Programmazione 2
Test di metà corso – 30/10/2006

Nome..... Cognome.....

Matricola..... Laurea in.....

Non si possono consultare appunti e libri. Dove previsto scrivere CHIARAMENTE la risposta nell'apposito spazio. ATTENZIONE: In tutti gli esercizi si intende la compilazione standard `g++` con il flag `-fno-elide-constructors`.

Esercizio 1

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;
```

Si supponga che tali definizioni siano visibili ad ognuna delle seguenti istruzioni. Per ognuna di tali istruzioni si scriva nell'apposito spazio contiguo:

- **COMPILA** se la compilazione dell'istruzione non provoca alcun errore;
- **NON COMPILA** se la compilazione dell'istruzione provoca un errore.

`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);`	

Ordine delle istruzioni:
1 - NO
2 - SI
3 - SI
4 - SI
5 - NO
6 - SI
7 - NO
8 - SI
9 - SI
10 - SI
11 - NO

Esercizio 2

```
class N {
  friend class Lista;
private:
  int x;
  N* prev; N* next;
public:
  N(int y, N* p = 0, N* q = 0): x(y), prev(p), next(q) {}
  ~N() {if(prev) delete prev; cout << x << " ~N ";}
};
class Lista {
private:
  N* first;
  static N* copia(N* p) {
    if(!p) return 0;
    N* primo = new N(p->x), *q = primo;
    while(p->next) {
      p=p->next;
      q->next = new N(p->x,q,0);
      q = q->next;
    }
    return primo;
  }
public:
  Lista() : first(0) {}
  Lista(const Lista& x) : first(copia(x.first)) {}
  ~Lista() {
    if(first) {
      while(first->next) first = first->next;
      delete first;
    }
  }
  void add(int x) {
    N* p = new N(x,0,first);
    if(first) first->prev = p;
    first = p;
  }
  int* operator++(int) {
    if(!first) return 0;
    int* p= &(first->x);
    first = first->next;
    return p;
  }
  static void print(const Lista& x) {
    N* p = x.first;
    while(p) {
      cout << p->x << " "; p = p->next;
    }
  }
};
void stampa(Lista x) {
  int* p = x++;
  while(p) {
    cout << *p << " "; p = x++;
  }
}
```

La compilazione delle precedenti definizioni non provoca errori (con gli opportuni `#include` e `using`). Si supponga che tali definizioni siano visibili ad ognuna delle seguenti funzioni `main()`, le cui compilazioni non provocano errori. Per ognuno di tali `main()` si scrivano nelle apposite righe numerate le stampe prodotte dalla sua esecuzione e si usi l'ultima riga non numerata per le eventuali stampe successive all'ultima stampa di `**n`. Se una riga non contiene alcuna stampa (oltre a quella già indicata) si scriva **NESSUNA STAMPA**. Se una esecuzione di un `main()` dovesse provocare un errore a run-time si scriva **ERRORE RUN-TIME**.

```
main() {
  Lista x;
  x.add(6); x.add(3); x.add(5); x.add(4);
  x++; x++;
  stampa(x); cout << " **1\n";
}
```

3 6 **1 ..... **1

4 ~N 5 ~N 3 ~N 6 ~N .....

```
main() {
  Lista x; x.add(3); x.add(5); x.add(3);
  Lista y(x); y++;
  Lista::print(x); cout << " **1\n";
  Lista::print(y); cout << " **2\n";
}
```

3 5 3 **1 ..... **1

5 3 **2 ..... **2

3 ~N 5 ~N 3 ~N 3 ~N 5 ~N 3 ~N .....

```
main() {
  Lista* p = new Lista; Lista y;
  p->add(3); p->add(5); p->add(3); p->add(6);
  y=*p; (y)++;
  Lista::print(*p); cout << " **1\n";
  Lista::print(y); cout << " **2\n";
}
```

6 3 5 3 **1 ..... **1

3 5 3 **2 ..... **2

6 ~N 3 ~N 5 ~N 3 ~N .....

Esercizio 3

```
class Nodo;

class Smart {
  friend class Lista;
private:
  Nodo* punt;
public:
  Smart(Nodo* p=0): punt(p) {}
  ~Smart();
  bool operator==(const Smart& s) const {return punt==s.punt;}
  Nodo* operator->() const {return punt;}
  Smart& operator=(const Smart&);
};

class Nodo {
  friend class Lista; friend class Smart;
private:
  int x;
  Smart next;
public:
  Nodo(int z =0, const Smart& n = 0): x(z), next(n) {}
  ~Nodo() {cout << " ~N";}
};

Smart& Smart::operator=(const Smart& s) {
  Nodo* t = punt;
  punt = new Nodo(s->x);
  delete t;
  return *this;
};

Smart::~Smart() {if(punt) delete punt; cout << " ~S";}

class Lista {
private:
  Smart first;
public:
  Lista() : first(0) {}
  Lista(int k): first(new Nodo(k)) {if(k>0) first->next = new Nodo(k+1);}
  void remove() {if(!(first==0)) first = first->next;}
};

main() {
  Lista x1; cout << " **1\n";
  Lista x2(5); cout << " **2\n";
  Lista* p = new Lista(3); cout << " **3\n";
  delete p; cout << " **4\n";
  Lista x3(0); cout << " **5\n";
  x2.remove(); cout << " **6\n";
}
```

Il precedente programma compila correttamente (con gli opportuni `#include` e `using`). Si scrivano nelle apposite righe numerate le stampe prodotte dalla sua esecuzione e si usi l'ultima riga non numerata per le eventuali stampe successive all'ultima stampa di `**6`. Se una riga non produce alcuna stampa (oltre a quella già indicata) si scriva **NESSUNA STAMPA**.

Esercizio 4

Si supponga che ognuno dei seguenti frammenti sia il codice di uno o piú metodi che appartengono alla parte pubblica di una qualsiasi classe `C`. Si scriva nell'apposito spazio contiguo:

- **COMPILA** se la compilazione del codice non provoca alcun errore;
- **NON COMPILA** se la compilazione del codice provoca un errore.

`C f(C& x) {return x;}`	Compila
`C& g() const {return *this;}`	Non compila
`C h() const {return *this;}`	Compila
`C* m() {return this;}`	Compila
`C* n() const {return this;}`	Non compila
`void p() {}` `void q() const {p();}`	Non compila
`void p() {}` `static void r(C *const x) {x->p();}`	Compila
`void s(C *const x) const {*this = *x;}`	Non compila
`static C& t() {return C();}`	Non compila
`static C *const u(C& x) {return &x;}`	Compila

Esercizio 5

```
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= ";}
  D y;
  static C x;
};
C E::x = 9;

main() {
  cout << "**1\n";
  C c(7); cout << "**2\n";
  D d; cout << "**3\n";
  c = *(d.getUno()); cout << "**4\n";
  c = d.z1.k; cout << "**5\n";
  E e1(d); cout << "**6\n";
  E e2 = e1; cout << "**7\n";
  e2 = e1; cout << "**8";
}
```

Il precedente programma compila correttamente (con gli opportuni `#include` e `using`). Si scrivano nelle apposite righe numerate le stampe prodotte dalla sua esecuzione. Se una riga non produce alcuna stampa (oltre a quella già indicata) si scriva **NESSUNA STAMPA**.

9 C01 **1 ..... **1

7 C01 **2 ..... **2

6 C01 8 C01 D0 **3 ..... **3

NESSUNA STAMPA **4

6 C01 **5 ..... **5

6 C01 6 C01 Dc 6 C01 8 C01 D0 E(D) **6 ..... **6

6 C01 8 C01 D0 Ec **7 ..... **7

E= **8 ..... **8