Recall the following BBPlayer class described in lecture.

```
class BBPlayer
{
public:
  BBPlayer();       // Constructor
  void shootFGoal(bool in);
  void shootFThrow(bool in);
  void shootTpointer(bool in);
  bool foulOpponent();
  unsigned int pointsScored() const;
  double fGoalPer() const;
  double fThrowPer() const;
  double tPointPer() const;
  bool fouledOut() const;
private:
  unsigned int FGoalAttempts;
  unsigned int FGoalSuccess;
  unsigned int FThrowAttempts;
  unsigned int FThrowSuccess;
  unsigned int TPointAttempts;
  unsigned int TPointSuccess;
  unsigned int Fouls;
};
```

Write the member function fThrowPer() which should return the percentage of the time that the player made their free throws.

Breifly and in your own words, describe how a **mutator** is used in the context of C++ classes.

Assume that the function printList(), when passed a list, will output each element in the list to a separate line. Indicate the output produced by the following code:

```
#include <iostream>
#include <list>

using namespace std;

void printList(const list<double>& aList);

int main()
{
  list<double> quiz4;
  for(int i=0; i<3; ++i)
    quiz4.push_back(i);
  quiz4.pop_front();
  quiz4.push_front(7.3);
  quiz4.insert(--quiz4.end(),2.5);
  printList(quiz4);

  return 0;
}
```

Below is a function that returns `true` if number is in the vector Vec.

```
bool FindIt(const vector<int>& Vec, int number)
{
  bool found=false;

  for(int i=0; i<Vec.size(); ++i) {
    if(Vec[i]==number) {
      found = true;
      break;
    }
  }

  return found;
}
```

Rewrite the function (including the parameters passed in) so that it returns `true` if number is in the list Lst. If number is not in Lst, the function should return `false`.

List all of the member functions in a base class that are **not** inherited by a class derived from the base class.

In lecture we defined the templated Array class. Listed below is the definition for the Array class with one additional member function, push_back().

```
template<class T>
class Array {
public:
  Array(int n=10, const T& val=T());
  Array(const T A[], int n);
  Array(const Array<T>& A);
  ~Array();
  int size() const;
  Array<T>& operator=(const Array<T>& rhs);
  const T& operator[](int i) const;
  T& operator[](int i);
  void push_back(const T& A);
private:
  int NumberValues;
  T* Values;
};
```

The push_back() member function should add one element to the end of the array. The value of that element is passed in as a parameter. Write the push_back() member function.

Assume Door is class and LockableDoor is a class which inherits the Door class. Given the following code:

```
Door aDoor;
LockableDoor aLckDoor;
Door* DrPtr;
LockableDoor* LckDrPtr;
```

For each expression below, indicate whether it is legal or illegal:

____ aDoor = aLckDoor;

____ aLckDoor = aDoor;

____ DrPtr = LckDrPtr;

____ LckDrPtr = DrPtr;

____ DrPtr = &aLckDoor;

____ LckDrPtr = &aLckDoor;

Briefly describe the meaning of a **pure virtual function**.