Università di Firenze – Scuola di Ingegneria
Corso di Laurea in Ingegneria Informatica - Prof. M. Bertini

Programmazione

PROVA SCRITTA – 18 Luglio 2022
Parte I – LINGUAGGIO C++ e OOP

1 (10 punti)

Si discuta come viene implementata la programmazione generica in C++, riferendosi sia a classi che funzioni. Indicare come si deve organizzare il codice.

2 (10 punti)

Spiegare quali differenze ci sono tra ereditarietà multipla e singola nel C++ e come trattare il caso di derivazione multipla da classi collegate (problema del diamante). Discutere la differenza tra ereditarietà pubblica e privata, con particolare riferimento agli effetti di design.

3 (10 punti)

Si discutano le caratteristiche ed i vantaggi del polimorfismo in C++. Discutere le caratteristiche dei distruttori in classi polimorfiche. Descrivere sintassi e uso dei metodi puramente virtuali.

Programmazione

PROVA SCRITTA – 18 Luglio 2022
Parte II – PROGRAMMAZIONE

4 (14 punti)

Sia data una struct `Pixel` che rappresenta i tre valori R, G e B di un pixel di un'immagine, compresi tra 0 e 255.
Sia data la seguente classe `Image` che rappresenta un'immagine. Il metodo `setPixel` imposta il valore di un pixel alle coordinate (x,y) modificando adeguatamente il contenuto del buffer dei Pixel (implementare il metodo), ed un metodo getPixel che ritorna il valore di un pixel alle coordinate richieste; i due metodi lanciano un'eccezione di tipo `std::out_of_bound` in caso le coordinate non siano accettabili (si devono implementare i metodi).
La classe ha un metodo `save()` che salva sul file dal nome specificato l'immagine, ed un metodo `load()` che legge un'immagine da file; questi metodi usano quattro funzioni che fanno parte di una qualche libreria per la lettura e scrittura di file di immagini, ovvero `decompressImage`, `compressBuffer`, `saveCompressedBuffer` e `freeBuffer` (non si devono implementare questi metodi) riportati nel codice.
Completare la classe per fare in modo che descriva un'immagine, con dimensione orizzontale e verticale; dato che l'operazione di compressione è lenta si evitino i salvataggi inutili dell'immagine. (9 punti).
Indicare se la classe necessita di deep copy motivando la risposta e nel caso implementarla. (5 punti)

```
class Image {
public:
      ......;
      void load(std::string name) {
         buffer = decompressImage(name);
      }
      void save(std::string name) {
         Pixel* compressedBuffer = compressBuffer(buffer);
         saveCompressedBuffer(name, compressedBuffer);
         freeBuffer(compressedBuffer);
      }
      void setPixel(int x, int y, const Pixel& pix);
      Pixel getPixel(int x, int y);
private:
       Pixel* buffer;
};
```

5 (12 punti)

Si implementi una classe `AudioTrack` che rappresenta una canzone con titolo, durata in secondi e nome del file dell'immagine della copertina dell'album di cui fa parte. La classe ha un metodo `play()` che suona la canzone. (2 punti)
Si implementi quindi una classe `Playlist`, che include un elenco ordinato di `AudioTrack` da suonare in sequenza; la playlist ha un nome. Si devono fornire metodi per aggiungere in coda una canzone (`addTrack()`), eliminare una canzone dato il titolo (`removeTrack()`) o suonare direttamente una canzone dato il titolo (`playTrack()`). La classe ha un metodo `play()` che suona le canzoni secondo la loro sequenza data. (6 punti)
Si implementi una classe `MusicLibrary` che contiene `Playlist` e consente di inserirle, cercarle e suonarle sulla base del loro nome. (4 punti)

- Nel caso si usi una `std::map<Key K, MappedType T>` si ricorda la presenza dei seguenti metodi:

  `T& operator[]( const Key& key );` ritorna un riferimento al valore mappato sulla chiave equivalente a quella passato come argomento, effettuando un inserimento se tale chiave non esiste.

  `iterator find( const Key& key );` cerca un elemento con chiave equivalente a quella indicata; rende l'iteratore all'elemento se presente o all'iteratore prima posizione dopo la fine se non presente (end()).

  `void erase( iterator pos );` rimuove l'elemento alla posizione indicata dall'iteratore

  e che i valori presenti nella mappa sono del tipo: `std::pair<const Key, T>` con attributi pubblici first per la chiave e second per il valore associato.
- Nel caso si usi una `std::list<ValueType T>` si ricorda la presenza dei seguenti metodi:

  `void push_back( const T& value );` aggiunge l'elemento in coda alla lista

  `iterator erase( iterator pos );` rimuove l'elemento alla posizione indicata dall'iteratore

  e l'algoritmo find:

  ```
  template< class InputIt, class T >
  InputIt find( InputIt first, InputIt last, const T& value );
  ```

  che cerca nell'intervallo tra l'iteratore first ed il last un valore. L'algoritmo usa l'operatore == che deve essere definito per il tipo su cui si applica.
  Si ricorda che la firma di tale operatore è:

  ```
  bool operator==(const T & right) const
  ```

6 (12+2 punti)

Si supponga di avere un programma di grafica che consente di disegnare forme geometriche, tra cui rettangoli. Per questo si ha una classe `Rectangle` il cui costruttore riceve in ingresso le coordinate X e Y dell'angolo in alto a sinistra e in basso a destra. La classe ha un metodo che disegna il rettangolo a partire da queste coordinate.
Nella nuova versione del programma si deve poter usare una nuova rappresentazione del rettangolo dove si usano le coordinate cartesiane dell'angolo in alto a sinistra oltre che larghezza e altezza.
Per compatibilità con i file dei disegni delle vecchie versioni si deve continuare ad usare la vecchia classe `Rectangle` nel nuovo programma che si aspetta di rappresentare i rettangoli usando la nuova rappresentazione. Si implementi una soluzione basata sul design pattern Adapter (12 punti).
Disegnare il diagramma di classe UML del pattern usato per risolvere il problema (2 punti).