Cosa stampa questo programma?

```
/*
  Lo scopo di questo esercizio è quello di verificare la conoscenza
  di alcune nozioni la cui comprensione è essenziale per affrontare
  (in maniera consapevole) le tematiche relative alla corretta gestione
  delle risorse e, in particolare, della exception safety.

  Le nozioni in questione sono:
    - tempo di vita degli oggetti
    - costruzione e distruzione di oggetti
    - composizione di tipi di dato (aggregazione e ereditarietà)
    - flussi di esecuzione eccezionali

  L'esercizio consiste nel *prevedere* l'output prodotto dal programma
  (indicando con precisione la sequenza corretta dei vari messaggi).
  Sebbene sia possibile (e facile) compilare ed eseguire il codice per
  *vedere* l'output, il consiglio è quello di farlo soltanto in un secondo
  momento, come utile feedback per verificare se le proprie previsioni
  erano accurate e, nel caso, chiedersi come mai non lo erano.
  Tipicamente, si scoprirà di essere incappati in qualche banale svista,
  ma in alcuni casi occorrerà ammettere che certi meccanismi di base
  non erano stati compresi a fondo.
*/

#include

// ----------------------------

struct C1 {
  int i1;
  C1() {
    std::cerr << "Constructor C1::C1()" << std::endl;
  }
  ~C1() {
    std::cerr << "Destructor C1::~C1()" << std::endl;
  }
};

// ----------------------------

struct C2 {
  int i2;
  C2() {
    std::cerr << "Constructor C2::C2()" << std::endl;
    throw 123;
  }
  ~C2() {
    std::cerr << "Destructor C2::~C2()" << std::endl;
  }
};

// ----------------------------

struct C3 : public C1 {
  int i3;
  C3() {
    std::cerr << "Constructor C3::C3()" << std::endl;
  }
  ~C3() {
    std::cerr << "Destructor C3::~C3()" << std::endl;
  }
};

// Memory layout per il tipo C3: { C1:{ i1 }, i3 }

// ----------------------------

class D : public C1 {
private:
  int i4;
  C3 c3;
  C2 c2;
  C1 c1;

public:
```

```
  D() : c1(), c2(), c3() {
    std::cerr << "Constructor D::D()" << std::endl;
  }
  ~D() {
    std::cerr << "Destructor D::~D()" << std::endl;
  }
};

// Memory layout per il tip D:
// { C1:{ i1 }, i4, c3:C3{ C1:{ i1 }, i3 }, c2:C2{ i2 }, c1:C1{ i1 } }

// ---------------------------

C3 c3;

// ---------------------------

int main() {
  std::cout << "Start" << std::endl;
  try {
    C1 c1;
    D d;
  }
  catch (char c) {
    std::cerr << "char " << c << std::endl;
  }
  catch (...) {
    std::cerr << "..." << std::endl;
  }
  std::cout << "End" << std::endl;
  return 0;
}

// ---------------------------

C1 c1;

// ---------------------------
```

Soluzione

```
Constructor C1::C1()
Constructor C3::C3()
Constructor C1::C1()
Start
Constructor C1::C1()
Constructor C1::C1()
Constructor C1::C1()
Constructor C3::C3()
Constructor C2::C2()
Destructor C3::~C3()
Destructor C1::~C1()
Destructor C1::~C1()
Destructor C1::~C1()
...
End
Destructor C1::~C1()
Destructor C3::~C3()
Destructor C1::~C1()
```