Svolgimento della traccia Ristorante – scritto 24 Luglio 2017

```
#include <iostream>
#include "OrdinazioneTakeAway.h"
#include "OrdinazioneDomicilio.h"
#include "Ristorante.h"
using namespace std;

int main() {
    Ordinazione* b1 = new OrdinazioneTakeAway(1,4,"Pizza Margherita");
    Ordinazione* b2 = new OrdinazioneDomicilio(2,15,"Hambuger","Via Rossi 11, Roma");
    Ristorante* ristorante1 = new Ristorante();
    ristorante1->aggiungiOrdinazioneDaServire(b1);
    ristorante1->aggiungiOrdinazioneDaServire(b2);
    Ristorante* ristorante2 = new Ristorante(*ristorante1);
    ristorante2->aggiungiOrdinazioneDaServire(b1);

    delete b1;
    delete b2;
    delete ristorante1;
    delete ristorante2;
}
```

```
void Ristorante::svuotaOrdinazioniServite() {
    delete [] ordinazioniServite;
    ordinazioniServite=0;
    numeroOrdinazioniServite=0;
    capacitaOrdinazioniServite=0;
}

bool Ristorante::aggiungiOrdinazioneDaServire(Ordinazione* o) {
    for(list<Ordinazione*>::iteratorit=ordinazioniDaServire.begin();
      it!=ordinazioniDaServire.end();it++)
        if((*o)==(*(*it)))
            return false;
    ordinazioniDaServire.push_back(o);
    return true;
}

bool Ristorante::aggiungiOrdinazioneServita(Ordinazione* o) {
    for(list<Ordinazione*>::iterator it=ordinazioniDaServire.begin();
      it!=ordinazioniDaServire.end();it++){
        if((*o)==(*(*it)))
        {
            if(capacitaOrdinazioniServite==numeroOrdinazioniServite){
                if(capacitaOrdinazioniServite==0)
                    capacitaOrdinazioniServite=1;
                else
                    capacitaOrdinazioniServite*=2;
                Ordinazione** tmp=new Ordinazione*[capacitaOrdinazioniServite];
                for(unsigned i=0;i<numeroOrdinazioniServite;i++)
                    tmp[i]=ordinazioniServite[i];
                delete [] ordinazioniServite;
                ordinazioniServite=tmp;
            }
            ordinazioniServite[numeroOrdinazioniServite++]=*it;
            ordinazioniDaServire.erase(it);
            return true;
        }
    }
    return false;
}
```

```cpp
#ifndef ORDINAZIONE_H_
#define ORDINAZIONE_H_

#include <iostream>
using namespace std;

class Ordinazione {
private:
    int codice;
    float prezzo;
    string descrizione;
public:
    Ordinazione(int c, float p, string d):codice(c),prezzo(p),descrizione(d){};
    int getCodice() const;
    void setCodice(int codice);
    const string& getDescrizione() const;
    void setDescrizione(const string& descrizione);
    int getPrezzo() const;
    void setPrezzo(int prezzo);
    bool operator==(const Ordinazione& o){return o.getCodice()==codice;}
    virtual string info();
    virtual ~Ordinazione(){};
};

#endif /* ORDINAZIONE_H_ */
```

```cpp
#ifndef ORDINAZIONEDOMICILIO_H_
#define ORDINAZIONEDOMICILIO_H_

#include "Ordinazione.h"

class OrdinazioneDomicilio : public Ordinazione {
private:
    string domicilio;
public:
    OrdinazioneDomicilio(int c, float p, string d, string dom):
      Ordinazione(c,p,d),domicilio(dom){}
    virtual string info() {return "Ordinazione a Domicilio "+getDescrizione();}
    const string& getDomicilio() const {return domicilio;}
    void setDomicilio(const string& domicilio) {this->domicilio = domicilio;}

    virtual ~OrdinazioneDomicilio(){}
};

#endif /* ORDINAZIONEDOMICILIO_H_ */
```

---

1. D
2. B
3. C
4. D
5. C
6. D

Implementazione completa dei metodi rimanenti

```
#ifndef RISTORANTE_H_
#define RISTORANTE_H_

#include "Ordinazione.h"
#include <list>

class Ristorante {
public:
    Ristorante():ordinazioniServite(0),numeroOrdinazioniServite(0),
      capacitaOrdinazioniServite(0){};
    Ristorante(const Ristorante& r);
    Ristorante& operator=(const Ristorante& r);
    ~Ristorante();
    bool aggiungiOrdinazioneDaServire(Ordinazione* o);
    bool aggiungiOrdinazioneServita(Ordinazione* o);
    friend ostream& operator<<(ostream& out, const Ristorante& r);
    unsigned getNumeroOrdinazioniServite() const;
    void svuotaOrdinazioniDaServire();
    void svuotaOrdinazioniServite();
protected:
    list<Ordinazione*> ordinazioniDaServire;
    Ordinazione** ordinazioniServite;
    unsigned numeroOrdinazioniServite;
    unsigned capacitaOrdinazioniServite;
};

#endif /* RISTORANTE_H_ */
```

```
#include "Ristorante.h"

Ristorante::Ristorante(const Ristorante& r) {
    for(list<Ordinazione*>::const_iterator
it=r.ordinazioniDaServire.begin();it!=r.ordinazioniDaServire.end();it++)
        ordinazioniDaServire.push_back(*it);

    numeroOrdinazioniServite=r.numeroOrdinazioniServite;
    capacitaOrdinazioniServite=r.capacitaOrdinazioniServite;
    ordinazioniServite=new Ordinazione*[capacitaOrdinazioniServite];
    for(unsigned i=0;i<numeroOrdinazioniServite;i++)
        ordinazioniServite[i]=r.ordinazioniServite[i];
}

Ristorante& Ristorante::operator =(const Ristorante& r) {
    if(&r!=this){
        ordinazioniDaServire.clear();
        for(list<Ordinazione*>::const_iterator  it=r.ordinazioniDaServire.begin();
            it!=r.ordinazioniDaServire.end();it++)
            ordinazioniDaServire.push_back(*it);
        delete [] ordinazioniServite;
        numeroOrdinazioniServite=r.numeroOrdinazioniServite;
        capacitaOrdinazioniServite=r.capacitaOrdinazioniServite;
        ordinazioniServite=new Ordinazione*[capacitaOrdinazioniServite];
        for(unsigned i=0;i<numeroOrdinazioniServite;i++)
            ordinazioniServite[i]=r.ordinazioniServite[i];
    }
    return *this;
}

Ristorante::~Ristorante() {
    delete [] ordinazioniServite;
}
```

```
unsigned Ristorante::getNumeroOrdinazioniServite() const {
    return numeroOrdinazioniServite;
}

void Ristorante::svuotaOrdinazioniDaServire() {
    ordinazioniDaServire.clear();
}

void Ristorante::svuotaOrdinazioniServite() {
    delete [] ordinazioniServite;
    ordinazioniServite=0;
    numeroOrdinazioniServite=0;
    capacitaOrdinazioniServite=0;
}

ostream& operator<<(ostream& out, const Ristorante& b){
    for(list<Ordinazione*>::const_iterator
it=b.ordinazioniDaServire.begin();it!=b.ordinazioniDaServire.end();it++)
        out<<(*it)->info();
    return out;
}
```

```
#include "Ordinazione.h"

int Ordinazione::getCodice() const {
    return codice;
}

void Ordinazione::setCodice(int codice) {
    this->codice = codice;
}

const string& Ordinazione::getDescrizione() const {
    return descrizione;
}

void Ordinazione::setDescrizione(const string& descrizione) {
    this->descrizione = descrizione;
}

int Ordinazione::getPrezzo() const {
    return prezzo;
}

void Ordinazione::setPrezzo(int prezzo) {
    this->prezzo = prezzo;
}

string Ordinazione::info(){
    return "Ordinazione Generica "+descrizione;
}
```

```
#ifndef ORDINAZIONETAKEAWAY_H_
#define ORDINAZIONETAKEAWAY_H_

#include "Ordinazione.h"

class OrdinazioneTakeAway : public Ordinazione {
public:
    OrdinazioneTakeAway(int c,float p,string d):Ordinazione(c,p,d){}
    virtual string info() {return "Ordinazione Take Away "+getDescrizione();}
    virtual ~OrdinazioneTakeAway(){}
};

#endif /* ORDINAZIONETAKEAWAY_H_ */
```