Programmazione ad oggetti – Laurea in Informatica
Appello d'Esame – 7/4/2011

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

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

Non si possono consultare appunti e libri. Dove previsto scrivere CHIARAMENTE la risposta nell'apposito spazio.

Quesito 1

```
class A {
  int x[10];
};

class B : virtual public A {
public:
  int y;
};

class C : virtual public A {
public:
  int z;
};

class D : public B, public C {
public:
  int w;
};

int main() {

  cout << sizeof(A) << endl;
  cout << sizeof(B) << endl;
  cout << sizeof(C) << endl;
  cout << sizeof(D) << endl;

  return 0;
}
```

Il precedente programma compila correttamente. (con gli opportuni `include` e `using`). Si scrivano le stampe prodotte dalla sua esecuzione in una architettura a 32 bit. 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.

.....

.....

.....

.....

Quesito 2

```
class A {
private:
  int x;
public:
  A(int k=0): x(k) { cout << "A(int) ";}
  ~A() { cout << "~A ";}
};

class B : public A {
public:
  static A a;
  A F(A a=A()) { return a; }
  B() : A(3) { cout << "B() ";}
  ~B() { cout << "~B ";}
};
A B::a(1);

int main() {
  A a(1); cout << "UNO" << endl;
  B b; cout << "DUE" << endl;
  b.F(); cout << "TRE" << endl;
  b.F(2); cout << "QUATTRO" << endl;
  return 0;
}
```

Il precedente programma compila correttamente (con gli opportuni `include` e `using`). Si scrivano le stampe prodotte dalla sua esecuzione. 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. Il programma si suppone compilato con l'opzione -fno-elide-constructors.

.....

.....

.....

.....

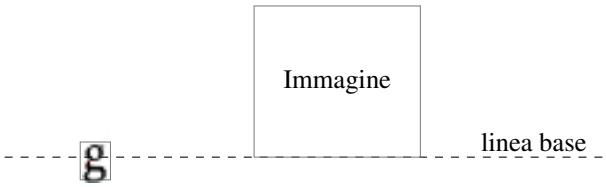
.....

Quesito 3

Si vuole affrontare il problema della suddivisione di un paragrafo in linee di testo all'interno di una pagina. A tale fine ogni paragrafo sarà caratterizzato dalla propria larghezza e dalla sequenza degli elementi di testo che contiene, ognuno dei quali potrà essere un carattere, un'immagine o un elemento di spaziatura.

I diversi elementi testuali, ognuno dotato di una propria altezza e larghezza, sono posti consecutivamente all'interno di ogni riga, la cui larghezza totale è esattamente la somma delle larghezze degli elementi testuali che contiene.

Ai fini della disposizione in verticale degli elementi testuali, ogni riga si pensa dotata di una linea ipotetica, detta linea base: ogni immagine poggia su tale linea, mentre nel caso dei caratteri, parte del simbolo deve stare sopra e parte sotto la linea base. La figura seguente chiarisce meglio la situazione:



Al fine di modellare tale realtà si definisca la seguente gerarchia di classi.

1. Definire una classe base `ElementoTestuale`, che fornirà l'interfaccia comune per le altre classi della gerarchia, ed i cui oggetti rappresentano un generico elemento che può essere inserito in un paragrafo. Ogni oggetto di `ElementoTestuale` è caratterizzato dalla sua larghezza (in pixel). D'altra parte un oggetto di `ElementoTestuale` non è dotato di altezza (che sarà invece definita specificamente per ogni sottotipo di `ElementoTestuale`), ma piuttosto la classe classe `ElementoTestuale` fornisce un metodo virtuale puro `altezza`.

2. Definire una classe `Spaziatura`, derivata da `ElementoTestuale`. Ogni oggetto di questa classe sarà caratterizzato convenzionalmente da una altezza zero.

3. Definire una classe `Carattere`, derivata da `ElementoTestuale`, i cui oggetti sono caratterizzati da

   (a) il codice `ASCII` del carattere (un intero in [0,255])

   (b) un'altezza superiore (l'altezza della parte del carattere al di sopra della linea base)

   (c) un'altezza inferiore (l'altezza della parte del carattere al di sopra della linea base).

   Quindi l'altezza di un `Carattere` sarà pari alla somma della sua altezze superiore ed inferiore.

4. Definire una classe `Immagine`, derivata da `ElementoTestuale`, i cui oggetti sono caratterizzati da

   (a) un'altezza

   (b) il nome del file ove l'immagine è fisicamente immagazzinata nel disco (il file va cercato rispetto alla cartella corrente del processo di esecuzione).

   Ogni immagine ha un margine fisso (quindi uguale per tutte le immagini) che deve essere lasciato vuoto intorno ad esse (sia a sinistra che a destra). Pertanto la larghezza di una immagine deve includere anche questi due margini.

5. Definire una classe `Paragrafo` i cui oggetti rappresentano un paragrafo all'interno di una pagina e sono caratterizzati dalla larghezza del paragrafo e dalla sequenza di elementi testuali che compongono il paragrafo.

   (a) definire un operatore di output che visualizzi il tipo e le caratteristiche di tutti gli elementi testuali di un paragrafo.

   (b) definire un metodo `searchMaxHeight` ritornante un elemento testuale del paragrafo che (A) sia un carattere e (B) abbia altezza massima tra tutti i caratteri. Se il paragrafo non contiene nessun elemento che sia un carattere, il metodo deve sollevare un'opportuna eccezione.

   (c) definire un metodo `render` che ritorna una sequenza di sequenze di elementi testuali, ognuna delle quali rappresenta una riga del paragrafo, con la condizione che la larghezza complessiva di ogni riga sia o minore di quella del paragrafo, o ecceda tale larghezza solo per l'ultimo elemento testuale.

   (d) definire un metodo `height` che ritorna l'altezza necessaria a rappresentare la renderizzazione di un paragrafo ottenuta con il precedente metodo `render`.

   (e) definire un esempio di metodo `main` che utilizzi tutti le funzionalità della classe `Paragrafo`.

NB: Scrivere la soluzione **chiaramente** nel foglio a quadretti. Per comodità di correzione, definire tutti i metodi inline.