Programmazione 2
Test di metà corso – 29/10/2007

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

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

Non si possono consultare appunti e libri. Ogni quesito a risposta multipla ha ESATTAMENTE una risposta corretta: segnare con una croce la risposta scelta. Le risposte errate vengono penalizzate. 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 consideri il seguente programma.

```
class A {
private:
  int x;
public:
  A(int k=0): x(k) {}
  A operator+(A a) const {A aux(x + a.x); return aux;}
  operator int() const {return x;}
  A F() {++x; return *this;}
};

class C {
public:
  static A a;
  static void G(A& x, A& y) {a = a + x.F() + y.F();}
};
A C::a(1);

main() {
  A p(2), q, r;
  C::G(p,q); cout << C::a << " ";
  C::G(q,r); cout << C::a << " ";
  C::G(p,r); cout << C::a;
}
```

1. stampa in output: `5 7 11`
2. stampa in output: `4 6 10`
3. stampa in output: `5 8 14`
4. stampa in output: `4 7 13`

Esercizio 2

Si completi la frase: "Il costruttore di default standard ..." in modo da ottenere un'affermazione corretta:

1. "... è sempre disponibile in ogni classe"
2. "... è disponibile quando non è definito esplicitamente il costruttore di default"
3. "... è disponibile quando non ha argomenti"
4. "... è disponibile se non sono stati definiti esplicitamente altri costruttori"

Esercizio 3

```
class N {
  friend class Lista;
private:
  int info;
  N* prev; N* next;
public:
  N(int y, N* p = 0, N* q = 0): info(y), prev(p), next(q) {}
  ~N() {if(next) delete next; cout << info << " ~N ";}
};

class Lista {
private:
     N* last;
public:
  Lista() : last(0) {}
  ~Lista() {
    if(last) {
       while(last->prev) last = last->prev;
       delete last;
    }
  }
  void add(int x) {
    N* p = new N(x,0,0);
    if(last) {last->next = p; p->prev = last;}
    last = p;
  }
  void operator--() { if(last) last = last->prev; }

  void operator--(int) {
    if(last)
      { N* p = last;
        last = last->prev;
        last->next = 0;
        delete p;   }
  }
  void print() {
    if(last) {
      N* p = last;
      while(p->prev) p = p->prev;
      while(p)
        { cout << p->info << " "; p = p->next; }
    } }
};
main() {
  Lista* p = new Lista; Lista* q = new Lista;
  p->add(3); p->add(4); p->add(5); p->add(6);
  q->add(7);
  *q=*p;
  --(*q);
  p->print(); cout << " **1\n";
  q->print(); cout << " **2\n";
  p->add(8);
  p->print(); cout << " **3\n";
  (*q)--; cout << " **4\n";
  q->print(); cout << " **5\n";
  --(*q); delete q; cout << " **6\n";  }
```

Il precedente programma compila correttamente. Si scrivano nelle apposite righe le stampe prodotte dalla sua esecuzione e si usi l'ultima riga 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**. Se una riga dovesse provocare un errore a run-time si scriva **ERRORE RUN-TIME**, e si lascino vuote le righe successive.

..... `**1`

..... `**2`

..... `**3`

..... `**4`

..... `**5`

..... `**6`

.....

Esercizio 4

Si considerino le classi `N` e `Lista` definite nell'Esercizio 3. Si supponga che il distruttore della classe `Lista` venga ridefinito nel seguente modo: `~Lista(){ delete last; }`. Si chiede di ridefinire il distruttore della classe `N` in modo tale che la distruzione di un oggetto di tipo `Lista` comporti la distruzione profonda della lista.

Esercizio 5

Si considerino le classi `N` e `Lista` definite nell'Esercizio 3. Si chiede di

- definire una classe `Iteratore` che permette di iterare sugli elementi di una lista. La classe deve contenere un solo metodo pubblico che ridefinisce l'operatore di incremento prefisso
- ridefinire per la classe `Lista` l'operatore di subscripting.

Considerando le definizioni della classe `Iteratore` e dell'operatore di subscripting appena scritte, indicare quali delle seguenti dichiarazioni `friend` sono **necessarie** (l'indicazione di dichiarazioni non necessarie sarà penalizzata).

1. nella classe `N` si deve dichiarare `friend class Iteratore`
2. nella classe `Lista` si deve dichiarare `friend class Iteratore`
3. nella classe `Iteratore` si deve dichiarare `friend class N`
4. nella classe `Iteratore` si deve dichiarare `friend class Lista`
5. nella classe `Lista` si deve dichiarare `friend class N`

Esercizio 6

```
class S {
public:
  int s;
  S(int x): s(x) {}
};

class N {
private:
  S x;
public:
  N* next;
  N(S t, N* p): x(t), next(p) {cout << "N2 ";}
  ~N() {if (next) delete next; cout << x.s << " ~N ";}
};

class C {
  N* pointer;
public:
  C(): pointer(0) {}
  ~C() {delete pointer; cout << "~C ";}
  void F(int a, int b = 9) {
    pointer = new N(S(a),pointer); pointer = new N(b,pointer);
  }
};

main(){
  C* p = new C; cout << "UNO\n";
  p->F(3,5); p->F(7); cout <<"DUE\n";
  delete p; cout <<"TRE\n";
}
```

Il precedente programma compila correttamente. Quali stampe produce la sua esecuzione? Se una istruzione provoca un errore a run-time si scriva **ERRORE RUN-TIME**, se non produce alcuna stampa si scriva **NESSUNA STAMPA**. Si usi l'ultima riga per eventuali stampe che dovessero seguire "`TRE`".

..... `UNO`

..... `DUE`

..... `TRE`

.....