Esercizi di Programmazione ad Oggetti

Lista n. 1

Esercizio 1

Definire una classe `IntMod` i cui oggetti rappresentano numeri interi modulo un dato intero n, che deve essere dichiarato come campo dati statico.
Definire metodi statici di `set_modulo()` e `get_modulo()` per tale campo dati statico.
Devono essere disponibili gli operatori di somma e moltiplicazione tra oggetti di `IntMod`.
Definire inoltre opportuni convertitori di tipo affiché questa classe sia liberamente usabile assieme al tipo primitivo `int` e valga la seguente condizione:
quando in una espressione compaiono interi e oggetti di `IntMod` il tipo dell'espressione dovrà essere intero.

Scrivere infine un programma d'esempio che utilizza tutti i metodi della classe.

Soluzione.

```
// IDEA DI RISOLUZIONE:
// i costruttori di IntMod devono essere dichiarati explicit per
// evitare ambiguita'

//file "intmod.h"
#ifndef INTMOD_H
#define INTMOD_H

class IntMod {
private:
  static int modulo;
  int val;
public:
  explicit IntMod(int n=0); // impedisce la conversione implicita int->IntMod

  operator int() const; // conversione implicita IntMod->int

  static int get_modulo();
  static void set_modulo(int);

  IntMod operator+(const IntMod&) const;
  IntMod operator*(const IntMod&) const;

  int print() const;
};
#endif
```

```
//file "intmod.cpp"
#include "intmod.h"

IntMod::IntMod(int n): val(n%modulo) {}

IntMod::operator int() const { return val; }

IntMod IntMod::operator+(const IntMod& m) const {
  cout << ``op + ridefinito''<< endl;
  return IntMod(val + m.val);
}

IntMod IntMod::operator*(const IntMod& m) const {
  cout << ``op * ridefinito''<< endl;
  return IntMod(val * m.val);
```

```
}

int IntMod::get_modulo() {
  return modulo;
}

void IntMod::set_modulo(int mod) {
  modulo=mod;
}

int IntMod::modulo=1;

int IntMod::print()const {return val;}

ostream& operator<<(ostream& os, const IntMod& n){
  return os << n.print() << `` mod `` << IntMod::get_modulo();
}
```

```
//main.cpp
#include <iostream>
#include "intmod.h"
using std::cout; using std::endl;

main() {
  IntMod::set_modulo(4);

  IntMod x,y(3);
  IntMod z=IntMod(7); // IntMod z=7 non va: non sa convertire int a IntMod!!

  cout << x << " " << y  << " " << z << endl; // usa << ridefinito su IntMod!!

  x=y+z; // chiama operator+ ridefinito su IntMod
  cout << "y+z= " << x <<endl;

  x=IntMod(y+10);  // chiama + tra interi e poi costruttore di IntMod
         //x=y+10 non va: cerca di convertire automat. int in IntMod
  cout << "y+10= " << x<<endl;


  x=y+IntMod(1); // usa op+ ridefinito su IntMod
  cout << "y+1= " << x<<endl;
              // come prima x= y+1 non va...

  cout << "y+55= " << y+55 <<endl<<endl; // stampa 58, che non e' modulo 4!!

}
```

Esercizio 2

Il seguente programma compila. Quali stampe produce la sua esecuzione?

```
#include<iostream>
using std::cout;

class A {
private:
  int x;
public:
  A(int k = 5): x(k) {cout << k <<  " A01 ";}
  A(const A& a): x(a.x) {cout << "Ac ";}
  A g() const {return *this;}
};

class B {
private:
  A ar[2];
  static A a;
public:
  B() {ar[1] = A(7); cout << "B0 ";}
  B(const B& b) {cout << "Bc ";}
};
A B::a = A(9);

A Fun(A* p, const A& a, B b) {
  *p = a;
  a.g();
  return *p;
};

main() {
  cout << ``ZERO\n'';
  A a1; cout << "UNO\n";
  A a2(3); cout << "DUE\n";
  A* p = &a1; cout << "TRE\n";
  B b; cout << "QUATTRO\n";

  a1 = Fun(p,a2,b); cout << "CINQUE\n";

  A a3 = Fun(&a1,*p,b); cout << "SEI";
}
```

Soluzione.

```
9 A01 Ac ZERO

5 A01 UNO

3 A01 DUE

NESSUNA STAMPA TRE

5 A01 5 A01 7 A01 B0 QUATTRO

5 A01 5 A01 Bc Ac Ac CINQUE

5 A01 5 A01 Bc Ac Ac Ac SEI
```

Esercizio 3

Perché il seguente programma non compila? Modificare o eliminare una e soltanto una delle righe 1-8 in modo che il programmi compili.

```
class C {
public:                          // 1
  int *const p;                  // 2
  C(int a=0): p(new int(a))      // 3
  { }                            // 4
};

main() {
  C x(3);                        // 5
  C y;                           // 6
  x=y;                           // 7
  C z(y);                        // 8
}
```

Soluzione.

Modificare `int* p; // 2` oppure togliere `7`

Esercizio 4

Il seguente programma compila ed esegue correttamente. Quale stampa di output provoca?

```
#include<iostream>
#include<string>
using std::string; using std::cout;

class C {
private:
  int d;
public:
  C(string s=""): d(s.size()) {}
  explicit C(int n): d(n) {}
  operator int() {return d;}
  C operator+(C x) {return C(d+x.d);}
};

main() {
  C a, b("pippo"), c(3);
  cout << a << ' ' << 1+b << ' ' << c+4 << ' ' << c+b;
}
```

Soluzione.

```
0 6 7 8
```

Esercizio 5

Definire, separando interfaccia ed implementazione, una classe `Raz` i cui oggetti rappresentano un numero razionale $\frac{num}{den}$ (naturalmente, i numeri razionali hanno sempre un denominatore diverso da 0). La classe deve includere:

1. opportuni costruttori;
2. un metodo `Raz inverso()` con il seguente comportamento: se l'oggetto di invocazione rappresenta $\frac{n}{m}$ allora `inverso` ritorna un oggetto che rappresenta $\frac{m}{n}$;
3. un operatore esplicito di conversione al tipo primitivo `double`;
4. l'overloading come metodi interni degli operatori di somma e moltiplicazione;
5. l'overloading come metodo interno dell'operatore di incremento postfisso, che, naturalmente, dovrà incrementare di 1 il razionale di invocazione;
6. l'overloading dell'operatore di output su `ostream`;
7. un metodo statico `Raz uno()` che ritorna il razionale 1.

Definire un esempio di `main()` che usi tutti i metodi della classe.

Esercizio 6

Il seguente programma compila ed esegue correttamente. Quali stampe provoca in output?

```
#include<iostream>
#include<string>
using std::string; using std::cout;

class B {
public:
  string s;
  B(char x='a', char y='b') {s += x; s += y; cout << "B012 ";}
  B(const B& obj): s(obj.s) {cout << "Bc "; }
};

class C {
private:
  B t;
  B* p;
  B u;
public:
  string s;
  C(char x='c', B y = B('d')): u(y), s(y.s) {
    s += x;
    cout << s[s.size()-2] << " C012 ";
  }
};

B F(B x, C& y) {
  (x.s) += (y.s)[0];
  return x;
}

main() {
  B b('e'); cout << "UNO\n";
  C c1('f',b); cout << "DUE\n";
  C c2; cout << "TRE\n";
  b=F(b,c2); cout <<"QUATTRO\n";
  cout << b.s  << " CINQUE";
}
```

Soluzione.

```
B012 UNO
Bc B012 Bc b C012 DUE
B012 Bc B012 Bc b C012 TRE
Bc Bc QUATTRO
ebd CINQUE
```

Esercizio 7

Si consideri il seguente programma.

```
#include<iostream>
using std::cout;

class C {
public:
  int x;
  C(int k=5): x(k) {};
  C* m(C& c) {
    if((c.x != 5) && (x==5)) return &c;
    return this;
  }
};

main() {
  C a, b(2), c(a);
  cout << (b.m(b))->x << ' ' << (a.m(a))->x << ' ' << (b.m(c))->x
       << ' ' << c.m(a) << ' ' << c.m(c);
}
```

Il seguente programma compila correttamente? Se sì, al sua esecuzione quali stampe provoca in output?
Soluzione: Compila e stampa in output `2 5 2` e di seguito 2 indirizzi uguali.

Esercizio 8

Definire, separando interfaccia ed implementazione, una classe `Data` i cui oggetti rappresentano una data con giorno della settimana (lun-mar-...-dom). La classe deve includere:

- opportuni costruttori
- metodi di selezione per ottenere giorno della settimana, giorno, mese, anno di una data
- l'overloading dell'operatore di output esternamente alla classe
- l'overloading dell'operatore di uguaglianza
- l'overloading dell'operatore relazionale < che ignori il giorno della settimana
- un metodo `aggiungi_uno()` che avanza di un giorno la data di invocazione. Esempi: `lun 21/10/2002 => mar 22/10/2002`; `gio 31/1/2002 => ven 1/2/2002`; `mar 31/12/2002 => mer 1/1/2003`. Ignorare gli anni bisestili

Esemplificare l'uso della classe e di tutti i suoi metodi tramite un esempio di `main()`.

Soluzione.

```
//data.h

#ifndef DATA_H
#define DATA_H
#include <iostream>
#include <string>
using std::string; using std::ostream;

class Data {
public:
  Data(string ="", int =1, int =1, int =0);

  string get_gsett() const;
  int get_giorno() const;
  int get_mese() const;
  int get_anno() const;

  bool operator==(const Data&) const;
  bool operator<(const Data&) const;
  void aggiungi_uno();
 private:
  int giorno, mese, anno;
  string gsett;
};

ostream& operator<<(ostream&, const Data&);
#endif
```

```
// data.cpp
#include "data.h"

Data::Data(string s, int g, int m, int a):
                                    gsett(s), giorno(g), mese(m), anno(a) {}

string Data::get_gsett() const {return gsett;}

int Data::get_giorno() const {return giorno;}
```

```
int Data::get_mese() const {return mese;}

int Data::get_anno() const {return anno;}

bool Data::operator==(const Data& d) const {
return ((gsett==d.gsett) && (giorno==d.giorno) &&
        (mese==d.mese) && (anno==d.anno));
}

bool Data::operator<(const Data& d) const {
  if(anno < d.anno) return true;
  if(anno==d.anno && mese<d.mese) return true;
  if ((anno == d.anno) && (mese==d.mese) && (giorno < d.giorno)) return true;
  return false;
}

void aggiungi_uno() { // per casa }

ostream& operator<<(ostream& os, const Data& d) {
  return os << d.get_gsett() << " " << d.get_giorno()
     << "/" << d.get_mese() << "/" << d.get_anno();
}
```

```
// main.cpp
#include "data.h"
#include <iostream>
using std:cout;

main() {
  Data d("lun",21,10,2002), e("mar",22,10,2002);
  cout << d << endl;
  cout << (d<e) << " " << (e==d) << endl;
}
```