
    public:
        Animale(string _nome, string _razza, int _eta) :
            nome(_nome), razza(_razza), eta(_eta){};
        ~Animale(){};
        string getNome() const{ return this->nome;}
        string getRazza() const{return this->razza;}
        int getEta()const {return this->eta; }
        virtual string verso() const = 0;
        virtual void stampa() const = 0;
        friend ostream& operator<<(ostream& out, const Animale& a);
};
ostream& operator<<(ostream& os, const Animale& a){
    a.Stampa(os);
    return os;
}

class Cane : public Animale{
        private:
            void Stampa(ostream &os) const;
        public:
            Cane(string _nome, string _razza, int _eta)
                : Animale(_nome,_razza, _eta){};
            ~Cane(){};
            string verso() const{ return "Bau";};
            void stampa() const;

};
void Cane::stampa() const{
    cout << "nome: " << getNome() << " , razza: "
    << getRazza() << ", eta': " << getEta()
    << ", verso: " << verso();
}
void Cane::Stampa(ostream &os) const{
    os << "nome: " << getNome() << " , razza: "
    << getRazza() << ", eta': " << getEta()
    << ", verso: " << verso();
}


class Gatto : public Animale{
            private:
                void Stampa(ostream &os) const;
            public:
```

```
                Gatto(string _nome, string _razza, int _eta)
                    : Animale( _nome, _razza, _eta){};
                ~Gatto(){};
                string verso() const { return "Miao";}
                void stampa() const;
};
void Gatto::stampa() const{
    cout << "nome: " << getNome() << " , razza: "
    << getRazza() << ", eta': " << getEta()
    << ", verso: " << verso();
}
void Gatto::Stampa(ostream &os) const{
    os << "nome: " << getNome() << " , razza: "
    << getRazza() << ", eta': " << getEta()
    << ", verso: " << verso();
}

#endif
```

BST.h

mercoledì 28 giugno 2023 21:31

main.cpp

mercoledì 28 giugno 2023 21:31

```cpp
#include <iostream>
#include <string>
#include "Animali.h"
#include "BST.h"
using namespace std;
int main(){
    Animale *a = new Cane("Adolfo", "Labrador", 4);
    Animale *a2 = new Cane("Bobby", "Pastore tedesco", 5);
    Animale *a3 = new Gatto("Ciondolino", "Siamese", 6);
    Animale *a4 = new Cane("Solo", "come un cane", 50);


    BST tree;
    tree.inserisci(a);
    tree.inserisci(a2);
    tree.inserisci(a3);
    cout << tree;

    string str;
    cout << "inserisci un nome da cercare: ";
    cin >> str;
//ricerca del nome
    if(tree.ricerca(str) != nullptr){
        cout <<"Trovato!" << endl;
        Nodo* temp = tree.ricerca(str);
        cout << "ecco tutte le informazioni di " << str << endl;
        cout << *(temp->dato) << endl;
    }
    else{
        cout << "questo nome non e' presente" << endl;
    }

    return 0;
}
```

Testo

mercoledì 28 giugno 2023 21:33

Programmazione 2

Si consideri una classe virtuale Animale che rappresenta un animale domestico. La classe ha i seguenti attributi privati:

- *nome, razza, età*

La classe ha i seguenti metodi pubblici:

- *costruttore, get degli attributi, stampa*
- *un metodo virtuale puro chiamato verso che restituisce il verso dell'animale*

Si considerino due classi derivate da Animale: Cane e Gatto. Ognuna di queste classi implementa il metodo *verso* in modo appropriato per il tipo di animale.

Si consideri una classe AlberoBinario che rappresenta una struttura dati ad albero binario di ricerca di animali domestici. 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 animale nell'albero in base al suo nome*
- *cerca: cerca un animale nell'albero dato il suo nome e restituisce un puntatore all'animale se trovato, altrimenti restituisce nullptr*
- *stampa: stampa tutti gli animali nell'albero in ordine alfabetico dei loro nomi*

Si scriva un programma che crea un albero binario di animali di dimensione 10 e inserisce nell'albero i seguenti animali:

- *nome Fido, razza Labrador, eta 3, tipo Cane*
- *nome Luna, razza Siamese, eta 2, tipo Gatto*
- *nome Rex, razza Pastore Tedesco, eta 5, tipo Cane*
- *nome Leo, razza Persiano, eta 4, tipo Gatto*
- *nome Milo, razza Beagle, eta 1, tipo Cane*
- *nome Nala, razza Maine Coon, eta 3, tipo Gatto*

Il programma poi chiede all'utente di inserire il nome di un animale da cercare nell'albero e stampa le informazioni sull'animale se trovato, altrimenti stampa un messaggio di errore. Il programma poi stampa il verso dell'animale trovato usando il metodo verso della classe corrispondente.

Output atteso inserendo il nome Luna:

Inserisci il nome di un animale da cercare: Luna

Luna è un gatto di razza Siamese di 2 anni.
Il verso di Luna è: Miao!

Output atteso inserendo il nome Bobo:
Inserisci il nome di un animale da cercare: Bobo
Nessun animale con questo nome trovato nell'albero.

main.cpp(tutto)

mercoledì 28 giugno 2023 21:32

```
/*Simulazione d'esame del 07/06/2023
*/
#include <iostream> // per usare cout e cin
#include <string> // per usare le stringhe
using namespace std;


// Funzione ausiliaria che riceve in input due stringhe che rappresentano due
date nel formato "gg/mm/aaaa" e restituisce true se la prima data è precedente
o uguale alla seconda, false altrimenti. Si assuma che le stringhe siano
sempre nel formato corretto e che le date siano valide.
bool data_precedente(string data1, string data2) {
  // Si convertono le stringhe in interi per confrontare le date
  int gg1 = stoi(data1.substr(0, 2));
  int mm1 = stoi(data1.substr(3, 2));
  int aaaa1 = stoi(data1.substr(6, 4));
  int gg2 = stoi(data2.substr(0, 2));
  int mm2 = stoi(data2.substr(3, 2));
  int aaaa2 = stoi(data2.substr(6, 4));

  // Si confrontano prima gli anni, poi i mesi e infine i giorni
  if (aaaa1 < aaaa2) {
    return true;
  } else if (aaaa1 == aaaa2) {
    if (mm1 < mm2) {
      return true;
    } else if (mm1 == mm2) {
      if (gg1 <= gg2) {
        return true;
      } else {
        return false;
      }
    } else {
      return false;
    }
  } else {
    return false;
  }
}
/*Si consideri una classe Prodotto che rappresenta un prodotto venduto in un
supermercato. La classe ha i seguenti attributi privati:
codice, nome, prezzo, scadenza
La classe ha i seguenti metodi pubblici:
              costruttore, get degli attributi, stampa
*/


class Prodotto {
  private:
    int codice;
    string nome;
    double prezzo;
    string scadenza;
  public:
    Prodotto(int c, string n, double p, string s) {
```

```
      codice = c;
      nome = n;
      prezzo = p;
      scadenza = s;
    }
    int get_codice() {
      return codice;
    }
    string get_nome() {
      return nome;
    }
    double get_prezzo() {
      return prezzo;
    }
    string get_scadenza() {
      return scadenza;
    }
    void stampa() {
      cout << codice << ": " << nome << " - " << prezzo << " euro (scadenza: "
<< scadenza << ")" << endl;
    }
    virtual ~Prodotto() {};
};
/*
Si consideri una classe Sconto che rappresenta uno sconto applicabile a un
prodotto.
La classe ha i seguenti attributi privati:
              percentuale, scadenza
*/
class Sconto {
  private:
    double percentuale;
    string scadenza;
  public:
    Sconto(double p, string s) {
      percentuale = p;
      scadenza = s;
    }
    double get_percentuale() {
      return percentuale;
    }
    string get_scadenza() {
      return scadenza;
    }
    double applica(double prezzo) {
      return prezzo * (1 - percentuale / 100);
    }
    void stampa() {
      cout << "[Sconto del " << percentuale << "% valido fino al " << scadenza
<< "]"<<endl;
    }
};

/*
Si consideri una classe ProdottoScontato che deriva dalla classe Prodotto
e ha un attributo privato aggiuntivo:
            - sconto: un puntatore a Sconto che indica lo sconto applicabile
al prodotto
La classe ha i seguenti metodi pubblici:
          - get_prezzo_scontato: restituisce il prezzo del prodotto dopo aver
applicato lo sconto, se lo sconto è valido rispetto alla data odierna,
altrimenti restituisce il prezzo originale del prodotto
```

```
          - stampa: stampa a video le informazioni del prodotto nel formato
"codice: nome - prezzo euro (scadenza: gg/mm/aaaa) [sconto del percentuale%
valido fino al scadenza]", se lo sconto è valido rispetto alla data odierna,
altrimenti stampa le informazioni del prodotto senza lo sconto
*/
class ProdottoScontato : public Prodotto {
  private:
    Sconto* sconto;
  public:
    ProdottoScontato(int c, string n, double p, string s, Sconto* sc) :
Prodotto(c, n, p, s) {
      sconto = sc;
    }
    ~ProdottoScontato() {
      delete sconto;
    }
    Sconto* get_sconto() {
      return sconto;
    }
    double get_prezzo_scontato(string oggi) {
      if (data_precedente(oggi, sconto->get_scadenza())) {
        // Lo sconto è valido, si applica al prezzo originale
        return sconto->applica(get_prezzo());
      } else {
        // Lo sconto non è valido, si restituisce il prezzo originale
        return get_prezzo();
      }
    }
    void stampa(string oggi) {
      //if (data_precedente(sconto->get_scadenza(), oggi)) {
        // Lo sconto è valido, si stampa con le informazioni dello sconto
        Prodotto::stampa();
        sconto->stampa();
      //} else {
        // Lo sconto non è valido, si stampa senza lo sconto
       // Prodotto::stampa();
      //}
    }
};
// Si consideri una classe Pila che rappresenta una pila di prodotti.

class Pila {
  private:
    Prodotto** dati;
    int dim;
    int testa;
  public:
    Pila(int d) {
      dim = d;
      testa = -1;
      dati = new Prodotto*[dim];
    }
    ~Pila() {
      while (!is_empty()) {
        delete pop();
      }
      delete[] dati;
    }
    void push(Prodotto* p) {
      if (!is_full()) {
        testa++;
        dati[testa] = p;
```

```
    } else {
      cout << "Pila piena, impossibile inserire il prodotto" << endl;
    }
  }
  Prodotto* pop() {
    if (!is_empty()) {
      Prodotto* p = dati[testa];
      testa--;
      return p;
    } else {
      cout << "Pila vuota, impossibile rimuovere il prodotto" << endl;
      return nullptr;
    }
  }
  Prodotto* top() {
    if (!is_empty()) {
      return dati[testa];
    } else {
      cout << "Pila vuota, impossibile visualizzare il prodotto" << endl;
      return nullptr;
    }
  }
  bool is_empty() {
    return testa == -1;
  }
  bool is_full() {
    return testa == dim - 1;
  }
};


// Si scriva un programma principale che crea una pila di prodotti di
dimensione 10 e inserisce nella pila i seguenti prodotti:
// - codice 1, nome "latte", prezzo 1.2, scadenza "10/06/2023"
// - codice 2, nome "pane", prezzo 0.8, scadenza "08/06/2023"
// - codice 3, nome "burro", prezzo 2.5, scadenza "15/06/2023"
// - codice 4, nome "marmellata", prezzo 3.0, scadenza "31/12/2023"
//  - codice 5, nome "yogurt", prezzo 1.5, scadenza "12/06/2023", sconto del
20% valido fino al "10/06/2023"
//  - codice 6, nome "succo di frutta", prezzo 2.0, scadenza "30/06/2023",
sconto del 10% valido fino al "15/06/2023"
// Il programma poi chiede all'utente di inserire la data odierna nel formato
"gg/mm/aaaa"
// e controlla se ci sono prodotti nella pila che sono scaduti rispetto a tale
data. In tal caso, il programma deve rimuovere dalla pila i prodotti scaduti e
stampare un messaggio di avviso con il nome e il codice dei prodotti rimossi.
// Il programma poi stampa il contenuto della pila e calcola il totale da
pagare per i prodotti nella pila usando i prezzi scontati dei prodotti
scontati, se lo sconto è valido rispetto alla data odierna, altrimenti usando
i prezzi originali.
int main() {
  Pila p(10);
  p.push(new Prodotto(1, "latte", 1.2, "10/06/2023"));
  p.push(new Prodotto(2, "pane", 0.8, "08/06/2023"));
  p.push(new Prodotto(3, "burro", 2.5, "15/06/2023"));
  p.push(new Prodotto(4, "marmellata", 3.0, "31/12/2023"));
  p.push(new ProdottoScontato(5, "yogurt", 1.5, "12/06/2023", new Sconto(20,
"10/06/2023")));
  p.push(new ProdottoScontato(6, "succo di frutta", 2.0, "30/06/2023", new
Sconto(10, "15/06/2023")));

  cout << "Inserisci la data odierna (gg/mm/aaaa): ";
```

```
  string oggi;
  cin >> oggi;

  // Si crea una pila ausiliaria per conservare i prodotti non scaduti
  Pila q(10);

  // Si scorre la pila originale e si controlla se ci sono prodotti scaduti
  while (!p.is_empty()) {
    Prodotto* prod = p.pop();
    if (data_precedente(prod->get_scadenza(), oggi)) {
      // Il prodotto è scaduto, lo si rimuove e si stampa un messaggio
      cout << "Attenzione: il prodotto " << prod->get_nome() << " (codice: "
<< prod->get_codice() << ") è scaduto e verrà rimosso dalla pila" << endl;
      delete prod;
    } else {
      // Il prodotto non è scaduto, lo si inserisce nella pila ausiliaria
      q.push(prod);
    }
  }

  cout << "Contenuto della pila:" << endl;
  double totale = 0;

  // Si riportano i prodotti non scaduti nella pila originale e si calcola il
totale
  while (!q.is_empty()) {
    Prodotto* prod = q.pop();
    p.push(prod);
    //prod->stampa();
    // Si controlla se il prodotto è scontato e se lo sconto è valido
    ProdottoScontato* prods = dynamic_cast<ProdottoScontato*>(prod);
    if (prods != nullptr && data_precedente(oggi, prods->get_scadenza())) {
      // Si usa il prezzo scontato
      totale += prods->get_prezzo_scontato(oggi);
      prods->stampa(oggi);
    } else {
      // Si usa il prezzo originale
      totale += prod->get_prezzo();
      prod->stampa();
    }

  }

  cout << "Totale da pagare: " << totale << " euro" << endl;

  return 0;
}
```

Testo

mercoledì 28 giugno 2023 21:35

Simulazione d'esame del 07/06/2023

Durata: 2 ore

Si consideri una classe Prodotto che rappresenta un prodotto venduto in un supermercato. La classe ha i seguenti attributi privati:

codice, nome, prezzo, scadenza

La classe ha i seguenti metodi pubblici:

costruttore, get degli attributi, stampa

Si consideri una classe Sconto che rappresenta uno sconto applicabile a un prodotto. La classe ha i seguenti attributi privati:

percentuale, scadenza

Si consideri una classe ProdottoScontato che deriva dalla classe Prodotto e ha un attributo privato aggiuntivo:

- *sconto: un puntatore a Sconto che indica lo sconto applicabile al prodotto*

La classe ha i seguenti metodi pubblici:

- get_prezzo_scontato: restituisce il prezzo del prodotto dopo aver applicato lo sconto, se lo sconto è valido rispetto alla data odierna, altrimenti restituisce il prezzo originale del prodotto
- stampa: stampa a video le informazioni del prodotto nel formato "codice: nome - prezzo euro (scadenza: gg/mm/aaaa) [sconto del percentuale% valido fino al scadenza]", se lo sconto è valido rispetto alla data odierna, altrimenti stampa le informazioni del prodotto senza lo sconto

Si scriva un programma che crea una pila di prodotti di dimensione 10 e inserisce nella pila i seguenti prodotti:

- *codice 1, nome "latte", prezzo 1.2, scadenza "10/06/2023"*
- *codice 2, nome "pane", prezzo 0.8, scadenza "08/06/2023"*
- *codice 3, nome "burro", prezzo 2.5, scadenza "15/06/2023"*
- *codice 4, nome "marmellata", prezzo 3.0, scadenza "31/12/2023"*
- *codice 5, nome "yogurt", prezzo 1.5, scadenza "12/06/2023", sconto del 20% valido fino al "10/06/2023"*
- *codice 6, nome "succo di frutta", prezzo 2.0, scadenza "30/06/2023", sconto del 10% valido fino al "15/06/2023"*

Il programma poi chiede all'utente di inserire la data odierna nel formato "gg/mm/aaaa" e controlla se ci sono prodotti nella pila che sono scaduti rispetto a tale data. In tal caso, il programma deve rimuovere dalla pila i prodotti scaduti e stampare un messaggio di avviso con il nome e il codice dei prodotti rimossi.

Il programma poi stampa il contenuto della pila e calcola il totale da pagare per i prodotti nella pila usando i prezzi scontati dei prodotti scontati, se lo sconto è valido rispetto alla data odierna, altrimenti usando i prezzi originali.

Output atteso inserendo la data odierna (07/06/2023):

Inserisci la data odierna (gg/mm/aaaa): 07/06/2023

Contenuto della pila:

1: latte - 1.2 euro (scadenza: 10/06/2023)

2: pane - 0.8 euro (scadenza: 08/06/2023)

3: burro - 2.5 euro (scadenza: 15/06/2023)

4: marmellata - 3 euro (scadenza: 31/12/2023)

5: yogurt - 1.5 euro (scadenza: 12/06/2023)

[Sconto del 20% valido fino al 10/06/2023]

6: succo di frutta - 2 euro (scadenza: 30/06/2023)

[Sconto del 10% valido fino al 15/06/2023]

Totale da pagare: 10.5 euro

Ritaglio schermata acquisito: 28/06/2023 21:37

Veicoli.h

mercoledì 28 giugno 2023 21:46

```
#include <iostream>
#include <string>

#ifndef VEICOLI_H
#define VEICOLI_H
class Veicolo{
    private:
        int targa;
        std::string nome_modello;
        int anno_fabbricazione;
        int cilindrata;
        std::string marca;
    public:

        Veicolo(int t, std::string nome_m, int anno_f, int cil, std::string
m) :
            targa(t), nome_modello(nome_m), anno_fabbricazione(anno_f),
cilindrata(cil), marca(m){};
        ~Veicolo(){};


        int getTarga() const{ return this->targa;}
        std::string getNomeModello() const{ return this->nome_modello;}
        int getAnnoFabbricazione() const{return this->anno_fabbricazione;}
        int getCilindrata() const{ return this->cilindrata;}
        std::string getMarca() const{ return this->marca;}
        virtual void stampa() = 0;
        Veicolo(Veicolo&v);

};
std::ostream& operator<<(std::ostream& os, const Veicolo& a) {
    os << "Veicolo: " << a.getTarga() << ", Modello:" << a.getNomeModello()
<< std::endl;
    os << "Marca: " << a.getMarca() << ", Anno Fabbricazione: "
<< a.getAnnoFabbricazione() << std::endl;
    os << "Cilindrata: " << a.getCilindrata();
    return os;
}


Veicolo::Veicolo(Veicolo& v){
    this->targa = v.targa;
    this->nome_modello = v.nome_modello;
    this->anno_fabbricazione = v.anno_fabbricazione;
    this->cilindrata = v.cilindrata;
    this->marca = v.marca;
}
class Auto : public Veicolo{
    private:
        int numero_porte;
    public:
        Auto(int t, std::string nome_m, int anno_f, int cil, std::string m,
int num_p) :
            Veicolo(t, nome_m, anno_f, cil, m), numero_porte(num_p){};
        Auto(Auto& a);
```

```
        ~Auto(){};

        int getNumeroPorte() const{ return this->numero_porte;}
        void stampa();
};
Auto::Auto(Auto& a) : Veicolo(a){
    this->numero_porte = numero_porte;
}

std::ostream& operator<<(std::ostream& os, const Auto& a) {
    os << "Auto: " << a.getTarga() << ", Modello:" << a.getNomeModello()
<< std::endl;
    os << "Marca: " << a.getMarca() << ", Anno Fabbricazione: "
<< a.getAnnoFabbricazione() << std::endl;
    os << "Cilindrata: " << a.getCilindrata() << "Numero porte: "
<< a.getNumeroPorte() << std::endl;
    return os;
}

void Auto::stampa()
{
    std::cout << "Veicolo: " << std::endl;
    std::cout << "targa: " << getTarga() << "\t nome_modello: "
<< getNomeModello()
    << "\n anno fabbricazione: " << getAnnoFabbricazione() << "\t cilindrata:
" << getCilindrata()
    << "\n marca: " << getMarca() << "\t numero porte: "
<< getNumeroPorte()  << std::endl;
    std::cout << std::endl;
}
class Moto : public Veicolo{
    private:
        std::string tipo_moto;
    public:
        Moto(int t, std::string nome_m, int anno_f, int cil, std::string m,
std::string tipo_m)
            : Veicolo(t, nome_m, anno_f, cil, m), tipo_moto(tipo_m){};
        ~Moto(){};
        std::string getTipoMoto() const{ return this->tipo_moto;}
        void stampa();
};
std::ostream& operator<<(std::ostream& os, const Moto& a) {
    os << "Auto: " << a.getTarga() << ", Modello:" << a.getNomeModello()
<< std::endl;
    os << "Marca: " << a.getMarca() << ", Anno Fabbricazione: "
<< a.getAnnoFabbricazione() << std::endl;
    os << "Cilindrata: " << a.getCilindrata() << "Tipo moto: "
<< a.getTipoMoto() << std::endl;
    return os;
}


void Moto::stampa()
{
    std::cout << "Veicolo: " << std::endl;
    std::cout << "targa: " << getTarga() << "\t nome_modello: "
<< getNomeModello()
    << "\n anno fabbricazione: " << getAnnoFabbricazione() << "\t cilindrata:
" << getCilindrata()
    << "\n marca: " << getMarca() << "\t tipo moto: " << getTipoMoto()
<< std::endl;
```

```
    std::cout << std::endl;
}
#endif
```

BST.h

mercoledì 28 giugno 2023 21:49

```
#include <iostream>
#include "veicoli.h"
using namespace std;
#ifndef BST_H
#define BST_H
class Nodo{
    public:
        Nodo* left;
        Nodo* right;
        Nodo* padre;
        Veicolo* dato;
        Nodo(Veicolo* v) : dato(v), left(nullptr), right(nullptr),
padre(nullptr){};
};
class BST{
    private:
        Nodo* radice;
        void inOrder(Nodo* ptr) const;
        void inOrder(Nodo* ptr, ostream& os) const;
        void preOrder(Nodo* ptr) const;
        void trapianta(Nodo*u, Nodo* v);
        void cancella(Nodo* z);
        Nodo* minimo(Nodo* x);
        void distruggi(Nodo* nodo);
        Nodo* ricerca(Nodo* x, int el) const;

    public:
        BST() : radice(nullptr){};
        ~BST();
        Nodo* getRadice() const{ return this->radice;}
        void inserisci(Veicolo* el);
        void inOrder() const;
        void preOrder() const;
        friend ostream &operator << (ostream &os, const BST &a);
        bool cancella(int x);
        Nodo* ricerca(int targa_) const;

};
void BST::inserisci(Veicolo* el){ //i veicoli verranno inseriti in funzione
del valore della loro targa.
    Nodo* y = nullptr;
    Nodo *nuovo = new Nodo(el);
    nuovo->left = nullptr;
    nuovo->right = nullptr;
    Nodo *x = radice;
    while(x != nullptr){
        y = x;
        if(el->getTarga() <  x->dato->getTarga()){
            x = x->left;
        }
        else
            x = x->right;
    }
    nuovo->padre = y;
    if(y == nullptr){
```

```
            radice = nuovo;
        }
        else if(el->getTarga() < y->dato->getTarga()){
            y->left = nuovo;
        }
        else{
            y->right = nuovo;
        }
}
void BST::inOrder(Nodo* ptr) const{
    if(ptr != nullptr){
        inOrder(ptr->left);
        ptr->dato->stampa();
        // cout << *(ptr->dato) << endl;
        inOrder(ptr->right);
    }
}
void BST::inOrder() const{
    inOrder(radice);
}
void BST::inOrder(Nodo* ptr, ostream& os) const { //stampa in ordine
alfabetico perchè i veicoli sono inseriti nell'albero in funzione della loro
targa

    if (ptr != nullptr) {
        inOrder(ptr->left, os);
        os << *(ptr->dato) << endl;
        inOrder(ptr->right, os);
    }
}

ostream &operator << (ostream &os, const BST &a){
    a.inOrder(a.radice, os);
    return os;
}
void BST::trapianta(Nodo*u, Nodo* v){
    if(u->padre == nullptr){
        radice = v;
    }
    else if(u->padre->left == u){
        u->padre->left = v;
    }
    else
        u->padre->right = v;
}
Nodo* BST::minimo(Nodo* x){
    Nodo* min = x;
    while(min->left != nullptr)
        min = min->left;
    return min;
}
void BST::cancella(Nodo *z){
    Nodo* y;
    if(z->left == nullptr){
        trapianta(z, z->right);
    }
    else if(z->right == nullptr){
        trapianta(z, z->left);
    }
    else{
        y = minimo(z->right); //y puntera' al successore di z
        if(y->padre != z){
```

```
            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;
}
bool BST::cancella(int x){
    Nodo* iter = radice;
    while((iter != nullptr) && (x != iter->dato->getTarga())){
        if( x < iter->dato->getTarga())
            iter = iter->left;
        else
            iter = iter->right;
    }
    if(iter == nullptr){
        return false;
    }
    else{
        cancella(iter);
        return true;
    }
}
void BST::preOrder(Nodo* ptr) const{
    if(ptr != nullptr){
        ptr->dato->stampa();
        preOrder(ptr->left);
        preOrder(ptr->right);
    }
}
void BST::preOrder() const{
    preOrder(radice);
}
BST::~BST() {
    distruggi(radice);
}
void BST::distruggi(Nodo* nodo) {
    if (nodo != nullptr) {
        distruggi(nodo->left);  // distruggi sottoalbero sinistro
        distruggi(nodo->right); // distruggi sottoalbero destro
        delete nodo->dato;      // distruggi il dato del nodo
        delete nodo;            // distruggi il nodo stesso
    }
}

 Nodo* BST::ricerca(Nodo* x, int el) const{
    if( x == nullptr || x->dato->getTarga() == el){
        return x;
    }
    if(el < x->dato->getTarga())
        return ricerca(x->left, el);
    else
        return ricerca(x->right, el);
}
 Nodo* BST::ricerca(int targa) const{
    return ricerca(radice, targa);
 }
#endif
```

main.cpp

mercoledì 28 giugno 2023 21:49

```
#include <iostream>
#include "veicoli.h"
#include "bst.h"
int main(){
    Veicolo *a = new Auto( 1, "Panda FIRE", 1986, 769, "Fiat", 3);
    Veicolo *a2 = new Auto(3, "Ferrari Testarossa", 1984, 4943, "Ferrari", 3);
    Veicolo *m = new Moto(4, "YZF-R3", 2014, 249, "Yamaha", "Sportiva" );
    Veicolo *a3 = new Auto(5, "Peugeot 208", 2019, 1199, "Peugeto", 5);
    Veicolo *m2 = new Moto(6,"Honda AfricaTwin 650", 1983, 749,"Honda",
"Enduro");
    // a->stampa();
    // m->stampa();

    BST *tree = new BST;;
    tree->inserisci(a2);
    tree->inserisci(a);
    tree->inserisci(a3);
    tree->inserisci(m);
    tree->inserisci(m2);
    // tree.inOrder();
    tree->cancella(3);
    tree->cancella(5);
    tree->preOrder();
    delete tree;
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 21:50

Simulazione d'esame

Si consideri una classe astratta Veicolo. La classe ha i seguenti attributi privati:

- *Targa, nome modello, anno di fabbricazione, marca.*

La classe ha i seguenti metodi pubblici:

- *costruttore, get degli attributi*
- *un metodo virtuale puro per la stampa*

Si considerino due classi derivate da Veicolo: Auto e Moto. Ognuna di queste classi implementa il metodo *stampa* in modo appropriato per il tipo di veicolo.

La classe Auto ha i seguenti attributi privati:

- *numero_porte*

La classe Moto ha i seguenti attributi privati:

- *tipo_moto (sportiva, chopper, enduro, ecc)*

Si consideri una classe AlberoBinario che rappresenta una struttura dati ad albero binario di ricerca di veicoli. La classe ha i seguenti attributi privati:

- *radice: un puntatore al nodo radice dell'albero*

La classe ha i seguenti metodi pubblici:

- *costruttore, distruttore*
- *inserisci: inserisce un nuovo veicolo nell'albero in base alla sua targa*
- *elimina: elimina un nuovo veicolo nell'albero in base alla sua targa*
- *cerca: cerca un veicolo nell'albero data la sua targa e restituisce un puntatore al veicolo se trovato, altrimenti restituisce nullptr*

trovato, altrimenti restituisce nullptr

- *stampa: stampa tutti i veicoli nell'albero in ordine alfabetico delle loro targe*
- *verifica stampa: stampa tutti i veicoli eseguendo una visita Pre-Order dell'albero*

Si scriva un programma che crei un albero binario di veicoli e inserisce nell'albero i seguenti veicoli:

Auto:
Targa: 1 Nome modello: Panda FIRE
Marca: Fiat Anno fabbricazione: 1986
Cilindrata: 769 Numero porte: 3

Auto:
Targa: 3 Nome modello: Ferrari Testarossa
Marca: Ferrari Anno fabbricazione: 1984
Cilindrata: 4943 Numero porte: 3
Moto:
Targa: 4 Nome modello: YZF-R3
Marca: Yamaha Anno fabbricazione: 2014
Cilindrata: 249 Tipo moto: Sportiva
Auto:
Targa: 5 Nome modello: Peugeot 208
Marca: Peugeot Anno fabbricazione: 2019
Cilindrata: 1199 Numero porte: 5
Moto:
Targa: 6 Nome modello: Honda Africa Twin 650
Marca: Honda Anno fabbricazione: 1983
Cilindrata: 749 Tipo moto: Enduro

Il programma dovrà eseguire in ordine:

- Si inseriscono i veicoli (si specifica la targa) nel seguente ordine: 3, 1, 5, 4, 6
- Si eliminano in ordine i veicoli con targa 3 e 5.
- Si stampa con una visita Pre-Order

La stampa risultate dovrà essere la seguente:

```
Moto:
```

La stampa risultate dovrà essere la seguente:

```
Moto:
    Targa: 4    Nome modello: YZF-R3
    Marca: Yamaha   Anno fabbricazione: 2014
    Cilindrata: 249 Tipo moto: Sportiva
Auto:
    Targa: 1    Nome modello: Panda FIRE
    Marca: Fiat Anno fabbricazione: 1986
    Cilindrata: 769 Numero porte: 3
Moto:
    Targa: 6    Nome modello: Honda Africa Twin 650
    Marca: Honda    Anno fabbricazione: 1983
    Cilindrata: 749 Tipo moto: Enduro
```

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;
}
```

Studente.h

mercoledì 28 giugno 2023 21:58

```
#include <iostream>
#include <string>
using namespace std;
#ifndef STUDENTE_H
#define STUDENTE_H
class Studente{
    private:
        int matricola;
        string nome;
        string cognome;
        float media;

    public:
        Studente(int _matricola, string _nome, string _cognome, float
_media) :
            matricola(_matricola), nome(_nome), cognome(_cognome),
media(_media){};

        virtual ~Studente(){};
        int getMatricola() const{ return this->matricola;}
        string getNome() const{return this->nome;}
        string getCognome() const{ return this->cognome;}
        float getMedia() const{ return this->media;}
        friend ostream& operator <<(ostream& os, const Studente &s);
        // virtual void stampa(ostream &os) const;
        virtual void stampa() const;
};
void Studente::stampa() const{
    cout << "Matricola: " << getMatricola() << ",\tnome: " << getNome()
    << ",\t cognome: " << getCognome() << ", \tmedia: " << getMedia() << endl;

}
class BorsaDiStudio{
    private:
        float importo;
        float durata;
    public:
        BorsaDiStudio(float _importo, float _durata) : importo(_importo),
durata(_durata){};
        ~BorsaDiStudio(){};
        float getImporto() const{ return this->importo;}
        float getDurata() const{return this->durata;}
};
class StudenteBorsista : public Studente{
    private:
        BorsaDiStudio* borsa; //indica la borsa di studio assegnata allo
studente
    public:
        StudenteBorsista(int _matricola, string _nome, string _cognome, float
_media, BorsaDiStudio* b) :
            Studente(_matricola, _nome, _cognome, _media),  borsa(b){};
        ~StudenteBorsista(){};
        float get_importo_borsa() const{ return borsa->getImporto(); }
        // friend ostream &operator <<(ostream &os, const StudenteBorsista&
s);
```

```
        // void stampa(ostream &os) const;
        void stampa() const;
};
void StudenteBorsista::stampa() const{
    cout << "Matricola: " << getMatricola() << ",\tnome: " << getNome()
    << ",\t cognome: " << getCognome() << ",\tmedia: " << getMedia() << ",
\tborsa di: "
    << get_importo_borsa() << " euro" << endl;
}


#endif
```

Coda.h

mercoledì 28 giugno 2023 21:59

```
#include <iostream>
#include "Studente.h"
using namespace std;
#ifndef CODA_H
#define CODA_H
class Nodo{
    public:
        Studente* dato;
        Nodo* succ;
    Nodo(Studente* el) : dato(el), succ(nullptr){};
};
class Coda{
    private:
        Nodo* testa;
        Nodo* fine;
    public:
        Coda() : testa(nullptr), fine(nullptr){};
        ~Coda();
        Nodo* getTesta() const{ return this->testa;}
        bool isEmpty() const;
        void enqueue(Studente* el); //inserimento in coda
        Studente* dequeue(); //rimozione
        Studente* peek() const;
        void stampaCoda() const;
};
void Coda::stampaCoda() const{
    Nodo* iter = testa;
    while(iter != nullptr){
        iter->dato->stampa();
        iter = iter->succ;
    }
}

bool Coda::isEmpty() const{
    if(fine == nullptr){
       return true;
    }
    else{
        return false;
    }
}
Coda::~Coda(){
    while(!isEmpty()){
        dequeue();
    }
}
void Coda::enqueue(Studente* el){
    Nodo* nuovo = new Nodo(el);
    nuovo->succ = nullptr;
    if(isEmpty()){
        fine = testa = nuovo;
    }
    fine->succ = nuovo;
    fine = nuovo;
}
```

```
Studente* Coda::dequeue(){
    if(isEmpty()){
        throw runtime_error("coda vuota, impossibile rimuovere elementi");
    }
    Nodo* iter;
    Studente* tmp = testa->dato;
    iter = testa;
    testa = testa->succ;
    if(testa == nullptr){
        fine = nullptr;
    }
    delete iter;
    return tmp;
}
Studente* Coda::peek() const{
    if(isEmpty()){
        throw runtime_error("la coda e' vuota!");
    }
    return testa->dato;
}

#endif
```

main.cpp

mercoledì 28 giugno 2023 21:59

```
#include <iostream>
#include <string>
#include "Studente.h"
#include "Coda.h"
using namespace std;

int main(){
    BorsaDiStudio *b = new BorsaDiStudio(880, 2);
    BorsaDiStudio *b2 = new BorsaDiStudio(500, 2);
    BorsaDiStudio *b3 = new BorsaDiStudio(600, 2);
    Studente *s1 = new Studente (1001, "Mario", "Rossi", 25.5);
    Studente *s2 = new StudenteBorsista(1002, "Anna", "Verdi", 28.0,  b);
    Studente *s3 = new Studente (1003, "Luca", "Bianchi", 26.0);
    Studente *s4 = new Studente (1004, "Mario", "Rossi", 27.5);
    Studente *s5 = new StudenteBorsista (1005, "Marco", "Gialli", 24.0, b2);
    Studente *s6 = new StudenteBorsista (1006, "Laura", "Marroni", 29.0, b3);

    s1->stampa();
    s2->stampa();
    s3->stampa();
    s4->stampa();
    s5->stampa();
    s6->stampa();

    Coda tail;
    tail.enqueue(s1);
    tail.enqueue(s2);
    tail.enqueue(s3);
    tail.enqueue(s4);
    tail.enqueue(s5);
    tail.enqueue(s6);
    cout << "\nSTUDENTI TOTALI:" << endl;
    tail.stampaCoda();


    Coda tail2;

    Studente* tmp;

    while(!tail.isEmpty()){
        tmp = tail.dequeue();
        if(tmp->getMedia() >= 25){
            tail2.enqueue(tmp);
        }
        else{
            cout << "\nlo studente seguente e' stato rimosso: " << endl;
            tmp->stampa();
        }
    }
    cout << "\n\nSTUDENTI CON MEDIA >= 25: " << endl;
    tail2.stampaCoda();
    float totaleBorseDiStudio = 0;
        for (Nodo* iter = tail2.getTesta(); iter != nullptr; iter = iter->
succ) {
            StudenteBorsista* sb = dynamic_cast<StudenteBorsista*>(iter->
```

```
dato);
            if (sb != nullptr) {
                totaleBorseDiStudio += sb->get_importo_borsa();
        }
            }
    cout << "Totale degli importi delle borse di studio: "
<< totaleBorseDiStudio << " euro" << endl;

    delete s1;
    delete s2;
    delete s3;
    delete s4;
    delete s5;
    delete s6;
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 22:00

PROVA D'ESAME 2

Si consideri una classe Studente che rappresenta uno studente universitario. La classe ha i seguenti attributi privati:

- *matricola, nome, cognome, media*

La classe ha i seguenti metodi pubblici:

- *costruttore, get degli attributi, stampa*

Si consideri una classe BorsaDiStudio che rappresenta una borsa di studio assegnata a uno studente. La classe ha i seguenti attributi privati:

- *importo, durata*

Si consideri una classe StudenteBorsista che deriva dalla classe Studente e ha un attributo privato aggiuntivo:

- *borsa: un puntatore a BorsaDiStudio che indica la borsa di studio assegnata allo studente*

La classe ha i seguenti metodi pubblici:

- *get_importo_borsa: restituisce l'importo della borsa di studio assegnata allo studente*
- *stampa: stampa a video le informazioni dello studente nel formato "matricola: nome cognome - media [borsa di importo euro]", se è titolare di borsa, altrimenti stampa le informazioni dello studente senza la borsa*

Si scriva un programma che crea una coda di studenti di dimensione 10 e inserisce nella coda i seguenti studenti:

- *matricola 1001, nome "Mario", cognome "Rossi", media 25.5*
- *matricola 1002, nome "Anna", cognome "Verdi", media 28.0, borsa di 880 euro*
- *matricola 1003, nome "Luca", cognome "Bianchi", media 26.0*
- *matricola 1004, nome "Sara", cognome "Neri", media 27.5*
- *matricola 1005, nome "Marco", cognome "Gialli", media 24.0, borsa di 500 euro*
- *matricola 1006, nome "Laura", cognome "Marroni", media 29.0, borsa di 600 euro*

Il programma poi controlla se ci sono studenti nella coda che hanno una media inferiore a 25. In tal caso, il programma deve rimuovere dalla coda gli studenti insufficienti e stampare un messaggio di avviso con il nome e la matricola degli studenti rimossi.

Il programma poi stampa il contenuto della coda e calcola il totale degli importi delle borse di studio assegnate agli studenti nella coda usando i metodi get_importo_borsa degli studenti borsisti.

Output atteso:

Contenuto della coda:

1001: Mario Rossi - 25.5

1002: Anna Verdi - 28.0 [borsa di 880 euro]

1003: Luca Bianchi - 26.0

1004: Sara Neri - 27.5

1006: Laura Marroni - 29.0 [borsa di 600 euro]

Totale degli importi delle borse di studio: 1480 euro

Veicolo.h

mercoledì 28 giugno 2023 22:03

```
#include <iostream>
#include <string>
using namespace std;
#ifndef VEICOLO_H
#define VEICOLO_H
class Veicolo{
    private:
        string modello;
        string marca;
        int anno;
    public:
        Veicolo(string _modello, string _marca, int _anno):
            modello(_modello), marca(_marca), anno(_anno){};
        virtual ~Veicolo(){};
        string getModello() const{return this->modello;}
        string getMarca() const {return this->marca;}
        int getAnno() const{return this->anno;}
        virtual void stampa() const;
        virtual string tipo() const = 0;
        friend ostream &operator <<(ostream &os, const Veicolo& v);
};
ostream &operator <<(ostream &os, const Veicolo& v){
    os << "Modello: " << v.getModello() << ",  Marca: " << v.getMarca()
    << ",  Anno:" << v.getAnno() << ",  Tipo:" << v.tipo() << endl;
    return os;
}
void Veicolo::stampa() const{
    cout << "Modello: " << getModello() << ",  Marca: " << getMarca()
    << ",  Anno:" << getAnno() << ",  Tipo: " << tipo() << endl;
}
class Auto : public Veicolo{
    public:
        Auto(string _modello, string _marca, int _anno) :
            Veicolo(_modello, _marca, _anno){};
        ~Auto(){};
        string tipo() const{ return "auto";}
};

class Moto : public Veicolo{
    public:
        Moto(string _modello, string _marca, int _anno) :
            Veicolo(_modello, _marca, _anno){};
        ~Moto(){};
        string tipo() const{ return "moto";}

};

#endif
```

Coda.h template

mercoledì 28 giugno 2023 22:03

```
#include <iostream>
#include "Veicolo.h"
using namespace std;
template <typename T = Veicolo*>
class Nodo{
    public:
        Nodo<T>* succ;
        T dato;
        Nodo(T el) : dato(el), succ(nullptr){};
};

template <typename T =  Veicolo*>
class Coda{
    private:
        Nodo<T>* testa;
        Nodo<T>* fine;
        int num_el = 0;
    public:
        Coda() : testa(nullptr), fine(nullptr){};
        ~Coda();
        bool isEmpty();
        Nodo<T>* getTesta() const{return this->testa;}
        void enqueue(T el);
        T dequeue();
        void stampaCoda() const;
        int dimensione() const{return this->num_el;}
        T front() const;
};
template <typename T>
bool Coda<T>::isEmpty(){
    if(fine == nullptr){
        return true;
    }
    else{
        return false;
    }
}

template <typename T>
Coda<T>::~Coda(){
    while(!isEmpty()){
        dequeue();
    }
}
template <typename T>
void Coda<T>::enqueue(T el){
    Nodo<T>* nuovo = new Nodo<T>(el);
    num_el++;
    if(fine == nullptr){
        testa = fine = nuovo;
    }
    else{
        fine->succ = nuovo;
        fine = nuovo;
    }
```

```
}
template <typename T>
T Coda<T>::dequeue(){
    if(isEmpty()){
        throw runtime_error("la coda e' vuota!");
    }
    T aux = testa->dato;
    Nodo<T>* iter = testa;
    testa = testa->succ;
    if(testa == nullptr){
        fine = nullptr;
    }
    delete iter;
    num_el--;
    return aux;
}
template <typename T>
T Coda<T>::front() const{
    if(isEmpty()){
        throw runtime_error("la coda e' vuota");
    }
    return testa->dato;
}
template <typename T>
void Coda<T>::stampaCoda() const{
    Nodo<T>* iter = testa;

    while(iter != nullptr){
        cout << *iter->dato;
        iter = iter->succ;
    }
}
```

main.cpp

mercoledì 28 giugno 2023 22:04

```
#include <iostream>
#include <string>
#include "Veicolo.h"
#include "CodaT.h"
int rimuovi(string &risp, Coda<Veicolo*> &tail){
    Nodo<Veicolo*> *iter;
    Coda tmp;
    string tmp2;
    int count = 0;
    for(iter = tail.getTesta(); iter != nullptr; iter = iter->succ){
        tmp2 = iter->dato->tipo();
        if(tmp2 != risp){
            tmp.enqueue(iter->dato);
        }
        else{
            count++;
        }
    }
    while(!tail.isEmpty()){
        tail.dequeue();
    }
    while(!tmp.isEmpty()){
        tail.enqueue(tmp.dequeue());
    }
    return count;
}

int main(){
    Veicolo* a1 = new Auto("Fiesta", "Ford", 2010);
    Veicolo* m1 = new Moto("CBR 600 RR", "Honda", 2008);
    Veicolo* a2 = new Auto("Panda", "Fiat", 2012);
    Veicolo* m2 = new Moto("Ninja ZX-10R", "Kawaski", 2019);
    Veicolo* a3 = new Auto("Golf", "Volkswagen", 2015);
    Veicolo* m3 = new Moto("R1", "Yamaha", 2020);
    Coda tail;
    tail.enqueue(a1);
    tail.enqueue(m1);
    tail.enqueue(a2);
    tail.enqueue(m2);
    tail.enqueue(a3);
    tail.enqueue(m3);

    cout << "STAMPA CODA: \n";
    tail.stampaCoda();


    string risp;
    cout << "che tipo di veicolo vuoi rimuovere?(auto/moto): ";
    cin >> risp;
    rimuovi(risp, tail);
    cout << "\n\nSTAMPA DOPO RIMOZIONE: \n";
    tail.stampaCoda();

    delete a1;
    delete m1;
    delete a2;
```

```
    delete m2;
    delete a3;
    delete m3;
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 22:01

Si consideri una classe virtuale Veicolo che rappresenta un veicolo.

La classe ha i seguenti attributi privati:

- *modello, marca, anno*

La classe ha i seguenti metodi pubblici:

- *costruttore, get degli attributi, stampa*
- *un metodo virtuale puro chiamato tipo che restituisce il tipo di veicolo*

Si considerino due classi derivate da Veicolo: Auto e Moto. Ognuna di queste classi implementa il metodo tipo in modo appropriato per il tipo di veicolo.

Si consideri una classe template Coda<T> che rappresenta una struttura dati a coda di elementi di tipo T.

La classe ha i seguenti attributi privati:

- *testa: un puntatore al nodo in testa alla coda*
- *coda: un puntatore al nodo in coda alla coda*
- *dimensione: il numero di nodi nella coda*

La classe ha i seguenti metodi pubblici:

- *costruttore, distruttore*
- *enqueue: inserisce un nuovo elemento in coda alla coda*
- *dequeue: rimuove e restituisce l'elemento in testa alla coda, se la coda non è vuota*
- *front: restituisce l'elemento in testa alla coda senza rimuoverlo, se la coda non è vuota*
- *stampa: stampa tutti gli elementi nella coda dal primo all'ultimo inserito*

Si scriva un programma che crea una coda di puntatori a veicoli e inserisce nella coda i seguenti veicoli usando il metodo *enqueue*:

- *Fiesta, Ford, 2010, tipo Auto*
- *CBR 600 RR, Honda, 2008, tipo Moto*
- *Panda, Fiat, 2012, tipo Auto*
- *Ninja ZX-10R, Kawasaki, 2019, tipo Moto*
- *Golf, Volkswagen, 2015, tipo Auto*
- *R1, Yamaha, 2020, tipo Moto*

Il programma poi chiede all'utente di inserire il tipo di veicoli da rimuovere dalla coda e li rimuove usando il metodo *dequeue* della classe Coda. Il programma poi stampa tutti i veicoli rimanenti nella coda usando il metodo stampa della classe Coda. Per rimuovere i veicoli di un certo tipo dalla

coda si scriva una funzione ricorsiva che riceve come parametri la coda e il tipo da rimuovere e restituisce il numero di veicoli rimossi.

Output atteso inserendo il tipo Auto:

Inserisci il tipo di veicoli da rimuovere: Auto

Sono stati rimossi 3 veicoli dalla coda.

Veicoli rimanenti nella coda:

CBR 600 RR - Honda - 2008 - Moto

Ninja ZX-10R - Kawasaki - 2019 - Moto

R1 - Yamaha - 2020 - Moto

Output atteso inserendo il tipo Moto:

Inserisci il tipo di veicoli da rimuovere: Moto

Sono stati rimossi 3 veicoli dalla coda.

Veicoli rimanenti nella coda:

Fiesta - Ford - 2010 - Auto

Panda - Fiat - 2012 - Auto

Golf - Volkswagen - 2015 – Auto