Problem 1 (10 Points)

Answer the following questions, and explain your answers thoroughly. The code that is the subject of this question is on page 6.

1. What is the minimum code that needs to be added to `ChildrensMovie` in order for this class to compile?

2. Do the two lines in the `print` function in `ChildrensMovie` behave identically?

3. Assume that you make the necessary changes to `ChildrensMovie` so that it compiles. Of the four declarations in function, `main`, which are legal and which are not?
Problem 2 (15 Points)

Insert the following values, one at a time, into an empty AVL tree. After each insertion, draw the tree and if necessary, balance it. Be sure to

1. adjust the balance-factor of the nodes that are affected by the insertion.

2. if the insertion triggers a rotation, identify the node whose balance factor is not within the acceptable range and apply the necessary rotation.

3. show clearly the balance factor of the nodes of the final tree.

100, 50, 40, 20, 10, 120, 400, 60, 30, 35, 25, 15
Problem 3 (25 Points)

For this problem, which is based on your third project, you are given the implementation of Artist, Artists, Album, and Albums and are asked to write the necessary code to assign to each artist the list of his/her albums. Specifically, your task is to write implementations for the following two functions. In doing so, you should assume that you have access to all other functions in these classes. The classes that your solution should be based on are on pages 7 and 8.

Albums *Albums::albumsForArtist(Artist *artist);
// Create a new instance of Albums, store all albums that belong to
// parameter "artist" in it based on the matching artist-ids, and return it.

void Artists::setAlbumsForArtists(Albums *albums);
// Using Albums::albumsForArtist and Artist::setAlbums,
// assign to each artist his/her albums.
Problem 4 (25 Points)

For this problem, you will write the partition function for Quick-sort. Recall that this function takes a vector and the index of the pivot element (the element around which the array is to be partitioned) and partitions the vector into three parts:

1. part 1: elements that are less than the pivot element.
2. part 2: the pivot element.
3. part 3: elements that are larger than the pivot element.

For example, suppose the vector contained the following values and that the index of the pivot element is 2 (the pivot element is 50).

23, 65, 50, 9, 70, 100, 33

Then, your function should partition the values around 50. The following is an acceptable partition.

23, 9, 33, 50, 65, 70, 100

Observe that the index of the pivot element after the partition is 3. After the partition, the order of the values that are less than the pivot element is not important so long as they appear before the pivot element. Likewise, the order of the elements that are larger than the pivot element is not important so long as they appear after the pivot element. Feel free to use a temporary vector if you find it helpful.

```
int partition(std::vector<int>& values, int n, int pivotIdx)
// "values" has "n" elements and "pivotIdx" is the index of the pivot element.
// Partition "values" using the value at "pivotIdx" and return the new position
// of the pivot element. Assume that the numbers in "values" are distinct.
```
Problem 5 (25 Points)

Given an binary search-tree, determine whether it is structurally an AVL tree or not. The definition of TreeNode appears on page 9.

bool isAVL(TreeNode *root)
// Is the tree rooted at "root" an AVL tree? Feel free to use helper
// functions in your solution.
Listing 1: Code for problem 1

class Movie
{
public:
    virtual int getPrice(int numDays) = 0; // pure virtual
    virtual int frequentRenterPoints(int numDays) { return 0; }
    std::string title() { return _title; }
private:
    std::string _title;
};

class ChildrensMovie: public Movie
{
public:
    void print()
    {
        std::cout << _title << std::endl;
        std::cout << title() << std::endl;
    }
};

int main()
{
    Movie m;
    Movie *dm;

    ChildrensMovie cm;
    ChildrensMovie *dcm;
    return 0;
}
Listing 2: Artist.hpp

class Artist : public JSONDataObject {
public:
    Artist();
    ~Artist();

    std::string profile();
    std::string artistName();
    std::string realName();
    std::string numImages();
    unsigned artistID();
    void setAlbums(Albums *albums) { _albums = albums; }
    Albums *albums() { return _albums; }

private:
    std::string toString(int v);

    Albums * _albums;
};

Listing 3: Artists.hpp

class Artists : public JSONArray
{
public:
    Artists();
    ~Artists();

    int numArtists();
    void addArtist(Artist *artist);
    Artist * artistWithID(unsigned int aID);
    JSONDataObject * jsonObjectNode() { return new Artist(); }
    void setAlbumsForArtists(Albums *albums);
    std::vector<Artist *> *listOfArtists();
};
Listing 4: Album.hpp

class Album: public JSONDataObject
{
public:
  Album();
  ~Album();

  std::string title();
  std::string genres();
  unsigned albumID();
  unsigned artistID();
  void parseFromJSONstream(std::fstream &stream);
  void setArtist(Artist *artist);

private:
  AlbumImage * _primaryAlbumImage, * _secondaryAlbumImage;
  Tracks * _tracks;
};

Listing 5: Albums.hpp

class Albums: public JSONArray
{
public:
  Albums();
  ~Albums();

  int numAlbums();
  void addAlbum(Album * album);
  void loadAlbumsFromFile(std::string fileName);
  Albums * albumsForArtist(Artist * artist);
  JSONDataObject * jsonObjectNode();
  std::vector < Album *> * listOfAlbums();
  std::string htmlString();
  void runAsserts();
};
Listing 6: TreeNode.hpp

class TreeNode {
  public:
    TreeNode(): left(0), right(0), data(0) {}
    TreeNode( int n ): left(0), right(0), data(n) {}

    TreeNode *leftSubtree() { return left; }
    TreeNode *rightSubtree() { return right; }

    void leftSubtree( TreeNode *left ) { this->left = left; }
    void rightSubtree(TreeNode *right) { this->right = right; }

    int& value() { return data; }

  private:
    TreeNode *left, *right;
    int data;
};