Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

class Raz {

private:

  int num, den; // INV: den != 0

public:

  explicit Raz(int n =0, int d =1): num(n), den(d==0? 1 : d)  {}

  // inverso di Raz(0,d) è Raz(0,1)

  Raz inverso() const {

    return Raz(num==0? 0 : den,num);

  }

  operator double() const {

    return static\_cast<double>(num)/static\_cast<double>(den);

  }

  Raz operator+(const Raz& r) const {

    return Raz(num\*r.den + r.num\*den, den\*r.den);

  }

  Raz operator\*(const Raz& r) const {

    return Raz(num\*r.num,den\*r.den);

  }

  bool operator==(const Raz& r) const {

    return num\*r.den == den\*r.num;

  }

  Raz& operator++() {

    num += den;

    return \*this;

  }

  Raz operator++(int) {

    Raz aux(\*this);

    num += den;

    return aux;

  }

  static Raz unTerzo() {

    return Raz(1,3);

  }

};

#include<iostream>

// output del razionale convertito a double

std::ostream& operator<<(std::ostream& os, const Raz& r) {

  return os << "il razionale in virgola mobile è: " <<

    r.operator double();

}

int main() {

  Raz x(2,3), y(2), z, u(1,8), v(3,2), w(8,4);

  std::cout << x+y+v\*u << std::endl; // 2.85417

  std::cout << u.inverso() << std::endl; // 8

  std::cout << (y == w) << std::endl; // true

  std::cout << y++ << " " << ++w << std::endl; // 2 3

  std::cout << (++y + Raz::unTerzo()) << std::endl; // 4.33333

  std::cout << 2 + Raz(1,2) << std::endl; // 2.5

}

Esempio didattico:

Immagine che contiene testo, schermata, schermo, software

Descrizione generata automaticamente

Soluzione

#include <iostream>

using namespace std;

class Z { private: int x;

};

class B { private:

    Z x; };

class D: public B { private:

    Z y; public:

    // ridefinizione di operator= in modo che il suo comportamento coincida con quello dell'assegnazione standard di D

    D& operator=(const D& d){

        if(this != &d){

            B::operator=(d);

            y = d.y;

        }

        return \*this;

    }

};

int main(){

    return 0;

}

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Soluzione:

*9 A01 ZERO*

*5 A01 UNO*

*3 A01 DUE*

*TRE*

*5 A01 5 A01 7 A01 B0 QUATTRO*

*5 A01 5 A01 Bc Ac Ac CINQUE*

*5 A01 5 A01 Bc Ac Ac SEI*

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Soluzione:

*B012 UNO*

*Bc B012 Bc b C012 DUE*

*B012 B012 Bc b C012 TRE*

*Bc Bc QUATTRO*

*ebd CINQUE*

*Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente*

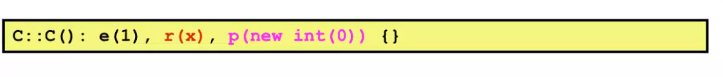
*Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente*

Immagine che contiene testo, schermata, design

Descrizione generata automaticamente

*Soluzione*



*Accenno alle classi container e loro operazioni (costruzione, metodo isEmpty, add, remove, trova\_in\_una\_posizione)*

*Accenno alle classi annidate*

*Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente*

*Accenno ai distruttori e loro uso rispetto ai costruttori*:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Soluzione:

*UNO*

*N2 N2 N2 N2 DUE*

*pluto~N paperino~N topolino~N pippo~N ~C() TRE*

*Accenno alle tipologie di esercizio da esame (gerarchie, sottotipi, funzioni, template – cosa stampa)*

Eventuali altri esercizi:

#include <iostream>

using namespace std;

class C {

    public:

    C() {}

    C(const C& r) {

        cout << "\*";

    }

};

C f(C a) {

  C b(a);

    C c = b;

    return c;

}

int main() {

    C x;

    C y = f(f(x));

}

Soluzione:

#include <iostream>

using namespace std;

class C {

    public:

    C() {}

    C(const C& r) {

        cout << "\*";

    }

};

C f(C a) {

  C b(a);

    C c = b;

    return c;

}

int main() {

    C x;

    C y = f(f(x)); // \*\*\*\*\*

}

// Si considerino le seguenti definizioni, la cui compilazione non provoca errori.

class C {

public:

static void f(const C& x) {}

};

class D {

public:

D(C x = C()) {}

void g() const {}

};

class E {

public:

E(D x = D()) {}

operator C() const {return C();}

static void h(const E& x) {C::f(x);}

void i() const {C::f(\*this);}

};

C c; D d; E e;

// Delle seguenti istruzioni, quali compilano e quali no?

// E::h(c);

// c.g();

// E::h(d);

// e.i();

// C::f(d);

// C::f(e);

// d.i();

// E e1(c);

// D d1(c);

// C c1(e);

// C c2(d);

Soluzione

// Si considerino le seguenti definizioni, la cui compilazione non provoca errori.

class C {

public:

static void f(const C& x) {}

};

class D {

public:

D(C x = C()) {}

void g() const {}

};

class E {

public:

E(D x = D()) {}

operator C() const {return C();}

static void h(const E& x) {C::f(x);}

void i() const {C::f(\*this);}

};

C c; D d; E e;

// Delle seguenti istruzioni, quali compilano e quali no?

E::h(c); // NON COMPILA

c.g(); // NON COMPILA

E::h(d); // COMPILA

e.i(); // COMPILA

C::f(d); // NON COMPILA

C::f(e); // COMPILA

d.i(); // NON COMPILA

E e1(c); // COMPILA

D d1(c); // COMPILA

C c1(e); // COMPILA

C c2(d); // NON COMPILA

//Cosa stampa?

#include <iostream>

using namespace std;

class C {

 public:

   C(int x=6): k(x) {cout << k << " C01 ";}

   C(const C& x): k(x.k) {cout << k << " Cc ";}

   C& operator=(const C& x) {return \*this; cout << "C= ";}

   int k;

 };

 class D {

 public:

   D(): z2(8) {cout << "D0 ";}

   D(const D& x): z1(x.z1.k) {cout << "Dc ";}

   D& operator=(const D& x) {z1=x.z1; z2=x.z2; return \*this; cout << "D= ";}

   C\* getUno() {return &z1;}

   C z1, z2;

 };

 class E {

 public:

   E(D x) {y=x; cout << "E(D) ";}

   E(const E& x) {cout << "Ec ";}

   E& operator=(const E& x) {cout << "E= "; return \*this;}

   D y;

   static C x;

};

C E::x = 9;

 main() {

   cout << "\*\*1\n"; // 9 C01

   C c(7); cout << "\*\*2\n"; // 7 C01

   D d; cout << "\*\*3\n"; // 6 C01 8 C01 D0

   c = \*(d.getUno()); cout << "\*\*4\n"; // ""

   c = d.z1.k; cout << "\*\*5\n"; // 6 C01

   E e1(d); cout << "\*\*6\n"; // 6 C01 6 C01 Dc 6 C01 8 C01 D0 E(D)

   E e2 = e1; cout << "\*\*7\n"; // 6 C01 8 C01 D0 Ec

   e2 = e1; cout << "\*\*8"; // E=

}

Soluzione

//Cosa stampa?

#include <iostream>

using namespace std;

class C {

 public:

   C(int x=6): k(x) {cout << k << " C01 ";}

   C(const C& x): k(x.k) {cout << k << " Cc ";}

   C& operator=(const C& x) {return \*this; cout << "C= ";}

   int k;

 };

 class D {

 public:

   D(): z2(8) {cout << "D0 ";}

   D(const D& x): z1(x.z1.k) {cout << "Dc ";}

   D& operator=(const D& x) {z1=x.z1; z2=x.z2; return \*this; cout << "D= ";}

   C\* getUno() {return &z1;}

   C z1, z2;

 };

 class E {

 public:

   E(D x) {y=x; cout << "E(D) ";}

   E(const E& x) {cout << "Ec ";}

   E& operator=(const E& x) {cout << "E= ";return \*this;}

   D y;

   static C x;

};

C E::x = 9;

 int main() {

   cout << "\*\*1\n"; // 9 C01 9 Cc-> Costruito oggetto anonimo (r.7), poi viene chiamato il

   // costruttore di copia (r.8) e assegnato ad x (r.28)

   C c(7); cout << "\*\*2\n"; // 7 C01 -> Chiamata espl. al costruttore C

   D d; cout << "\*\*3\n"; // 6 C01 8 C01 D0 -> Creo un D (r.14), inizializza prima z1 poi z2 (r.18)

   // con il costruttore D (r.7)

   c = \*(d.getUno()); cout << "\*\*4\n"; // "" -> richiama getUno (r.17) ritornando z1 per rif.,

   // poi lo deferenzia e fa assegnazione a c. NB tutto ciò che sta dopo il return non viene considerato

   c = d.z1.k; cout << "\*\*5\n"; // 6 C01 -> Viene preso il valore k attraverso il costruttore di C (r.7),

   // creando il nuovo oggetto avremmo assegnato 6

   E e1(d); cout << "\*\*6\n"; // 6 C01 6 C01 Dc 6 C01 8 C01 D0 E(D) ->Viene richiamato il costruttore E (r.22),

   prende un parametro per valore, quindi chiama costruttore di copia D (r.15)

   // costruendo z1 con il parametro x che è a sua volta il valore d(=6), poi costruisce z2, chiamando il

    costruttore C; ora si creano tutti i campi dati della classe E

   // non statici, chiamando costruttore D (r.25), arrivando al corpo del costruttore E

   E e2 = e1; cout << "\*\*7\n"; // 6 C01 8 C01 D0 Ec -> Chiamato costruttore copia di E (r.23) e creato oggetto y, poi richiamati

   // i costruttori di C e D

   e2 = e1; cout << "\*\*8"; // E= -> Chiamato operatore E& (r.24)

}